IVANICA, ADINA D., M.S. Interoperability of XML and Relational Data – Optimization Algorithm. (2005)
Directed by Dr. Fereidoon Sadri. 113 pp.

Within the past six years, Extensible Markup Language (XML) has spread rapidly and has gained popularity in the database community with its primary focus in the design of query languages and storage methods to select data from vast amounts of XML data efficiently. In this respect, I discuss some of the research that has been done by presenting three papers that describe different approaches to querying XML documents.

This thesis concentrates on the method used by Sadri and Lakshmanan in [1]: viewing an XML document as a relational database upon which the user can write simple SQL queries that can be translated into equivalent XQuery queries. Taking the output of the translation algorithm presented, I further develop an optimization algorithm meant to decrease the running time of the translated queries. I mainly focus on two aspects: the need of the *distinct-values()* function and the minimization of the number of variables.

INTEROPERABILITY OF XML AND RELATIONAL DATA – OPTIMIZATION ALGORITHM

by

Adina D. Ivanica

A Thesis Submitted to
the Faculty of The Graduate School at
The Univeristy of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2005

Approved by

_____

Committee Chair

APPROVAL PAGE

This thesis has been approved by the following committee of the Faculty of the Graduate School at the University of North Carolina at Greensboro.

Committee Chair _____

Committee Members _____

_____

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGEMENTS

I want to thank my advisor, Dr. Sadri for his assistance and guidance in writing this thesis. This research was aided by National Science Foundation, grant IIS-0083312.

TABLE OF CONTENTS

CHAPTER I

PRELIMINARIES

1.1 Introduction

Over the past six years, Extensible Markup Language (XML) has spread rapidly and has gained great popularity as a universal data exchange format. As it grows towards becoming the standard for e-business, the amount of exchanged XML data grows exponentially. As a direct implication, the requirement to store and query XML efficiently increases rapidly. In this context, the researchers have oriented their attention finding ways of querying XML data, and many query languages for XML such as Lorel, Quilt, XQL, XML-QL, XPath and XQuery have appeared. In this respect, I have reviewed different approaches in the next section of this chapter.

The most popular and generally accepted language for querying XML is XQuery, a superset of XPath, which uses a structured query approach. Its syntax is complicated, not appropriate for a novice user, and requires extensive knowledge of underlying schema.

In [1], Lakshmanan and Sadri develop a new way of approaching XML documents – through their information content, which can be represented as a relational database. SQL queries can easily be expressed on the new generated table, and the paper also provides an algorithm for translating them into XQuery queries against the original

XML document. This work concentrates on the optimization of the translation algorithm.

1.2 General and actual context of XML in the databases' world

XML is defined by the XML Core Working Group as a "simple, very flexible text format derived from SGML (ISO 8879)". Although it was originally designed to meet the challenges of large-scale electronic publishing, XML is now playing an increasingly important role in the exchange of a wide variety of data on the Web. Since the traditional search engines are not well suited for querying XML, due to their lack in exploiting the structure of the documents, the database community has focused its attention lately on developing ways of querying the large amounts of XML data that are already accessible to the public.

I will briefly present three papers that discuss different approaches in reviewing and querying XML documents. They have been introduced in the proceedings of different international database conferences in the last two years.

1.2.1 "Generating Relations from XML Documents",

by Sara Cohen, Yaron Kanza and Yehoshua Sagiv

The paper was presented in the Proceedings of the 9th International Conference on Database Theory in January 2003. In this paper the authors introduce several mechanisms for creating relations out of XML documents. Their belief is that a universal relation interface that would hold for a set of documents can facilitate a natural language approach in querying over XML data. In this context, they focused on finding ways to

determine pairs of nodes that are *meaningfully related* and present different methods for solving the problem. The choice of using the best approach for a specific domain is left to the user's judgment. The method is intended for the experienced user rather than the public at large.

The assumption behind this work is that the XML document can be represented as a tree. Given a tuple of path expressions, their aim is to find tuples of nodes from a given document that match the path expression and are meaningfully related.

Framework

The following definitions and results are taken directly form [5].

- *Relations*

Definition 1.2.1.1 A **tuple** has the form $t = (c_1:a_1, \ldots, c_k:a_k)$ where c_i and a_i are a column name and a value. Tuples can have columns with null values, denoted $\perp$, and multiple columns with the same name. $(c_1, \ldots, c_k)$ is called signature of t.

A ***relation*** R is a bag of tuples with the same signature, called the signature of R

- *Trees*

Let *L* be a set of labels and A a set of constants.

An XML document is a tree T in which each interior node is associated with a label from *L* and each leaf node is associated with a value from A. A label of an interior node n is label(n) and the value of a leaf node m is val(m).

The val function can be extended to interior nodes by defining val(n) to be the concatenation of the values of its leaf descendents.

- *Relationship Tree*

Definition 1.2.1.2 Let T be a tree and n_1, n_2 nodes in T. Let n be the lowest common ancestor of the two nodes and T_n the subtree rooted at n. $T_{|n1,n2}$ is the tree obtained by pruning from T_n all nodes other than n_1 and n_2 that are not ancestors of either n_1 or n_2. This is the ***relationship tree*** of n_1 and n_2.

Example. In Fig. 1 the lowest common ancestor of 8 and 13 is 7. The relationship tree of 8 and 13 is comprised of the nodes 7, 8, 10 and 13.

- *Graphs and Interconnection Graphs*

Definition 1.2.1.3 Let n and m be nodes in T. We say that n and m are *interconnected* if one of the following conditions holds:

  - The relationship tree of n and m does not contain two distinct nodes with the same label, or
  - The relationship tree of n and m contains exactly one pair of distinct nodes with the same label and this pair is comprised of n and m

Given a tree T, the *interconnection graph* IG(T) is an undirected graph that has the same number of nodes as T and has an edge between any nodes n and m that are interconnected.

Creating Relations from Trees

Definition 1.2.1.4 An **atomic path** (α) is either a nonempty disjunction (l_1 | … | l_k) of labels from *L* or the wildcard symbol *.

Definition 1.2.1.5 A **path expression** (ω) is either an atomic path expression or of the form ω'/α or $\omega'//\alpha$, whereω' is a path expression and α is an atomic path expression

A node n in a tree T matches a path expression (n |= ω) if:

- ω = (l_1 | … | l_k), the root of T is n and label(n) = l_i for some i ≤ k;
- ω = * and the root of T is n;
- ω = ω'/α, the parent of n matches ω' and n |= α in the subtree rooted at n;
- ω = ω'// α, there is an ancestor of n that matches ω' and n |= α in the subtree rooted at n.

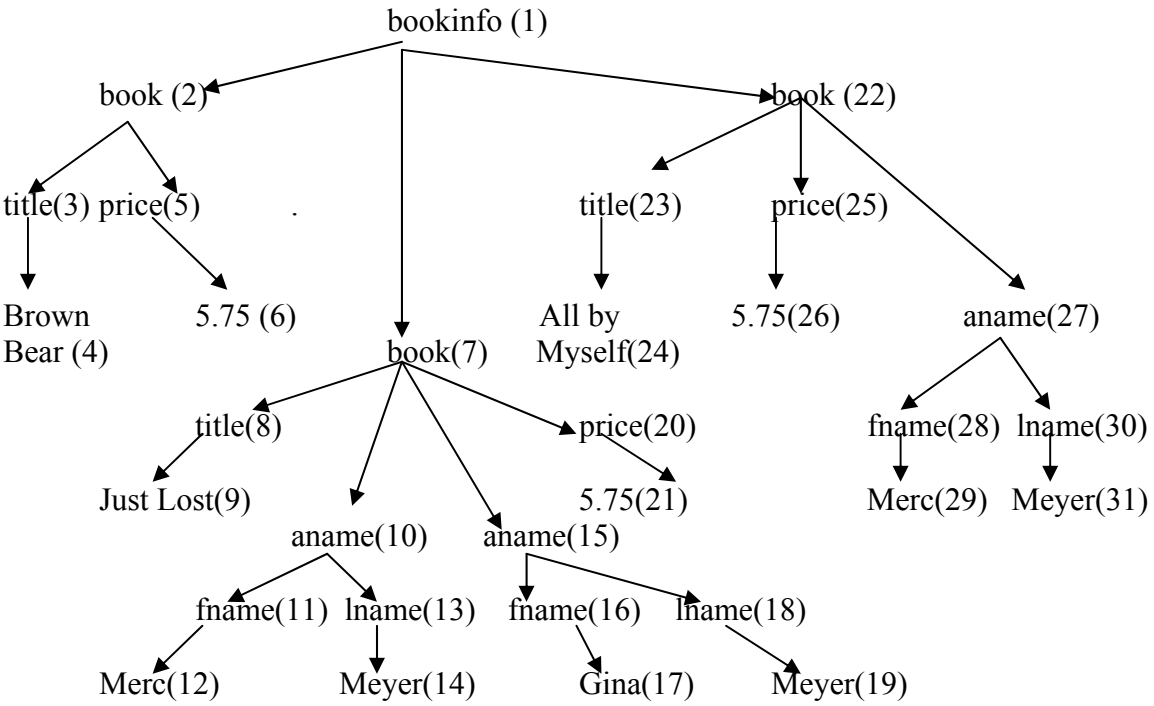


Fig. 1 – a bibliography grouped by book (T_{bk})

Ex.: bookinfo/* - matches any child of a root with label *bookinfo* , *//book/(aname | price) - matches any *aname* or *price* node in a tree that is a child of a *book* node, *//* - matches any node in any tree

Let T be a tree and (ω_1, …, ω_m) be a tuple of path expressions. We are interested in finding tuples of nodes (n_1, …, n_m) from T that satisfy:

- For all $i \leq m$, the node n_i matches the path expression ω_i and
- The nodes n_1, …, n_m are meaningfully related

The interconnection graph contains information about pair of nodes that are related. There are three different semantics that enable to decide whether larger tuples of nodes are related:

1. **Completely-Interconnected**: a set of nodes N are completely-interconnected in a tree $T \approx_c \{N\}$ if the interconnection graph of T, projected on N is a complete graph;
2. **Reachably-Interconnected**: a set of nodes N are reachably-interconnected in a tree $T \approx_r \{N\}$ if the interconnection graph of T, projected on N is a connected graph; Meaningful relationships are transitive;
3. **Star-Interconnected**: a set of nodes N are star-interconnected in a tree $T \approx_s \{N\}$ if the interconnection graph of T, projected on N is a star graph;

Definition 1.2.1.6 Let $\Omega = (\omega_1, \ldots, \omega_m)$ be an m-tuple of path expressions and T be a tree with a set of nodes N. We say that a function $\mu : \{1,..,m\} \rightarrow N \cup \{\perp\}$ is ***a matching of Ω to T*** if for all $i \leq m$, either $\mu(i) \models \omega_i$ or $\mu(i) = \perp$. The set of nodes in the image of a matching μ is denoted by Image(μ).

$Mat_T^c(\Omega)$ is used to denote the set of matchings μ, such that the nodes in Image(μ) are completely-interconnected ($\approx c\{Image(\mu)\}$); $Mat_T^r(\Omega)$ is the set of matchings μ, such that the nodes in Image(μ) are reachably-interconnected ($\approx r\{Image(\mu)\}$) and $Mat_T^s(\Omega)$ is the set of matchings μ, such that the nodes in Image(μ) are star-interconnected ($\approx s\{Image(\mu)\}$). Observe that:

$$Mat_T^c(\Omega) \subseteq Mat_T^s(\Omega) \subseteq Mat_T^r(\Omega)$$

- A matching μ' subsumes μ, denoted $\mu \prec \mu$', if μ' gives the same value for every index that is assigned a non-null value by μ: for all $i \leq m$, either $\mu'(i) = \mu(i)$ of $\mu(i) = \perp$.
- An interconnection assignment μ is maximal if for every matching μ', we have $\mu \prec \mu$' implies $\mu = \mu$'

Definition 1.2.1.7 The pair $\Delta = (\Omega,k)$, where Ω is an m-tuple of path expressions and $k \leq m$, is called ***relation generator***. For $\square \in \{c, r, s\}$, $MMat_T^{\square}(\Omega,k)$ denotes the set of maximal elements in $Mat_T^{\square}(\Omega)$ that do not map any of the first k path expressions to the null values.

Evaluating a relation generator involves finding tuples of nodes that satisfy the path expressions and are related to one another in a meaningful fashion.

A relation generator Δ can be used to create a relation out of parts of an XML document. Given $\Delta = (\Omega,k)$ and a tree T, compute $MMat_T^{\square}(\Omega,k)$ with $\square \in \{c, r, s\}$ chosen as desired.

