Cost Framework for a Distributed Semi-Structured Environment

Tianxiao Liu, Tuyet-Tram Dang-Ngoc, Dominique Laurent

Cost Framework for a Heterogeneous Distributed Semi-Structured Environment

Tianxiao Liu[1], Tuyêt Trâm Dang Ngoc[2], and Dominique Laurent[3]

[1] ETIS Laboratory - University of Cergy-Pontoise & XCalia S.A, France.
Tianxiao.Liu@u-cergy.fr;Tianxiao.Liu@xcalia.com
[2] ETIS Laboratory-University of Cergy-Pontoise, France.
Tuyet-Tram.Dang-Ngoc@u-cergy.fr
[3] ETIS Laboratory-University of Cergy-Pontoise, France.
Dominique.Laurent@u-cergy.fr

Abstract. This paper proposes a generic cost framework for query optimization in an XML-based mediation system called XLive, which integrates distributed, heterogeneous and autonomous data sources. Our approach relies on cost annotation on an XQuery logical representation called Tree Graph View (TGV). A generic cost communication language is used to give an XML-based uniform format for cost communication within the XLive system. This cost framework is suitable for various search strategies to choose the best execution plan for the sake of minimizing the execution cost.

Keywords: mediation system, query optimization, cost model, Tree Graph View, cost annotation

1 Introduction

The architecture of mediation system has been proposed in [Wie92] for solving the problem of integration of heterogeneous data sources. In such an architecture, users send queries to the mediator, and the mediator processes these queries with the help of wrappers associated to data sources. Currently, the semi-structured data model represented by XML format is considered as a standard data exchange model. XLive [NJT05], mediation system based on XML standard, has a mediator which can accept queries in the form of XQuery [W3C05] and return answers. The wrappers give the mediator an XML-based uniform access to heterogeneous data sources.

For a given user query, the mediator can generate various execution plans (referred to as "plan" in the remainder of this paper) to execute it, and these plans can differ widely in execution cost (execution time, price of costly connections, communication cost, etc. An optimization procedure is thus necessary to determine the most efficient plan with the least execution cost. However, how to choose the best plan based on the cost is still an open issue. In relational or object-oriented databases, the cost of a plan can be estimated by using a cost

model. This estimation is processed with database statistics and cost formulas for each operator appearing in the plan. But in a heterogeneous and distributed environment, the cost estimation is much more difficult, due to the lack of underlying databases statistics and cost formulas.

Various solutions for processing the overall cost estimation at the mediator level have been proposed. In [DKS92], a calibration procedure is described to estimate the coefficients of a generic cost model, which can be specialized for a class of systems. This solution is extended for object database systems in [GGT96][GST96]. The approach proposed in [ACP96] records cost information (results) for every query executed and reuses that information for the subsequent queries. [NGT98] uses a cost-based optimization approach which combines a generic cost model with specific cost information exported by wrappers. However, none of these solutions has addressed the problem of overall cost estimation in a semi-structured environment integrating heterogeneous data sources.

In this paper, we propose a generic cost framework for an XML-based mediation system, which integrates distributed, heterogeneous and autonomous data sources. This framework allows to take into account various cost models for different types of data sources with diverse autonomy degrees. These cost models are stored as annotations in an XQuery logical representation called Tree Graph View (TGV) [DNGT04][TDNL06]. Moreover, cost models are exchanged between different components of the XLive system. We apply our cost framework to compare the execution cost of candidate plans in order to choose the best one.

First, we summarize different cost models for different types of data sources (relational, object oriented and semi-structured) and different autonomy degrees of these sources (proprietary, non-proprietary and autonomous). The overall cost estimation relies on the cost annotation stored in corresponding components TGV. This cost annotation derives from a generic annotation model which can annotate any component (i.e. one or a group of operators) of a TGV.

Second, in order to perform the cost communication within the XLive system during query optimization, we define an XML-based language to express the cost information in a uniform, complete and generic manner. This language, which is generic enough to take into account any type of cost information, is the standard format for the exchange of cost information in XLive.

