Querying and Browsing XML and Relational Data Sources*

James J. Lu
Mathematics and Computer Science, Emory University, Atlanta, GA, U.S.A.
jlu@mathcs.emory.edu

Chia-Hsin Huang
Institute of Information Science
Academia Sinica, Taipei, Taiwan
jashing@iis.sinica.edu.tw

Tyng-Ruey Chuang
Institute of Information Science
Academia Sinica, Taipei, Taiwan
trc@iis.sinica.edu.tw

Abstract

A lightweight method for querying and browsing multiple relational and XML data sources is presented. The approach is based on a simple abstraction of relational and XML data models. A query language for the abstraction to which XPath and SQL map to naturally is introduced. A unique feature of the query language is that it unifies structured queries and keyword-based searches.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*

General Terms

Schemaless Data Model, XML, Query Processing

1 Introduction

Most modern database applications require the simultaneous management of diverse types of data. In particular, the need to access both relational and XML data is ubiquitous, and numerous techniques for extending RDBMS as well as native XML and hybrid systems have been developed to address this need. With the exception of the System RX [2], most approaches to integrating XML and relational data are to translate data in one form to data in another, or to extend one data model with new data types to accommodate the other data model (e.g. [5, 6, 18, 20], Tamino[1], Oracle[2]). The goal of these techniques and systems is to provide a full range of data management features for the *precise* handling of both XML and relational data.

In many practical settings, however, a more lightweight approach to interfacing the data sources may suffice. In medical and public-health informatics, for instance, it is often the case that researchers perform *focused browsing* of both relational data (e.g., Electronic Medical Records) along with medical reports structured as XML documents [8]. The researchers do not necessarily know, a priori, the kinds of information for which they are searching. Consequently, they do not demand the precise querying capabilities of XQuery or SQL, nor do they require an "integrated" database of the two sources. At the same time, researchers often wish to perform keyword searches that take into account certain contextual information (e.g., structural or domain specific knowledge of the reports or relations).

In this paper, we present a simple technique for viewing/browsing existing data as a conceptually integrated database, but that can be implemented without the explicit transformation of the data sources. The technique provides a unifying view of XML and relational data, and enables the use of an existing source query over other sources. In particular, existing SQL queries may be executed over XML data sources, and XPath queries over relational data sources.

The key to the approach is a schemaless data model, the Universal Data Model (UDM), in which the purpose of metadata is to describe, not prescribe, the information that exist in the data sources. The same assumption has been adopted elsewhere for data integration and modeling (e.g., [16]). The basic unit of data in the UDM is a generalization of relational tuples, and we introduce a relational-like algebraic language, the UDA, to extract information from the UDM. A unique aspect of the UDA is that it encompasses both structured queries as well as keyword-based search queries naturally. We show how the data model abstracts XML and relational data, and present transformations from SQL and XPath to the UDA. It is through these transformations that the UDM can be implemented as a lightweight abstraction of relational and XML data sources without the need for an explicit materialization of the data sources.

The work is a continuation of our investigation into data integration methods for heterogeneous data models [13] and efficient query processing techniques for XML data [9]. The next section introduces the basics of our data model — the universal data model, including an algebraic language for data manipulation. In Section 3, mappings from relational and XML data are defined. Section 4 details sound and mostly complete translations of relational and XPath queries to the UDA, and a simple implementation for storing data units of the UDM is outlined in Section 4.4. Section 5 concludes with some discussion of related work.

2 The Universal Data Model

Let $\mathcal{S}$ denote a countably infinite set, called *properties*. A *context over* $\mathcal{S}$ is any string over $\mathcal{S}$ (i.e., finite concatenation of properties). Suppose c is the context $p_1.p_2.\cdots.p_n$, $n \geq 1$. We write $v(c)$ to denote p_n. We assume the existence of a binary relation, $\sim$ — the matching relation — on contexts. A *data unit* (or simply unit) is a finite set of contexts, and a universal database (UDB) is a pair $(\mathbf{U}, \sim)$ where $\mathbf{U}$ is a finite set of data units.

