Three Light-Weight Execution Engines in Java for Web Data-Intensive data source contents
(Extended Abstract)

Ricardo Ambrose, Philippe Bonnet,
Stéphane Bressan, and Jean-Robert Gruser

Sloan WP #4014 CISL WP #98-03
March 1998

The Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02142

Three Light-Weight Execution Engines in Java for Web Data-Intensive Applications (Extended Abstract)

Ricardo Ambrose, Philippe Bonnet,
Stéphane Bressan, and Jean-Robert Gruser

ambrosia@mit.edu,philippe.bonnet@dyade.fr,
steph@context.mit.edu,gruser@umiacs.umd.edu

1 Introduction

Everything is on the Web. Almost. Large amounts of data are available for potential use for scientific and business purposes. Applications range from the search and consolidation of the publicly available scientific data such as the human genome data, to, as in the example below, investment research involving stock prices, economic indicators, and analysts recommendations. Nevertheless, the question remains of how to use this wealth of data in applications that require a structured access as opposed to the casual browsing supported by the usual World Wide Web interfaces.

It seems indeed oxymoronic to expect a structured query paradigm for accessing the Web when the original paradigm is the one of hypertext navigation. Of course, the Web is a structured set (a graph) of "documents", some of which, in their turn, are structured from the standpoint of their layout defined by the Hypertext Markup Language (HTML). Recent advances in the study of so called *semi-structured data* [Suc97] enable pragmatic solutions for identifying and extracting structured content in the absence of an explicit reference data model or of a known schema. For accessing Web sources, the prototypes presented in this paper rely on such solutions one of which is discussed in [BB97].

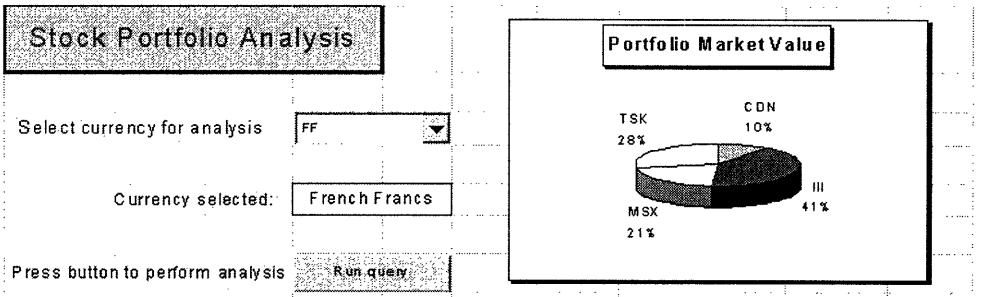
When we accept the hypothesis that data from the World Wide Web are available, we are still confronted with two problems posed at an unprecedented scale: heterogeneity and distribution. In other words, as the Web and its widely accepted protocol HTTP offer a first level of physical connectivity, both logical and semantic connectivity need to be achieved. In provocative terms, we ask if we could turn the Web into a global federated database.

The COIN project at MIT [GBMS97], the Disco project at Dyade [TRV96], and the WinMWrap project at the University of Maryland [RVG] are developing prototype mediation ([Wie92]) networks.

In each of these projects, one of the challenges we are confronted with is the design and implementation of a structured query processing engine providing the basic functionalities for the construction of mediators and wrappers.

The research issue is to devise new planning and optimization strategies by extending existing techniques for database systems [JK84] to support the efficient processing of queries to multiple distributed and independent sources such as Web sites. The pragmatic issue is to design and implement a query execution engine under the constraints and requirements introduce by the nature and use of the global information infrastructure.

In this paper we present a synthesis of the major design decisions taken for the implementation of the query execution engines of these three prototypes. Section 2 briefly outlines an example of a practical application that illustrates the kind of scenarios we are considering. Section 3 reviews the generic architecture of the three prototypes. In section 4 we focus on the design and implementation of



Prices, market value, and gain/loss expressed in French Francs

Stock	Ticker Symbol	Shares Held	Purchase Price	Purchase Opinion	Current Price	Current Opinion	Market Value	Gain / Loss
Cadence Design Systems, Inc.	CDN	5	202.59	1.3	212.07	1.3	1,060.35	47.40
Insteel Industries, Inc.	III	100	44.45	1.0	45.18	1.0	4,518.00	73.00
MascoTech, Inc.	MSX	20	112.96	1.4	114.41	1.6	2,288.20	29.00
Computer Task Group, Incorporated	TSK	10	284.94	1.0	303.16	1.0	3,031.60	182.20
						Portfolio Totals	10,898.15	331.60

Figure 1: Excel Spreadsheet

the execution engines emphasizing the commonalities between the three approaches and identifying some of the original contributions. Section 5 overviews some of the dynamic optimizations features present in the three systems.