The paper is organized as follows: In Section 2, we introduce XLive system with its TGV modeling of XQuery and we motivate our approach to cost-based optimization. In Section 3, we describe the summarized cost models and show how to represent and exchange these cost models using our XML-based generic language. Section 4 provides the description of TGV cost annotation and the procedure for the overall cost estimation at the mediator level. We conclude and give directions for future work in Section 5.

2 Background

XQuery processing in XLive A user's XQuery submitted to the XLive mediator is first transformed into a canonical form. Then the canonized XQuery is

modeled in an internal structure called TGV. We annotate the TGV with information on evaluation, such as the data source locations, cost models, sources functional capabilities of sources, etc. The *optimal* annotated TGV is then selected based on a cost-based optimization strategy. In this optimization procedure, TGV is processed as *the logical execution plan* and the cost estimation of TGV is performed with cooperation between different components of XLive. This *optimal* TGV is then transformed into an execution plan using a physical algebra. To this end, we have chosen the XAlgebra [DNG03] that is an extension to XML of the relational algebra. Finally, the physical execution plan is evaluated and an XML result is produced, Fig.1 depicts the different steps of this processing.

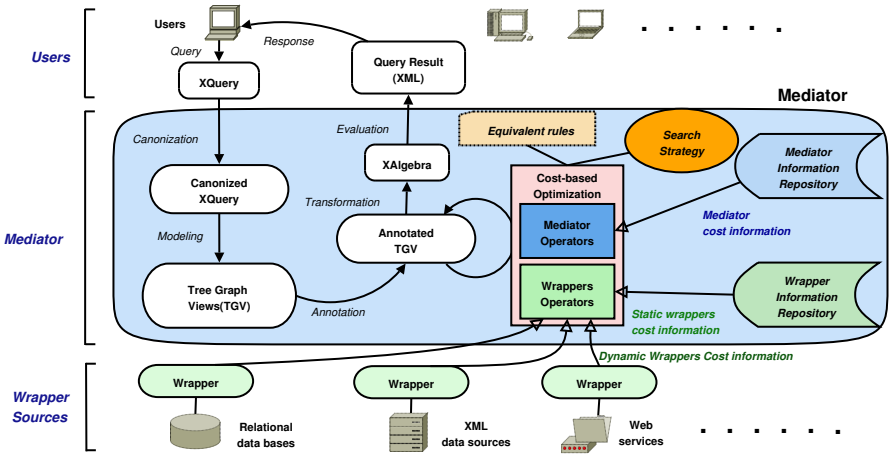


Fig. 1. Cost-based optimization in processing of XQuery in the XLive system

Tree Graph View TGV is a logical structure model implemented in the XLive mediator for XQuery processing, which can be manipulated, optimized and evaluated [TDNL06]. TGV takes into account the whole functionality of XQuery (collection, XPath, predicate, aggregate, conditional part, etc.) and uses an intuitive representation that provides a global view of the request in a mediation context. Each element in the TGV model has been defined formally using Abstract Data Type in [Tra06] and has a graphical representation. In Fig. 2 (a), we give an example of XQuery which declares two FOR clauses ($a and $b), a join constraint between authors and a contains function, then a return clause projects the title value of the first variable. This query is represented by a TGV in Fig. 2 (b). We can distinguish the two domain variables $a and $b of the XQuery, defining each nodes corresponding to the given XPaths. A join hyperlink links the two *author* nodes with an equality annotation. The *contains* function is linked

to the $b "author" node, and a projection hyperlink links the node *title* to the ReturnTreePattern in projection purposes.