A context is a variation of the notion of property in Parsons and

SAC'07, March 11–15, 2007, Seoul, Korea.

[1] www.softwareag.com/corporate/products/tamino
[2] www.oracle.com

Wand's data model [15]. First, it aggregates related properties, thus enabling each property to be understood in the context of other properties. For instance, the property `name` takes on distinctly different meanings in the contexts `name.firstname.John` and `company.name.Microsoft`. Second, the sequencing within each context provides a rudimentary structuring mechanism for capturing "is-a" and "has-a" relationships. For instance, `firstname` "is-a" `name` in `name.firstname.John`, as well `John` "is-a" `firstname`. On the other hand, `company` "has-a" `name` in `company.name.Microsoft`.

The above examples give a very simple intuition for the idea of context. A context $c = p_1.\cdots.p_n$ is a concatenation of "attributes" (i.e., $p_1.\cdots.p_{n-1}$) that describe a "value" (i.e., p_n). The philosophical distinction between attribute and value is not sharp; they correspond, roughly, to what Parsons and Wand refer to as *generic* and *specific* properties. The distinction may help to provide some concreteness when reading, but it is not important in the formal development of the UDM.

As in WHIRL [4], original local names (of both instance data and metadata constants) are retained in a UDB consisting of data derived from multiple sources. Collectively these names form the set $\mathcal{S}$. Determining whether two properties are co-referent is based the matching relation $\sim$ — an abstraction of some arbitrary but fixed matching technique. Choices for the relation include information retrieval techniques such as tf-idf: each context may be represented as a vector of weights, with each weight denoting the relative importance of the corresponding property. The context `name.John`, for instance, will be a (very sparse) vector of two weights, with probably a higher value for `John` given its likely infrequent occurrence in $\mathcal{S}$. Treating each context as a vector of weights ignores the ordering among properties, and in this paper, we are interested utilizing the ordering. To that end, we assume $\sim$ to be string equality in the remainder of the paper.

We may define familiar relational algebraic operations for the UDM. We call the language UDA – the Universal Data Algebra. First, we give the syntax of boolean expressions inductively. Define a *term* to be any finite set of regular expressions over $\mathcal{S}$. Singleton sets are often written without braces.

1. A term is a boolean expression.
2. The expression $b_1 \circ b_2$ is a boolean expression if either b_1 (respectively b_2) is a term, and b_2 (respectively b_1) is a property (written inside double quotes), and $\circ$ is a comparison operator (e.g., $=, <$). We call this a *selection expression*.
3. The expression $b_1 \circ b_2$ is a boolean expression if both b_1 and b_2 are terms, and $\circ$ is a comparison operator. We call this a *join expression*.
4. Expressions formed from boolean expressions and logical connectives (e.g., $\wedge$, $\neg$) in the standard ways are boolean expressions.

Some remarks on notation. We use the symbol `?` to stand for any property. The operator `*` affixed to a context c indicates zero or more repetition of c, but by itself denotes any string of properties. For example, `a*.b` denotes strings with zero or more `a`'s followed by `b`, while `a.*.b` consists of strings that begin with `a`, followed by any context, and ends with `b`. As a special case, we often want to write expressions that begin and end with `*` (e.g., `*.name.*`). For these, we introduce a new symbol, `$`, to simplify notation. For example, `$author.name` denotes `*.author.name.*`.

Suppose b is a term. We say that a context c satisfies b if for each element $b' \in b$, $c \sim c'$ for some $c' \in L(b')$ — the language generated by b'. For instance, `a.b` satisfies the terms `{$a,$b}`, `a*.b`, and `a.*.b`. It neither satisfies `b.a` nor `{a,b}`.

