Architecture and Implementation of an XQuery-based Information Integration Platform

Yannis Papakonstantinou
Computer Science and Engineering
University of California, San Diego
yannis@cs.ucsd.edu

Vasilis Vassalos
Information Systems Group, Stern School of Business
New York University
vassalos@stern.nyu.edu

Abstract

An increasing number of business users and software applications need to process information that is accessible via multiple diverse information systems, such as database systems, file systems, legacy applications or web services. We describe the Enosys XML Integration Platform (EXIP), a commercial XQuery-based data integration software platform that provides a queryable integrated view of such information. We describe the platform architecture and describe what the main principles and challenges are for the query engine. In particular, we discuss the query engine architecture and the underlying semistructured algebra, which is tuned for enabling query plan optimizations.

1 Introduction

A large variety of Web-based applications demand access and integration of up-to-date information from multiple distributed and heterogeneous information systems. The relevant data are often owned by different organizations, and the information sources represent, maintain, and export the information using a variety of formats, interfaces and semantics. The ability to appropriately assemble information represented in different data models and available on sources with varying capabilities is a necessary first step to realize the Semantic Web [3], where diverse information is given coherent and well-defined meaning. The Enosys XML Integration Platform (EXIP) addresses the significant challenges present in information integration:

- Data of different sources change at different rates, making the data warehousing approach to integration hard to develop and maintain. In addition, Web sources may not provide their full data in advance. The platform we describe resolves this challenge by being based on the on-demand mediator approach [49, 9, 34, 7, 46, 29]: information is collected dynamically from the sources, in response to application requests.
- The Mediator, which is the query processing core of the EXIP platform, has to decompose application requests into an efficient series of requests targeted to the sources. These requests have to be compatible with the query capabilities of the underlying sources. For example, if the underlying source is an XML

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

file, data may only be retrieved from the file sequentially. All data processing operations on the data of the file will have to be performed by the Mediator. On the other hand, if the underlying source is a powerful SQL DBMS, the Mediator can send to it SQL queries that delegate most of the data processing operations to the SQL query processor, hence providing multiple efficiencies: The amount of data retrieved from the database is potentially much smaller, and the source's processing power and optimization ability is harnessed to answer the integrated query.

In order to address this challenge, the Mediator rewrites the query plan in order to push the most efficient supported query to the underlying sources. In the algebra-based EXIP query processor this is done by having the rewriter/optimizer transform the algebraic expressions to appropriate sub-expressions and "delegate" them to the sources. Wrappers are responsible for translating these subexpressions into SQL or other queries/commands acceptable to the information source [15, 27].

- Integrated views are the key abstraction offered by the EXIP platform, and they often need to be defined over a variety of sources, with different capabilities and access methods, and for use by different applications. The platform makes source metadata available for view definition in an automatic way and simplifies the view definition process via graphical tools.

- Information assets available for integration reside in diverse systems, have different structure, and are usually in heterogeneous formats. The Mediator enables and facilitates the resolution of the heterogeneities by using XQuery to perform complex structural transformations. Furthermore, extensibility of the query engine is required in order to allow easy interface with function libraries built in other programming languages, such as Java, that perform special-purpose, domain-specific transformations.

- The heterogeneity and Web-orientation of modern applications that make use of the EXIP platform for integrated access to diverse information again require a lot of flexibility from EXIP. Different applications use different XML views and queries, which need to structure the XML data as close as possible to the application needs. We have found in particular that, for presentation-oriented Web applications, XML views and queries that structure the data in a way that is "isomorphic" to the HTML structure of the produced Web pages lead to huge time savings in building the applications. Providing the flexibility to produce structures that fit the application requirements again requires a powerful language for selection, join, and transformation.

The EXIP platform uses the XPath/XQuery data model [17], augmented with Skolem functions, which were first proposed in [34] in the context of the OEM model [35]. The platform enables applications and users to access and integrate information using XQuery, the W3C draft proposal for a high-level, declarative XML query language [8]. XQuery statements can reference integrated views, also defined in XQuery with small extensions. Queries and views are translated into the semistructured algrebra used by EXIP, henceforth refered to as *XAlgebra*, are combined into a single algebra expression/plan, and are rewritten/optimized to effect query composition with the views and decomposition into plans appropriate for delegation to the underlying sources.