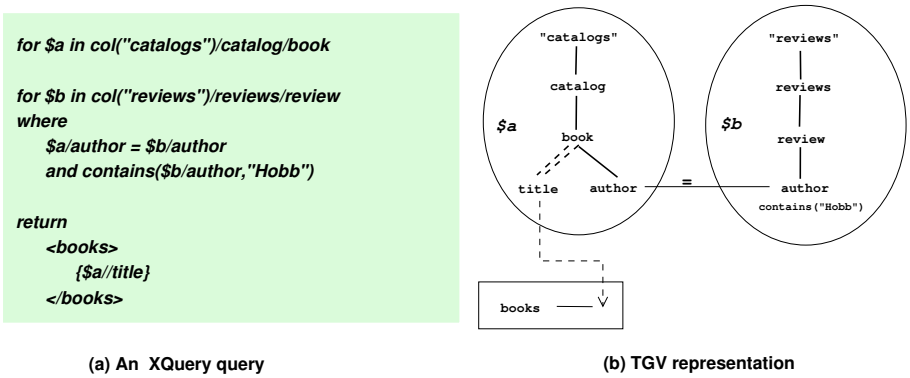


Fig. 2. An example of XQuery and its TGV representation

TGV generic annotation The motivation to annotate a TGV is to allow annotating subsets of elements of a TGV model with various information. Precisely, for each arbitrary component (i.e. one or a group of operators of TGV), we add some additional information such as cost information, system performance information, source localization, etc. Our annotation model is generic and allows annotation of any type of information. The set of annotation based on the same annotation type is called an *annotated view*. There can be several annotated views for the same TGV, for example, time-cost annotated view, algorithm annotated view, sources-localization annotated view, etc.

3 Heterogeneous Cost Models and Cost Communication within XLive

3.1 Cost Models for Heterogeneous Autonomous Data Sources

Cost models summary We summarize different existing cost models for various types of data sources in Fig. 3. This summary is not only based on types of data sources but also on autonomy degrees of these sources. In addition, this summary gives some relations between different works on cost-based query optimization. The cost models with the name "operation" contain accurate cost formulas for calculating the execution cost of operators appearing in the plan. Generally, cost information such as source statistics is necessary for these cost models, because these statistics are used to derive the value of coefficients in cost formulas. It is often data sources implementers who are able to give accurate cost formulas with indispensable sources statistics.

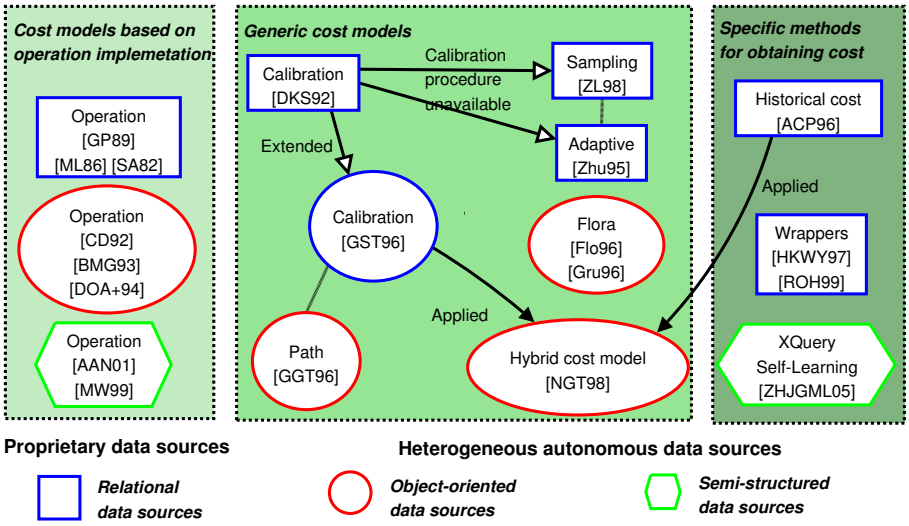


Fig. 3. Cost models for heterogeneous sources

When the data sources are autonomous, cost formulas and source statistics are unavailable. For obtaining cost models we need some special methods that vary with the autonomy degree of data sources. For example, the method by *Calibration* [DKS92] estimates the coefficients of a generic cost model for each type of relational data sources. This calibration needs to know access methods used by the source. This method is extended to object-oriented databases by [GST96]. If this calibration procedure can not be processed due to data source constraints, a sampling method proposed in [ZL98] can derive a cost model for each type of query. The query classification in [ZL98] is based on a set of common rules adopted by many DBMSs. When no implementation algorithm and cost information are available, we can use the method described in [ACP96], in which cost estimation of new queries is based on the history of queries evaluated so far.