Example Uses for Relation Generating Mechanism

- Querying Trees and Relations Using SQL

The authors suggest an extension of the FROM clause of SQL to allow "on-the-fly" creation of the relations from trees. If we wish to query on the document presented in the figure above and a table UserRatings(title,user,rating) stored in a relational database for titles of books with a rating of at least 8 and author "Smith" the SQL query looks like:

```
SELECT    Book.title
FROM      Complete(T_bk, ('*//title','*//aname'),0) as Book(title,author), UserRatings
WHERE     Book.title = UserRatings.title and
          Book.author like '%Smith%' and rating ≥ 8
```

- An XML Search Engine

A simple search query can have the form:

$path\_expression_1$: $search\_phrase_1$ … $path\_expression_n$: $search\_phrase_n$

Example. Search for books with a title containing the word XML and either the author *Smith* or no specified author (+ prefaces a path expression):

+ *//title : XML *//aname : Smith

The $MMat_T^c((*//title,*//aname),1)$ (or $MMat_T^r((*//title,*//aname),1)$ or $MMat_T^s((*//title,*//aname),1)$ is computed on the documents T available. Then, only the tuples that satisfy the search conditions would be retuned.

In the last section of the paper the authors prove that $MMat_T^{\square}(\Omega,k)$ can be computed in polynomial time under input-output complexity.

One obvious problem with their approach is the underlying assumption of a tree representation for XML documents, ignoring the existence of IDREF/IDREFS attributes

or recursive definitions in the Document Type Definition (DTD). As I already mentioned, the authors argue that the relation generator allows naive users to retrieve "interesting and naturally related portions of a document," but still the user should have some knowledge about which of the three cases (completely-, reachably- or star- interconnected) his document falls under. Another open problem they discuss is related to the indexing over a XML document in a fashion that will allow relation generators to be quickly evaluated.

1.2.2 "XSEarch: A Semantic Search Engine for XML",

by Sara Cohen, Jonathan Mamou, Yaron Kanza and Yehoshua Sagiv

The paper was introduced at the Proceedings of the 29th Very Large Data Bases (VLDB) Conference in 2003. It is a continuation of the work described above, and it shows how the theoretical results from [5] can be efficiently combined with information retrieval techniques to yield XSEarch – a search engine for XML, built on a keyword-based approach that always returns, as answers, *document fragments* that are semantically related, even when only keywords (and no tags) are specified in the query.

Query Syntax

The query language for standard search engines is just a list of keywords. Keywords with a plus sign prefacing them must appear in the satisfying document. Besides these requirements, the XSEarch query language allows the user to specify labels and keyword-label combinations that must or may appear in a satisfying document. A search term has the form *l:k*, *l:* or *:k* where *l* is a label and *k* is a keyword. Required terms have a plus sign prepended, otherwise they are optional terms.

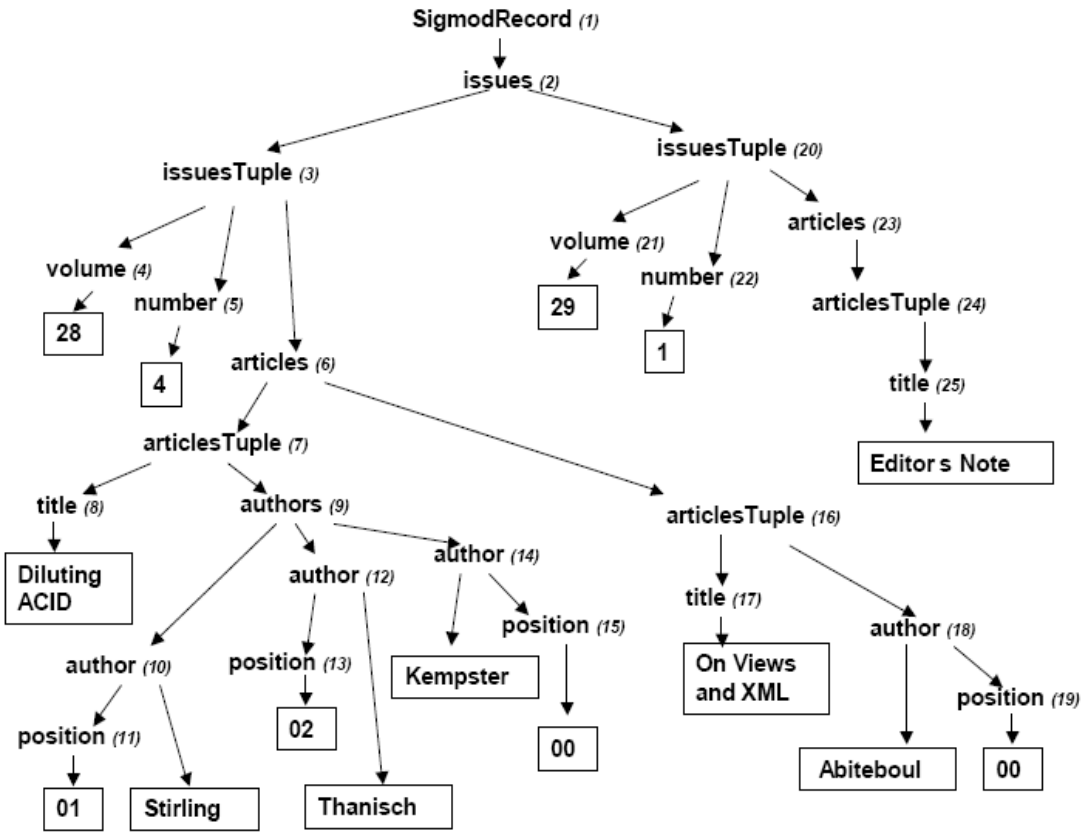


Fig. 2 – Part of the Sigmond record document tree (T_{sr}).

In order to satisfy a query Q, each required term in Q must be satisfied. In addition, they must be meaningfully related. The authors assume there exists a given relationship *R* that determines when two nodes are related. *R* can be extended to an arbitrary set of nodes. Going back to the tree representation they used in [5], an interior node is associated with a label and each leaf node with a sequence of keywords.

Definition 1.2.2.1 Let *n* be an interior node in a tree T. Then:

- *n* satisfies the search term *l:k* if *n* is labeled with *l* and has a descendant that contains the keyword *k*
- *n* satisfies the search term *l:* if n is labeled with *l*
- *n* satisfies the search term *:k* if *n* has a leaf child that contains the keyword *k*

Example In Fig. 2 node number 14 satisfies *:Kempster* and node number 9 satisfies *authors:Kempster.*

Let T be a tree and R be a binary, reflexive and symmetric relationship on the nodes of T.

Definition 1.2.2.2. A set of nodes N is *all-pairs R-related*, denoted $\approx_a^R\{N\}$, if (n_1,n_2) is in R for every pair of nodes n_1, n_2. This is equivalent to saying that a set of nodes is meaningfully related when every pair of nodes in the set is meaningfully related.

Definition 1.2.2.3 A set of nodes N is *star R-related*, denoted $\approx_s^R\{N\}$, if there is a node $n_* \in N$ such that the pair (n_*,n) is in R, for all nodes $n \in N$. n_* is called star center. Intuitively, this states that the nodes of a set are meaningfully related if all these nodes are meaningfully related to a node in the set.

Query Answers

Definition 1.2.2.4 Let $Q(t_1,\ldots,t_m)$ be a query, where $t_1,\ldots,t_m$ represent the required and optional terms. The sequence $N = n_1,\ldots,n_m$ of nodes and null values is an *all-pairs R-answer* for Q if the nodes in N are all-pairs R-related and for all $1 \le i \le m$:

1. n_i is not the null value if t_i is a required term
2. n_i satisfies t_i if it is not the null value

Similarly, N is a star R-answer when the nodes in N are *star R-related*. $Ans_T^{a,R}(Q)$ denotes the set of all-pairs R-answers for the query Q over the tree T and $Ans_T^{a,R}(Q)$ denotes the set of star R-answers for Q over T. Observe that: $Ans_T^{a,R}(Q) \subseteq Ans_T^{a,R}(Q)$.

Definition 1.2.2.4 An answer N' *subsumes* N if N' is equal to N on all non null values of N.

The answer N is *maximal* if every answer that subsumes N is actually equal to N. For $\square \in \{a, s\}$, $MaxAns_T^{\square,R}(q)$ denotes the set of maximal answers in $Ans_T^{\square,R}(Q)$. Either relation described above can be used for getting a query answer, but depending on the document's structure one can be more appropriate and lead to better results.

Definition 1.2.2.5 Let n and n' be nodes in T and let $T_{|n,n'}$ be the relationship tree as described in *Definition 1.2.1.2*. It is said that n and n' are interconnected if one of the following conditions holds:

1. $T_{|n,n'}$ does not contain two distinct nodes with the same label or
2. the only two distinct nodes in $T_{|n,n'}$ with the same label are n and n'.

R_i is used to denote the interconnection relationship.

For a better intuition of these results the authors present some examples which I reproduce below.

Examples.

- Consider the query Q_1(+ title:, author:) that finds pairs of titles and authors belonging to the same article in the document presented in Fig. 2. Only the tuples with non-null title values are returned and these are: (8, 10), (8, 12), (8, 14), (17, 18) and (25, ⊥).

- Consider now the same query as above expressed on the document represented in Fig. 3. The answer consists of (6, 3) and (6, 4). Although the documents are different in structure, the correct pairs are found in both cases.

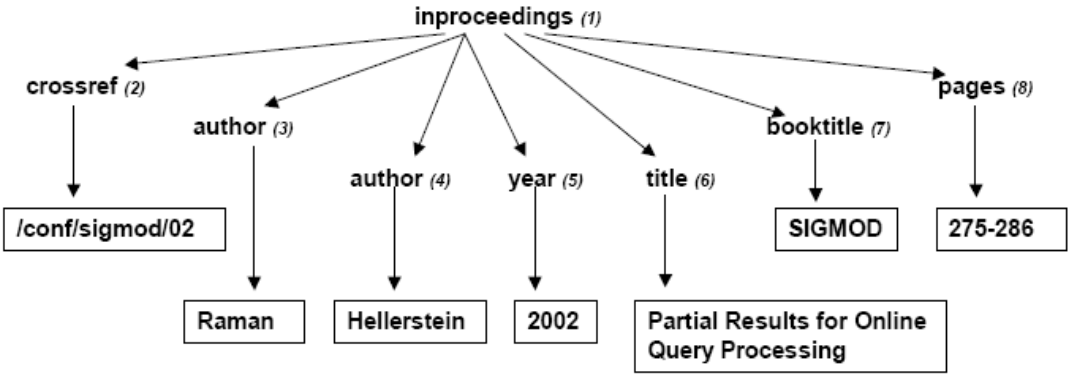


Fig. 3

- Consider query Q_2 that looks for volumes, authors with the name Kempster, and authors who have published with Kempster:

  Q_2(+ volume:, + author:Kempster, author:)

  The set of maximal all-pairs answers for Q_2 over T_{sr} only contains answers for which both authors appear under the same articlesTuple node, which are the correct results. However, the set of maximal star answers for Q_2 contains authors that appear in the same issue, but under different articles. And this is not the desired answer.

- Consider the following document fragment:

```
<proceedings>
    <inproceedings>
            <author>Moshe Y. Vardi</author>
            <title>Querying Logical Databases</title>
    </inproceedings>
    <inproceedings>
            <author>Victor Vianu</author>
            <title>A Web Odyssey: From Codd toXML</title>
    </inproceedings>
</proceedings>
```

and query Q_3(+ :Vianu, + :logical, + :databases) which assumes no knowledge of the tags in the document. The answer to this query, as expected, will be empty for both all-pair and star-semantics.

The complexity results from [5] apply here also. Besides, the XSEarch system has implemented a method that ranks the answers by their estimated relevance. The XSEarch ranker gives a score to each query answer N by taking into consideration both the structure of the result as well as its contents.

The paper describes in detail the ranking factors, system implementation, as well as experimental results. It is not the purpose of this thesis to analyze in detail the XSEarch system architecture, but I want to present a certain situation when XSEarch fails.

Consider the document presented in Fig.4 and query Q_4 that finds title of the publications for which Mary is an author.

Q_4(+ title:, + author:Mary)

The answer to Q_4 comprises of (2, 3) and (10, 3). The problem arises form the multiple logical hierarchies of the XML document. There is also a case in which XSEarch does not recognizes meaningful answers: when the tag names involved in the

query are different but refer to the same entity (ex: XSEarch will treat article and publication differently). These issues are addressed by the Schema-Free XQuery [6] which I present in the next section.

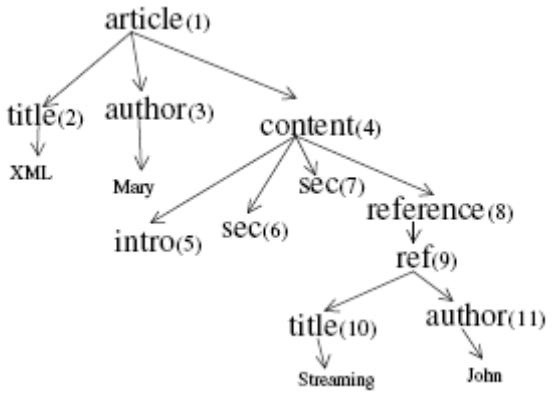


Fig. 4 – An XML document with multiple hierarchies

1.2.3 "Schema-Free XQuery",

by Yunyao Li, Cong Yu and H.V. Jagadish

Schema-Free XQuery was presented in the Proceedings of the 30th VLDB Conference in 2004. Addressing the same problem of querying over XML data, the authors propose the addition of a new functionality to XQuery that enables users to take full advantage of XQuery in querying XML data precisely without requiring complete knowledge of the document structure.