A unit u satisfies a boolean expression b, written $u \models b$, if

1. b is a term, and there exists a context $c \in u$ that satisfies b.
2. b is a selection expression $b_1 \circ b_2$ where b_1 is a term, that there exists a context $c_1 \in u$ that satisfies b_1, and $v(c_1) \circ b_2$ holds. The case where b_2 is a term and b_1 a property is defined analogously.
3. b is join expression $b_1 \circ b_2$, that there exist contexts $c_1, c_2 \in u$ that satisfy b_1 and b_2, respectively, and $v(c_1) \circ v(c_2)$ holds.
4. b consists of boolean sub-expressions and logical connectives, and the recursive truth-functional evaluations of the sub-expressions holds.

EXAMPLE 1. Consider the following unit (contexts numbered for reference).

```
{ bib.author.name.firstname.'Karen',        (1)
  bib.author.name.lastname.'Armstrong',     (2)
  bib.book.title.'The Spiral Staircase'     (3)
  bib.book.genre.'Biography'                (4)
}
```

The unit satisfies the boolean expressions `$name` (by contexts (1) and (2)), `$author = "Karen"` (by context (1)), and `{$title, $book} = "The Spiral Staircase"` (by context (3)). As the latter example shows, a set is useful for specifying, without indicating order, properties that must appear in the same context. The unit does not satisfy the expression `$title.book = "The Spiral Staircase"`, for instance.

The basic operations of relational algebra are easily generalized.

Selection The selection of a set of units U on the boolean expression b, $\sigma_b(U)$, is the set $\{u \in U | u \models b\}$.

Projection Given a unit u and a set of terms $b_1,...,b_n$, $(u|\{b_1,...,b_n\})$ is the unit $\{c \in u | b_i \sim c, \text{ for some } 1 \leq i \leq n\}$.

Then, the projection of a set of units U on terms $b_1,...,b_n$, $\pi_{b_1,...,b_n}(U)$, is the set $\{u \in U | (u|\{b_1,...,b_n\})\}$.

Renaming Given a set of units U and contexts c_1, c_2, $\rho_{c_1 \to c_2}$ is the set of units obtained from U by replacing, for each unit in U, each occurrence of c_1 by c_2. As a special case, c_2 may be the empty string.

Join The join of unit sets U_1 and U_2 on a boolean expression b, $U_1 \bowtie_b U_2$, is the set $\{u | u_1 \in U_1, u_2 \in U_2, u = u_1 \cup u_2, u \models b\}$.

Set Operation Given two sets of units U_1 and U_2, the standard set-theoretic union, intersection, and difference of the two sets are all well defined.

3 Abstracting Relational and XML Data

One may define data units directly in the UDM, but more importantly, the notion of a data unit abstracts familiar concepts such as a tuple in relational databases and a document in XML databases. A tuple t over the schema $r(a_1,...,a_n)$ in a relational database maps to the unit $\{r.a_1.t(a_1),...,r.a_n.t(a_n)\}$. An XML document (tree) T maps to a unit consisting of the set of all complete paths in T (i.e. paths that originates at the root and terminates at a leaf).[3] It follows that the UDB provides a unifying view of relational and XML that uses the names appearing in the sources.

As an example, the unit shown in Example 1 may be the result of mapping the following XML bibliographic entry.

[3]For the purpose of our discussion, we do not distinguish between node types in the document, and PCDATA is regarded as a leaf node.

```
<bib>
  <author>
     <name> <firstname> Karen </firstname>
            <lastname> Armstrong </lastname>
     </name>
  </author>
  <book genre = "Biography">
    <title> The Spiral Staircase </title>
  </book>
</bib>
```

A relational representation of similar information may take on one of many designs. Some possibilities are,

Design A A single relation encompassing author names and book title.

```
author(firstname,lastname,book,genre)
```

The tuple

```
<Karen,Armstrong,'The Spiral Staircase',
 Biography>
```

over the schema is represented as the unit

```
{ author.firstname.Karen,
  author.lastname.Armstrong,
  author.book.'The Spiral Staircase',
  author.genre.Biography
}
```

