ACE-XQ: A CachE-aware XQuery Answering System *

Li Chen and Elke A. Rundensteiner
Department of Computer Science,
Worcester Polytechnic Institute, Worcester, MA 01609
{lichen|rundenst}@cs.wpi.edu

Abstract. Caching popular queries and reusing results of these previously computed queries to speed up query processing is one important query optimization technique for distributed environments such as the Web. However, existing query-based cache systems, based on query containment and rewriting techniques developed for relational queries, are not appropriate for supporting the more powerful XML queries. We hence propose the first solution for XML query processing using cached XQuery views. In particular, we describe in this paper an XQuery-based semantic caching system called ACE-XQ, that we have implemented to realize the proposed containment mapping and query rewriting techniques. Preliminary experiments confirm the feasibility of our approach and also illustrate the performance gains achievable by ACE-XQ over the original XQuery query engine.

1 Introduction

Due to the growing popularity of XML, many Web applications retrieve desired information from multiple XML sources by issuing queries against remote data sources across the Internet. However, delayed data transmissions and lost data packages often hinder query-embedded Web applications from retrieving such desired remote information in an efficient manner. Hence, the idea of *query-based caching* known for improving performance by orders of magnitude in relational distributed systems [7, 9] could also provide a viable solution for efficient XML query processing in the Web environment.

Related Work. As a fundamental technique underlying *view-based query answering* (or *semantic caching*), *query containment* has been extensively studied for relational conjunctive queries [3, 12, 13]. Even though the topic of semantic caching of web queries has recently gained the attention of researchers, assumptions are usually made that web queries are simple form-based queries. This means they can be either translated into selection SQL queries with simple predicates over form attributes [14] or reduced to a traditional Datalog query evaluation for simple boolean queries [6]. Such an assumption is reasonable when web data sources are stored in back-end relational database systems or when plain web pages are being wrapped to expose limited querying capabilities via a form-based interface.

Other work tackles the query containment problem for semi-structured data [1, 2]. Papakonstantinou et. al. [17] have addressed the query containment and rewriting problem for the Tree Specification Language (TSL). They generalize the relational containment mapping technique [3] to establish mappings between complex object pattern variables in two TSL queries. Florescu et. al. showed in [8] that query containment for a union-free, negation-free subset of StruQL is decidable, and even NP-complete for a significant subset of this language restricting the regular path expressions to contain only * or label constants.

However, no work in view-based query answering for semi-structured data has investigated this issue for state-of-the-art XML query languages such as the recent W3C proposal of XQuery [18], which is capable of expressing complex structural pattern matching, joins of multiple XML data sources and output restructuring. No semantic caching system developed thus far has been targeting XQuery.

XQuery is heavily based on the notation of *expression* which may be nested with full generality. For example, a nested XQuery expression can interleave result construction with further pattern matchings, which is not trivial to be reduced to a datalog-like language like TSL [17]. Also, the regular expression type forms the foundation for XQuery semantics, whereas TSL does not support regular path expressions. Similar to [8], we only deal with the conjunctive, negation-

*This work was supported in part by the NSF NYI grant #IRI 94-57609. Li Chen would like to thank IBM for the IBM corporate fellowship.

free subset of XQuery, where the regular path expressions are restricted to range over tag names and either * or //. On the other hand, we consider the rewriting of a new XQuery in terms of the restructured view schema of a containing query, whereas [8] does not deal with the restructuring capability of StruQL. Furthermore, [8] focuses on the fundamental decidability of query containment for StruQL. Our focus instead is to provide the first practical framework for a cache-aware XQuery answering system, including XQuery pre-processing by normalization, type-enhanced XQuery containment mapping and rewriting, and cache management. A semantic caching system called ACE-XQ [1] [5] has been implemented to realize the proposed techniques.