The techniques we are discussing could also apply to the implementation of other systems and languages for accessing data over the Internet such as WebLog [LSS96], Tsimmis [HGMC+97], or WebSQL [MMM96], for instance.

2 Example

In this section, we present an example application adapted from the demonstration of MIT's prototype (called MINT [BFG+97]). An investor wishes to monitor the activity of his portfolio. The portfolio is composed of stocks from the New York Stock Exchange (NYSE). He also wishes to watch some analysts' recommendations for the stocks he owns. In addition, he would like to monitor the market value of his portfolio in various currencies in order to integrate the variation of the currency exchange rates as he considers to transfer his assets abroad. Finally he would like to visualize the daily results into a spreadsheet accompanied by various charts and pies (see figure 1).

The desired information is available from various sites on the World Wide Web. The NYSE Web site [NYS98] provides general data about the stocks such as ticker symbols, company names, industry sectors, or number of outstanding shares. Among many other Web sites, the CNN Financial Network [CNN98] provides, free of charge, 15 minutes delayed stock last prices and related data (high, low, P/E ratio, etc). Zacks Investment Research publishes quantified recommendations for stocks in the American stock exchanges on its Web site [Res98]. Finally, Olsen Associates [Ols98] provides current and historical exchange rates between hundreds of major currencies.

The user might stores his portfolio data in a local database (as simple as a file or a local Web page if no database management system is available). With this data at hand the user can compose and program his spreadsheet, copying the latest data by hand from the numerous individual Web pages. The task is repetitive and tedious. The availability or the construction of a structured query interface to these data highly simplifies the task and enables more ambitious applications.

Indeed, provided that structured views of the Web sites are available, the information needed to refresh the spreadsheet can be expressed by one or more SQL queries. Let us for instance retrieve the

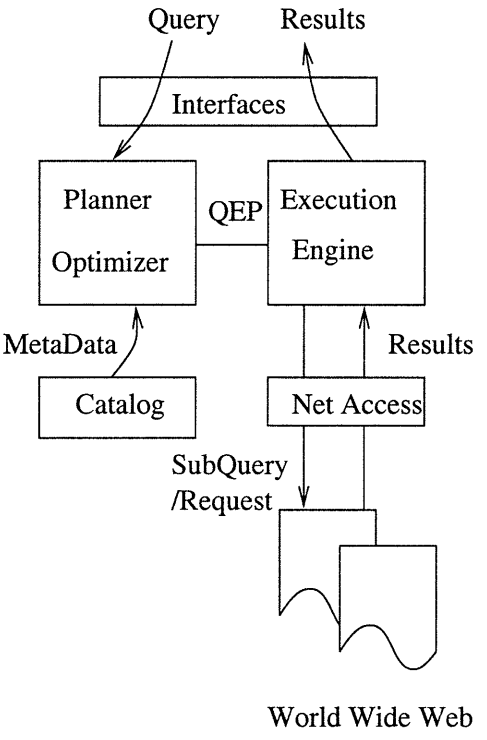


Figure 2: Architecture

current market value in French Francs and analysts' recommendations for each of the stocks owned:

```
Select Local.Ticker, Nyse.CompanyName, Local.Qty * Cnn.Last * Olsen.rate, Zacks.Rec
From Nyse, Cnn, Zacks, Olsen
Where Olsen.Expressed='USD' and Olsen.Exchanged = 'FRF' and
      Nyse.Ticker = Local.Ticker and
      Cnn.Ticker = Nyse.Ticker and  Zacks.Ticker = Nyse.Ticker;
```

As illustrated on figure 1, after selecting a currency, the content of the spreadsheet is refreshed by a single click on the `run query` button. All embedded SQL queries are submitted to the query processing engine using its ODBC front-end interface. At the back end, the execution engine retrieves data from the various Web sources and recombines them according to the queries. The results of the queries are then displayed in the spreadsheet. They are aggregated and presented in a pie chart representing the repartition of the market value of the portfolio among the various stocks. More data such as business news headlines about the companies, number of outstanding shares, 52 week high, or earnings per share are available for more a realistic portfolio management.