Design B Normalized relations that separates author and book information.

```
author(id,firstname,lastname)
book(title,genre,aid)
```

Design C A relation for each genre consisting of book title and author name as attributes.

```
biography(firstname,lastname,title)
```

Represented as UDB data, we may write generic queries that retrieve relevant data from the XML format as well as each of the above relational designs. The queries may be written without any knowledge of the structural aspects of the schema designs. For example, the query

$$\pi_{\texttt{\$book}}(U \bowtie_{\texttt{\$author=\$book} \wedge \texttt{\$author = "Armstrong"}} U)$$

retrieves the title of all books written by `Armstrong` from Designs A and B. Here U denotes the set of all units derived from the two sources.

If we have additional knowledge about the structures of the designs, then we may write more precise — and often more efficient — queries. For example, assuming that our data sources consist of the XML document above and Design A. Then the same query may be written $\pi_{\texttt{\$book}}(\sigma_{\texttt{\$lastname = "Armstrong"}} U)$.

Lastly, we may write information retrieval type queries based on keyword searches. A query to find information about `Armstrong`, in any schema design, is $\sigma_{\texttt{\$'Armstrong'}}(U)$, while $\sigma_{\texttt{\$Biography}}(U)$ retrieves all units pertaining to biography.

A SQL-like query language based on the UDA is under design and will be reported in the full version of this paper. The language provides a friendlier interface to the UDB abstraction for users. As an example, the UDA query that finds all books written by `Armstrong` may be expressed:

```
retrieve $book
with $lastname = "Armstrong";
```

4 Query Processing

Operationalizing the UDM can be accomplished by explicitly materialing XML and relational data into units and implementing a query processor for the UDA. A more lightweight approach is to compile UDA queries into source query languages, and to combine the results dynamically upon return. This is the approach that we take in an ongoing prototype implementation effort. Clearly, while any UDA query will return some answer over a set of units derived from multiple data sources, the query will not, in general, have meaningful interpretations over these data sources. Instead, we focus on cases where the UDA query is formed in some way, denoted ψ, from an existing source SQL or XPath query Q_i over source D_i. In addition to the answers retrieved from D_i, $\psi(Q_i)$ may be used to compute potentially useful answers from other data sources. The question is, what would be an appropriate definition for ψ?

First, let ϕ denote the abstract, polymorphic mapping from data sources to the UDB defined in Section 3. Then a UDB U consisting of data derived from data sources $D_1, ..., D_n$ may be expressed as $U = \phi(D_1) \cup ... \cup \phi(D_n)$. Two properties of ψ, *soundness* and *completeness*, can be studied.

Soundness The mapping ψ is sound if for any query Q_i over D_i, $\cup_{u \in \phi(Q_i)} u \subseteq \cup_{u \in \psi(Q_i)} u$. Paraphrased: the set of all contexts in the result of Q_i over D_i mapped under ϕ is contained in the set of all contexts in the result of $\psi(Q)$ over U.

Completeness The mapping ψ is complete if for any query Q_i over D_i, $\cup_{u \in \psi(Q_i)} u \subseteq \cup_{u \in \phi(Q_i)} u$.

A sound and complete mapping is desirable since it allows one to use a source query for exploring additional answers in the integrated database, knowing that the mapping preserves the semantics of the query with respect to the original data source. Note that the two properties do not specify a one-to-one correspondence between units of $\psi(Q_i)$ and units in $\phi(Q_i)$. Below, we give an instance of ψ for each of relational and XML queries.

4.1 Relational Queries

Suppose Q is a relational algebraic query over the relational database D, and U is a UDB that contains $\phi(D)$. Then $\psi(Q)$ is the UDA query obtained from Q as follows.

1. Replace each occurrence of a base relation r in Q by $\rho_{\texttt{r} \to \varepsilon}(\sigma_{r.*}(U))$.
2. Replace each attribute a in Q by $a.*$.