2 Motivating Example

Suppose we have the cached XQuery `V1` and the new query `Q` as shown in Figure 1. Both `V1` and `Q` involve a "join" of two XML documents which conform to the two DTD structures shown in Figure 2.

```
FOR $b IN $bib0/book, $a IN $b/*[last="James"]
LET $w := (FOR $b' IN $reviews0/book
             WHERE $b'/title=$b/title
             RETURN ⟨result⟩ b' ⟨/result⟩)
RETURN
  ⟨entry⟩ $b/@year, $b/title, $a,
      ⟨reviews⟩
        { FOR $u IN $w/result
          RETURN $u/review }
      ⟨/reviews⟩
  ⟨/entry⟩                                  V1
```

```
FOR $b IN $bib0/book, $a IN $b/author
WHERE $a/last="James"
RETURN $b/title, $a,
  ⟨favorable_reviewers⟩
    { FOR $b' IN $reviews0/book[title=$b/title],
          $r IN b'/review
      WHERE $r/rate >4
      RETURN $r/reviewer }
  ⟨/favorable_reviewers⟩                    Q
```

Figure 1: Cached Query V1 and New User Query Q

`V1` retrieves the `year`,`title`,`author` and `editor` information of the books from `bib.xml` at a remote site (`bib0` denotes the variable bound to the root element of `bib.xml`) whose `author` or `editor` has the `last` name of "James", and the `review` about such books from another remote document `reviews.xml` (the root variable is `review0`). `Q` retrieves the books that are authored by a person with the last name of "James" and highly rated (rate>4) by the reviewers.

[1] Our system was previously called XCache, but has been renamed since that name has been registered with a different product already

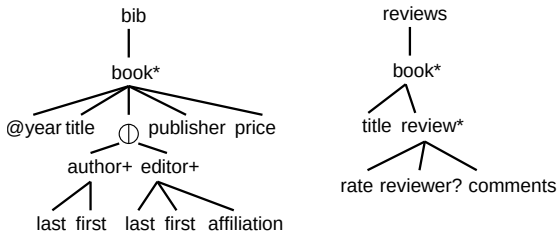

Figure 2: bib.dtd and reviews.dtd

In this example, we can see that the answer required of `Q` is totally subsumed in that of `V1`. In this paper, we sketch out the basics of our XQuery containment and rewriting strategies using this running example.

3 The ACE-XQ Approach

The basic idea of our approach is to explore the containment mappings between the variables specified in the matching patterns of two XQueries for query containment decision and for query rewriting. Since an XQuery defines its variables using regular expressions, our containment mapping strategy incorporates a type-related query analysis, in particular type inference and subtyping relationships, for the purpose of matching regular-expression-type-based variables. To facilitate query reasoning for containment mapping, we first propose a normalized form for XQuery queries, which helps to separate the pattern matching semantics of a query from its restructuring semantics.

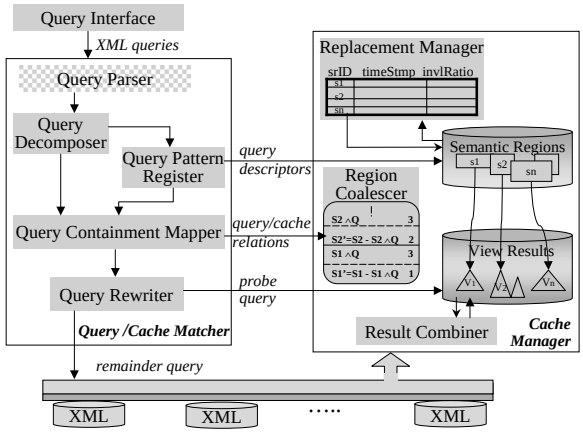

Figure 3: The ACE-XQ Architecture

The framework of the ACE-XQ system is depicted in Figure 3. The **Query Decomposer** first applies normalization rules to an input XQuery to derive a normalized nesting form revealing the query's matching patterns through the specified variables and their dependency hierarchy. It further regroups the conditions and return expressions so to center them around

their referring pattern variables to form variable-specific subqueries. The **Query Pattern Register** uses the pattern variables, their dependency relationships as well as associated conditions and return expressions extracted from the decomposed query to construct the query semantics descriptor. The query's restructuring semantics are also captured separately by nested relational restructuring operators.

For a pair of new and cached queries, the **Query Containment Mapper** explores containment mappings between their pattern variables. It makes the query containment decision depending on whether one-to-one containment mappings can be established. Type inference and subtyping mechanisms supported by a static type checker of XQuery are utilized for this containment mapping between the regular-expression-type-based variables specified in the two queries. Based on the established containment mappings, the **Query Rewriter** rewrites the new query with respect to the possibly severely restructured view structures of the containing cached queries.

The **Replacement Manager** incorporates replacement strategies for purging both complete and partial regions to make space for new queries. For better cache space utilization, the **Region Coalescer** merges and splits views to control region granularity over time.

