XML: Some Papers in a Haystack

Mirella M. Moro
Univ. Fed. Minas Gerais, UFMG
mirella@dcc.ufmg.br

Vanessa Braganholo
Univ. Fed. Rio de Janeiro, UFRJ
braganholo@dcc.ufrj.br

Carina F. Dorneles
Universidade de Passo Fundo, UPF
dorneles@upf.br

Denio Duarte
Univ. do Estado de Santa Catarina
denio@unochapeco.edu.br

Renata Galante
Univ. Fed. Rio Grande Sul, UFRGS
galante@inf.ufrgs.br

Ronaldo S. Mello
Univ. Fed. Santa Catarina, UFSC
ronaldo@inf.ufsc.br

ABSTRACT

XML has been explored by both research and industry communities. More than 5500 papers were published on different aspects of XML. With so many publications, it is hard for someone to decide where to start. Hence, this paper presents some of the research topics on XML, namely: XML on relational databases, query processing, views, data matching, and schema evolution. It then summarizes some (some!) of the most relevant or traditional papers on those subjects.

1. INTRODUCTION

XML is a language for specifying semi or completely structured data. It has been explored over and over by both research and industry communities. More than ten years after its proposal, its initial expectation of "solving all the problems in the world" has not been fulfilled. However, we cannot say XML failed, since it solved some really important problems very efficiently. Some of those problems include data integration, data interoperability, and data publishing on the Web. Moreover, XML is adopted as a standard language by many industries (from retail to healthcare) for exchanging data. For those and other reasons, XML is the most successful, ubiquous technology for the Web, at the same level as URI, HTTP, and HTML [98].

If someone wants to start a research on XML, the first questions that come to mind are "what papers one must read" and "what part of the XML literature one needs". There is one problem though with such questions. Much research has already been done, and a considerable amount of papers have already been published about XML. If a person needs to start any study on XML, s/he will probably start reading the W3C documents, but that is not enough.

Specifically, the DBLP Computer Science Bibliography[1] Faceted Search points out XML as the 10^{th} most popular keyword[2], appearing in 1250 publications. Using the DBLP CompleteSearch with *XML* as search word returns 5529 publications. How does one find the desired publication over so many? One solution is to specify a second search word such as *query*. In this case (search words *XML* and *query*), the CompleteSearch returns 604 publications, which is still a lot.

This paper is intended to people who would like to start a research on XML or simply start studying XML from the research point-of-view. It summarizes **some** of the research topics on XML: XML on relational databases (Section 2), views (Section 3), query processing (Section 4), data matching (Section 5), and schema evolution (Section 6). It then presents some of the most relevant or traditional papers on those subjects. By "relevant or traditional papers", we mean those that are more frequently cited or that considerably changed the way XML research was done.

Note that this paper is not intended to cover the whole vast literature on XML. For example, there are papers about the three major commercial DBMSs (*i.e.* IBM DB2, Microsoft SQL Server, Oracle) that discuss many practical aspects of supporting XML management, for example [12, 62, 81, 82]. Other industrial papers discuss specific topics, such as schema evolution [11]. Furthermore, some research groups and companies have also their own XML engines [14, 42, 43, 54]. We do not plan to detail those solutions. Instead, we focus on a very selected set of papers on some XML research areas. Nevertheless, this paper intends to be a starting point for any person who decided to study this wonderful, versatile, powerful language, called XML.

2. XML ON RELATIONAL DATABASES

The hierarchical nature of XML brought several challenges to the database community. One of the first ones was simple, but very tricky: How to store XML data? Relational databases seemed to be a good alternative, since most of the data (both in academia and industry) was/is stored in RDBMS (Relational Database Management Systems).

The first paper to suggest the use of relational database engines for managing data in files was [1]. The file contents should then be stored in the relational database using some mapping schema. For this to be possible, the file should have a "strict inner structure". This requirement resembles XML documents that have a DTD or schema.

2.1 Storage

Based on such observation, many publications followed with the goal of storing [24, 25, 35, 44, 57, 83] and querying [32, 83, 85, 89] XML documents using relational databases.

XML-based approaches. The first proposals to store generic XML documents (schemaless) appeared in [44]. The *Edge* and *Attribute* approaches are able to store any XML document (regardless of a schema) in relations. Later, a different solution, based on *dynamic intervals*, introduced both storage and query translation algorithms [32].