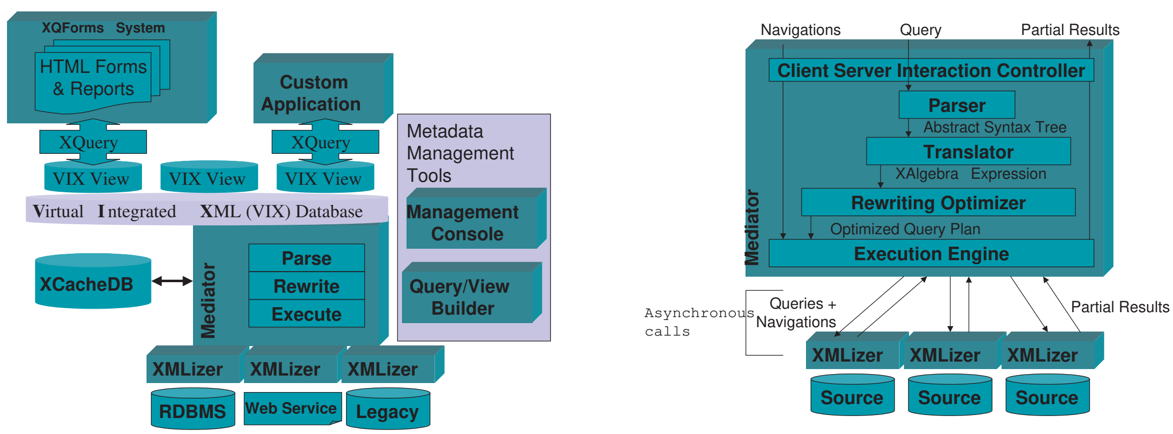
Processing XML query statements in a dynamic mediator context in a way that addresses the challenges mentioned above creates new query processing challenges, as we discuss in Section 3. In addition, the EXIP platform surrounds the Mediator with a series of components that raise the usability of the platform and increase its efficiency.

The EXIP platform is currently in use for a variety of integration projects in large corporations and is being evaluated by two leading software companies for incorporation in their data processing and application server products.

Roadmap Section 2 presents the architecture and components of the EXIP platform. Section 3 presents the internal architecture of the Mediator query processor and introduces the reader to XQuery and the semistructured algebra used in the Mediator's query processing. Section 4 discusses related work.

2 High-Level Platform Architecture and Components

The Enosys XML Integration Platform is based on the wrapper-mediator architecture, as shown in Figure 1(a). Each wrapper, called XMLizer, accesses an information source, such as a relational DBMS or a web service, and exports a Virtual XML view of it. In particular, a wrapper exports an XML Schema describing the content of the source as XML, and it allows querying of that content by the Mediator. It also obtains and exports metadata describing the capabilities of the sources, such as the existence of special access structures or adornments [21] for web services. The Mediator exports the Virtual Integrated XML (VIX) database, which provides access to all the individual views exported by the wrappers. Virtual integrated XML views and queries can be built on top of the VIX database. The views organize information from the distributed sources into XML objects that conform to an application's needs. For example, to a marketplace application the integrated XML view can provide front-end access to an integrated catalog, where the heterogeneities between the suppliers' products are resolved, and the products are integrated and classified according to the needs of the marketplace. Each product object contains catalog data along with attributes from the pricing, delivery, service and other databases. The views provide distribution transparency, *i.e.*, the originating sources and methods of access are transparent to the application. For example, it is transparent to the application that the product specifications in the catalog come from a product database, while the pricing and service information may be coming from a Customer Relationship Management (CRM) system, which often provides customized pricing and service options for each customer.