The framework developed in the paper exploits whatever partial knowledge of the schema the user has. Knowing the full schema allows a user familiarized with XQuery to acquire all the meaningful information of interest to him. Without this knowledge, they

can specify keywords, and the system developed in the paper will respect only the given specifications.

The starting point is the same that [6] uses: the tree representation of the XML document and the Lowest Common Ancestor (LCA) of individual term/tag matches. The authors refine the LCA of nodes belonging to an XML document by defining the concept of Meaningful Lowest Common Ancestor Structure (MLCAS), which meaningfully relates together the nodes corresponding to the relevant variables in an XQuery expression. The addition to standard XQuery consists of the "*mlcas*" function. While the use of new *mlcas*-embedded XQuery is trivial and gives the user the full power of XQuery in the absence of the schema specification, the actual background computation is very complicated. The hidden algorithm for computing all MLCASs for a given set of nodes is reproduced in Appendix D.

The following definitions are taken directly from the paper. For a better intuition of the MLCAS concept, additional examples are presented after defining the key terms.

Definition 1.2.3.1 An entity type of a node in an XML tree is defined as the tag name of the node. Two nodes n_1 and n_2 are of the same entity type if and only if they have the same tag name.

Definition 1.2.3.2 (MLCA of two nodes) Let the set of nodes in an XML document be N. Given A,B $\subseteq$ N, where A is comprised of nodes of type $\mathcal{A}$ and B is comprised of nodes of type $\mathcal{B}$, the Meaningful Lowest Common Ancestor Set C $\subseteq$ N of A and B satisfies the following conditions:

- $\forall\, c_k \in C,\ \exists\ a_i \in A,\ b_j \in B$ such that $c_k = LCA(a_i,b_j)$;

c_k is denoted as MLCA(a_i, b_j)

- $\forall a_i \in A$, $b_j \in B$, if d_{ij} = LCA(a_i, b_j) and $dij \notin C$, then $\exists c_k \in C$, descendant(c_k,d_{ij})=true, where the function descendant(a,b) returns true if a is a descendant of b.

Definition 1.2.3.3 (MLCA of multiple nodes) Let the set of nodes in an XML document be N. Given $A_1,A_2,\ldots,A_m \subseteq N$, where $\forall j, a_{ij} \in A_i$ is of type $\mathcal{A}_i$ ($i \in [1,\ldots,m]$), a Meaningful Lowest Common Ancestor c = MLCA($a_1,\ldots,a_m$), where $a_i \in A_i$ ($i \in [1,\ldots,m]$), satisfies the following conditions:

- $\forall$ j, $k \in [1,\ldots,m]$ ($j \neq k$), $\exists$ m = MLCA(a_j,a_k), m ≠ null and descendant-or-self(m,c) =true;
- $\exists$ j, k $\in [1,\ldots,m]$ ($j \neq k$), c = MLCA(a_j,a_k).

Definition 1.2.3.4 (MLCAS) Let the set of nodes in an XML document be N. Given $A_1,A_2,\ldots,A_m \subseteq N$, where $\forall i, a_{ij} \in A_i$ is of type $\mathcal{A}_j$ ($j \in [1,\ldots m]$), a Meaningful Lowest Common Ancestor Structure Set S = { (r, $a_1,\ldots,a_m$)| $r \in N$, $a_i \in A_i$ ($i \in [1,\ldots,m]$), r= MLCA($a_1,\ldots,a_m$)}. Each element of this set is denoted as MLCAS($a_1,\ldots,a_m$), with r as its root.

Each MLCAS contains only the nodes that are meaningfully related to each other and it is a refined context for query evaluation.

Definition 1.2.3.5 (*mlcas* Fuction) mclas($a_1,\ldots,a_n$) is a function that returns (i) root node of MLCAS($a_1,\ldots,a_n$) if it exists, (ii) null otherwise.

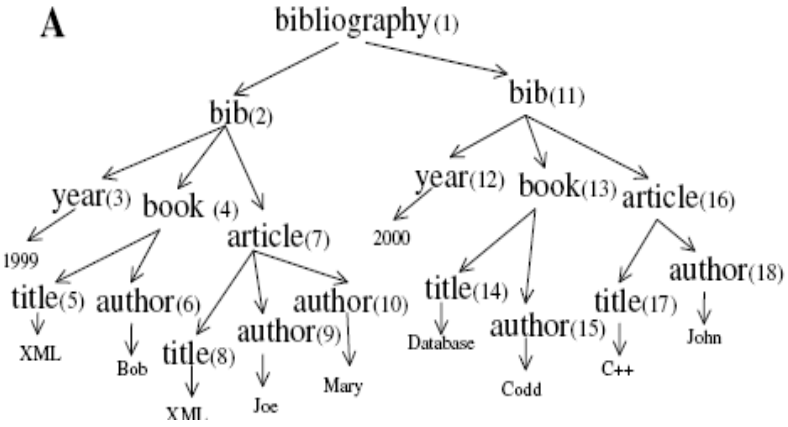


Fig. 5 – A bibliography document

Example. In running the following query over the XML document in Fig. 5: find title and year of the publications of which Mary is an author, the MLCAS is comprised of the following elements: (2, 6, 5, 3), (2, 10, 8, 3), (11, 15, 14, 12) and (11, 18, 17, 12). The only MLCAS satisfying the original search condition, author = ”Mary”, is (2, 10, 8, 3), hence the result is (title = “XML”, year = “1999”).

Without structural knowledge of the document, this query can be written in an XQuery syntax as:

```
for     $a in doc(“bib.xml”)//author,
        $b in doc(“bib.xml”)//title,
        $c in doc(“bib.xml”)//year
where   $a/text() = “Mary”
return  <result> { $b, $c } </result>
```

The result will retrieve numerous meaningless answers because the context is too general. Enriched with the *mlcas* function, the query becomes:

```
for     $a in doc("bib.xml")//author,
        $b in doc("bib.xml")//title,
        $c in doc("bib.xml")//year
where   $a/text() = "Mary"
        and exists mlcas($a, $b, $c)
return  <result> { $b, $c } </result>
```

and it is guaranteed to exclude the unrelated results.

To further simplify the syntax of Schema-Free XQuery, the actual implementation requires a simple addition of the *mlcas* keyword for each variable defined in the "for" clause. The same query, in a simple *mlcas*-enhanced XQuery format, becomes:

```
for     $a in mlcas doc("bib.xml")//author,
        $b in mlcas doc("bib.xml")//title,
        $c in mlcas doc("bib.xml")//year
where   $a/text() = "Mary"
return  <result> { $b, $c } </result>
```

While the *mlcas*-enhanced XQuery addresses the issue of structure ambiguity, it relies on the correctness of the element tag names in a given query. If the creator of the XML document uses *au* or *auth* instead of *author*, the query will return unwanted empty results. To resolve the "tag name ambiguity," the authors of Schema-Free XQuery system propose the addition of another simple function, *expand()*, meant to solve the user's lack of knowledge of the exact tag name. *expand(k)* will search for k equivalents in a domain-specific thesaurus, returning matches also found within the document. The above query rewritten is:

```
for     $a in doc("bib.xml")//expand(author),
        $b in doc("bib.xml")//title,
        $c in doc("bib.xml")//year
where   $a/text() = "Mary"
        and exists mlcas($a, $b, $c)
return  <result> { $b, $c } </result>
```

The principle of XQuery is very simple: to help the users construct meaningful queries when the knowledge of the schema (in terms of structure and tag name ambiguity) is missing. However, the time cost for adding the *expand()* function to the *mlcas*-enriched XQuery is proportional to the total number of synonyms of all the *expand* marked tag names. Although the authors come up with a "term normalization" approach to solve this issue, the method seems unrealistic, as it tries to standardize the tags that one can use in an XML document, or to fetch at the validation time the standard version of the tag name if the one used is non-standard.

One important drawback of the MLCAS approach is in the running time. Based on the experiments with XMark queries, the overhead for *mlcas*-embedded XQuery, compared to schema-aware XQuery varied from 0% to 250%.

1.2.4 Examples

A walk-through example for the methods described above

For all the examples, I will consider the XML document in Fig.6, which conforms to the DTD below. Note the presence of an ID attribute that does not appear in the representation of the XML tree structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE dept [ -->
  <!ELEMENT department     (faculty+,staff*,lecturer*)>
  <!ELEMENT faculty        (name,secretary,RA*,TA*)>
  <!ELEMENT staff          (name)>
  <!ELEMENT lecturer       (name,RA*,TA*)>
  <!ELEMENT name           (#PCDATA)>
  <!ELEMENT RA             (#PCDATA)>
  <!ELEMENT TA             (#PCDATA)>
  <!ATTLIST faculty faculty-id ID #REQUIRED>
  <!ATTLIST staff staff-id ID #REQUIRED>
  <!ATTLIST lecturer lecturer-id ID #REQUIRED>
<!-- ]> -->
```

Suppose that we are interested in finding all the teacher's assistants (TA) of the faculties listed under each department. For simplification we look at one department, as depicted in Fig. 6. Our query result should list faculty ids, names and their teacher assistants if they exist (have non-null values).

- Generating relations from XML documents

With the method described in [5], the query will be transformed to the relation generator similar to: ((*//faculty-id, *//name, *//TA),3). The first three xpaths will give us triplets of nodes of interest to us, which are meaningfully related. The 3 in the relation generator indicates that nodes with null values are not acceptable for any of the specified

paths. For example, we want to eliminate a potential answer that lists a faculty name with no teacher assistant.

There are four *faculty* nodes, six *name* nodes and three *TA* nodes. The next step is to build the interconnection graph.

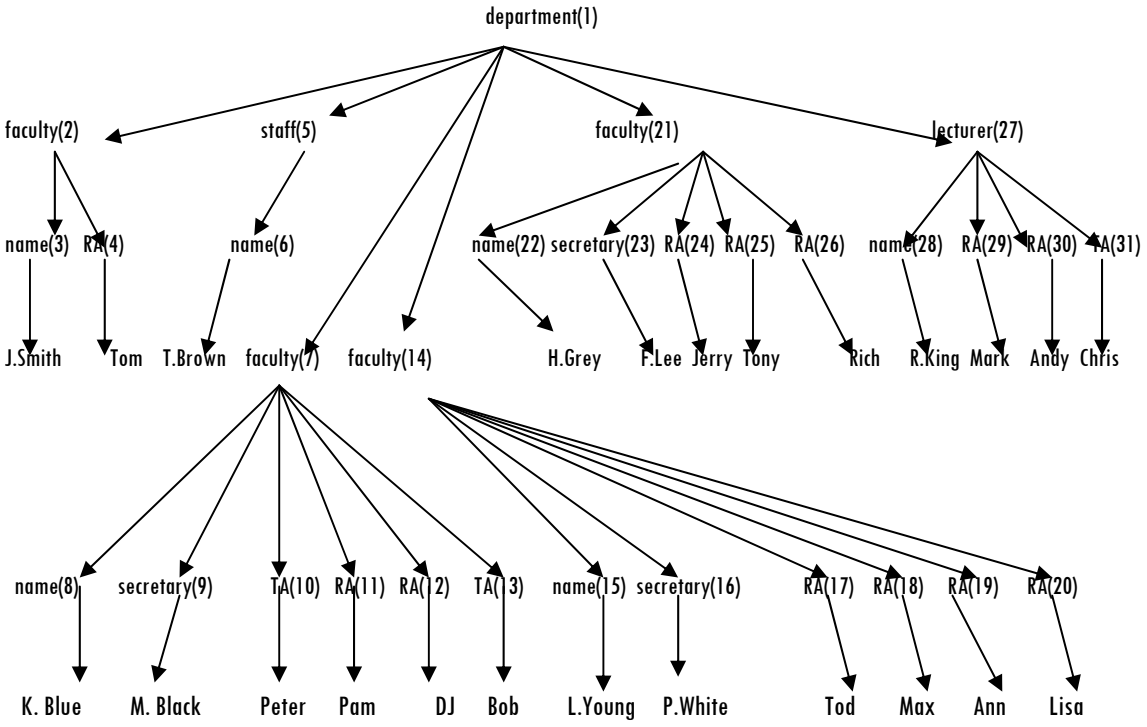


Fig. 6 – An XML document

Conforming to definitions 1.2.1.3 – 7, some of the relations we get are presented in Table 1(a).

After eliminating the tuples with null values and applying the *val* function we reduce the answer to tuples presented in Table 1(b).

Faculty-id	name	TA
id (2)	3	null
id (2)	6	null
id (2)	28	31
id (7)	6	null
id (7)	8	10
id (7)	8	13
id (7)	28	31
id (14)	6	null
id (14)	15	null
id (14)	28	31
id (21)	6	null
id (21)	22	null
id (21)	28	31

Table 1(a)

faculty-id	name	TA
id (2)	R.King	Chris
id (7)	K.Blue	Peter
id (7)	K.Blue	Bob
id (7)	R.King	Chris
id (14)	R.King	Chris
id (21)	R.King	Chris

Table 1(b)

Going back to the document and analyzing the answer we find out that the only correct answers are:

faculty-id	name	TA
id (7)	K.Blue	Peter
id (7)	K.Blue	Bob

Table 2

Why did we get four more unwanted tuples? The problem arises from the DTD itself and the fact that faculty, staff and lecturer all have a subelement named *name*. When we look at different *name* nodes in the XML tree, in order to establish their interconnection (if they are meaningfully related or not, see definition 1.2.1.3), the tree rooted at their LCA will give us an interconnection relation for (3,6), (3,28), (6,8), (6,15), (6,22), (6,28), (8,28), (15,28) and (22,28). Since all of the name nodes aren't

meaningfully related to each other, we should make the system differentiate between faculty-name, staff-name and lecturer-name. By renaming them fname, sname and lname respectfully, the procedure described in [5] works successfully. The query will be transformed into a relation generator similar to: ((*//faculty-id, *//fname, *//TA),3) which will return only the right answers: (id(7), K.Blue, Peter), (id(7), K.Blue, Bob).

- XSEarch

The query described above translated in an XSEarch language looks like:

Q(+ faculty-id:, + name:, + TA:)

Since the idea behind the XSEarch is the same generation of the relationship tree as in [5], we expect that the methods will produce the same results. See Table 1(b). If we use the renaming as above, the XSEarch will again return the correct answers.

- Schema-Free XQuery

The *mlcas*-enhanced XQuery is:

```
for     $i in mlcas doc("dept.xml")//faculty-id,
        $n in mlcas doc("dept.xml")//name,
        $t in mlcas doc("dept.xml")//TA
return <result> {$i, $n, $t} </result>
```

Unlike the other two methods, the Meaningful Lowest Common Ancestor of nodes name and *TA* will work correctly, producing the expected answers. For example the MLCA of nodes 3 and 10 does not exist because node 8, with the same entity type as 3 (name) is more related to 10. So, the answers are: (7,id(7),8,10) and (7,id(7),8,13).

1.3 A review of "On the Information Content of an XML Database"

The paper contributes with a new way for querying XML data – through the information content of the XML database. The authors argue that querying a traditional relational model using SQL is much simpler than querying XML data using the powerful, but complex and complicated, XQuery.

The starting point in deriving the information content of an XML document is the document itself and its schema, which can be specified either as a DTD or XML Schema. If unknown, it can be obtained with one of the available tools for generating DTD/XML Schema out of an XML document (see Appendix A: relaxer and Stylus Studio both have this feature).

The document schema is a graph, which may be a tree, a directed acyclic graph (DAG) or can even contain cycles. Through an algorithm that performs a DTD transformation, we obtain an equivalent DTD, whose graphical representation always resembles a tree (schema tree in [1]). Associated with each XML document there is a document tree defined as "The document tree D(V,E) is a rooted ordered tree with set of nodes V and set of edges E. Each node $v \in V$ represents an element or attribute, and is labeled by the element tag or the attribute name. We also assume that each node has a unique identifier (which could be the pre-order number of the node). An edge $(v_i, v_j) \in E$ is a subelement or an attribute of v_i."

The information content of the XML document can be captured in two ways: a semantic approach, using the notion of valuations as described in the paper, or a syntactic

approach, based on combining parent/child relationships (or hierarchical relationships). Both ways will equivalently produce the same XML database, called Hierarchical Information Content Tableaux (HICT) – the relational model representation for the XML document. The HICT is used as a view upon which user queries are formulated. The system translates the SQL queries back to XML and executes them against the original document.

The paper takes care also of the IDREF/IDREFS relationship representations (with their equivalent key/keyref in XML Schema). This is done using separate relational representations called Linked Information Content Tableaux (LICT) – which capture only the portion of the information content of the XML data that is represented through references.

Algorithm 4.1 for generating XPATH expressions (XGA)

Let T be the schema tree of the document being analyzed, A a set of given attributes of T and N_A the set of nodes in T that correspond to attributes in A. The algorithm uses a priority queue of entries, P, which is initialized to:

P = { (n, l, -) | n$\in$ N_A and l is the level number of n};

- ($\forall$) (n, l, p) entry in the P that has the lowest level (largest l) replace (n, l, p) with (parent, l-1, n/p) where is the parent of node n and n/p is an xpath expression;
- If a node m is listed more than once in P we found a least common ancestor (lca)

- For each occurrence (m, k, p) of a repeated node, generate the following XPATH: $O : $M/p, where O is the last node of the xpath p. The algorithm records the lca, M , for further inline optimization issues;
- Replace all entries corresponding to a repeated node m (m, k, p) by a single entry of the form (m, k, -)

- Repeat the first two steps until P has only one entry;
- When P has a single entry of the form (m, k, -) add the declaration

  for $M in doc(…)/path

  where path is the path from root to m in the schema tree T. Remove the last entry from P.
- If the DTD transformation algorithm was used to obtain the schema tree T, the tags in the XPATH expressions should be restored to their original names.

Algorithm 4.2 for translation of single-block user queries

Let Q be a user query in SQL of the following form:

select target
from HICT as h_1, HICT as h_2, …, HICT as h_k
where conditions

the translated query will have the form:

(variable declaration)
where (translated conditions)
return (translated target)

The algorithm is divided into three cases:

- *Variable Declarations*

Let A_i be the set of attributes that appear in the SQL user query on the HICT u_i. Let $A_i' \subseteq A_i$ be the set of attributes that appear in select clause of the user query.

The translated XQuery has variable declarations for:

- Auxiliary variables corresponding to lca nodes as discussed in Algorithm 4.1 (XGA)
- Let $A \in A_i'$ and P be the lowest node among the set of lca's that is an ancestor of A. If there exists a node on the path from P to A which has a * or + in the DTD (called multi-valued) than A's corresponding variable is declared: "for $A in $P/…/A"
- Let $A \in A_i'$. If A also appears in the where clause conditions (except for constrains of the form A in NULL), then its corresponding variable is also declared. These declarations have the form "for $A in xpathA" where xpathA is the XPATH obtained with XGA Algorithm for variable $A

➢ *Where-clause generation*

Under the assumption that the conditions in the where clause are in disjunctive normal form, A is an attribute, op comparison operator and v a value, a condition of the form "A op v" are replaced by "tr(A) op v" or tr(A)/text() depending on whether A is a leaf node or an internal node, where tr(A) is the translation of A (for more details refer to Conclusions and Future Work Chapter). "v op A" and "A1 op A2" are treated similarly; "A is not NULL" and "A is NULL" simply translates to "tr(A)" and "not(tr(A))"

➢ *Return-clause generation*

The translated xquery return clause for variable A in the target list of the select clause will contain a declaration like $A/text() if A has been declared or xpathA/text() otherwise. If A is an XML attribute, the text() function can be omitted.

Algorithm 4.3 for Translating Aggregate Queries

Translation refers to queries of form:

```
select h.A, aggregate-part
from HICT as h
where conditions
group by h.A
```

- Generate the declaration for the group-by variable $A using distinct-values function with where-clause conditions incorporated in the declaration
- Declarations for variables corresponding to SQL query attributes are generated using a slightly modified version of Algorithm 4.1. Variables corresponding to aggregate attributes and their auxiliary variables should be declared using let. The aggregate variable declaration should also include aggregation and formatting
- XQuery return clause is generated by a simple translation of the SQL return clause.

CHAPTER II

EXPERIMENTS WITH XMARK QUERIES

2.1 Introduction

The XML Benchmark Project [3], dated April, 2001, is an important reference point in helping with the transition from traditional databases to XML databases and the optimization of the XML, as it is addressed to both users and implementers required to compare XML databases independent of their specific application scenario. The project framework consists of a scalable document and a broad spectrum of different queries, covering the major aspects of query processing. The benchmark is specifically intended for XML query processors.

The set of queries presents challenges ranging from stressing the textual character of the document to data analysis queries. The benchmark results take into account only the query processor and its interaction with the data, ignoring any network overhead, communication costs, or transformations of the output that might be required. The authors argue that they based their example database on a model that makes the behavior of queries "predictable" and allows for a natural formulation of queries that "present concise challenges." A copy of the DTD they use is presented in Appendix A, and a sample XML document based on that schema is presented in Appendix B.

2.2 XMark queries

The twenty benchmark queries are grouped in fourteen categories and they will be presented below.

- *Exact Match* - tests the database ability to handle simple string lookups with a fully specified path.

Q1. "Return the name of the item with ID 'item20748' registered in North America."

```
for $b in
document("auction.xml")/site/regions/namerica/item[@id="item20748"]
return $b/name/text()
```

- *Ordered Access* – refers to the intrinsic order of XML documents and how the database management system (DBMS) handles queries with order constraints(Q2); the cost of array look-ups(Q3, Q4).

Q2. "Return the initial increases of all open auctions."

```
for $b in document("auction.xml")/site/open_auctions/open_auction
return <increase> $b/bidder[1]/increase/text() </increase>
```

Q3. "Return the first and current increases of all open auctions whose current increase is at least twice as high as the initial increase."

```
for $b in document("auction.xml")/site/open_auctions/open_auction
where $b/bidder[0]/increase/text() * 2 <=
      $b/bidder[last()]/increase/text()
return <increase first=$b/bidder[0]/increase/text()
                 last=$b/bidder[last()]/increase/text()>
```

Q4. "List the reserves of those open auctions where a certain person issued a bid before another person."

```
for $b in document("auction.xml")/site/open_auctions/open_auction
where $b/bidder/personref[id="person18829"] before
```

```
$b/bidder/personref[id="person10487"]
return <history> $b/initial/text() </history>
```

- *Casting* - challenges the DBMS in terms of the casting primitives it provides.

Q5. "How many sold items cost more than 40?"

```
count (for $i in
document("auction.xml")/site/closed_auctions/closed_auction
where $i/price/text() >= 40
return $i/price)
```

- *Regular Path Expressions* - investigate how well the query processor can optimize path expressions and prune traversals of irrelevant parts of the tree.

Q6. "How many items are listed on all continents?"

```
for $b in document("auction.xml")/site/regions
return count ($b//item)
```

Q7. "How many pieces of prose are in our database?"

```
for $p in document("auction.xml")/site
let   $c1 := count($p//description),
      $c2 := count($p//mail),
      $c3 := count($p//email),
      $sum := $c1 + $c2 + $c3
return $sum;
```

- *Chasing References* - horizontal traversals with increasing complexity

Q8. "List the names of persons and the number of items they bought. (joins person, closed auction)"

```
for $p in document("auction.xml")/site/people/person
let $a :=    for $t in
           document("auction.xml")/site/closed_auctions/closed_auction
             where $t/buyer/@person = $p/@id
             return $t
return <item person=$p/name/text()> count ($a) </item>
```

Q9. “List the names of persons and the names of the items they bought in Europe. (joins person, closed auction, item)”

```
for $p in document("auction.xml")/site/people/person
let $a :=    for $t in
          document("auction.xml")/site/closed_auctions/closed_auction
             let $n :=      for $t2 in
                      document("auction.xml")/site/regions/europe/item
                            where $t/itemref/@item = $t2/@id
                            return $t2
             where $p/@id = $t/buyer/@person
             return <item> $n/name/text() </item>
return <person name=$p/name/text()> $a </person>
```

5.6 *Construction of Complex Results*

Q10. “List all persons according to their interest; use French markup in the result.”

```
for $i in distinct
document("auction.xml")/site/people/person/profile/interest/@category
let $p :=    for $t in document("auction.xml")/site/people/person
             where $t/profile/interest/@category = $i
             return <personne>
                <statistiques>
                       <sexe> $t/gender/text() </sexe>,
                       <age> $t/age/text() </age>,
                       <education> $t/education/text()</education>,
                       <revenu> $t/income/text() </revenu>
                </statistiques>,
                <coordonnees>
                       <nom> $t/name/text() </nom>,
                       <rue> $t/street/text() </rue>,
                       <ville> $t/city/text() </ville>,
                       <pays> $t/country/text() </pays>,
                       <reseau>
                          <courrier> $t/email/text() </courrier>,
                          <pageperso> $t/homepage/text()</pageperso>
                       </reseau>,
                    </coordonnees>
                 <cartepaiement> $t/creditcard/text()</cartepaiement>
                    </personne>
return <categorie>
             <id> $i </id>,
             $p
      </categorie>
```

- *Joins on Values* – tests the ability of database to handle large (intermediate) results.

Q11. "For each person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income."

```
for $p in document("auction.xml")/site/people/person
let $l :=    for $i in
      document("auction.xml")/site/open_auctions/open_auction/initial
            where $p/profile/@income > (5000 * $i/text())
            return $i
return <items name=$p/profile/@income> count ($l) </items>
```

Q12. "For each richer-than-average person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income."