DTD-based approaches. Then, the pioneer proposals to store XML documents in relations by exploring their

[1] http://www.informatik.uni-trier.de/~ley/db access on 02/23/09
[2] http://dblp.l3s.de/browse.php?browse=mostPopularKeywords access on 02/23/09

.

schema (DTD) are in [83]. It proposes *Basic*, *Shared*, and *Hybrid Inlining* and discusses their pros and cons. However, query translation is not discussed.

Constraint-based approaches. Some approaches construct the relational schema based on constraints of the XML documents. For example, [57] explores DTD constraints such as cardinalities and referential integrity. On the other hand, [24] uses XML key and keyref, while [25] explores XML functional dependencies (XFDs) to generate a relational schema that has as few redundancy as possible.

Discussion. It is very important to notice that querying depends on the method for storing XML data in the relations. The Edge and Attribute approaches require many joins to recompute the document. The dynamic interval uses sub-queries and the *union all* operator to reproduce the original XML instance.

All of those publications have influenced the native and hybrid storage alternatives currently in use.

2.2 Updates

Once the storage problem was handled, it was necessary to update the documents stored in the RDBMS. There are three main proposals for updating XML documents stored in relational databases [88, 89, 91]. All of them assume a default mapping of the XML document to relations, such as the *shared inlining method* [83] and the approach of [57].

Tatarinov et al. [88] propose an extension to XQuery to support updates. This extension comprises primitives such as `UPDATE`, `DELETE`, `INSERT`, `REPLACE`, and `RENAME`. They also discuss how to translate those updates to the underlying RDBMS for updating XML documents using the shared inlining method. In [89], Tatarinov et al. use the XQuery extension proposed in [88] to deal with ordered XML documents. They propose a storage method for ordered XML documents in relations, and show how updates that affect order can be translated. Finally, [91] provides an updatability study for XML documents stored in RDBMS.

It is important to notice that [88] presented the first proposal of an update language for XML. Many of the concerns risen on that paper served as starting points to the study of update-related problems regarding XML [8].

3. XML VIEWS

A natural follow-up to store XML data as relations is to generate XML *from* relational data. The literature deals with this problem by using the concept of views. Many approaches explore building and querying XML views over relational databases [6, 13, 23, 41, 61, 84, 86]. Most of them tackle the problem by building a *default* XML view from the relational source, and then using an XML query language to query the default view [13, 23, 41, 86]. Their concerns are basically the following: *(i)* how to compose queries with view definitions, in order to get a *single* resulting expression that represents both the query and the view; *(ii)* how to use the relational engine to execute the query resulting from step *i*; and *(iii)* how to use the relational tuples resulting from step *ii* to build the resulting XML document.

Each approach has a different technique to construct the XML view using the relational engine to retrieve data. Some transform the XML view definition into extended SQL [23, 84, 86], others use internal representations to map the XML view to several SQL queries [13, 41] or to map the relational schema to an XML Schema [61]. For speeding-up queries, some approaches maintain materialized views [6]. The classical papers on this subject are *SilkRoute* [41] and *XPERANTO* [86]. The SQL language has also been extended to support generating XML documents from relational data. The extensions include SQL/XML, FOR XML, SQLX, and are supported by most database vendors.

Once a view is defined, there are also the problems of updating it, and translating it back to the underlying relational database. The pioneer work is [19], where the solutions is based on the *Nested Relational Algebra* to define the XML views. Later, the idea was extended to support an XQuery-based view definition language [20, 21]. The main idea of those works is to map XML view updates to Relational View updates, and then map the updates back to the database by using existing View Update work for relational databases.

Then, the approach in [92] presents an updatability study of XQuery views published over relational databases. Their results are analogous to the results of [19]. However, neither of those papers discusses how updates are translated to the underlying relational database. Recently, a two step approach to the problem was proposed in [95]: the first step is schema checking, which uses schema information to decide if the update is translatable to the database. In some cases, a second step may be required to make the decision: data checking, which is an expensive operation, and thus must be avoided whenever possible.

4. XML QUERY PROCESSING

The success of XML databases depends heavily on efficient methods for manipulating XML data. XML requires efficient query processing techniques over tree-structured data. Given the many differences between that type of data and relational data, the well known and efficient algorithms from relational databases cannot be (directly) employed to process XML data efficiently.