(a) The High-Level Architecture of the Enosys Platform

(b) The Query Processor Architecture

Figure 1: EXIP architecture

The virtual database and views enable the front-end applications, which may be the XQForms system [40] for declarative generation of query forms and reports or custom applications, to seamlessly access distributed heterogeneous information sources as if they were a single XML database. In particular, the application can issue an XML query against either the XML database or the views. The query typically selects information from the views and structures it in ways that are convenient for the application. For example, a HTML application will create queries that structure the XML results in a way that easily translate into the target HTML pages. In the catalog example, if the resulting HTML page presents the data grouped by product family then the XML result will greatly facilitate the construction of the HTML page if the results are organized hierarchically by product family.

An XML view can be cached into the XCacheDB, which is an XQuery database implemented on top of a relational back-end, as in Niagara [44], employing proprietary storage and query processing and translation algorithms. Typically, the data of slow and relatively static sources are collected, integrated, and cached in advance, while the information originating at fast dynamic sources is collected by accessing the sources dynamically. It is transparent to the application which pieces of the view originate from the underlying sources and which ones originate from XCacheDB.

The Mediator is accessible by applications through a query language API and a DOM-based (Document Object Model) API. Finally, the platform includes management tools that enable the user to graphically create and manage query forms, view definitions, queries, source connections and other metadata.

Example 1: Assume we want to find information about available houses, to guide our home-buying decision-making. Two important considerations are size and price, and we also want to know the quality of the schools in the home's neighborhood. Also assume that

- information on houses and their neighborhoods is accesible via a realtor's database system
- information about school performance is available from an external web service

The following XQuery retrieves the appropriate integrated information:

```
FOR $h IN database("homesDB")/*/homes/home
WHERE
     $h/area > 1,500 AND
     $h/price <500,000
RETURN <home_with_schools>
    {$h},
    {FOR  $s IN database("schoolsDB")/*/schools/school
     WHERE $s/zip = $h/zip
     RETURN
      {$s},
      {ws:schoolreview($s/name)}
    }
</home_with_schools>
```

The XQuery produces `home_with_schools` elements that group houses with the required size and within the right price range with schools in the same zip code and with their reviews. The web service `schoolreview`, offered by an external entity, *e.g.*, the Department of Education, is invoked as an external function, via a simple extension to the function naming mechanism of XQuery. The physical details of the Web service are given in its WSDL description. The school reviews, if the data were available, would also be a good candidate for storage in XCacheDB, for improved performance. A query such as this can be built graphically with EXIP's metadata-aware query and view builder.

For more detail on the components of EXIP, please refer to [39]. In the next section, we discuss briefly how such a query is processed by the EXIP Mediator.

3 Mediator Query Processing Architecture

As we described earlier, the Mediator inputs an XQuery, one or more optional view definitions (also in XQuery), a description of the sources and optional XML Schemas of the sources, and a description of available user-defined functions. It returns the XML query result. The Mediator architecture, shown in Figure 1(b), is similar to the architecture of a traditional DBMS query engine [21], with a few significant differences that are discussed in this section. A query is first parsed into an Abstract Syntax Tree. During parsing, references to data sources and external functions are resolved, using the metadata information available to the Mediator. References to views are also resolved, and view definitions are parsed. Moreover, the parser performs type checks and infers the type of the final as well as of intermediate results [14].

The produced AST is then translated by the *translator* into an XAlgebra expression. The principles of XAlgebra are described in Section 3.2. Consequently, the *rewriting optimizer* applies a series of algebraic and cost-based rewritings on the algebra expression in order to optimize it. We use a rule-based heuristic rewriter that performs optimizations such as

- constant folding
- pushing selections down the algebra expression
- optimizing the join order based on the characterisitcs of the sources (such as the existence of fast access structures)
- unfolding nested XAlgebra expressions
- optimizations targeted for navigation-based evaluation, which is described further in Section 3.1
- choosing the right kind among semantically equivalent operators, *e.g.*, sort-merge join operator or nested loops join operator.