```
for $p in document("auction.xml")/site/people/person
let $l :=    for $i in
      document("auction.xml")/site/open_auctions/open_auction/initial
            where $p/profile/@income > (5000 * $i/text())
            return $i
where $p/profile/@income > 50000
return <items person=$p/profile/@income> count ($l) </person>
```

- *Reconstruction* - tests for the ability of the database to reconstruct portions of the original XML document.

Q13. "List the names of items registered in Australia along with their descriptions."

```
for $i in document("auction.xml")/site/regions/australia/item
return <item name=$i/name/text()> $i/description </item>
```

- *Full Text* - full-text search in the form of keyword search

Q14. "Return the names of all items whose description contains the word 'gold'."

```
for $i in document("auction.xml")/site//item
where contains ($i/description,"gold")
return $i/name/text()
```

- *Path Traversals* - tries to quantify the costs of long path traversals that don't include wildcards.

Q15. "Print the keywords in emphasis in annotations of closed auctions."

```
for $a in
document("auction.xml")/site/closed_auctions/closed_auction/annotation/
\description/parlist/listitem/parlist/listitem/text/emph/keyword/text()
return <text> $a <text>
```

Q16. "Confer Q15. Return the IDs of the sellers of those auctions that have one or more keywords in emphasis."

```
for $a in document("auction.xml")/site/closed_auctions/closed_auction
where not empty ($a/annotation/description/parlist/listitem/parlist/\
                 listitem/text/emph/keyword/text())
return <person id=$a/seller/@person />
```

- *Missing Elements* - tests how well the query processors knows to deal with the semi-structured aspect of XML data, especially elements that are declared optional in the DTD.

Q17. "Which persons don't have a homepage?"

```
for $p in document("auction.xml")/site/people/person
where empty($p/homepage/text())
return <person name=$p/name/text()/>
```

- *Function Application* - user defined functions

Q18. "Convert the currency of the reserve of all open auctions to another currency."

```
function convert ($v)
     {
            return 2.20371 * $v -- convert dfl to euros
     }

for $i in document("auction.xml")/site/open_auctions/open_auction/
return convert($i/reserve/text())
```

- *Sorting* - sort on generic strings

Q19. "Give an alphabetically ordered list of all items along with their location."

```
for $b in document("auction.xml")/site/regions//item
let $k := $b/name/text()
return <item name=$k> $b/location/text() </item>
sortby (.)
```

- *Aggregation*

Q20. "Group customers by their income and output the cardinality of each group."

```
<result>
 <preferred>
  count (document("auction.xml")/site/people/person/profile[@income
                                                          >= 100000])
 </preferred>,
 <standard>
  count (document("auction.xml")/site/people/person/profile[@income <
                                        100000 and @income >= 30000])
 </standard>,
 <challenge>
  count (document("auction.xml")/site/people/person/profile[@income <
                                                              30000])
 </challenge>,
 <na>
  count (    for $p in document("auction.xml")/site/people/person
             where empty($p/@income)
             return $p)
 </na>
</result>
```

2.3 Translation algorithm [1] applied to XMark queries

Q1. "Return the name of the item with ID "item20748" registered in North America."

Solution in ICT:

SELECT namerica-item-name
FROM HICT
WHERE namerica-item-@id = 'item20748'

Translated ICT query:

```
for     $r in document(“auction.xml”)//namerica/item,
        $i in $r/@ id,
        $n in $r/name
where   $i = “item20748”
return  $n/text()
```

Optimized query:

```
for     $r in
doc("auction.xml")/site/regions/namerica/item[@id="item20748"]
return  $r/name/text()
```

Q2. “Return the initial increases of all open auctions.”

Solution in ICT:

SELECT increase
FROM HICT
WHERE bidder-pos = 1

Translated ICT query:

```
for $b in doc("auction.xml")//bidder[1]
return <increase> { $b/increase/text() } </increase>
```

Optimized query:

```
for $b in doc("auction.xml")/site/open_auctions/open_auction/bidder[1]
return <increase> { $b/increase/text() } </increase>
```

Q3. “Return the first and current increases of all open auctions whose current increase is at least twice as high as the initial increase.”

Solution in ICT:

SELECT h1.increase, h2.increase
FROM HICT h1, HICT h2
WHERE h1.open_auction-@id = h2.open_auction-@id and
first(h1.bidder) and
last(h2.bidder) and
h1.increase*2 <= h2.increase

Translated ICT query:

```
for     $h1 in doc("auction.xml")//open_auction,
        $id1 in $h1/@id,
        $b1 in $h1/bidder[1],
        $inc1 in $b1/increase,
        $h2 in doc("auction.xml")//open_auction,
        $id2 in $h2/@id,
        $b2 in $h2/bidder[last()],
        $inc2 in $b2/increase
where   $id1=$id2 and
        $inc1/text()*2 <= $inc2/text()
return
        <increase
                first= "{$inc1/text()}"
                last="{$inc2/text()}"/>
```

Optimized query:

```
for $h1 in doc("auction.xml")/site/open_auctions/open_auction
where $h1/bidder[1]/increase/text()*2 <=
$h1/bidder[last()]/increase/text()
return
        <increase
                first= "{$h1/bidder[1]/increase/text()}"
                last="{$h1/bidder[last()]/increase/text()}"/>
```

Q4. “List the reserves of those open auctions where a certain person issued a bid before another person.”

Solution in ICT:

SELECT h1.reserve
FROM HICT h1, HICT h2
WHERE h1.open_auction = h2.open_auction and
h1.@person = ‘18829’ and
h2.@person = ‘person10487’ and
h1bidder < h2.bidder

Note:

The restriction in the “where” clause: h1.open_auction = h2.open_auction is equivalent to h1.open_auction-@id = h2.open_auction-@id, which was stated in the previous query. The first one ran without the id, refers to internal nodes. For SQL

querying over the HICT purposes we can use the associated node values for internal nodes with no text value. For details on this subject refer to Chapter 5, Conclusions and Future Work.

Translated ICT query:

```
for     $h1 in doc("auction.xml")//open_auction,
        $r1 in $h1/reserve,
        $b1 in $h1/bidder,
        $p1 in $b1/personref/@person,
        $h2 in doc("auction.xml")//open_auction,
        $b2 in $h2/bidder,
        $p2 in $b2/personref/@person
where  $b1 << $b2 and
        $p1="person18829" and $p2="person10487" and $h1=$h2
return
        <history> { $r1/text() }</history>
```

Optimized query:

```
for     $h1 in doc("auction.xml")/site/open_auctions/open_auction,
        $b1 in $h1/bidder,
        $b2 in $h1/bidder
where   $b1 << $b2 and
        $b1/personref/@person="person18829" and
$b2/personref/@person="person10487"
return
        <history> { $h1/reserve/text() }</history>
```

Q5. “How many sold items cost more than 40?”

Solution in ICT:

SELECT count(price)
FROM HICT
WHERE price >= 40

Translated ICT query:

```
let $p:=for $p1 in
doc("auction.xml")/site/closed_auctions/closed_auction/price
        where $p1/text() >=40
        return $p1
return count($p)
```

The query is already in an optimized form.

Q6. “How many items are listed on all continents?”

Solution in ICT:

```
SELECT    count(africa-item) + count(asia-item) + count(namerica-item) +
          count(europe-item) + count(australia-item) + count(samerica-item)
FROM      HICT
```

Translated ICT query:

```
for     $r in doc("auction1.xml")/site/regions,
        $b1 in $r/africa/item,
        $b2 in $r/namerica/item,
        $b3 in $r/europe/item,
        $b4 in $r/samerica/item,
        $b5 in $r/asia/item,
        $b6 in $r/australia/item
return  count($b1) + count($b2) + count($b3) + count($b4) + count($b5)
        + count($b6)
```

No optimization can be performed.

Q7. “How many pieces of prose are in our database?”

Note:

The authors refer to all the elements of type annotation, description and email.

Some of them appear under different path in the DTD (see Appendix A).

Solution in ICT:

```
SELECT    count(closed_auction-annotation) +
          count(closed_auction-annotation-description) +
          count(open_auction-annotation) +
          count(open_auction-annotation-description) +
          count(category-description) +
          count(item-description) +
          count(email)
FROM      HICT
```

Translated ICT query:

```
for $p in doc("auction.xml")/site
return  count($p//closed_auction//annotation) +
```

```
        count($p//closed_auction//description) +
        count($p//open_auction//annotation) +
        count($p//open_auction//description) +
        count($p//category//annotation) +
        count($p//category//description) +
        count ($p//emailaddress) +
        count($p//item//description)
```

Again, no optimization can be carried out.

Q8. "List the names of persons and the number of items they bought. (joins person, closed auction)"

Solution in ICT:

SELECT @person-id, count(closed-auction)
FROM LICT (@buyer-person, person)
GROUP BY @person-id

Translated ICT query:

```
for $i in doc("auction.xml")/site/people/person/@id
let $a:=for $t in doc("auction.xml")//closed_auction,
        $i2 in $t/buyer/@person
        where $i2 = $i
        return  $t
return <item person = "{ $i }"> {count($a)} </item>
```

Optimized query:

```
for $i in doc("auction.xml")/site/people/person/@id
let $a:=for $t in
doc("auction.xml")/site/closed_auctions/closed_auction
        where $t/buyer/@person = $i
        return  $t
return <item person = "{ $i }"> {count($a)} </item>
```

Q9. "List the names of persons and the names of the items they bought in Europe. (joins person, closed auction, item)"

Solution in ICT:
SELECT l1.@person-id, collect(l2.item-name)

FROM LICT (@buyer-person, person) l1, LICT(@itemref,europe-item) l2
WHERE l1.closed_auction = l2.closed.auction
GROUP BY @person-id

Translated ICT query:

```
for $i in doc("auction.xml")/site/people/person/@id
let $items:=for $t1 in doc("auction.xml")//closed_auction,
                $t2 in doc("auction.xml")//closed_auction,
               $i1 in $t1/buyer/@person,
               $b2 in $t1/itemref/@item
            where $t1 = $t2 and $i1 = $i and
               ( some $item in doc("auction.xml")//europe/item
               satisfies ($item/@id = $b2))
            return <item> {$b2} </item>
return <person person-id="{$i}"> {$items} </person>
```

Optimized query:

```
for $i in doc("auction.xml")/site/people/person/@id
let $items:=for $t1 in
               doc("auction.xml")/site/closed_auctions/closed_auction,
               $i1 in $t1/buyer/@person,
               $b2 in $t1/itemref/@item
            where $i1 = $i and
               ( some $item in
                           doc("auction.xml")/site/regions/europe/item
               satisfies ($item/@id = $b2))
            return <item> {$b2} </item>
return <person person-id="{$i}"> {$items} </person>
```

Q10. “List all persons according to their interest; use French markup in the result.”

Solution in ICT:

SELECT @category, collect(gender, age, education, income, person-name, street, city, country, email, homepage, creditcard)
FROM HICT
GROUP BY @category

Translated ICT and optimized query:

```
for $i in distinct-values(
document("auction.xml")/site/people/person/profile/interest/@category
)
let $p :=    for $t in document("auction.xml")/site/people/person
```

```
              where $t/profile/interest/@category = $i
              return <personne>
                 <statistiques>
                        <sexe> $t/gender/text() </sexe>,
                        <age> $t/age/text() </age>,
                        <education> $t/education/text()</education>,
                        <revenu> $t/income/text() </revenu>
                 </statistiques>,
                 <coordonnees>
                        <nom> $t/name/text() </nom>,
                        <rue> $t/street/text() </rue>,
                        <ville> $t/city/text() </ville>,
                        <pays> $t/country/text() </pays>,
                        <reseau>
                           <courrier> $t/email/text() </courrier>,
                           <pageperso> $t/homepage/text()</pageperso>
                        </reseau>,
                     </coordonnees>
                  <cartepaiement> $t/creditcard/text()</cartepaiement>
                     </personne>
return <categorie>
              <id> $i </id>,
              $p
       </categorie>
```

Q11. "For each person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income."

Solution in ICT:

SELECT person, count(initial)
FROM HICT
WHERE @income > initial * 5000
GROUP BY person

Translated ICT query:

```
for $g in distinct-values(doc("auction.xml")/site/people/person/name)
let $i:=for    $s in doc("auction0.xml")/site,
               $ p in $s/people/person,
               $n in $p/name,
               $income in $profile/@income,
               $init in $s/open_auctions/open_auction/initial
       where $g = $n and $income > $init/text() * 5000
       return $init
return <items name="{$g}" > {count ($i)} </items>
```

Optimized query:

```
for $g in distinct-values(doc("auction.xml")/site/people/person/name)
let $i:=for    $s in doc("auction.xml")/site,
               $ p in $s/people/person,
               $n in $p/name,
               $income in $p/profile/@income,
               $init in $s/open_auctions/open_auction/initial
       where $g = $n and $income > $init/text() * 5000
       return $init
return <items name="{$g}" > {count ($i)} </items>
```

Q12. "For each richer-than-average person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income."

Solution in ICT:

SELECT person, count(initial)
FROM HICT
WHERE @income > 50000 and initial * 5000 < @income
GROUP BY person

Translated ICT query:

```
for $p in doc("auction.xml")/site/people/person
let $i := for $init in
            doc("auction.xml")/site/open_auctions/open_auction/initial
          where $init * 5000 < $p/profile/@income
          return $init
where  $p/profile/@income > 50000
return <items person="{$p/profile/@income}"> {count($i)}</items>
```

Optimized query:

```
for $p in doc("auction.xml")/site/people/person,
     $income in $p/profile/@income
let $i := for $init in
            doc("auction.xml")/site/open_auctions/open_auction/initial
          where $init * 5000 < $income
          return $init
where  $income > 50000
return <items person="{$income}"> {count($i)}</items>
```

Q13. "List the names of items registered in Australia along with their descriptions."

Solution in ICT:

SELECT australia-item-name , australia-item-description
FROM HICT

Translated ICT query:

```
for $i in doc("auction.xml")/site/regions/australia/item,
       $n in $i/name,
       $d in $i/description
return <item name="{$n/text()}"> {$d} </item>
```

Optimized query:

```
for $i in doc("auction.xml")/site/regions/australia/item,
return <item name="{$i/name/text()}"> {$i/description} </item>
```

Q14. "Return the names of all items whose description contains the word 'gold'."

Solution in ICT:

SELECT item-name
FROM HICT
WHERE item-description like *gold*

Translated ICT query:

```
for    $i in doc("auction.xml")//item,
       $n in $i/name
where  contains($i/description, "gold")
return $n
```

Optimized query:

```
for    $i in doc("auction.xml")/site//item,
where  contains($i/description, "gold")
return $i/name/text()
```

Q15. "Print the keywords in emphasis in annotations of closed auctions."

Solution in ICT:

SELECT emph-keyword
FROM HICT

Translated ICT query is similar to the XMark's Q15:

```
for $a in
document("auction.xml")/site/closed_auctions/closed_auction/annotation/
\description/parlist/listitem/parlist/listitem/text/emph/keyword/text()
return <text> $a <text>
```

There is a problem with the high level of recursion in the XMark DTD for the auction XML file. The authors automatically consider only two levels of recursion. The HICT method can cover any level of recursion through the renaming of the subelements, but the complexity cost we have to pay for that is very high.

Q16. "Confer Q15. Return the IDs of the sellers of those auctions that have one or more keywords in emphasis."

Solution in ICT:

SELECT @seller-person
FROM HICT
WHERE emph-keyword is not NULL

Translated ICT query:

```
for     $a in doc("auction.xml")/site/closed_auctions/closed_auction
where
not(empty($a/annotation/description/parlist/listitem/parlist/listitem/
text/emph/keyword/text()))
return  <person id="{$a/seller/@person}"/>
```

Q17. "Which persons don't have a homepage?"

Solution in ICT:

SELECT person-name
FROM HICT
WHERE homepage is NULL

Translated ICT query:

```
for     $p in doc("auction.xml")/site/people/person
where   empty($p/homepage/text())
return  <person name="{$p/name/text()}"/>
```

Q18. "Convert the currency of the reserve of all open auctions to another currency."

Solution in ICT:

SELECT convert(reserve)
FROM HICT

Where convert() is the conversion function.

Translated ICT query:

```
declare namespace my="my.uri";

declare function my:convert($v)
{
  2.20371 * $v
};

for     $r in
doc("auction.xml")/site/open_auctions/open_auction//reserve/text()
return my:convert($r)
```

Q19. "Give an alphabetically ordered list of all items along with their location."

Solution in ICT:

SELECT item, item-name
FROM HICT
ORDER BY item-name

Translated ICT query:

```
for     $b in doc("auction.xml")/site/regions//item,
        $l in $b/location,
        $n in $b/name/text()
order by $n
return <item name="{$n}"> { $l/text()} </item>
```

Optimized query:

```
for     $b in doc("auction.xml")/site/regions//item
order by $b/name/text()
return <item name="{$b/name/text()}"> { $b/location/text()} </item>
```

Q20. “Group customers by their income and output the cardinality of each group.”

Solution in ICT:

SELECT “preferred”, count(person)
FROM HICT
WHERE @income >= 100000
UNION
SELECT “standard”, count(person)
FROM HICT
WHERE @income < 100000 and @income >= 30000
UNION
SELECT “challenge”, count(person)
FROM HICT
WHERE @income < 30000
UNION
SELECT “na”, count(person)
FROM HICT
WHERE @income is NULL

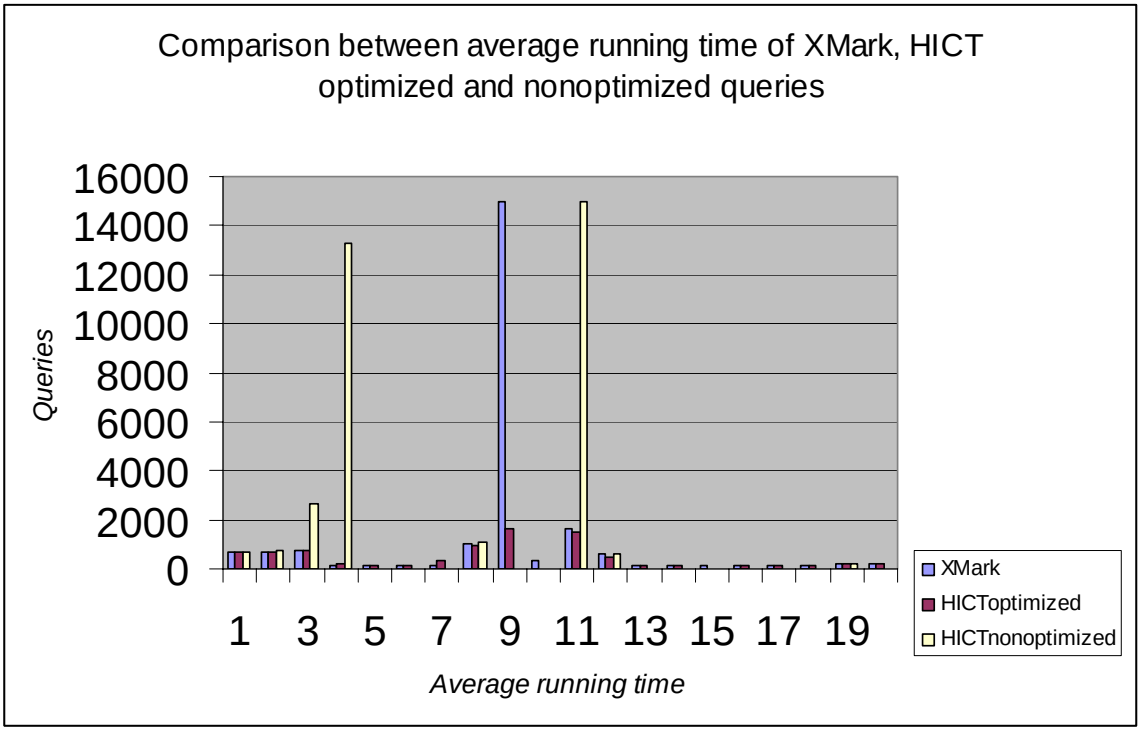
Translated ICT query:

```
<result>
{
let $n:=for $p in doc("auction.xml")/site/people/person
      where $p/profile/@income >=100000
      return $p
return <preffered> {count($n)} </preffered>}
{
let $n:=for $p in doc("auction.xml")/site/people/person
      where $p/profile/@income >=30000 and $p/profile/@income < 100000
      return $p
return  <standard> {count ($n)} </standard>}
{
let $n:=for $p in doc("auction.xml")/site/people/person
      where $p/profile/@income < 30000
      return $p
return <challenge> {count ($n)} </challenge>}
{
let $n:=for $p in doc("auction.xml")/site/people/person
```

```
      where empty($p/profile/@income)
      return $p
return <na> {count($n)} </na>
}
</result>
```

2.4 Conclusions

The XMark queries were run on a 5.6 MB XML document generated with the XML data generator – XMLgen – developed under the XML Benchmark Project [4]. The following results reproduce the average running times expressed in milliseconds*10^{-1}.



As we can see, the optimization process has considerably reduced the running time for some of the queries. Tracking down the issues that caused the increase in

execution time, we found that elimination of *distinct-values* (Query11) and variable reduction/in-lining (Query 3, 4, 6 and 11) can significantly help in the optimization process.

The issues discussed above brought up by our experiments with the translated XMark queries narrowed down to two main topics in the optimizing algorithm: *distinct-values()* function with unique values and unnecessary variable declarations.

CHAPTER III

OPTIMIZATION

3.1 Uniqueness

3.1.1 Introduction

Specifying uniqueness in a database is done using keys. The keys represent an essential part of database design, provide the means by which one tuple in a relational database may refer to another tuple, and enable us to guarantee that an update will affect precisely one tuple.

If we assign a double functionality to an XML document, including that of a relational database, then we need to understand the mean of a key in terms of XML. Through the use of ID attributes in a DTD, we can uniquely identify an element within an XML document. However, [3] argues that ID attributes are intended to be used as internal "pointers" rather than keys. The DTD's ID attributes are not scoped and in contrast to keys, they are unique within the entire document rather than among a designated set of elements. For example, we cannot allow a student (element) and a person (element) to use the same SSN as an ID. Moreover using ID attributes as keys means that we are limited to unary keys and, of course, to using attributes rather than elements. XML Schema has a more elaborate proposal, which is our starting point.

3.1.2 Some Important Differences Between DTDs and Schemas

One of the important features XML Schema is equipped with is that it is well-formed XML, while the DTDs have their own syntax. This constitutes one of the main differences between the two of them. Strictly speaking of this critical difference, the advantage of Schema over the DTD is that, on a developmental level, XML Schemas can be parsed like regular XML documents. The disadvantage is that XML Schemas are verbose.

The most significant difference between DTDs and XML Schemas is in the definition of datatypes. Datatypes are very useful in schema design and in the validation of the XML documents. If we take, for example, a zip code element using a DTD, we can only specify that a <zip> element is text. That means that someone could enter @4982FdbgWa as a zip code, and it would still be considered valid. Using XML Schemas, we can actually create a datatype as a standard five-digit zip code. The ability to get that specific with the datatypes of the content value for both elements and attributes is a very powerful aspect of XML Schemas.

Another aspect in the differences between DTDs and XML Schemas is expression of cardinality. With DTDs, it's only possible to express the occurrence of an element within a content model by using one of three symbols:

- **"*" :** an element can occur any number of times
- **"+" :** an element can occur one or more times
- **"?":** an element can occur zero or one time

This doesn't allow for a great deal of flexibility. If we have a DTD for a bus and want to specify that the bus has to have at least 10 passengers to be cost-effective, but can have up to 25, the DTD looks like this:

```
<!ELEMENT bus (passenger, passenger, passenger, passenger, passenger,
               passenger, passenger, passenger, passenger, passenger,
               passenger?, passenger?, passenger?, passenger?,
               passenger?, passenger?, passenger?, passenger?,
               passenger?, passenger?, passenger?, passenger?,
               passenger?, passenger?, passenger?)>
```

In a schema, this is more compact:

```
<xs:element name="bus">
      <xs:element name="passenger" minOccurs="10" maxOccurs="25"/>
</xs:element>
```

This is a much cleaner mechanism for defining elements that have specific occurrence constraints.

Another advantage of XML Schemas is that both attributes and elements can be enumerations. With DTDs, if we want to restrict our values to a list of choices, this must be done as an attribute, which adds complexity to the content model.

3.1.3 Specifying uniqueness using DTD

The only way of dealing with uniqueness when the XML file is developed conforming to a DTD grammar is through the ID attributes. The value of an attribute of ID type can contain only characters permitted for NMTOKEN[1] and must start with a

[1] NMTOKEN represents the NMTOKEN attribute type from [XML 1.0 (Second Edition)]..The base type of NMTOKEN is token. Token represents tokenized strings and the value space of token is the set of strings that do not contain the carriage return (#xD), line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces.

letter. No element type may have more than one ID attribute specified. The value of an ID attribute must be unique between all values of all ID attributes. Some of these requirements are inconvenient, but for our optimization purposes the existence of a required attribute of type ID helps with the elimination of the *distinct-values()* function from the XQuery and significantly reduces the query running time.

3.1.4 Examples

Although it is very intuitive and straightforward that the removal of a *distinct-value()* associated with an attribute of type ID has no effect on the result of the query, I will use an example to illustrate this hypothesis. The queries will be expressed on the following XML file[1]:

```
<?xml version = "1.0"?>
<!DOCTYPE CONTACTS [
      <!ELEMENT CONTACTS ANY>
      <!ELEMENT CONTACT (NAME, EMAIL)>
      <!ELEMENT NAME (#PCDATA)>
      <!ELEMENT EMAIL (#PCDATA)>
      <!ATTLIST CONTACT CONTACT_NUM ID #REQUIRED>
      ]>
<CONTACTS>
    <CONTACT CONTACT_NUM = "c1">
    <NAME>Lok Siu</NAME>
    <EMAIL>siu@lok.com</EMAIL>
    </CONTACT>