The effectiveness of such an approach depends on the features and the performance of a structured query processing tool for Web sources.

3 Architecture

The generic architecture of the three prototypes we are presenting is illustrated in figure 2. We see four main groups of processes: the interfaces, the catalog manager, the planner-optimizer, and the execution engine. The interfaces are application programming interfaces or user-interfaces connecting the systems to the required front-ends (desktop applications, proprietary application programs, scripts,

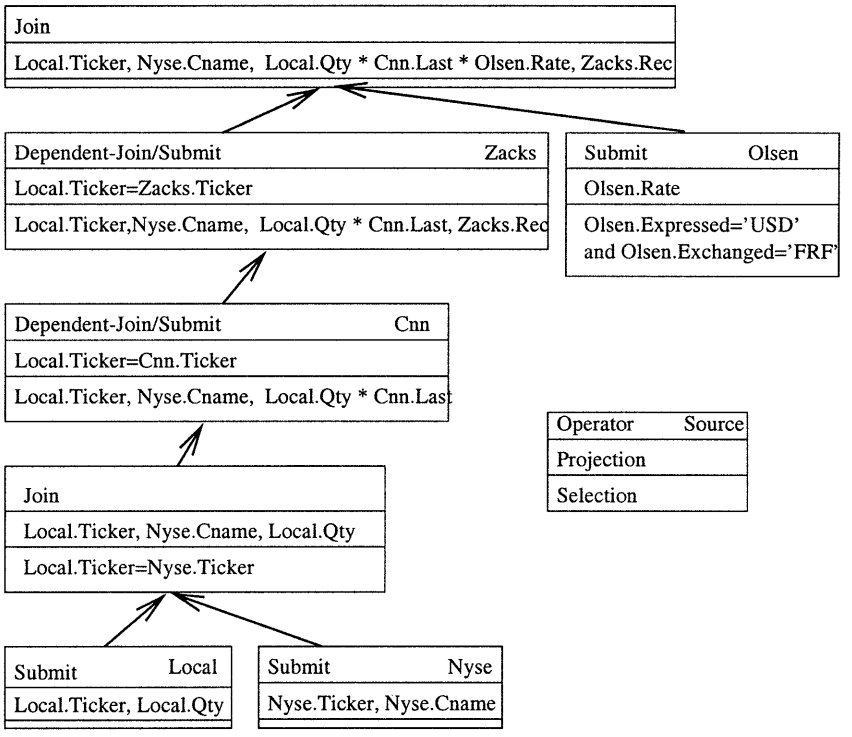


Figure 3: Query Execution Plan

or Web browsers). The catalog manager provides the other processes with the necessary meta-data such as the address and specifications of the available sources. The planner-optimizer compiles the query into a query execution plan (QEP) according to the constraints imposed by the capability restrictions of the sources (what sub-queries/requests a source can answer [PGH96]), and to heuristics and possibly a cost model of the physical operators and network access (e.g. [NGT98]). The execution engine is in charge of the evaluation of the QEP, i.e of accessing the sources over the network, extracting the data, and composing an answer to the query. The data extraction capabilities may alternatively be carried out by external components called wrappers not represented on the figure.

Let us, examine the steps involved in the processing of the query in the example of the previous section. The query from the spreadsheet is submitted through the ODBC interface (or in the case of Excel'97 through the new Web query interface) to the planner-optimizer. The planner-optimizer parses the query and gathers the necessary meta-data about the relevant sources: the local database `Local`, and the four Web sites. The planner recognizes that the currency must be known before querying the `Olsen` site. The query provides the two currencies. The planner recognizes that the Web sites `Cnn` and `Zacks` require that the ticker symbol is known. Such a binding can be provided by either the local database or the NYSE site. The optimizer decides that the local database is more selective than the complete listing of the NYSE. The plan on figure 3 is generated.