There are many algorithms for XML query processing, which can be divided in two categories: *(i)* the ones that employ the infra-structure of a relational database (*i.e.*, based on the mapping from XML to relational tables/columns) [32, 83, 85, 89, 90], and *(ii)* the ones that employ a native XML engine (*i.e.*, data is handled in its original tree format; access methods consider the native tree structure of the data; new, XML-structure-aware indexes may be employed) [3, 22, 27, 28, 55, 60, 70, 79, 94, 99, 102]. This section focuses on XML query processing using a *native* engine. Moreover, there are different semantics for XML queries. Here, we consider the basic structural constraints: structural join and tree-pattern query (or twig). Other semantics, such as keyword search and document filtering, are not covered due to space constraints. Finally, we present some other very specific topics related to query processing.

Structural Joins. This type of join is the simplest possible structure and considers only ancestor//descendant and parent/child pairs. The query result may be the pair or one of its components. The first native approach to implement structural joins is the *MPMGJN* algorithm [102]. Then, *StackTree* [3] is the state-of-the-art algorithm for structural joins, since it defined the problem and presented a very nice initial solution. Different nuances of structural joins use partitioning techniques, which include the work on [60], first partition-based technique for structural joins. Finally, there is also some work on processing structural joins when the document suffers insertions [31].

Tree-Pattern Query, or Twig. This type of query considers path and subtree structures, *i.e.*, combinations of the basic axis (ancestor//descendant and parent/child) with predicates. The query result may be an element, a path, or a subtree of the document that satisfies the constraints. The XML twig algorithms may be divided into four very distinct categories (see [70] for definition of the categories and an experimental comparison). Note that those categories do not consider optimizations according to the query workload. The only optimizations allowed are those based on the data, and the data alone (*i.e.*, clustering, indexes, and so on).

The simplest (possibly the most naïve) way to process twigs is by merging the results of structural joins. Both *StackTree* [3] and *MPMGJN* [102] (from structural joins) are binary join algorithms, *i.e.*, they join only a pair of inverted lists (or only one edge in the query twig). Since a complete twig query consists of a series of binary joins, the problem of join order selection has to be considered seriously.

A dynamic programming method to select an optimal or sub-optimal order of binary structural joins for XML twig queries is proposed in [99]. The *StackTree* binary join algorithm and the corresponding dynamic-programming-based join order selection algorithm have been integrated into Timber [54] (the native XML database prototype from the University of Michigan). Other techniques followed those by aggregating indexes to the lists to be merged, *e.g.*, [27, 55].

The state-of-the art algorithm for processing twigs is a holistic approach, the *TwigStack* [22]. It performs a pipelining join by joining multiple inverted lists at one time without generating intermediate results. That paper also introduces *XB-Twigstack*, a *TwigStack* with indexes.

Other algorithms followed by proposing optimizations for different nuances (restrictions, requirements, and profiles), specialized numbering schemes such as Dewey (*e.g.*, [63]), or using different holistics, such as subsequence matching (*e.g.*, [79, 94]). Other techniques consider the query workload, such as *APEX* [28], which defines an index and considers the query workload in order to compact even more the index.

A third way to process twigs is through a finite state machine (FSM). Each XML element is processed only once through sequential access. For example, an extension to FSM with stacks (similar to *TwigStack*) is proposed on [70].

Other XML Query-related Topics. It is very important to notice that it is impossible to cover 604 papers on XML query processing. Besides the work aforementioned, we could also cite other punctual papers, which deal with very specific, query-related topics. Examples include: index advisors [39], unordered XQuery processing [48], query unnesting [65], evolving queries [69], structural summaries [71], among many others.

5. XML DATA MATCHING

The central problem on data matching is to find out heterogeneous data that represent the same real world object. Solutions to this matter are required by several applications, like data integration, database design, and querying heterogeneous data sources [37]. XML data matching is a complex task, and involves at least two levels: schema and instance.

5.1 Schema Matching

Schema matching is an active research area since the Eighties, but with more theoretical than effective proposals until today. Two approaches are usually considered for schema matching [40]: *(i)* to define a global schema that maintains mappings to a set of local schemas; or *(ii)* to define, for each pair of schemas, a set of mappings among their components. The first one is classified into two main categories [59]: *(i.1) Global-as-View* (*GAV*), and *(i.2) Local-as-View* (*LAV*). A *GAV* approach builds a global schema from the semantic matching of the local schemas, *i.e.*, the global schema is a unified view of the local schemas. The *LAV* approach builds a global schema from scratch, and then defines a set of mappings to the local schemas, *i.e.*, the local schemas are views of the global schema. Despite of being initially proposed for databases, these approaches have been applied to XML schema matching, with adaptations that consider the semistructured nature of XML data. Nonetheless, this is still an active research area, since most of the existing work do not deliver satisfactory results [45].