Generic cost model Here, we show how to reuse the summary in Fig. 3 to define our generic cost model used for XQuery optimization in the XLive system. First, a cost model is generally designed for some type of data source (but there are also some methods that can be used for different types of sources, for example, the method by history [ACP96]). Second, this cost model can contain some accurate cost formulas with coefficients' value derived from data sources statistics, or a specific method for deriving the cost formulas. This cost model may also have only a constant value for giving directly the execution cost of operators. The possible attributes of our generic cost model are described in Table 1. This descriptive definition of cost model is used for TGV cost annotation for the purpose of overall cost estimation in the mediator level (ref. Section 4).

For a cost model, all attributes are optional by reason of generality. We apply the principle *as accurate as possible*. For example, the method by *calibration* can

normally provide more accurate cost models than the method based on *historical costs*, but it has a lower accuracy level than cost models based on operation implementation. That means if the cost models based on operations implementation are available, we use neither the method by *calibration* nor *history*.

Attribute	Description
Data source type	This type can be relational, object-oriented, semi-structured, files, Web services, etc.
Method	The specific method stored in this field can be used to derive the practicable cost formulas. These cost formulas may be inaccurate, but can at least roughly estimate the execution cost. This respect our *as accurate as possible* principle. Generally, some APIs corresponding to the specific method are available in this field, these APIs are implemented by XLive system and can give some useful services such as "provide the value of coefficients in the formulas".
Formulas	This is the core of a cost model, but they are often unavailable in a heterogeneous environment. These formulas are given in form of equations. The values of coefficients appearing in the formulas can also be represented in form of equations, for example, *Cardinality = 10000*. All these formulas forms an equations system. For some cost models, only a constant cost value is available. This value can be provided by data source (stored in *wrapper information repository*), or derived from results of executed queries (*historical cost*)

Table 1. Definition of generic cost model

3.2 Generic Language for Cost Communication (GLCC)

XML-based generic language To perform cost communication within our XLive system, we define a language to express the cost information in a uniform, complete and generic manner. This language fits to our XML environment, to avoid costly format converting. It considers every cost model type and allows wrappers to export their specific cost information. In our XLive context, this language is generic enough to express cost information of different parts of a TGV and is capable to express cost for various optimization goals, for example, response time, price, energy consummation, etc.

Our language extends the MathML language [W3C03], which allows us to define all mathematical functions in XML form. MathML fits to cost communication within XLive due to its semi-structured nature. We use the *Content Markup* in MathML to provide explicit encoding for cost formulas. We just add some rules to MathML to define the grammar of our language. Furthermore, this grammar is extensible so that users can always define its own tags for any type of cost.

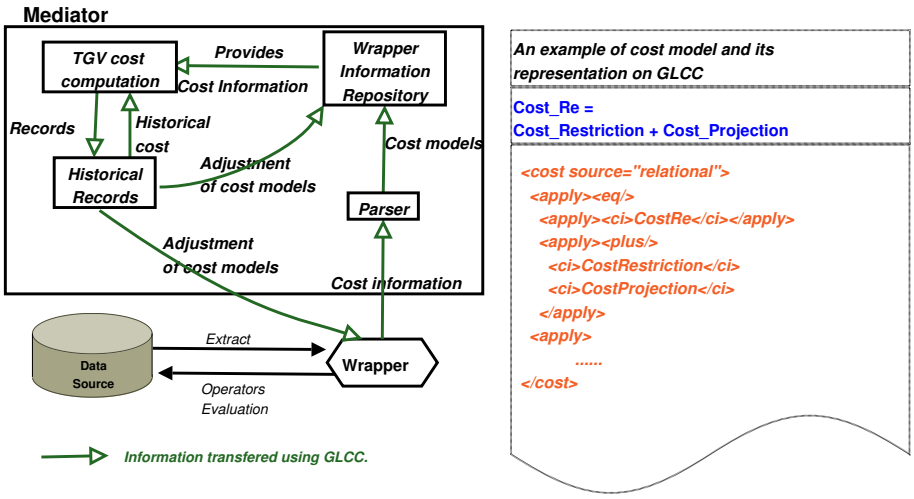


Fig. 4. Dynamic cost evaluation with GLCC in XLive system

Cost formulas are represented in the form of equations set. Each equation corresponds to a cost function that may be defined by the source or by the mediator. Each component of TGV is annotated with an equation set in which the number of equations is undefined. One function in a set may use variables defined in other sets. We define some rules to ensure the consistency of the equations system. First, every variable should have somewhere a definition. Second, by reason of generality, there are no predefined variable names. For example, in the grammar, we do not define a name "time" for a cost variable because the cost metric can be a price unit. It is the user of the language who gives the specific significant names to variables. This gives a much more generic cost definition model compared to the language defined in [NGT98].