    <CONTACT CONTACT_NUM = "c2">
    <NAME>Joseph Misuraca</NAME>
    <EMAIL>joe@misuraca.com</EMAIL>
    </CONTACT>

    <CONTACT CONTACT_NUM = "c3">
    <NAME>John Darwin</NAME>
    <EMAIL>john@hotmail.com</EMAIL>
```

[1] All the XML files were checked for validity and well-formedness using Stylus Studio

```
    </CONTACT>

    <CONTACT CONTACT_NUM = "c4">
    <NAME>Alina Florescu</NAME>
    <EMAIL>alina_flo@hotmail.com</EMAIL>
    </CONTACT>
</CONTACTS>
```

By running the query:

```
for $c in distinct-
values(doc("testID.xml")/CONTACTS/CONTACT/@CONTACT_NUM)
return <contact_number> {$c} </contact_number>
```

We get the following answer:

```
<contact_number>c1</contact_number>
<contact_number>c2</contact_number>
<contact_number>c3</contact_number>
<contact_number>c4</contact_number>
```

The numbers presented above in the query result are the ID attributes for element CONTACT. As we know, they are unique throughout the document, so we cannot have a value that appears twice. Since the role of the *distinct-value()* is to remove the duplicates, it is clear that this situation is not one that requires any elimination process.

3.1.5 Specifying uniqueness using XML Schema

The paper [1] talks and uses the DTD associated with an XML document. In our effort to optimize the translated XQuery query, we turned to the XML Schema. The reason is that while the DTD provides a mechanism for ensuring uniqueness using the ID attribute and its associated IDREF and IDREFS attributes, XML Schema is equipped with these features and even more flexible and powerful mechanisms for representation of identity-constraint schema components through either a `<unique>`, a `<key>` or a `<keyref>` element.

XML Schema's mechanisms can be applied to any element or attribute content, regardless of its type. In contrast, the ID that DTD uses is an attribute type and it cannot be applied to attributes, elements or their content. Furthermore, XML Schema enables you to specify the scope within which uniqueness applies, whereas the scope of an ID is fixed to be the whole document. Another feature of XML Schema we plan to use is that it allows you to create *keys* or a *keyref* from combinations of element and attribute content, whereas ID has no such facility.

XML Schema enables us to indicate that any attribute or element value must be unique within a certain scope. To specify that an attribute or element value is unique, we use the unique element first to select a set of elements through an xpath selection. To identify the specific attribute or element we use the field option relative to each selected attribute/element that has to be unique within the scope of the set of selected elements.

The syntax, described in the XML Schema Primer of w3.org, is as follows:

```
<unique
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (selector, field+))
</unique>
```

An example of use taken from the same source:

```
<unique name="dummy1">
      <selector xpath="r:regions/r:zip"/>
      <field xpath="@code"/>
</unique>
```

In the above definition of unique types, the NCName represents XML "non-colonized" names: "the set of all strings which match the NCName production of

Namespaces in XML. The lexical space of NCName is the set of all strings which match the NCName production."

The base type of NCName is Name, which finally redirects us to normalizedString - the set of strings that do not contain the carriage return, line feed or tab characters.

One more case we need to be careful with is when our unique definition in the schema contains multiple field elements, meaning that the unique value is given by a group of elements and each one can appear multiple times as long as their combination appears once. As an example from the same source as above, we have:

```
<element name="items" type="Items">
  <unique name="partNumAndName">
    <selector xpath="item"/>
    <field    xpath="@partNum"/>
    <field    xpath="productName"/>
  </unique>
</element>
```

We can enforce a constraint using the *key* and *keyref* elements.

```
<key
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (selector, field+))
</key>
```

```
<keyref
  id = ID
  name = NCName
  refer = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (selector, field+))
</keyref>
```

More examples from w3.org:

```
<key name="pNumKey">
      <selector xpath="r:parts/r:part"/>
      <field xpath="@number"/>
```

```
</key>
```

```
<keyref name="dummy2" refer="r:pNumKey">
      <selector xpath="r:regions/r:zip/r:part"/>
      <field xpath="@number"/>
</keyref>
```

The same idea of multiple field elements is applicable to the *key* definitions and as an example the following statement:

```
<key name="pNumKey1">
      <selector xpath="r:regions/r:zip/r:part"/>
      <field xpath="@number"/>
       <field xpath="@quantity"/>
</key>
```

was validated by the Stylus Studio XML Schema Validator, as part of a valid XML Schema.

3.1.6 Comments

As long as the variable declared in the “for” clause is an attribute or an atomic value (not an interior node) we can use the *distinct-values()* feature of Xquery, which is similar to the distinct function of SQL. This feature allows us to eliminate the duplicates in the output. Although this is convenient, the query running-time increases in the presence of the *distinct-values* method, with the increase of the XML file size and also the complexity of the query. There are still situations when we can avoid the use of this feature and optimize the query in terms of running time.

To further develop this idea lets look at the example used in the paper:

```
for   $D in doc(…)//D,
      $F in $D/F,
      $E in $D/E,
      $G in $E/G,
      $H in $E/H,
```

The first step in the optimization process should be the expansion of the //D path – wildcard path expansion. Then we need to check if D is in any field xpath, uniquely defined, of a unique or key declaration in the XML Shema. If so, we need to further compare the expanded path that follows the document declaration to the selector xpath of that unique or key statement.

Theorem 1 (*XML Schema***)**

Let

```
for   $D in distinct-values(doc(…)/A/B/C/D),
```

be the part of the "for" clause of the translated XQuery that resulted from our algorithm [1]. If D corresponds in the XML Schema of doc(…) to a *unique* or *key* field xpath value, with a single field declaration, and the path A/B/C is the same as the selector xpath in the same declaration of the schema, or if D is an attribute of type ID, then the part of XQuery presented above is equivalent to:

```
for   $D in doc(…)/A/B/C/D,
```

Proof

Case 1 – D is declared as a unique "field" value of an unique or key element, and the path A/B/C is the same as the selector xpath in the same declaration of the schema

D is an element of type *unique* or *key* ⇔

```
<unique name="Dummy1">
   <selector xpath="A/B/C"/>
   <field xpath="D"/>
</unique>
```

or

```
<key name="Dummy2">
   <selector xpath="A/B/C"/>
   <field xpath="D"/>
</key>
```

Suppose that queries are not equivalent and the outcomes are different. That means that the second query will return some duplicates since the only difference between them is the "distinct-values" removal. This will obviously contradict the definition of D in the schema.

Case 2 – D is an attribute of type ID

The definition of the ID type, as a W3C Recommendation says that "an **ID** must not appear more than once in an XML document as a value of this type; i.e., **ID** values must uniquely identify the elements which bear them". If D complies with this definition then there are not two equal values of D throughout the document, and the use of the distinct-value function associated with D is not justified. So the result of Theorem 1 is true.

Corollary (*DTD*)

Let

```
for   $D in distinct-values(doc(…)/A/B/C/D),
```

be, as described above, the part of the "for" clause of the translated XQuery resulted from our algorithm. If D is declared as an attribute of type ID in the DTD associated with our XML document, then the distinct-values method can be removed and our XQuery is equivalent to:

```
for   $D in doc(…)/A/B/C/D,
```

Proof

The XML schema is a superset of DTD. The ID type was embedded in the XML Schema with exactly the same meaning and properties as it carries in the DTD. The proof resumes to Case 2 of Theorem 1.

3.1.7 Examples

Using the following schema, taken from the w3.org, I will express different queries to exemplify the above statements:

```
<schema targetNamespace="http://www.example.com/Report"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:r="http://www.example.com/Report"
        xmlns:xipo="http://www.example.com/IPO"
        elementFormDefault="qualified">

 <!-- for SKU -->
 <import namespace="http://www.example.com/IPO"/>

 <annotation>
  <documentation xml:lang="en">
   Report schema for Example.com
   Copyright 2000 Example.com. All rights reserved.
  </documentation>
 </annotation>

 <element name="purchaseReport">
  <complexType>
   <sequence>
    <element name="regions" type="r:RegionsType">
     <keyref name="dummy2" refer="r:pNumKey">
      <selector xpath="r:zip/r:part"/>
      <field xpath="@number"/>
     </keyref>
    </element>

    <element name="parts" type="r:PartsType"/>
   </sequence>
   <attribute name="period"       type="duration"/>
   <attribute name="periodEnding" type="date"/>
  </complexType>

  <unique name="dummy1">
   <selector xpath="r:regions/r:zip"/>
```

```
   <field xpath="@code"/>
  </unique>

  <key name="pNumKey">
   <selector xpath="r:parts/r:part"/>
   <field xpath="@number"/>
  </key>
 </element>

 <complexType name="RegionsType">
  <sequence>
   <element name="zip" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="part" maxOccurs="unbounded">
       <complexType>
        <complexContent>
         <restriction base="anyType">
          <attribute name="number"   type="xipo:SKU"/>
          <attribute name="quantity" type="positiveInteger"/>
         </restriction>
        </complexContent>
       </complexType>
      </element>
     </sequence>
     <attribute name="code" type="positiveInteger"/>
    </complexType>
   </element>
  </sequence>
 </complexType>

 <complexType name="PartsType">
  <sequence>
   <element name="part" maxOccurs="unbounded">
    <complexType>
     <simpleContent>
      <extension base="string">
       <attribute name="number" type="xipo:SKU"/>
      </extension>
     </simpleContent>
    </complexType>
   </element>
  </sequence>
 </complexType>

</schema>

The associated XML document, that I will query and it conforms to
this schema is -  report.xml:

<purchaseReport
  xmlns="http://www.example.com/Report"
  period="P3M" periodEnding="1999-12-31">
```

```
 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
   <part number="926-AA" quantity="1"/>
   <part number="833-AA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="455-BX" quantity="4"/>
  </zip>
 </regions>

 <parts>
  <part number="872-AA">Lawnmower</part>
  <part number="926-AA">Baby Monitor</part>
  <part number="833-AA">Lapis Necklace</part>
  <part number="455-BX">Sturdy Shelves</part>
 </parts>

</purchaseReport>
```

- **<key> example**

The part number attribute is defined as a key value under the path *parts/part/@number*.

That does not restrict it to appear in different other places multiple times.

Running the query:

```
for $n in distinct-
values(doc("report.xml")/purchaseReport/parts/part/@number)
return <part number="{$n}"/>
```

The outcome of this query is:

```
<part number="872-AA"/>
<part number="926-AA"/>
<part number="833-AA"/>
<part number="455-BX"/>
```

Without the *distinct-values()*:

```
for $n in doc("report.xml")/purchaseReport/parts/part/@number
return <part number="{$n}"/>
```

we get the same answer:

```
<part number="872-AA"/>
<part number="926-AA"/>
```

```
<part number="833-AA"/>
<part number="455-BX"/>
```

To emphasize once more the importance of the *distinct-values()* function, the role of the selector field in the schema definition of the *key* and *unique* elements, which are of interest for our researches, as well as the differences between the DTDs and Schemas, I ran the above query without wildcard expansion, with and without the *distinct-values()* restriction.