4 The Query Execution Engine

Let us digress at this point on some of the main reasons that motivated our design and implementation choices and in particular the choice of Java. The primary objective was to develop a *light weight* execution engine, i.e. an engine that could seamlessly be integrated into several architectures including client software deployed on personal computers. This requirement justified the choice of Java from two

points of view. First Java's *portability*, i.e. the fact that its abstract machine is now almost transparently supported for most operating systems, conferred portability to our code. Second, the various Java APIs, JavaBeans, JDBC, RMI, allow the seamless integration of our code in many configurations. We were ready to abandon, at least temporarily, some of the difficult requirements for a conventional query execution engine and designed main memory, single user systems (although these two aspects appear now quite simple to introduce). As a high level object oriented language, Java offered the necessary constructs for the implementation of our prototype. The absence of explicit memory management allowed the rapid prototyping of the systems although we may expect it becomes a pitfall in the future as we may wish more control on the system's use of the resources. Finally, the availability of threads and synchronization primitives greatly simplified the implementation of concurrency.

4.1 The Logical Operators

The logical algebra we consider is the set of the standard operators on collections of structured data. It applies, at this level of generality, to both relational and object-oriented data models. Namely we consider operators such as selection, projection, union, or join. Depending on the type of collection, these operators will or will not realize duplicate elimination (Dyade's and Maryland's prototype consider an object oriented data model and the corresponding algebra, MIT's prototype considers a strict relational model). Operators or operations combined with the above mentioned operators which are specific to certain data models such as the nest and unnest operators for nested data structures or path expressions are not considered in this discussion. Similarly object methods are not discussed. We only assume that a minimum set of operations are available for the basic types of the model allowing for instance the evaluation of arithmetic expressions and the evaluation of comparisons and boolean expressions to verify the conditions of a selection or a join. Also we recognize the importance of aggregation functions and operators in the kind of applications we are targeting, we will not elaborate on their implementation.The architecture, the design, and the implementation of the three systems we are presenting have made provision for the extension of the prototypes to these features. As a matter of fact some of the features mentioned are already implemented in one or the other system.

4.2 The Physical Operators

A QEP is a tree (or possibly a directed graph in MIT's prototype) of physical operators. The execution engine evaluates the QEP from the leaves to the root. Each physical operator has zero, one, or two sub-trees ordered from left to right. The algebra of logical operators is implemented by means of a set of physical operators. The correspondence between logical and physical operators is many to many. For instance a Theta-join followed by a selection in the algebraic representation of a query is likely to be implemented by a single join-project physical operator. Furthermore, the join-project operator can be one of nested loop join, hash join, merge-join, etc. The design of the physical algebra is guided by the necessity to provide the planner with a sufficient set of operations for the evaluation of a query and by the attempt to provide the optimizer with a sufficient range of options to construct an efficient plan.

The main operators implemented are the selection, the union, the nested loop join, and the hash join. While Dyade's prototype offers an explicit projection operator, projection is combined with each operator in Maryland's and MIT's prototypes. Duplicate elimination is explicitly perform by a separate operator.

At the leaves of the QEP, the submit operators transforms sub-queries into requests to the external information sources and produces the initial data.

Maryland's prototype implements the *dependent-join* operator. This operator is necessary for the compilation and execution of OQL queries. The dependent-join operator transfers data resulting from the evaluation of its first argument (left sub-tree) into its second argument reversing the otherwise general bottom-up flow of data. These data can parameterize the predicates in the conditions of the operators in the right sub-tree. Such an operator is used to implement OQL queries in which collections in the from clause are defined by means of conditional expressions.

When answering queries over Web sites it very often occurs that a set of uniform results is obtained by means of several parameterized requests. In other words, a set of documents obtained by an HTTP varying only by some parameters in the URL or object body is often seen as a single collection. This is the case, in our running example for instance, of the set of stocks prices or the set of recommendations. They are obtained by generating a separate request for each ticker symbol corresponding to a stock in the local database. Practically, such a situation occurs almost systematically when the data is accessible through fill-out forms. In Maryland's prototype this situation is handled by combining a dependent-join and a submit. In MIT's prototype, a specific operator (we call it a posteriori *dependent join-submit*). Dyade's prototype does not currently handles such a situation.

Finally results are accessible from the interfaces by means of cursors.

4.3 The Execution Model

The execution model is the iterator model describe by Graefe in [Gra93]. This model is characterized by a pipelined production of the results. In its sequential version, implemented in Dyade's prototype, each operator is equipped with three methods: `open()`, `close()`, and `get_next()`, to initialize, finalize, and request the production of data. Maryland's and MIT's prototypes implement inter-operator concurrency resulting in a data driven model. The operators are concurrent processes eager to consume the available data and communicating by means of buffered data streams. The size of each buffer given in numbers of data elements (tuples) is fixed in the QEP.