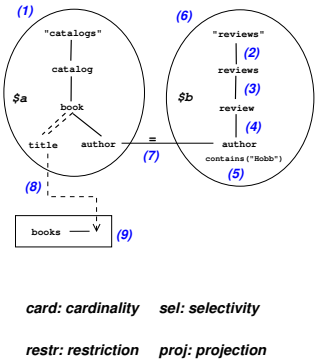
Dynamic cost evaluation Fig. 4 gives a simple example for the expression of a cost model and shows the role of our language in cost communication. After extracting cost information from data source, the wrapper exports that information using our language to the parser, which derives cost models that will be stored in the wrapper information repository. When the mediator needs to compute the execution cost of a plan (TGV), the wrapper information repository provides necessary cost information for operators executed on wrappers. We have a cache for storing historical execution cost of queries evaluated, which can be used to adjust the exported cost information from the wrapper. All these communications are processed in the form of our language. Our language completes the interface between different components of XLive.

4 Overall Cost Estimation

4.1 TGV cost annotation

As mentioned in Section 2, the TGV is the logical execution plan of XQuery within the query processing in XLive. The purpose of our query optimization is to find the *optimal* TGV with the least execution cost. For estimating the overall cost of a TGV, we annotate different components (one or a group of operators) of TGV. For an operator or a group of operators appearing in a TGV, the following cost information can be annotated:

- Localization: The operator(s) can be executed on the mediator or on the wrappers (data sources).
- Cost Model: Used to calculate the execution cost of the component.
- Other information: Contains supplementary information that is useful for cost estimation. For example, several operators' (such as join operator) implementation allows parallel execution between its related operators.



N°	Localization	Cost Model			Other information
		Type	Method	Cost formulas	
(1)	Wrapper	Relational	History	Cost_s1 = return value of API	Dynamic
(2)	Wrapper	XML		Cost_link_review = Cost_IO*card(reviews) Cost_IO = 0.05 reviews = 10000	
(3)	Wrapper	XML		Cost_link_review = Cost_IO*card(catalogs) card(catalogs) = 6000	
(4)	Wrapper	XML	Calibration	Cost_link_author = 2*card(catalogs)*sel sel = 0.2	
(5)	Wrapper	XML		Cost_restr_author = Cost_IO*sel	
(6)	Wrapper	XML		Cost_s2 = (2) + (3) + (4) + (5)	
(7)	Mediator	XML	Calibration	Cost_join = Cost_CPU*(card_s1*card_s2)	Parallel execution enable
(8)	Mediator	XML		Cost_proj = result_card * 0.02	
(9)	Mediator	XML		Cost_global = Max((1), (6)) + (7) + 2*(8)	

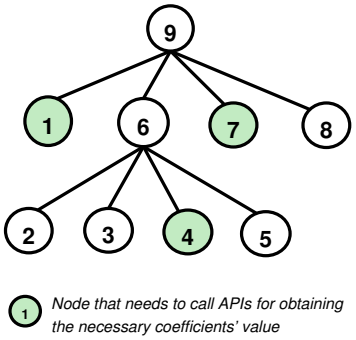
Fig. 5. An example for TGV cost annotation

Fig. 5 gives an example for TGV cost annotation. In this example, different components of the TGV introduced in Fig. 2 (Ref. Section 2) are annotated. We can see for the operators executed on Source1(S1), we have only the historical cost to use for estimate the total execution cost of all the these operators; in contrast, for each operator executed on Source2(S2), we have a cost model for estimating its execution cost. For the join operator(numbered (7)) executed on the mediator, the operators linked to it can be executed in parallel.