Due to space limitations, we focus on the description of the query decomposition and containment strategies based on the introduced motivating example. Interested readers are referred to [4] for more details about the containment mapping and rewriting algorithms used in ACE-XQ.

4 XQuery Semantics Analysis

XQuery Pre-processing via Normalization. To facilitate the reasoning between XQueries, we exploit *normalization* rules for transforming an XQuery arbitrarily composed of `FLWR (FOR-LET-WHERE-RETURN)` expressions into a form which separates its pattern matching and restructuring semantics and explicitly exposes the interdependencies between those two semantics.

The XQuery normalization has been addressed both in the W3C proposal for XQuery formal semantics [19] and by Manolescu et. al. [15] to obtain an appropriate XQuery form for which the XQuery-SQL translation can be initiated. The normalization rules proposed by W3C are used for transforming FLWR expressions in the full XQuery syntax into their simplified form in the XQuery Core syntax [19]. Although aiming at different goals, these two normalization techniques do not necessarily exclude each other.

Using the normalization rules provided by [15], the `LET` bound variables can be eliminated by substituting their occurrences for expression definitions. Also, the `FWR` expressions that are nested within a `FOR` or `WHERE` clause can be unnested and put inside of the `RETURN` clause. Reversely, a `RETURN` clause could be unnested as well. For our interest in the XQuery containment problem, we select the unnesting rule for the `FOR` and `WHERE` clauses which simplifies them to contain only path expressions while still capturing the `FWR` hierarchy specified in the original query by nesting `FWR` expressions only within the `RETURN` clauses. An `FWR` expression at any nesting level of such a form can be represented as below:

$$\texttt{FOR } \vec{x} \texttt{ IN } \vec{E}(\vec{v}) \texttt{ WHERE C}(\vec{v'}) \texttt{ RETURN R}(\vec{v'}),$$

where the arity of $\vec{x}$ is the same as that of $\vec{E}(\vec{v})$. The definition of $\vec{x}$ by $\vec{E}(\vec{v})$ can be expanded as `FOR` $\$x_1$ `IN PE1`$(\overrightarrow{v_{preX}})$, $\$x_2$ `IN PE2`$(\overrightarrow{v_{preX}}, \$x_1)$, ..., $\$x_n$ `IN PEn`$(\overrightarrow{v_{preX}}, \$x_1, \ldots, \$x_{n-1})$, where `PEi` denotes the path expression used to define variable $\$x_i$, and $\overrightarrow{v_{preX}}$ represents the variables defined in the outer `FWR` expressions w.r.t. the current `FWR` expression.

For example, during the normalization process for `V1`, the `LET` bound variable $\$w$ is first eliminated and its occurrence in the nested `FOR` clause used to define $\$u$ is substituted for its `FWR` definition. Then the unnesting rule is applied to derive the required normalized form of `V1`, as depicted in Figure 4:

```
FOR $b IN $bib0/book, $a IN $b/*, $t IN $b/title
WHERE $a/last="James"
RETURN
  ⟨entry⟩ $b/@year, $t, $a,
    ⟨reviews⟩
      { FOR $b' IN $reviews0/book
        WHERE $b'/title = t
        RETURN $b'/review }
    ⟨/reviews⟩
  ⟨/entry⟩
```

Figure 4: The Normalized Form for V1

We also provide additional rules to eliminate unnecessary `FOR` expressions if the defined variables do not occur elsewhere, and to add `FOR` and `WHERE` clauses to keep variable definitions free of filter expressions:

```
FOR $a IN $b/*[last=''James'']   ⟹
    FOR $a IN $b/*
    WHERE $a/last=''James''
```

```
FOR $f IN $b/publisher[last=''James'']/first   ⟹
    FOR $p IN $b/publisher, $f IN $p/first
    WHERE $p/last=''James''
```

Variable Dependency. We use $\$bib0 \xrightarrow{/book} \$b$ to denote that variable $\$b$ is dependent on $\$bib0$ via the navigation path `/book`. Similarly, $\$b \xrightarrow{/*} \$a$, $\$b \xrightarrow{/title} \$t$ and $\$reviews0 \xrightarrow{/book} \$b'$. The *pattern matching* semantics of a query are captured by the specified variables and their dependency relationships.