Initial work on XML schema matching focused on defining a global schema for XML data. This happened because XML data sources are usually on the *Web* and have no associated schemas. Therefore, *mediator-based* architectures have been developed [97]. There are two layers in this kind of architecture: *wrapping* and *mediation*. The wrapping layer is responsible for extracting a schema for each XML source, or providing mappings from a predefined global schema to the XML data sources. The mediaton layer is responsible for managing a global schema according to a GAV [67, 73, 75, 80, 101] or LAV [34, 36] approach. For example, in the *Enosys* data integration platform [73], the mediator module allows the user to define XML views from the global schema. A drawback on some architectures is that the matching process is manual, *i.e.*, the mediators act as a tool that helps the user to define global schema or mappings among XML local sources [29, 66, 73, 80]. This limitation has been solved by semi-automatic approaches [33, 34, 36, 67, 75, 101].

Another idea is to adopt a *canonical model*, *i.e.*, a data model on which the XML global schema is defined. Some proprosals consider the proper XML model to define the canonical schema, which is useful for applications that need to persist and/or to manage unified data in XML format [34, 36, 66, 73, 101]. Others define the global schema based on a conceptual model, which is more suitable to capture data semantics, and useful for applications that query heterogeneous XML data sources [33, 64, 67, 75, 80].

Regarding the XML schema matching process, differences among the proposals are mainly on their XML model constructs. For example, most approaches do not deal with the matching of mixed elements or an element content model specified as a choice among several nested elements [29, 34, 36, 64, 66, 75, 80]. Others try to overcome such limitation, *e.g. BInXS* [67]. Regarding the comparison strategy for defining schema components to be matched, there are usually dictionaries or *Thesauri* (*linguistic* strategy) [34, 64, 67], and/or *metrics* (functions and/or algorithms) applied on structural and/or constraint properties of the XML schema, like data types and cardinality constraints [64, 67].

Few approaches define binary schema mappings among XML local schemas instead of creating a global schema, *e.g. HDM* [66], *Xere* [33], and *YAT* system [29]. In the first two, XML schema of local sources are converted to schemas in an intermediate conceptual model. Semantic mappings are further defined between conceptual and XML schemas. *YAT* supports the manual definition of mappings between XML schemas represented through a graph-based model.

These proposals illustrate that schema mapping approaches for XML can be provided at a conceptual or logical level.

Different from those traditional schema matching approaches, other systems consider information about XML instances and machine learning techniques to deduce semantic correspondences between heterogeneous schemas' elements [36]. Despite of the additional analysis of XML data contents (besides schema information, which can contribute to more exact schemas component matchings) approaches like that are criticized by their complexity.

5.2 Instance Matching

Due to the hierarchical nature of XML data, most approaches on instance matching consider both structure and content aspects. The techniques are very different from each other. Some employ *diff* algorithm [30, 93] or a TF/IDF weighting scheme [76], while others use a tree-edit distance-based metric [4, 51, 52, 58], or a similarity function [96].

Regarding diff algorithms, the *XyDiff* algorithm [30] detects changes in XML documents that are stored in a data warehouse, and uses signatures to match (large) subtrees that were left unchanged between the old and the new versions. Then, it propagates those matches to ancestors and descendants to obtain more matches. It also uses XML specific information such as ID attributes. Another diff algorithm is the *XDiff* algorithm [93], which integrates key XML structure characteristics with standard tree-to-tree correction techniques. Such algorithms focus on identifying changes in versions of a document and introduce cost models to quantify those changes.

Likewise, the tree-edit distance metric is widely used to match XML documents [4, 51, 52, 58]. Specifically, in [51, 52], the approximate matching is quantified in terms of distance notions. The central idea is to incorporate the tree-edit distance in a join framework, which is used in XML integration processes. In [4], the authors propose the concept of approximate tree join, which intuitively assesses that two objects are the same if they have (almost) the same tree. They propose the *pq-grams* distance to approximately match hierarchical information. Such measure is defined as the distance between ordered labeled trees using the tree-edit distance and similarity functions. In [100], the authors propose to transform tree-structured data into an approximate numerical multidimensional vector, which encodes the original structure information. The algorithm embeds the proposed distance into a filter-and-refine framework to process similarity search on tree-structured data.