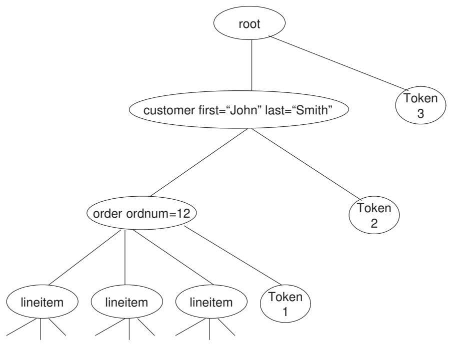
The rewriter performs *capability-sensitive decomposition*, *i.e.*, it decomposes the logical plan into the *maximal* fragments that can be executed by the data sources and the Mediator and respect the capabilities of the data sources. That is, the decomposition pushes as much processing as possible to the data sources, based on their query capabilities. In the case of relational databases, the rewriter also uses heuristics that allow it to take into account the abilities of the query optimizer of the underlying relational database, so that the decomposition produces fragments that are efficiently optimizable by the underlying system (and hence, not necessarily maximal.)

The plan is finally run by the *execution engine*. Conceptually, the execution engine receives the query plan, sends requests/queries to the wrappers, and performs necessary local processing, such as joins across data sources and XML tagging and composition of the result. In practice, the query result object is not fully materialized immediately. Instead, query evaluation is driven by the client navigations, as described below.

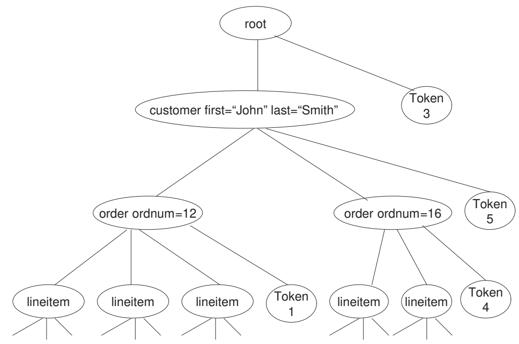
3.1 Navigation-Driven Evaluation

Since Web-based applications are very often accessed by large numbers of users, and often generate correspondingly large numbers of requests to the Mediator, good use of available resources, and in particular main memory and bandwidth, is of high importance. The EXIP Mediator allows "just-in-time" generation of the necessary (parts of the) query result, by integrating querying and result object navigation. In particular, the Mediator, following an initial negotiation with the client, may only send parts of a query result to the client. In place of the missing parts of the results, appropriate tokens are included that the client may send back to the server if it needs one of the missing parts. The tokens contain information needed for tne Mediator to produce the missing parts. The effects of the on-demand, navigation-driven execution model are lower memory footprint and reduced data exchanges from the data sources to the Mediator [32].

A partial result has multiple lists of elements that are *incomplete*, as in Figure 2(a), from where navigation can continue because of the information encoded in the tokens. For example, the Mediator may have exported the partial result of Figure 2(a). If the client navigates to the second child of customer John Smith, the `Token 2` is passed to the Mediator and the Mediator produces another partial result, which leads to a tree such as the one of Figure 2(b).



(a) A partial result



(b) The partial result after navigation to the right of order "12"

Figure 2: Navigation-driven evaluation

Different clients are exposed differently to result fragmentation and navigation-based evaluation, depending on the interaction model used between the client and the Mediator server. Web-based query applications, which use the Mediator as a servlet, [1] benefit from being aware of the existence of tokens and can be made responsible for returning the appropriate tokens to the Mediator via a POST request. This approach fits well with the navigational nature of such applications, *e.g.*, with the existence of "Next" or "Drill Down" hyperlinks that are a common feature of these applications. On the other hand, a general purpose application can be shielded from the existence of tokens: a thin client library is in charge of passing the appropriate tokens to the Mediator server. The client navigates in the result fragment, unaware that parts of the result are not yet in client memory. If the client happens to navigate into a missing part, the library will send the relevant token to the server and fetch the required fragment.