We also break down the `WHERE` and `RETURN` clauses and move condition and return expressions close to the `FOR` clauses where the referring variables are defined. We hence obtain nesting subqueries, each of which clusters all the conditions and return expressions with locally defined variables. Each variable-specific subquery hence encapsulates the *selection* and *projection* semantics of local variables.

In ACE-XQ, we utilize the data structures in Figure 5 to represent the semantic descriptor of a normalized query.

V: variables defined in a local subquery
$VarECRs$: variable specific semantics
$VarDep$: parent and children variables
V_{top}: referred earlier-defined variables

V_{top}
{$bib0}

V
{$b,$a,$t}

$VarDep$

v_p	V_c
$bib0	{$b}
$b	{$a,$t}

$VarECRs$

v_c	$DefExpr$	$Conds$	$Rets$
$b	$bib0/book	{}	{$b/@year}
$a	$b/*	{$a/last="James"}	{$a}
$t	$b/title	{$t=$t'}	{$t}

Figure 5: Variable Dependencies in V1's $subQ_1$

The original nesting `FWR` expressions imply the view structure, which may be affected by the movements of the `WHERE` and `RETURN` clauses into different nesting levels. For example, assuming `V1` returns $\$a$ in the inner `FWR` expression where $\$r$ bound with the `reviewer` objects is to be returned. This indicates a "product" of objects bound to $\$a$ with those *reviewer* objects. We utilize some restructuring operators, such as "product", "nest", "unnest" from the nested relational context, to annotate the effect caused by moving $\$a$ up to the outer `FWR` expression in order to form the $\$a$-specific subquery.

5 Subtyping Enhanced XQuery Containment Mapping

As a regular expression language, XML processing is closely related to tree automata theory [16, 11]. Tree-automata-based regular expression types can be inferred for the variables specified in the matching pattern of an XQuery and for the return type composed of these variables. XDuce [10] provides core functions for such static type checking of XQuery through a functional programming language, and we thus make use of XDuce in our ACE-XQ system.

Our containment mapping strategy proceeds in a progressive fashion guided by the variable dependencies of the new query `Q`. It first matches each variable in `Q` to a variable of the cached query `V1` based on their inferred types from the definitions and their subtyping relationships. It then establishes a containment mapping between a matched variable pair by checking their *selection* and *projection* semantics, similar to the relational view utility conditions.

We now walk through the subtyping-enhanced containment mapping process using the example queries `V1` and `Q` in Figure 1. `V1` and `Q` both query the same XML documents, `bib.xml` rooted at $bib0 and `reviews.xml` rooted at $review0. We illustrate in Figure 6 the graphic presentation of the two queries' variable dependencies as well as the associated conditions and return expressions.

Since variable $\$b$ in `V1` (denoted by $\$b^{V1}$) and $\$b$ in `Q` ($\$b^{Q}$) are defined the same using `PE1($bib0/book)`, we set up a containment mapping from $\$b^{V1}$ to $\$b^{Q}$. We then check whether any condition specified in the $\$b$-specific subquery of `V1` corresponds to some equally or more restricted condition posed on $\$b^{Q}$. In this case, there is no condition for either $\$b^{V1}$ or $\$b^{Q}$. We now check whether the document nodes to be returned by the `RETURN` expression referring to $\$b^{Q}$ are also required in any `RETURN` expression with the occurrence of $\$b^{V1}$. In this case, `R1($b/title)` in `Q` is also required to be returned in `V1`.