4.2 Overall cost estimation

Cost Annotation Tree (CAT) We have seen how to annotate a TGV with cost information. Now we are concentrated on how to use this cost annotation

for the overall cost estimation of a TGV. As illustrated in Fig. 5, the cost of an annotated component of TGV generally depends on the cost of other components. For example, for the cost formula annotated in (6), we see that it depends on the cost of (2), (3), (4) and (5). From the cost formulas annotated for each component of TGV, we obtain a *Cost Annotation Tree (CAT)*. In a CAT, each node represents a component of TGV annotated by cost information and this CAT describes the hierarchical relations between these different components. Fig. 6 (a) illustrates the CAT of the TGV annotated in Fig. 5.



(a) Cost Annotation Tree (CAT)

```
1  associateCost (node) {
2      node.analyzeCostModel ( );
3      if (node.hasSpecialMethod( )) {
4          node.callAPI( );
5      }
6      for (each child of node) {
7          associateCost(child);
8      }
9      node.configCostFormula( );
10     node.calculateCost( );
11 }
```

(b) Overall cost estimation algorithm

Fig. 6. Cost Annotation Tree and the algorithm for overall cost estimation

Overall cost estimation algorithm We now show how to use the CAT of a TGV to perform the overall cost estimation. We use the recursive breadth-first search algorithm of a tree for performing cost estimation of each node. For each node of CAT, we define a procedure called *associateCost* (Fig. 6 (b)) for operating the cost annotation of a node. This procedure first analyzes the cost annotation of the node and derives its cost model (line 2); If a specific cost method is found, it calls an API implemented by XLive for obtaining the necessary values of coefficients or cost formulas for computing the cost (line 3-5); if the cost of this node depends on the cost of its child nodes, it executes recursively the *associateCost* procedure on its child nodes (line 6-8). When these 3 steps are terminated, a procedure *configCostFormula* completes the cost formulas with obtained values of coefficients (line 9) and execution cost of this node will be calculated (line 10). By using this algorithm, we can obtain the overall cost of a TGV, which is the cost of the root of CAT.

4.3 Application: plan comparison and generation

It has been shown in [TDNL06] that for processing a given XQuery, a number of candidate plans (i.e. TGV) can be generated using transformation rules that

operate on TGVs. These rules have been defined for modifying the TGV without changing the result. The execution cost of a TGV can be computed by using our generic cost framework and thus we can compare the costs of these plans to choose the best one to execute the query.

However, as the number of rules is huge, this implies an exponential blow-up of the candidate plans. It is impossible to calculate the cost of all these candidate plans, because the cost computation and the subsequent comparisons will be even more costly than the execution of the plan. Thus, we need a search strategy to reduce the size of the search space containing candidate execution plans. We note in this respect that our cost framework is generic enough to be applied to various search strategies such as exhaustive, iterative, simulated annealing, genetic, etc.

5 Conclusion

In this paper, we described our cost framework for the overall cost estimation of candidate execution plans in an XML-based mediation system. The closest related work is DISCO system [NGT98], which defines a generic cost model for an object-based mediation system. Compared to DISCO work and other mediation systems, we have the following contributions: First, to our knowledge, our cost framework is the first approach proposed for addressing the costing problem in XML-based mediation systems. Second, our cost communication language is completely generic to express any type of cost, which is an improvement compared to the language proposed in DISCO. Third, our cost framework is generic enough to fit to overall cost computation within various mediation systems.

As futur work, we plan to define a generic cost model for XML sources with cost formulas that can compute the cost with given parameters that are components in TGV. This cost model would be generic for all types of XML sources. We will also concentrate on the design of an efficient search strategy that will be used in our cost-based optimization procedure.

Acknowledgment

This work is supported by *Xcalia S.A.* (France) and by ANR *PADAWAN* project.

References

[AAN01] A. Aboulnaga, A. Alameldeen, and J. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. *VLDB*, 2001.

[ACP96] S. Adali, K. Candan, and Y. Papakonstantinou. Query Caching and Optimization in Distributed Mediator Systems. In *ACM SIGMOD*, 1996.