The renaming of `r` to ε reflects the fact that querying a base relation in the relational algebra results in an anonymous relation [10]. For instance, the query $\pi_{\texttt{book}}(\sigma_{\texttt{lastname = "Armstrong"}}(\texttt{author}))$ over the schema of Design A in Section 3 maps to the UDA query

$$\pi_{\texttt{book.*}}(\sigma_{\texttt{lastname.* = "Armstrong"}}(\rho_{\texttt{author} \to \varepsilon}(\sigma_{\texttt{author.*}}(U)))).$$

The soundness and completeness of the mapping for relational queries follows by induction on the form of the given query under a simple assumption.

THEOREM 1. *Suppose D is a relational database in which no relation name appears as either an attribute or a data value. Then ψ is sound and complete for any query over D.*

Indeed, for relational databases, we may prove the stronger property that a one-to-one correspondence exists between units of $\phi(Q)$ and $\psi(Q)$.

4.2 XPath Queries

A complete translation of XPath queries is more difficult to pin down due to XPath's set of complex features for navigating a tree. Moreover, the UDM offers very little in the way of structuring data.

Hence properties such as position and ordering cannot be determined when querying a UDB. We consider here a small but important subset of XPath. The XPath subset, adapted from [1], is given below.[4] Other "core" subsets have been studied elsewhere (e.g., [7]).

```
path =  step | step/path
step =  axis::(m|*) | axis::(m|*)[p]
   p =  p or p | p and p | path | path op literal
axis =  root | child | parent | descendent | ancestor | attribute
  op =  < | <= | = | > | >=
```

Call an XPath query *simple* if it contains no disjunctive predicates, and *linear* if it consists of only steps with axes in the same direction (i.e., child, descendent and attribute, or parent and ancestor). To translate an XPath query Q, we first rewrite each boolean expression in Q into an equivalent disjunctive normal form (DNF), and replace Q with a set of simple queries, each obtained from Q by replacing a boolean expression in DNF by one of its disjuncts. Then, for each simple query, we extract a set of a regular expressions that correspond to labels associated with a maximal contiguous sequence of forward steps embedded in the given query. This computation can be performed via an abstract interpretation of the set of simple queries from the first step to produce a set of linear queries. Outputs from the computation are combined to form a UDB query in which the final sequence computed is used for projection, and all sequences computed form a conjunctive boolean expression for selection.[5] Some example queries, outputs from the abstract interpretation, and the eventual UDA query $\psi(Q)$ are shown below.

Input Q:	`descendent:a/child:b[@c = "1"]/child:d`
Output:	`$a.b.c`
	`$a.b.d`
$\psi(Q)$:	$\pi_{\texttt{\$a.b.d}}(\sigma_{\texttt{\$a.b.d} \wedge \texttt{\$a.b.c="1"}}(U))$

Input Q:	`root:a/child:*/child:b/parent:*/child:c`
Output	`a.?.b.*`
	`a.?.c.*`
$\psi(Q)$	$\pi_{\texttt{a.?.c.*}}(\sigma_{\texttt{a.?.c.*} \wedge \texttt{a.?.b.*}}(U))$

Input Q:	`descendent:a/ancestor:b/descendent:c`
Output	`$a`
	`$b.c`
$\psi(Q)$	$\pi_{\texttt{\$b.c}}(\sigma_{\texttt{\$a} \wedge \texttt{\$b.c}}(U))$

THEOREM 2. *The above translation of XPath queries to UDA query is sound.*

Completeness is not guaranteed. For XML documents and queries that occur in practice, however, experiments below show that incorrect answers are rarely retrieved. For the experiments, we first use two sets of data-centric XML documents. The first is auction data created using XMark [17], and the second is a set of DBLP data created by Christoph Koch:

`www.infosys.uni-sb.de/teaching/dbs05/tests/`

Queries for the experiments are obtained from the same source as the dataset in each case. To illustrate, two example queries for the auction data are

```
Q1: /site/regions/namerica/item[@id="item101"]/name/
Q14: /site//item/name/text()[../../description//text
     [contains(text(),"gold")]]
```

For each query Q, the number of all contexts contained in the XML fragments computed by an XPath processor and the number of contexts computed by the UDA query under ψ are listed. From Theorem 2, we know that any (and only) additional answers computed by $\psi(Q)$ are irrelevant answers.