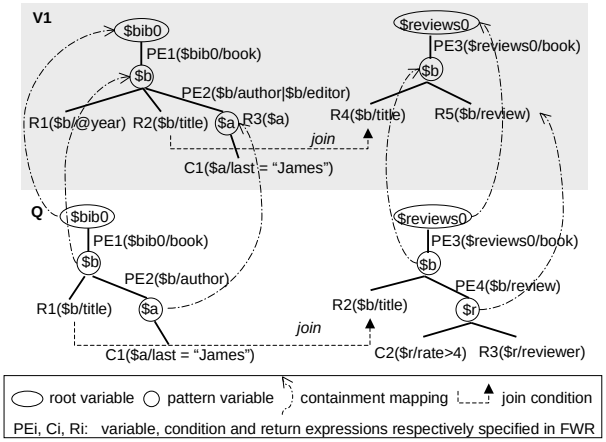

Figure 6: Containment Mappings Between V1 and Q

The join condition on Q's R1($b/title) in source bib.xml and R3($b/title) in reviews.xml match the join condition specified on R2($b/title) and R4($b/title) in V1. We hence continue the containment mapping process to check whether V1's a^{V1} defined as PE2($b/*) matches Q's a^{Q} defined as PE2($b/author). The regular expression type inferred for a^{V1} is (Author+|Editor+) and the type for a^{Q} is Author. Based on the subtyping theory for regular expression types, we find that $a^{Q} <: a^{V1}$, where <: denotes *subtypeOf*. We thus can set up a containment mapping from a^{V1} to a^{Q} and annotate their subtyping relation with a type constraint typeswitch(a^{V1}) as author. If a^{V1} is also returned by V1, which is the case, we rewrite the variable definition of a^{Q} to obtain the document nodes bound to it from those returned by a^{V1} using such a type constraint. We further compare the strictness of conditions attached to a^{V1} and a^{Q}.

A similar containment mapping process proceeds for all the variables defined in reviews.xml by V1 and those in Q. Although r^{Q} has no corresponding variable match in V1, it can be specified on R5($b/review) returned by V1. Therefore, our containment mapping strategy allows for subtyping relations between Q's variables and V1's variables or return expressions, in addition to requiring stricter conditions on Q than on V1. Based on the successful containment mappings between pattern variables of V1 and Q by exploiting their subtyping relations, we determine Q ⊆ V1 [2].

XQuery Rewriting. We now exploit the established containment mappings to obtain a rewriting of Q Q^{rew} (shown in Figure 7) with respect to V1's view structure. In V1's view structure, the association relationship between title and author or editor is preserved inside of a newly constructed element entry. We hence specify a variable $e in Q^{rew} by PE($V1Res0/entry), where $V1Res0 represents the variable for the root element of V1's result document. We then return the title-author pair of each object bound to $e as required by Q. For favorable reviewers, we specify a variable $r in Q^{rew} to be bound to its match in V1, the descendant element <review> of <entry>, restrict its rate with the condition of >4 and return the corresponding reviewer.

```
FOR $e IN $V1Res0/entry
RETURN $e/title, $e/author,
    ⟨favorable_reviewers⟩
        { FOR $r IN $e/reviews/review
          WHERE $r/rate >4
          RETURN $r/reviewer }
    ⟨favorable_reviewers⟩                    Q^rew
```

Figure 7: A Rewriting of Q with Respect to V1

6 Preliminary Evaluation

Our ACE-XQ system is developed using Java JDK 1.3. The front end of the ACE-XQ system is running on top of Apache Web Server and Tomcat servlet engine. We use the Quilt parser and query engine available at *http://cheops.cis.upenn.edu/Kweelt* to analyze and evaluate the input XQuery. We also utilize the type inference and subtyping mechanisms provided by the XDuce system [11] for containment mapping.

Our ACE-XQ system checks the correctness of our rewritten queries by comparing their results with those produced by directly evaluating the original query against the remote documents. It also serves as a testbed for further conducting in-depth experimental studies investigating the query performance gain achieved by answering queries using cached views.

In a distributed system like the Internet, we expect that a query-based caching system like ACE-XQ would save the fetching costs for queries answerable by cached ones. In our experiments, we install the Kweelt query engine on two UNIX machines across a Local Area Network (LAN). One also has the ACE-XQ system installed and allows users to input queries, while the other hosts a set of XML documents as a remote data server.

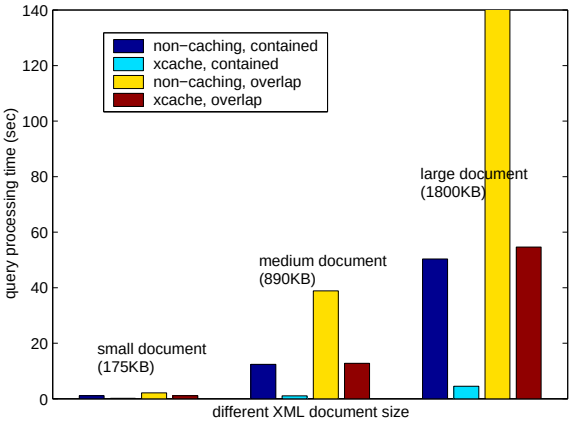

Figure 8: Comparison of Query Processing Times

In our initial experiments, we test different XML document sizes. We warm up ACE-XQ with several queries and design the new queries to be either totally contained in or overlapping with one of the cached

[2] We refer the readers who are interested in other subtle issues of our containment mapping strategy to [4].