The basic model for a buffered data stream is the one of a producer consumer monitor. The following is an outline of (a possible) code for the buffered data stream. The main methods are `get_next()`, and `put_next()` which requests a data element from the buffer or inserts a data element into the buffer, respectively. The `wait()` and `notify()` methods of the synchronization variables respectively stops and restarts the threads accessing the monitor.

```
class buffered_data_stream {
private Buffer Buffer;              % storage
private int n = 0;                  % number of elts in the storage
private sync non_empty = new sync();  % synchronization variable
private sync non_empty = new sync();  % synchronization variable

public buffered_data_stream(int: Size) {% object constructor}
public Data get_next() {
        if (n == 0) {non_empty.wait();}
        d = Buffer.Get(); n--;
non_full.notify();
        return d;}
public void put_next(Data d) {
        if (n == self.MaxSize) {non_full.wait();}
        n++; Buffer.put(d);
non_empty.notify();}}

class sync {
public void synchronized notify() {notify();}
public void synchronized wait() {wait();}}
```

On a single processor machine the concurrent version will not be more efficient than sequential one as far as the local processing is concerned. On the contrary, there is an overhead in scheduling the various threads. However the model is fairer in that it produces early results as soon as possible and is able, in this task, to avoid bottlenecks created by datasources with low data rates (e.g. "slow" Web sites). However, as most of the time is expected to be spent on the network retrieving data, a concurrent

broadcasting of the requests often results in an improved performance (although this is bound by the network bandwidth and the servers' capability to handle multiple requests).

The reader has noticed that no special effort has been made to explicitly control the concurrency. The task is left to the Java thread scheduler. However, several elements in the plan, such as the choice of an operator or the size of a buffered data stream will influence (in a distributed manner as opposed to a centralized scheduling) the concurrent behavior. For instance, the systematic choice of buffers of size 1 synchronizes waves of single data production.

In order to further increase the concurrency, MIT's prototype offers a concurrent nested-join operator which opportunistically uses the next available data from produced by either of its subtrees. This form of concurrency is called intra-operator concurrency.

Finally, we have not been able to experiment with parallel implementations of Java in which the threads can take advantage of multiple processors. It must be clear that the standard Java abstract machines do not necessarily take advantage of the multiple processors available on a machine nor do they usually take advantage of the system native threads (as for instance the new Sun Solaris Java Thread library). However, Dyade's prototype has provision for the parallel evaluation of a QEP on a set of distributed machines. This feature takes the form of a control operator `exec()` delegating a portion of the plan to a remote machine.

5 Dynamic Optimization

Whether the engines we are describing are going to be used for ad hoc queries , i.e queries which are not known ahead of time and therefore need to go through the entire process of compilation, planning and optimization, and execution, or queries which can be compiled, planned and optimized in advance (queries which are systematically reused such as, for instance, a query monitoring the activities of a portfolio), the availability of dynamic optimization mechanism is critical for the system to be able to react to unpredictable and uncontrolled behaviors and performances of the network and sources. Every user of the Web has experienced slow or non-responding servers and networks being randomly congested. Given that many of these phenomena are largely unpredictable, provision must be made for allowing the system to react at dynamically.

One simple way of reducing the risk incurred is to eliminate unnecessary access. As sub-queries and requests, as we have seen may not be fully determined at planning-optimization time, factoring may not be optimum in the QEP. A simple predicate caching mechanism at the back-end of the execution recognizes sub-queries and requests whose answers can be obtained from previously cached results. Our experience has demonstrated that such a features drastically improve the overall performances of the system despite the cost of the query subsumption tests. This feature is implemented in MIT's prototype.

A more general solution to the problem of unexpected behaviors of the network and sources is the dynamic reorganization of the QEP. The prototype of the University of Maryland has provision for the implementation of a *query scrambling* mechanism [AFTU96] which dynamically reorganizes a sub-tree as runtime information is available about the sources.

In many cases sources are not-responding. recognizing that useful work towards answering the original query can still be done with the remaining sites, Dyade's prototype implements a particular form of query scrambling [ABF+97] which computes a *partial answer*. A partial answer is twofold: it contains an incremental query that can be later processed in order to obtain the complete answer, as well as some data that could be obtained from the available sites using an auxiliary query. The auxiliary query is called a *parachute query*. The algorithm for computing valid parachute queries is described in [BT97]. Assuming, in our running example, that the source for recommendations (Zacks) is unavailable, a parachute query would, for instance, retrieve only the last prices of stocks for the portfolio.