Some work consider both content and structure during the instance matching [74, 77]. For example, in [77], the authors extend the classical approach to duplicate detection in flat relational data, that is the sorted neighborhood method to be applied in nested XML elements, detecting duplicates at each level of the XML hierarchy. In addition, the framework proposed in [96] characterizes duplicates in a matching process by considering the description of data instances, and a similarity score using a similarity measure.

The choice of an appropriated similarity function is an important task in a matching process, since it defines the result quality. A problem related to the use of disparate similarity functions appears when the similarities scores of the XML nodes are combined. As the individual functions usually generate scores that are not comparable, there is no general straightforward way for combining independent distinct functions into a single measure. Other recent work has addressed such a problem [38, 53].

6. XML SCHEMA EVOLUTION

Schema evolution deals with updating a schema when it no longer meets the user or the application requirements. The main problem is how to allow schemas to change while maintaining the access to the existing XML data.

The approach employed to allow schemas to evolve depends on the XML schema language. The two most used schema languages are DTD and XML Schema (XSD) [10, 68]. However, XSD is more powerful than DTD because XSD, unlike DTD, permits decoupling an element tag from its type (or content model), and an element may have different types depending on the context [72, 78, 56].

The structure employed to represent XML documents and schemas is important to implement an *evolution framework*. XML documents are mostly represented as unranked labeled trees (*i.e.*, their nodes have no priory bound on the number of children). Each tree node represents an element (or an attribute). XML schemas, on the other hand, do not have a standard representation (as XML documents have). Thus, the representations found in the literature are: regular tree grammars (RTG) [17, 26, 72], direct graphs (DG) [2, 87], and mixed representation (*e.g.*, based on applying more than one structure to model a schema) [50, 78]. Although XML schemas may have different representations, the primitives for evolving them follow, as expected, the same basis: objects may be inserted, deleted and/or updated.

Update Primitives. Update primitives for XML schemas are slightly different from those to update schemas on traditional databases (*e.g.*, relational databases). XML schema objects may still be inserted, deleted, and/or replaced. However, XML schemas have their own characteristics: *(i)* objects have cardinality, for example, an element may repeat n times ($n \geq 0$ or $n \geq 1$); *(ii)* objects may be moved, for example, a sub-tree may be moved from a parent node to another one; and, *(iii)* objects may be complex and may have their complex type changed.

Non-conservative Primitives. Non-conservative schema updates define a set of operations that, when applied to a schema, may lead to an inconsistent state. In this case, revalidation and adaptation processes are necessary. The revalidation of an XML document identifies whether an initially valid document is still valid with respect to the updated schema definitions (*e.g.*, [5, 7, 15]). The adaptation of an XML document is required if the revalidation process fails. This process operates on the document structure so it respects the most up to date schema definitions. Initial approaches were based on object-oriented databases [2, 87] and schema versioning [46]. Using information from the DTD, schema evolution may be triggered by patterns detected on documents through usage of data mining techniques [9]. *MXML* [47] is a multidimensional model for representing the history of XML documents and the evolution of their schema. Finally, [49] describes a document evolution and adaptation proposal, and [50] extends it for the incremental validation of documents upon schema evolution.

Conservative Primitives. This kind of primitives is important in XML-based applications since: *(i)* it is not necessary to revalidate all XML document collections (or database); *(ii)* data loss can be avoided because it may be necessary to delete tags (and their information) from

initially valid documents; and *(iii)* when documents to be revalidated are stored in different sites, not only their transfer cost should be considered (in addition to the whole revalidation cost) but also problems due to the access control should be faced. However, conservative primitives are not complete, that is, some update primitives cannot be mapped into conservative ones. Several approaches have been proposed to evolve XML schemas. Some of them keep the instances validity without revalidation [16, 18] (*e.g.*, all existing valid instances are guaranteed to be valid with relation to updated schema), while others have to test, in some cases, the validity of all existing valid instances [2, 49, 87].

7. CONCLUDING REMARKS

XML has been explored over and over by both research and industry communities. This paper summarized some of the research topics on XML by presenting some of the most relevant/traditional papers on those subjects. It was not intended to cover the vast XML literature, but to point out some directions to those who are interested in learning this powerful, versatile language called XML. The topics addressed on this paper were not randomly chosen. They reflect the area of expertise of the authors. Overtime, we plan to expand it to include more XML research areas.

Acknowledgements. The authors thank the reviewers for their feedback. This work was partially supported by CNPq, CAPES, FAPERJ, and FAPEMIG, Brazil.