Auction Data

Query	Q	$\psi(Q)$
Q1	1	1
Q2	55	60
Q6	108	108
Q14	4	5
Q15	1	1
Q17	61	61
Q20	1	1

DBLP

Query	Q	$\psi(Q)$
Q1	1	1
Q2	73	73
Q3	0	0
Q4	73	73
Q6	5	5

To validate the accuracy of the translated query further, we compare in table **Treebank** the result of five queries, executed over a large and complex document-centric treebank of natural language sentences [3]. The dataset contains 19,274 sentences with 223,281 nodes occupying 9.125MB. While the UDM had not been motivated by querying document-centric data, the results show that its simplistic representation of XML documents and the translation of XPath to UDA queries produce, nevertheless, highly precise results.

Treebank

Query	Q	$\psi(Q)$
Q1	1	1
Q2	767	813
Q3	17603	17603
Q4	39	49
Q5	96	96

4.3 Querying Other Sources

Aside from returning answers from the original source, the query $\psi(Q)$ will not, in most cases, yield many answers from other sources. For instance, lengths of contexts associated with XPath queries will typically be larger than the lengths of contexts in data units derived from relational data source. Some simplifications and modifications of the query are thus useful if the query is posed to other sources. We are currently developing a UDB system for the domain of pathology reports and will experiment with different query simplification/modification heuristics.

4.4 Unit Materialization

The purpose of UDM is to provide a lightweight solution for accessing relational and XML data sources. In particular, units need not be materialized to process queries. In certain situations, however, materializing a small set of computed units is useful. These may be used to answer frequently executed queries more efficiently, or to perform local joins of data subsets derived from several sources.

A simple storage approach is to represent all units as tuples over the binary relation `ur(Id:integer,Context:string)`. Each unit is identified by a unique integer, and each context in the unit is represented as a value under the attribute `Context`. A set of mutually recursive rewrite rules can been defined for compiling each UDA query Q to an answer preserving relational query, $\alpha(Q)$, in the sense that $u \in Q$ iff there exists an integer i such that $u = \cup_{c \in \pi_{Context}(\sigma_{Id=i}(\alpha(Q)))} c$. An alternative implementation approach under investigation is to adapt the indexing techniques developed in [9] for prefiltering XML documents.

[4] While the grammar gives no provision, we assume that root may appear at most once in a query and at the beginning.

[5] Formal details of the algorithms will be included in the full paper.

5 Related Work and Conclusion

The UDM adopts the principle suggested by [15, 16], namely data, not metadata, are the primary objects of focus in a database. Our work aims to provide a simple, unifying query interface for multiple relational and XML data sources. We conclude with some summary comments and discussions related work.

1. The UDM is parameterized by a matching relation that enables similarity techniques to be used in determining coreference between properties. In this, our initial study of the UDM, we have chosen string equality as the matching relation.

   Other information retrieval-based querying techniques over structured data have been considered. The most notable is the system WHIRL by Cohen [4]. The key difference compared to our approach is that matching in WHIRL is performed over instance data, and writing queries in WHIRL still requires full knowledge of the schemas of each data source.

2. The UDM provides property concatenation as a rudimentary mechanism for structuring related properties. This allows one to write UDA queries that are structure-aware with respect to the data sources, but still retains the flexibility for information-retrieval type queries that ignore structural concerns.

   While most relational database systems today offer special operators (e.g., `contains`) for keyword searching over designated columns, these are adhoc extensions, however, and they provide no mechanism for uniformly searching over meta and instance data.