XML Doc Size (KB)	Decomposition Time (ms)	Rewriting Time (ms)	Evaluation Time (ms)
for Contained Case			
175	0.8	5.2	173.6
890	0.8	5.4	1068.8
1800	0.8	5.2	4525.4
for Overlapping Case			
175	0.8	1.6	1126.2
890	0.7	1.6	12778
1800	0.8	1.7	54660

Table 1: ACE-XQ Processing Time Components

queries. We compare the query processing time spent when using ACE-XQ versus when by-passing it.

The result in Figure 8 is consistent with our expectation. The query performance improvement for the totally contained case using ACE-XQ is the most significant, by an order of magnitude in a LAN environment. The performance improvement is about 1.8x times for the overlapping cases.

We break down the query processing time used in the ACE-XQ system into three components, i.e., the time used for query decomposition, for query containment mapping and rewriting and for query evaluation respectively. As observed in Table 1, the computational overhead for ACE-XQ is comparatively small with respect to the overall cost.

7 Conclusions

The XQuery-based ACE-XQ query system aims to minimize the fetching cost of XQuery results by answering new queries using cached queries whenever possible. Our preliminary experimental results are promising. Given that our "remote" data sources are on a LAN and hence network delay is minimal, we expect that the performance improvement would be even more dramatic in Web systems over the Internet.

References

[1] D. Calvanese, D. Giacomo, and M. Lenzerini. Rewriting of Regular Expressions and Regular Path Queries. In *PODS, Philadephia, PA*, pages 194–204, 1999.

[2] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Answering Regular Path Queries Using Views. In *ICDE, San Diego, CA*, pages 389–398, 2000.

[3] A. K. Chandra and P. M. Merlin. Optimal Implementations of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.

[4] L. Chen and E. A. Rundensteiner. A Framework for Answering XML Queries Using Views, In Progress. Technical Report 2002-6, Worcester Polytechnic Institute, 2002.

[5] L. Chen, E. A. Rundensteiner, and S. Wang. XCache - A Semantic Caching System for XML Queries. In *SIGMOD demonstration paper, Madison, Wisconsin*, June 2002.

[6] B. Chidlovskii and U. M. Borghoff. Semantic Caching of Web Queries. *The VLDB Journal*, 9(1):2–17, 2000.

[7] S. Dar, M. J. Franklin, and B. Jonsson. Semantic Data Caching and Replacement. In *VLDB, Bombay, India*, pages 330–341, 1996.

[8] D. Florescu, A. Levy, and D. Suciu. Query Containment for Conjunctive Queries with Regular Expressions. In *PODS, Seattle, WA*, pages 139–148, 1998.

[9] J. Goldstein and P. Larson. Optimizing Queries Using Materialized Views: A Practical, Scalable Solution. In *SIGMOD, Santa Barbara, California*, pages 331–342, 2001.

[10] H. Hosoya and B. C. Pierce. XDuce: A Typed XML Processing Language (Preliminary Report). In *WebDB, Dallas, Texas*, pages 111–116, May 2000.

[11] H. Hosoya and J. Vouillon and B. C. Pierce. Regular Expression Types for XML, Montreal, Canada. In *ICFP*, pages 11–22, 2000.

[12] P. A. Larson and H. Z. Yang. Computing Queries from Derived Relation. In *VLDB, Stockholm, Sweden*, pages 259–269, Aug. 1985.

[13] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *PODS, San Jose, CA*, pages 95–104, June 1995.

[14] Q. Luo, J. F. Naughton, R. Krishnamurthy, P. Cao, and Y. Li. Active Query Caching for Database Web Servers. In *WebDB, Dallas, TX*, pages 29–34, 2000.

[15] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries on Heterogeneous Data Sources. In *VLDB, Roma, Italy*, pages 241–250, Sept. 2001.

[16] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML Transformers. In *PODS*, pages 11–22, May 2000.

[17] Y. Papakonstantinou and V. Vassalos. Query Rewriting for Semistructured Views. In *SIGMOD, Philadelphia, USA*, pages 455–466, 1999.

[18] W3C. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, December 2001.

[19] W3C. XQuery 1.0 Formal Semantics. http://www.w3.org/TR/query-semantics/, June 2001.