8. REFERENCES

[1] S. Abiteboul, S. Cluet, and T. Milo. Querying and Updating the File. In *VLDB*, 1993.

[2] L. Al-Jadir and F. El-Moukaddem. Once Upon a Time a DTD Evolved into Another DTD... In *OOIS*, 2003.

[3] S. Al-Khalifa et. al. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *ICDE*, 2002.

[4] N. Augsten, M. Böhlen, and J. Gamper. Approximate matching of hierarchical data using pq-grams. In *VLDB*, 2005.

[5] A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental Validation of XML Documents. *ACM TODS*, 29(4):710–751, 2004.

[6] A. Balmin et. al. A Framework for Using Materialized XPath Views in XML Query Processing. In *VLDB*, 2004.

[7] D. Barbosa et. al. Efficient Incremental Validation of XML Documents. In *ICDE*, 2004.

[8] M. Benedikt et. al. Adding Updates to XQuery: Semantics, Optimization, and Static Analysis. In *XIME-P*, 2005.

[9] E. Bertino et. al. Evolving a Set of DTDs According to a Dynamic Set of XML Documents. In *EDBT*, 2002.

[10] G. J. Bex, F. Neven, and J. V. Bussche. DTDs versus XML schema: a practical study. In *WebDB*, 2004.

[11] K. S. Beyer et. al. DB2/XML: Designing for Evolution. In *SIGMOD*, 2005.

[12] K. S. Beyer et. al. DB2 Goes Hybrid: Integrating Native XML and XQuery with Relational Data and SQL. *IBM Systems Journal*, 45(2):271–298, 2006.

[13] P. Bohannon et. al. Optimizing View Queries in ROLEX to Support Navigable Result Trees. In *VLDB*, 2002.

[14] P. Boncz et. al. MonetDB/XQuery: a fast xquery processor powered by a relational engine. In *SIGMOD*, 2006.

[15] B. Bouchou and M. H. F. Alves. Updates and Incremental Validation of XML Documents. In *DBPL*, 2003.

[16] B. Bouchou and D. Duarte. Assisting XML Schema Evolution that Preserves Validity. In *Brazilian Symp. on Data Base (SBBD)*, 2007.

[17] B. Bouchou et al. Extending Tree Automata to Model XML Validation under Element and Attribute Constraints. In *ICEIS*, 2003.

[18] B. Bouchou et al. Schema Evolution for XML: A Consistency-Preserving Approach. In *Mathematical Foundations of Computer Science*, 2004.

[19] V. Braganholo, S. B. Davidson, and C. A. Heuser. On the Updatability of XML Views over Relational Databases. In *WebDB*, 2003.

[20] V. Braganholo, S. B. Davidson, and C. A. Heuser. From XML View Updates to Relational View Updates: Old Solutions to a New Problem. In *VLDB*, 2004.

[21] V. Braganholo, S. B. Davidson, and C. A. Heuser. PATAXÓ: a Framework to Allow Updates Through XML Views. *ACM TODS*, 31(3):839–886, 2006.

[22] N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. In *SIGMOD*, 2002.

[23] S. Chaudhuri, R. Kaushik, and J. Naughton. On Relational Support for XML Publishing: Beyond Sorting and Tagging. In *SIGMOD*, 2003.

[24] Y. Chen, S. B. Davidson, and Y. Zheng. Constraint Preserving XML storage in Relations. In *WebDB*, 2002.

[25] Y. Chen, S. B. Davidson, and Y. Zheng. RRXS: redundancy reducing XML storage in relations. In *VLDB*, 2003.

[26] B. Chidlovskii. Using Regular Tree Automata as XML Schemas. In *Proc. IEEE Advances in Digital Libraries Conference*, May 2000.

[27] S.-Y. Chien et. al. Efficient Structural Joins on Indexed XML Documents. In *VLDB*, 2002.

[28] C.-W. Chung, J.-K. Min, and K. Shim. APEX: an adaptive path index for XML data. In *SIGMOD*, 2002.

[29] S. Cluet et. al. Your Mediators Need Data Conversion! In *SIGMOD*, 1998.

[30] G. Cobéna, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *ICDE*, 2002.

[31] E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML Trees. In *PODS*, 2002.

[32] D. DeHaan et.al. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *SIGMOD*, 2003.

[33] G. Della Penna et. al. Interoperability Mapping from XML Schemas to ER Diagrams. *Elsevier DKE*, 59:166–188, 2006.