6 Conclusion

Many reason can be advocated to argue that attempting to provide a structured query mechanism on top of semi-structured Web sources is foolish. Primarily, there is a possibility that the development of the Web will favor the definition and adoption of an Electronic Data Interchange standard. In the philosophy of the Web and HTML, XML could be the candidate. Information providers would then provide direct access to the data in the databases behind their Web site directly through this standard. Commercial applications would provide some of the functionalities we have been discussing here. In the meantime, our prototypes offer practical alternatives. The choice of Java as the implementation language enables a simple installation of our tools as independent client applications. We have been able to build several practical applications such as bargain finders, customized newspapers, etc.

Interestingly enough, users may be disappointed by the performances of our systems. Paradoxally, as they now just need to formulate their query and wait for the results, not being entertained by the bells and whistles of the various pages they would need to visit while browsing, they often become more sensitive to evaluation time although the system can perform several factors a magnitude faster than they would.

This observation makes more critical than ever the need for effective optimization strategies. Not only logical (general heuristics) and physical (cost based models) optimizations techniques should be developed or adapted, but also, semantic optimization should be considered in order to take advantage of knowledge and assumptions about the network, the sources and their content just as a user browsing the Web does. Finally, the fundamental difficulty will reside in the combination of the three forms of optimization.

References

[ABF+97] L. Amsaleg, Ph. Bonnet, M. Franklin, A. Tomasic, and T. Urhan. Improving responsiveness for wide-area data access. *Bulletin of the IEEE Technical Committee on Data Engineering*, 1997.

[AFTU96] L. Amsaleg, M. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. *in Proceedings of Intl. Conf. on Parallel and Distribution Information Systems (PDIS)*, 1996.

[BB97] S. Bressan and Ph. Bonnet. Extraction and integration of data from semi-structured documents into business applications. In *in Proceedings of the Intl. Conference on Industrial Applications of Prolog*, 1997.

[BFG+97] S. Bressan, K. Fynn, C. Goh, M. Jakobisiak, K. Hussein, H. Kon, S. Lee, T. Madnick, T. Pena, J. Qu, A. Shum, and M. Siegel. The context interchange mediator prototype. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, 1997. (see also http://context.mit.edu/demos/sigmod).

[BT97] Ph. Bonnet and A. Tomasic. Partial answers for unavailable data sources. Technical Report RR-3127, GIE Dyade-INRIA, 1997.

[CNN98] CNN. www.cnnfn.com, 1998.

[GBMS97] C. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. Technical Report CISL WP97-03, MIT Sloan School of Management, 1997.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2), 1993.

[HGMC+97] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semi-structured information from the web. In *Proceedings of the Workshop on Management of Semi-structured Data*, 1997.

[JK84] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2), 1984.

[LSS96] L. Lakshmanan, F. Sadr, and I. Subramanian. A declarative language for querying and restructuring the web. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering (RIDE'96)*, 1996.

[MMM96] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings PDIS'96*, 1996.

[NGT98] H. Naacke, G. Gardarin, and A. Tomasic. Leveraging mediator cost models with heterogeneous data sources. In *Proceeding of the Intl. Conf. on Data Engineering (ICDE)*, 1998.

[NYS98] NYSE. www.nyse.com, 1998.

[Ols98] Olsen. www.oanda.com, 1998.

[PGH96] Y. Papakonstantinou, A. Gupta, and L. Haas. Capabilities-based query rewriting in mediator systems. In *Proceedings of the Intl. Conf. on Parallel and Distributed Information Systems*, 1996.

[Res98] Zacks Investment Research. www.zacks.com, 1998.

[RVG] L. Raschid, M. Vidal, and J. Gruser. A flexible meta-wrapper interface for autonomous distributed information source. submitted.

[Suc97] D. Suciu. Proceedings of the workshop on management of semi-structured data, 1997.

[TRV96] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous database and the design of DISCO. In *Proceedings of the 16th Intl. Conf. on Distributed Computing Systems*, 1996.

[Wie92] G. Wiederhold. Mediation in the architecture of future information systems. *Computer*, 1992.