A key challenge in optimizing navigation-driven query evaluation is the choice of the size and shape of fragments produced. At the one extreme, one may choose each node of the data model to be a fragment and encode the relevant token in the node itself. This approach, described in [32], is elegant but it is very inefficient in the number of round-trips that will be needed between the client and the server. Moreover, it penalizes the server with unacceptable overhead in creating tokens.

Instead, the Mediator employs a complex algorithm for *client-server interaction control (CLSIC)*, as shown in Figure 1(b). The CLSIC algorithm is reponsible for choosing the size (and shape) of the fragment that will be returned to the client. The algorithm takes as input configuration parameters provided by the client.

The Mediator execution engine supports CLSIC through a pipelined, iterator-based execution model, which additionally allows the computation state of operators to be exported and imported. Each operator can respond to a call for the "next" tuple (as we discuss in the next section, XAlgebra is tuple-oriented.) Moreover, each operator enables the production of tokens by being able to produce information about its state on command. This state information is encoded by CLSIC in tokens. Each operator is able to reproduce a prior state and continue its computation from that point on. Upon receipt of a token, CLSIC produces and imposes the appropriate state on each operator.

Note that, depending on the capabilities of the underlying sources for on-demand evaluation, the Mediator (and the corresponding XMLizer) sets up and invokes appropriate access methods of these sources, such as SAX calls or SQL cursors. For example, assume that a query produces `customer` elements, from a `customer` table of an underlying relational source. When the client issues the query, the corresponding XMLizer establishes a

[1] Such applications are often implemented using technologies such as Java Server Pages.

cursor and passes this cursor to the Mediator, together with the first fragment of the result, *e.g.*, the first 100 tuples. The Mediator encodes the cursor state information in the token. When the client asks for the 101st `customer` element, the Mediator passes the cursor information back to the XMLizer and asks for the "next" tuple. Note also that the optimization target of the rewriting optimizer can be set to either speed up response time to navigation commands or to speed up production of the full result.

3.2 Principles of XAlgebra

XAlgrebra is a fine-grained algebra used to represent and manipulate queries in the Mediator. XAlgebra is functional, and algebra expressions are fully composable. XAlgebra is also tuple-oriented, meaning that the result of most operators is a set of tuples, and draws on the relational and nested relational algebras [21]. Tuple orientation allows the construction of an iterator-based execution model, which extends the iterator model of relational databases and enables navigation-driven partial evaluation, as described in the previous section. Moreover, it enables the inclusion in the algebra of powerful operators for join and grouping operations, which are much easier specified in terms of tuples and tuple partitions [2, 12]. Finally, it fits better with the underlying relational databases that make an important category of sources for an integration platform. On the flip side, typing becomes more difficult as both tuples and lists need to be typed.

A logical query plan is simply an XAlgebra expression. The input and output of most operators is a set of tuples $\{b_i \mid i = 1, \ldots, n\}$. Each tuple $b_i \equiv [\$var_1 = val_1^i, \ldots, \$var_k = val_k^i]$ consists of variable-value pairs, also referred to as bindings. We say that the variable $\$var_j$ is bound to the value val_j^i in the tuple b_i if the pair $\$var_j = val_j^i$ appears in b_i. All input (resp. output) tuples of an operator have the same list of variables and no variable appears more than once in a tuple. Each value val_j^i can either be a constant, NULL, a single element, a list of elements or a set of tuples. Support for NULL bindings enables easier handling of semistructured data as well as easier implementation of "traditional" operators that are defined using NULLs, such as outerjoins.

XAlgebra contains different implementations of the same logical operation (*e.g.*, grouping, join) as separate operators. This allows the rewriting optimizer to consider the choice of the particular implementation together with other optimization opportunties.

4 Related Work

Data integration has been an important database topic for many years. Most of the early works focused on the integration of structured sources - primarily relational databases. A survey and summary of such works can be found in [47, 22, 31]. In the 90's the scope of data integration systems was extended to include the integration of autonomous and non-structured sources and the "mediator" concept was created [49]. EXIP follows the architecture of earlier virtual view mediators, such as TSIMMIS [20], YAT [9], HERMES [46], Garlic [7], and the Information Manifold [29].