[34] C. Delobel et. al. Semantic Integration in Xyleme: a Uniform Tree-based Approach. *Elsevier DKE*, 44(3):267–298, 2003.

[35] A. Deutsch, M. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *SIGMOD*, 1999.

[36] A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In *SIGMOD*, 2001.

[37] A. Doan and A. Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26(1):83–94, 2005.

[38] C. F. Dorneles et. al. A Strategy for Allowing Meaningful and Comparable Scores in Approximate Matching. In *CIKM*, 2007.

[39] I. Elghandour et al. An XML Index Advisor for DB2. In *SIGMOD*, 2008.

[40] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann, San Francisco, CA, 1999.

[41] M. Fernandez et. al. SilkRoute: A framework for publishing relational data in XML. *ACM TODS*, 27(4):438–493, 2002.

[42] T. Fiebig and H. Schöning. Software AG's Tamino XQuery Processor. In *XIME-P*, 2004.

[43] T. Fiebig et. al. Natix: A Technology Overview. In *NODe - Web and Database-Related Workshops*, 2002.

[44] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a re- lational database. *Tech.Report 3684*, INRIA, 1999.
[45] A. Gal. The Generation Y of XML Schema Matching Panel Description. In *XSym*, 2007.
[46] R. M. Galante et. al. Temporal and versioning model for schema evolution in object-oriented databases. *Data Knowl. Eng.*, 53(2):99–128, 2005.
[47] M. Gergatsoulis and Y. Stavrakas. Representing Changes in XML Documents using Dimensions. In *Xsym*, 2003.
[48] T. Grust, J. Rittinger, and J. Teubner. eXrQuy: Order Indifference in XQuery. In *ICDE*, 2007.
[49] G. Guerrini, M. Mesiti, and D. Rossi. Impact of XML schema evolution on valid documents. In *WIDM*, 2005.
[50] G. Guerrini, M. Mesiti, and D. Sorrenti. XML schema evolution: Incremental validation and efficient document adaptation. In *XSym*, 2007.
[51] S. Guha et. al. Approximate XML joins. In *SIGMOD*, 2002.
[52] S. Guha et. al. Integrating XML Data Sources using Approximate Joins. *ACM TODS*, 31(1):161–207, 2006.
[53] S. Guha et.al. Merging the Results of Approximate Match Operations. In *VLDB*, 2004.
[54] H. V. Jagadish et.al. TIMBER: A native XML database. *The VLDB Journal*, 11(4):274–291, 2002.
[55] H. Jiang et. al. XR-Tree: Indexing XML data for efficient structural joins. In *ICDE*, 2003.
[56] A. H. F. Laender et. al. An X-ray on Web-available XML Schemas. *SIGMOD Record*, 38(1):37–42, 2009.
[57] D. Lee and W. W. Chu. Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. In *ER*, 2000.
[58] K.-H. Lee, Y.-C. Choy, and S.-B. Cho. An Efficient Algorithm to Compute Differences between Structured Documents. *IEEE TKDE*, 16(8), 2004.
[59] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, 2002.
[60] Q. Li and B. Moon. Partition Based Path Join Algorithms for XML Data. In *DEXA*, 2003.
[61] C. Liu, M. W. Vincent, J. Liu, and M. Guo. A Virtual XML Database Engine for Relational Databases. In *XSym*, 2003.
[62] Z. H. Liu, M. Krishnaprasad, and V. Arora. Native Xquery Processing in Oracle XMLDB. In *SIGMOD*, 2005.
[63] J. Lu et al. From Region Encoding to Extended Dewey: on Efficient Processing of XML Twig Pattern Matching. In *VLDB*, 2005.
[64] J. Madhavan, P. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB*, 2001.
[65] N. May, S. Helmer, and G. Moerkotte. Strategies for Query Unnesting in XML Databases. *ACM Trans. Database Syst.*, 31(3):968–1013, 2006.
[66] P. McBrien and A.Poulovassilis. A semantic approach to integrating XML and structured data sources. In *CAISE*, 2001.
[67] R. S. Mello and C. A. Heuser. BInXS: A Process for Integration of XML Schemata. In *CAISE*, 2005.
[68] L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW*, 2003.
[69] M. M. Moro, S. Malaika, and L. Lim. Preserving XML queries during schema evolution. In *WWW*, 2007.