[BMG93] J. A. Blakeley, W.J. McKenna, and G. Graefe. Experiences Building the Open OODB Query Optimizer. In *ACM SIGMOD*, 1993.

[CD92] S. Cluet and C. Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *ACM SIGMOD*, 1992.

[DKS92] W. Du, R. Krishnamurthy, and M.C. Shan. Query Optimization in a Heterogeneous DBMS. In *VLDB*, 1992.

[DNG03] T.T. Dang-Ngoc and G. Gardarin. Federating Heterogeneous Data Sources with XML. In *Proc. of IASTED IKS Conf.*, 2003.

[DNGT04] T.T. Dang-Ngoc, G. Gardarin, and N. Travers. Tree Graph View: On Efficient Evaluation of XQuery in an XML Mediator. In *BDA*, 2004.

[DOA+94] A. Dogac, C. Ozkan, B. Arpinar, T. Okay, and C. Evrendilek. *Advances in Object-Oriented Database Systems*. Springer-Verlag, 1994.

[Flo96] D. Florescu. *Espace de Recherche pour l'Optimisation de Requêtes Objet*. PhD thesis, University of Paris IV, 1996.

[GGT96] G. Gardarin, J.R. Gruser, and Z.H. Tang. Cost-based Selection of Path Expression Algorithms in Object-Oriented Databases. In *VLDB*, 1996.

[GM93] G. Graefe and W.J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *ICDE*, 1993.

[GP89] D. Gardy and C. Puech. On the Effects of Join Operations on Relation Sizes. *ACM Transactions on Database Systems (TODS)*, 1989.

[Gru96] J.R. Gruser. *Modèle de Coût pour l'Optimisation de Requêtes Objet*. PhD thesis, University of Paris IV, 1996.

[GST96] G. Gardarin, F. Sha, and Z.H. Tang. Calibrating the Query Optimizer Cost Model of IRO-DB. In *VLDB*, 1996.

[HKWY97] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimization Queries Across Diverse Data Sources. In *VLDB*, 1997.

[ML86] L. F. Mackert and G. M. Lohman. R* Optimizer Validation and Performance Evaluation for Local Queries. In *ACM SIGMOD*, 1986.

[MW99] J. McHugh and J. Widom. Query Optimization for Semistructured Data. Technical report, Stanford University Database Group, 1999.

[NGT98] H. Naacke, G. Gardarin, and A. Tomasic. Leveraging Mediator Cost Models with Heterogeneous Data Sources. In *ICDE*, 1998.

[NJT05] T.T. Dang Ngoc, C. Jamard., and N. Travers. XLive: An XML Light Integration Virtual Engine. In *Bases de Données Avancées (BDA)*, 2005.

[ROH99] M.T. Roth, F. Ozcan, and L. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *VLDB*, 1999.

[RS97] M. Tork Roth and P.M. Schwarz. Don't Scrap It, Wrap it! A Wrapper Architecture for Legacy Data Sources. In *VLDB*, 1997.

[SA82] P. G. Selinger and M. E. Adiba. Access path selection in distributed database management systems. In *ICOD*, 1982.

[TDNL06] N. Travers, T.T. Dang-Ngoc, and T. Liu. TGV: An Efficient Model for XQuery Evaluation within an Interoperable System. *Int. Journal of Interoperability in Business Information Systems (IBIS)*, 3, 2006.

[Tra06] Nicolas Travers. *Optimization Extensible dans un Médiateur de Données XML*. PhD thesis, University of Versailles, 2006.

[W3C03] W3C. Mathematical Markup Language (Mathml TM) Version 2.0, 2003.

[W3C05] W3C. An XML Query Language (XQuery 1.0), 2005.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, March 1992.

[ZHJGML05] N. Zhang, P. J. Haas, V. Josifovski, and C. Zhang G. M. Lohman. Statistical Learning Techniques for Costing XML Queries. In *VLDB*, 2005.

[Zhu95] Q. Zhu. *Estimating Local Cost Parameters for Global Query Optimization in a Multidatabase System*. PhD thesis, University of Waterloo, 1995.

[ZL98] Q. Zhu and P.A. Larson. Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems. *Distributed and Parallel Databases*, 1998.