XAlgebra is the cornerstone of query processing in EXIP, similarly to the roles that algebras play in relational [21] and object-oriented databases [12]. Algebras were also designed for the nested relational [42] and complex value models [1]. Recently XML Query Algebras have been proposed as the underlying infrastructure of XML databases and mediators [19, 9, 32]. The common characteristic of those algebras is that the operators input and output sets of tuples. This should be contrasted with functional programming-based XML algebras, such as [4] and [14], which serves as the core of the semantics of the emerging XQuery W3C XML querying standard [8]. In those approaches the operators input and output lists of XML elements. The tuple orientation, which is also present in object-oriented algebras, allows one to carry proven aspects of relational, nested relational and object-oriented algebras into XML processing. For example, it facilitates the specification of a join operator similar to the one of relational algebra. The XAlgebra relates most to the OQL algebra [12] and the XMAS algebra [32].

The component of the rewriting optimizer that is responsible for capability-sensitive decomposition is based on the conceptual background set up in [36, 48, 41], which, in turn, are related to the background created by works on answering queries using views either in the relational model (see [26] for a comprehensive survey) or semistructured/XML models [38, 18]. The system architecture of that component is related to the ones of [24, 9] in the sense that it is built around a rewriting optimizer, such as the one of Starburst [23].

Many of the query processing challenges of the Mediator's query processor are also faced by systems that export an XML view of a single relational source [43, 44, 16, 15].

On the commercial front, other data integration companies and corresponding systems have emerged during the last few years, such as Callixa (`www.callixa.com`) and Metamatrix (`www.metamatrix.com`). More recently, the adoption of XML and its perfect fit to integration applications has led to the emergence of other XML-based information integration companies, such as Xyleme [50] and Nimble [13].

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] F. Bancilhon, S. Cluet, and C. Delobel. A query language for the O_2 object-oriented database system. In *DBPL*, pages 123–138, 1989.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001. Available at `http://www.sciam.com/2001/0501issue/0501berners-lee.html`.

[4] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. ACM SIGMOD*, 1996.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Rewriting of regular expressions and regular path queries. In *ACM PODS Conf.*, pages 194–204, 1999.

[6] M.J. Carey et al. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proc. RIDE-DOM Workshop*, pages 124–31, 1995.

[7] D. Chamberlin et al. XQuery 1.0: An XML query language. W3C working draft, available at `http://www.w3.org/TR/xquery/`.

[8] V. Christophides, S. Cluet, G. Moerkotte, and J. Simeon. On wrapping query languages and efficient xml integration. In *ACM SIGMOD Conf.*, pages 141–152, 2000.

[9] V. Christophides, S. Cluet, and G. Moerkotte. Evaluating queries with generalized path expressions. In *ACM SIGMOD Conf.*, pages 413–423, 1996.

[10] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *ACM SIGMOD Conf.*, pages 177–188, 1998.

[11] S. Cluet and G. Moerkotte. Nested queries in object bases. In *Database Programming Languages (DBPL-4), Proceedings of the 4th Int'l Workshop on Database Programming Languages - Object Models and Languages*, Workshops in Computing, pages 236–242. Springer, 1993.

[12] D. Draper, A. Y. Halevy, and D. S. Weld. The Nimble integration engine. In *ACM SIGMOD Conf.*, 2001.

[13] P. Fankhauser et al. XQuery 1.0 formal semantics. W3C working draft, available at `http://www.w3.org/TR/query-semantics/`.

[14] M. Fernandez, A. Morishima, and D. Suciu. Efficient evaluation of xml middle-ware queries. In *ACM SIGMOD Conf.*, 2001.

[15] M. Fernandez, A. Morishima, D. Suciu, and W. Chiew Tan. Publishing relational data in xml: the silkroute approach. *IEEE Data Engineering Bulletin*, 24(2):12–19, 2001.

[16] Mary Fernndez, Jonathan Marsh, and Marton Nagy. XQuery 1.0 and XPath 2.0 data model. W3C working draft, available at `http://www.w3.org/TR/query-datamodel/`.