[70] M. M. Moro, Z. Vagena, and V. J. Tsotras. Tree-Pattern Queries on a Lightweight XML Processor. In *VLDB*, 2005.
[71] M. M. Moro, Z. Vagena, and V. J. Tsotras. XML Structural Summaries. *PVLDB*, 1(2):1524–1525, 2008.
[72] M. Murata et al. Taxonomy of XML Schema Language using Formal Language Theory. *ACM TOIT*, 5(4), 2005.
[73] Y. Papakonstantinou and V. Vassalos. Architecture and implementation of an XQuery-based information integration platform. *IEEE Data Eng. Bull.*, 25(1):18–26, 2002.
[74] U. Park and Y. Seo. An Implementation of XML Documents Search System based on Similarity in Structure and Semantics. In *Int Workshop on Challenges in Web Information Retrieval and Integration*, pages 97–103. IEEE Computer Society, 2005.
[75] K. Passi et. al. A Model for XML Schema Integration. In *EC-Web*, 2002.
[76] E. Popovici, G. Ménier, and P.-F. Marteau. SIRIUS XML IR System: Approximate Matching Structure and Textual Content. In *INEX*, 2006.
[77] S. Puhlmann, M. Weis, and F. Naumann. XML duplicate detection using sorted neighborhoods. In *EDBT*, 2006.
[78] M. Raghavachari and O. Shmueli. Efficient Schema-Based Revalidation of XML. In *EDBT*, 2004.
[79] P. Rao and B. Moon. Sequencing XML data and query twigs for fast pattern matching. *ACM TODS*, 31(1):299–345, 2006.
[80] P. Rodriguez-Gianolli and J. Mylopoulos. A Semantic Approach to XML-Based Data Integration. In *ER*, 2001.
[81] M. Rys. XML and relational database management systems: inside microsoft sql server 2005. In *SIGMOD*, 2005.
[82] M. Rys, D. D. Chamberlin, and D. Florescu. XML and Relational Database Management Systems: the Inside Story. In *SIGMOD*, 2005.
[83] J. Shanmugasundaram et. al. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB*, 1999.
[84] J. Shanmugasundaram et. al. Efficiently Publishing Relational Data as XML Documents. *The VLDB Journal*, pages 65–76, 2000.
[85] J. Shanmugasundaram et. al. A general technique for querying XML documents using a relational database system. *SIGMOD Record*, 30(3):20–26, Sept. 2001.
[86] J. Shanmugasundaram et.al. Querying XML Views of Relational Data. In *VLDB*, 2001.
[87] H. Su et al. XEM: Managing the evolution of XML documents. In *RIDE-DM*, 2001.
[88] I. Tatarinov et. al. Updating XML. In *SIGMOD*, 2001.
[89] I. Tatarinov et. al. Storing and Querying Ordered XML Using a Relational Database System. In *SIGMOD*, 2002.
[90] A. Vyas, M. F. Fernández, and J. Siméon. The Simplest XML Storage Manager Ever. In *XIME-P*, 2004.
[91] L. Wang, M. Mulchandani, and E. A. Rundensteiner. Updating XQuery Views Published over Relational Data: A Round-trip Case Study. In *XSym*, 2003.
[92] L. Wang and E. A. Rundensteiner. On the updatability of XML Views Published over Relational Data. In *ER*, 2004.
[93] Y. Wang, D. DeWitt, and J. Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *ICDE*, 2003.
[94] H. Wang et. al. ViST: a dynamic index method for querying XML data by tree structures. In *SIGMOD*, 2003.
[95] L. Wang et. al. An Optimized Two-Step Solution for Updating XML Views. In *DASFAA*, 2008.
[96] M. Weis and F. Naumann. DogmatiX Tracks down Duplicates in XML. In *SIGMOD*, 2005.
[97] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, Mar 1992.
[98] E. Wilde and R. J. Glushko. XML Fever. *Commun. ACM*, 51(7):40–46, 2008.
[99] Y. Wu, J. M. Patel, and H. V. Jagadish. Structural join order selection for XML query optimization. In *ICDE*, 2003.
[100] R. Yang, P. Kalnis, and A. K. H. Tung. Similarity Evaluation on Tree-structured Data. In *SIGMOD*, 2005.
[101] X. Yang, M. L. Lee, and T. W. Ling. Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach. In *ER*, 2003.
[102] C. Zhang et. al. On Supporting Containment Queries in Relational Database Management Systems. *SIGMOD Record*, 30(2):425–436, 2001.