3. Source relational and XPath queries may be meaningfully interpreted over a UDB comprised of several data sources. Each query maps naturally to a UDA query that computes either exact (in the case of relational query) or highly precise answers with respect to the original data source.

   The use of source queries over multiple data sources is a relatively unexplored area of research. To the best of our knowledge, Norrie and Kerr's Universal Contextual Query System (UCQS) [14] is, conceptually, the closest work. The UCQS heuristically translates queries written for one data source into queries for other data sources without reference to a mediated schema. No inter-model translation nor soundness and completeness studies exist, however.

Language studies for querying multiple data sources have emphasized providing relational interoperability [11, 12, 19]. They extend relational query languages with higher-order features for extracting and manipulating metadata. Special operators are often needed, however, for restructuring due to the assumed separation of metadata and data.

Acknowledgement. This work was completed while J.J. Lu visited the Institute of Information Science, Academia Sinica, Taiwan under the support of grant NSC94-2811-Z-001 from the National Science Council of Taiwan.

6 References

[1] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke. CTL Model Checking for Processing Simple XPath Queries. In *Proceedings Temporal Representation and Reasoning (TIME 2004)*. IEEE Computer Society Press, 2004.

[2] K. Beyer, R. J. Cochrane, V. Josifovski, J. Kleewein, G. Lapis, G. Lohman, B. Lyle, F. Özcan, H. Pirahesh, N. Seemann, T. Truong, B. V. der Linden, B. Vickery, and C. Zhang. System RX: one part relational, one part XML. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 347–358, New York, NY, USA, 2005. ACM Press.

[3] K.-J. Chen, C.-C. Luo, Z.-M. Gao, M.-C. Chang, F.-Y. Chen, C.-J. Chen, and C.-R. Huang. The CKIP Chinese Treebank. In *The Journees ATALA sur les Corpus annotes pour la syntaxe*, Paris, 1999.

[4] W. W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000.

[5] D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu. A comprehensive xquery to sql translation using dynamic interval encoding. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 623–634, New York, NY, USA, 2003. ACM Press.

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

[7] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

[8] M. Graiser, L. Hill, M. Keehan, J. Simons, and C. Flowers. Utilization of an integrated information system linking legacy databases for oncology outcomes research. In *Frontiers in Oncology and Pathology Informatics*, 2006.

[9] C.-H. Huang, T.-R. Chuang, and H.-M. Lee. Prefiltering techniques for efficient XML document processing. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 149–158, New York, NY, USA, 2005. ACM Press.

[10] P. C. Kanellakis. Elements of Relational Database Theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 1073–1156. Elsevier, Amsterdam, 1990.

[11] R. Krishnamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. In *SIGMOD Conference*, pages 40–49, 1991.

[12] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM Transactions on Database Systems*, 26(4):476–519, 2001.

[13] J. J. Lu. A data model for data integration. *Electr. Notes Theor. Comput. Sci.*, 150(2):3–19, 2006.

[14] M. C. Norrie and D. Kerr. Universal Contextual Queries in Database Networks. In *CoopIS*, pages 180–191, 1995.

[15] J. Parsons and Y. Wand. Emancipating Instances from the Tyranny of Classes in Information Modeling. *ACM Transactions on Database Systems*, 25(2):228–268, 2000.

[16] J. Parsons and Y. Wand. Property-based semantic reconciliation of heterogeneous information sources. In *ER*, pages 351–364, 2002.

[17] A. Schmidt, F. Waas, M. Kersten, D. Florescu, I. Manolescu, M. Carey, and R. Busse. The XML benchmark project. Technical Report INS-R0103, CWI, April 2001.

[18] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 302–314. Morgan Kaufmann, 1999.

[19] C. M. Wyss and E. L. Robertson. Relational Languages for Metadata Integration. *ACM Transactions on Database Systems*, 30(2):1–33, 2005.

[20] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemara. XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. *ACM Transactions on Internet Technology*, 1(1):110–141, August 2001.