[17] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *ACM PODS Conf.*, pages 139–148, 1998.

[18] L. Galanis et al. Following the paths of XML data: An algebraic framework for XML query evaluation. Available at `http://www.cs.wisc.edu/niagara/`.

[19] H. Garcia-Molina et al. The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, 8:117–132, 1997.

[20] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall, 2001.

[21] A. Gupta. *Integration of Information Systems: Bridging Heterogeneous Databases*. IEEE Press, 1989.

[22] L. Haas, J. Freytag, G. Lohman, and H. Pirahesh. Extensible query processing in Starburst. In *Proc. ACM SIGMOD Conf.*, pages 377–88, 1989.

[23] L. Haas, D. Kossman, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. VLDB*, 1997.

[24] Laura Haas, Donald Kossman, Edward Wimmers, and Jun Yang. An optimizer for heterogeneous systems with non-standard data and search capabilities. *Special Issue on Query Processing for Non-Standard Data, IEEE Data Engineering Bulletin*, 19:37–43, December 1996.

[25] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 2001.

[26] J.Hammer, M. Breunig, H.Garcia-Molina, S.Nestorov, V.Vassalos, and R. Yerneni. Template-based wrappers in the tsimmis system. In *Proc. ACM SIGMOD*, pages 532–535, 1997.

[27] H. F. Korth and M. A. Roth. Query languages for nested relational databases. In *Nested Relations and Complex Objects in Databases*, pages 190–204. Springer-Verlag, 1989.

[28] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB*, pages 251–262, 1996.

[29] A. Levy, A. Rajaraman, and J. Ullman. Answering queries using limited external processors. In *Proc. PODS*, pages 237–37, 1996.

[30] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 23:267–293, 1990.

[31] B. Ludascher, Y. Papakonstantinou, and P. Velikhov. Navigation-driven evaluation of virtual mediated views. In *Proc. EDBT Conf.*, 2000.

[32] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. VLDB Conf.*, 1996.

[33] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. ICDE Conf.*, pages 251–60, 1995.

[34] Y. Papakonstantinou, A. Gupta, and L. Haas. Capabilities-based query rewriting in mediator systems. In *Proc. PDIS Conf.*, 1996.

[35] Y. Papakonstantinou, A. Gupta, and L. Haas. Capabilities-based query rewriting in mediator systems. *Distributed and Parallel Databases*, 6:73–110, 1998.

[36] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *ACM SIGMOD Conf.*, pages 455–466, 1999.

[37] Y. Papakonstantinou and V. Vassalos. The Enosys Markets Data Integration Platform: Lessons from the trenches. In *Proc. CIKM Conf.*, 2001.

[38] M. Petropoulos, V. Vassalos, and Y. Papakonstantinou. XML query forms (XQForms): Declarative specification of XML query interfaces. In *Proc. WWW10 Conf.*, 2001.

[39] A. Rajaraman, Y. Sagiv, and J. Ullman. Answering queries using templates with binding patterns. In *Proc. PODS Conf.*, pages 105–112, 1995.

[40] M. A. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13:389–417, 1988.

[41] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In *VLDB Conf.*, pages 261–270, 2001.

[42] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. In *VLDB Conf.*, pages 65–76, 2000.

[43] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proc. VLDB Conference*, 1999.

[44] V.S. Subrahmanian et al. HERMES: A heterogeneous reasoning and mediator system. Available at `http://www.cs.umd.edu//projects/hermes/publications/abstracts/hermes.html`.

[45] G. Thomas et al. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 23:237–266, 1990.

[46] V. Vassalos and Y. Papakonstantinou. Expressive capabilities description languages and query rewriting algorithms. *Journal of Logic Programming*, 43(1):75–123, 2000.

[47] G. Wiederhold. Intelligent integration of information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 434–437, 1993.

[48] L. Xyleme. A dynamic warehouse for XML data of the web. In *IDEAS*, pages 3–7, 2001.