```
Query:
for $n in distinct-values(doc("report.xml")//@number)
return <part number="{$n}"/>
```

```
Result:
<part number="872-AA"/>
<part number="926-AA"/>
<part number="833-AA"/>
<part number="455-BX"/>
```

```
Query:
for $n in doc("report.xml")//@number
return <part number="{$n}"/>
```

```
Result:
<part number="872-AA"/>
<part number="926-AA"/>
<part number="833-AA"/>
<part number="455-BX"/>
<part number="455-BX"/>
<part number="872-AA"/>
<part number="926-AA"/>
<part number="833-AA"/>
<part number="455-BX"/>
```

The results speak for themselves and there is no need for commenting them out.

- **<unique> example**

The zip code attribute is defined as a unique value under the path *regions/zip/@code*. Again, that does not restrict it to appear in other places multiple times.

The query:

```
for $c in distinct-
values(doc("report.xml")/purchaseReport/regions/zip/@code)
return <zipcode>{$c}</zipcode>
```

returns:

```
<zipcode>95819</zipcode>
<zipcode>63143</zipcode>
```

Running it without the distinct values returns the same result:

```
<zipcode code="95819"/>
<zipcode code="63143"/>
```

- **non-key or non-unique values example**

If the attribute or element we are trying to analyze is not defined in the context as a unique or key value, then we can expect to get duplicates if they exist in the document, which is perfectly legal and conforms with the schema.

The same attribute for part number, when it appears under the *regions/zip/part* path is no more restricted to be a key value. The following query, with and without the *distinct-values()*, returns different results:

```
for $n in distinct-
values(doc("report.xml")/purchaseReport/regions/zip/part/@number)
return <parts part="{$n}"/>
```

Answer:

```
<parts part="872-AA"/>
<parts part="926-AA"/>
<parts part="833-AA"/>
<parts part="455-BX"/>
```

```
for $n in doc("report.xml")/purchaseReport/regions/zip/part/@number
return <parts part="{$n}"/>
```

Answer:

```
<parts part="872-AA"/>
<parts part="926-AA"/>
<parts part="833-AA"/>
```

```
<parts part="455-BX"/>
<parts part="455-BX"/>
```

Obviously, we get duplicates.

- **ID example (DTD)**

Examples and comments were presented in the beginning of this section ("Specifying uniqueness using DTD")

3.2 Minimization of the number of variables

In practical situations it can happen that a user query involves self-joins of our HICT or multiple HICT/LICT tables. The paper [1] briefly describes such situations and how the algorithm works in that case.

Using the DTD:

```
<!ELEMENT  dept (division*)>
<!ELEMENT  division (faculty*)>
<!ELEMENT  faculty (name, ra*, ta*)>
<!ELEMENT  ra (name)>
<!ELEMENT  ta (name)>
```

Example 4.3 in [1] presents the SQL query for listing pairs of faculty in the same division:

```
select h1.fname, h2.fname
from HICT h1 h2
where h1.division=h2.division
```

The translated query is:

```
for     $D1 in doc(…)//division,
        $F1 in $D1/faculty/fname,
        $D2 in doc(…)//division,
        $F2 in $D2/faculty/fname
```

where $D1=$D2
return {F1} {F2}

The key observation is in realizing that the condition in the "where" clause $D1=$D2 implies that the two variables referred to are bound to the same element. So, through the process of elimination of one of the equivalent variables, the query is optimized as follows:

```
for     $D1 in doc(…)//division,
        $F1 in $D1/faculty/fname,
        $F2 in $D1/faculty/fname
return {F1} {F2}
```

Fig. 7 Optimized query

3.2.1 Self-joins for HICTs and the optimization process

As with any join, a self-join requires at least two tables. The difference is that, instead of adding a second table to the query, we add a second instance of the same table, for our purposes, HICT.

We have seen above how a duplicate declaration for a variable can be removed when that variable is an internal node and there exists a condition in the "where" clause which imposes them to be equal. It can be inferred that the ancestors of the specified nodes are also equal. That is obvious and the following examples give us a hint in that direction.

Consider the following DTD:

```
<!ELEMENT  univ (uname,id,dept+)>
<!ELEMENT  dept (dept_name,division*)>
<!ELEMENT  division (dname,faculty*)>
<!ELEMENT  faculty (name, ra*)>
<!ELEMENT ra(name)>
<!ELEMENT name(#PCDATA)>
```

After the DTD transformation (Algorithm 2.1 in [1]) takes place, the schema graph is mapped into its corresponding schema tree:

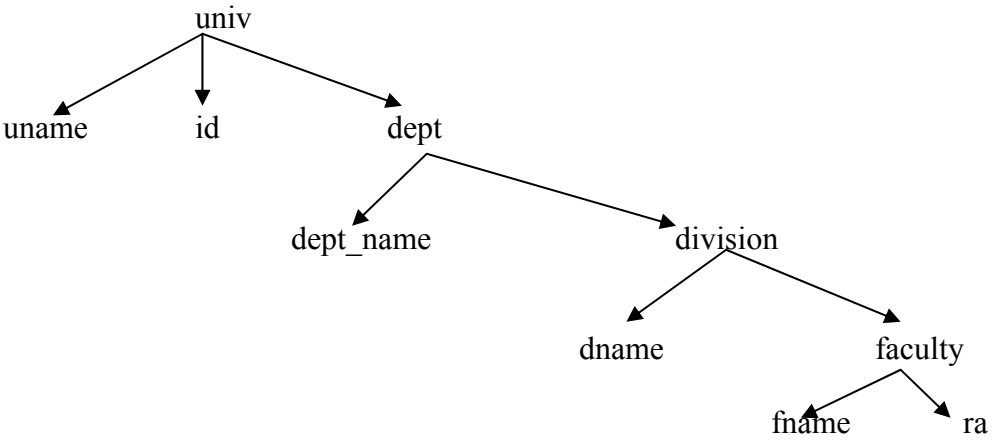


Fig. 8 Schema graph for univ.xml

I ran several queries on the univ.xml document for exemplification of the ideas presented in this section.

univ.xml

```
<?xml version="1.0"?>
<univ>
     <uname>UNCG</uname>
     <id>012563254</id>
     <dept>
            <dept_name>Mathematical Science</dept_name>
            <division>
                   <dname>CSC</dname>
                   <faculty>
                          <fname>Lea, S.</fname>
                          <fname>Fu, L.</fname>
                          <fname>Sadri, F.</fname>
                          <fname>Green, N.</fname>
                   </faculty>
            </division>
            <division>
                   <dname>STA</dname>
```

```
                    <faculty>
                            <fname>Richter, S.</fname>
                            <fname>Gupta, S.</fname>
                    </faculty>
            </division>
      </dept>
      <dept>
            <dept_name>Economics</dept_name>
            <division>
                    <dname>ECO</dname>
                    <faculty>
                            <fname>Snodwen, K.</fname>
                            <fname>Stuart, A.</fname>
                            <fname>Bearse, P.</fname>
                    </faculty>
            </division>
            <division>
                    <dname>STA</dname>
                    <faculty>
                            <fname>Swann, C.</fname>
                            <fname>Rosenbaum, D.</fname>
                    </faculty>
            </division>
      </dept>
</univ>
```

The query presented in the beginning of this chapter that lists pairs of faculty in the same division and the name of the department they belong to, before optimization takes place is:

```
for $d in doc("univ.xml")/univ/dept,
    $f1 in $d/division,
    $f2 in $d/division,
    $dn1 in $f1/faculty/fname,
    $dn2 in $f2/faculty/fname
where $f1 = $f2 and $dn1 << $dn2
return <result> {$d/dept_name}{ $dn1 } { $dn2 }</result>
```

The answer lists eleven pairs of faculty in two different departments and is presented below in Fig. 9. The ordering condition in the “where” clause was introduced to eliminate the duplicate values and is part of the XQuery optimization.

Fig. 9

```
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Lea, S.</fname>
      <fname>Fu, L.</fname>
</result>
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Lea, S.</fname>
      <fname>Sadri, F.</fname>
</result>
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Lea, S.</fname>
      <fname>Green, N.</fname>
</result>
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Fu, L.</fname>
      <fname>Sadri, F.</fname>
</result>
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Fu, L.</fname>
      <fname>Green, N.</fname>
</result>
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Sadri, F.</fname>
      <fname>Green, N.</fname>
</result>
<result>
      <dept_name>Mathematical Sciences</dept_name>
      <fname>Richter, S.</fname>
      <fname>Gupta, S.</fname>
</result>
<result>
      <dept_name>Economics</dept_name>
      <fname>Snowden, K.</fname>
      <fname>Stuart, A.</fname>
</result>
<result>
      <dept_name>Economics</dept_name>
      <fname>Snowden, K.</fname>
      <fname>Bearse, P.</fname>
</result>
<result>
      <dept_name>Economics</dept_name>
      <fname>Stuart, A.</fname>
      <fname>Bearse, P.</fname>
</result>
<result>
```

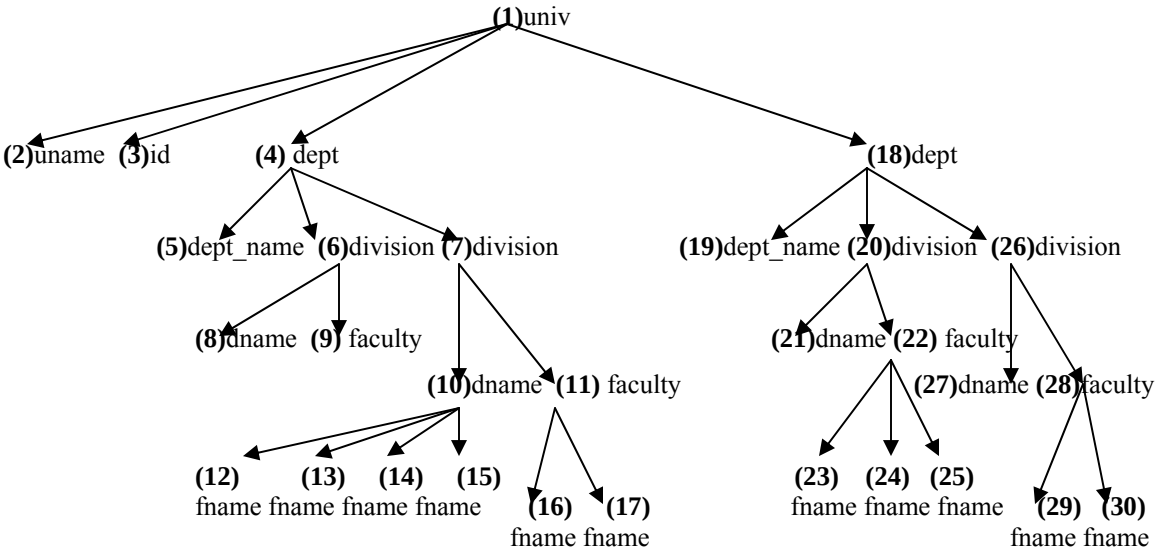
```
      <dept_name>Economics</dept_name>
      <fname>Swann, C.</fname>
      <fname>Rosenbaum, D.</fname>
</result>
```

This is the desired result for the query that was formulated directly on the xml document, by a user familiarized with the underlying schema.

Having the HICT as our starting point and ignoring the DTD, one can ask for a list of pairs of faculties in the same division that work in the same department as follows:

```
select  h1.fname, h2.fname
from    HICT h1 h2
where   h1.division=h2.division and
        h1.dept=h2.dept and
        h1.fname<h2.fname
```

It is important to understand that under the assumption that the underlying schema is unknown, the relation between departments and divisions cannot be reconstituted. Thus, we need to specify two equalities is the “where” clause.



For gaining a better understanding, let's parse our document into a database in the form of HICT. The document tree, with numbers associated with the nodes is presented above and the corresponding binary tables with their final join into the HICT for this document are:

univ	uname
1	2

univ	id
1	3

univ	dept
1	4
1	18

dept	dept_name
4	5
18	19

division	faculty
6	8
13	15
20	22
26	28

dept	division
4	6
4	13
18	20
18	26

faculty	fname
8	9
8	10
8	11
8	12
15	16
15	17
22	23
22	24
22	25
28	29
28	30

division	dname
6	7
13	14
20	21
26	27

The final join (▷◁) will produce :

HICT

univ	uname	id	dept	dept_name	division	dname	faculty	fname
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	10
1	2	3	4	5	6	7	8	11
1	2	3	4	5	6	7	8	12
1	2	3	4	5	13	14	15	16
1	2	3	4	5	13	14	15	17
1	2	3	18	19	20	21	22	23
1	2	3	18	19	20	21	22	24
1	2	3	18	19	20	21	22	25
1	2	3	18	19	26	27	28	29
1	2	3	18	19	26	27	28	30

Or if we want to see it with the applied function *value()*:

univ	uname	id	dept	dept_name	division	dname	faculty	fname
1	UNCG	12563254	4	Mathematical Sciences	6	CSC	8	Lea S.
1	UNCG	12563254	4	Mathematical Sciences	6	CSC	8	Fu L.
1	UNCG	12563254	4	Mathematical Sciences	6	CSC	8	Sadri F.
1	UNCG	12563254	4	Mathematical Sciences	6	CSC	8	Green N.
1	UNCG	12563254	4	Mathematical Sciences	13	STA	15	Richter S.
1	UNCG	12563254	4	Mathematical Sciences	13	STA	15	Gupta S.
1	UNCG	12563254	18	Economics	20	ECO	22	Snoden K.
1	UNCG	12563254	18	Economics	20	ECO	22	Stuart A.
1	UNCG	12563254	18	Economics	20	ECO	22	Bearse P.
1	UNCG	12563254	18	Economics	26	STA	28	Swann C.
1	UNCG	12563254	18	Economics	26	STA	28	Rosenbaum D.

The result of the above mentioned SQL query:

Lea S.	Sadri F.
Fu L.	Green N.
Fu L.	Lea S.
Fu L.	Sadri F.
Green N.	Lea S.
Green N.	Sadri F.
Gupta S.	Richter S.
Snoden K.	Stuart A.
Bearse P.	Snoden K.
Bearse P.	Stuart A.
Rosenbaum D.	Swann C.

As we can see we get again the eleven pairs listed above. Returning to the SQL query, the translation algorithm will produce the following query:

```
for $d1 in doc("univ.xml")/univ/dept,
    $d2 in doc("univ.xml")/univ/dept,
    $f1 in $d1/division,
    $f2 in $d2/division,
    $dn1 in $f1/faculty/fname,
    $dn2 in $f2/faculty/fname
where $f1 = $f2 and $dn1 << $dn2 and $d1 = $d2
return <result>{ $d1/dept_name }{ $dn1 } { $dn2 }</result>
```

The result is the same as above as it lists eleven pairs, seven in the Mathematical Science department, and four in the Economics department. Being so lengthy, it is not reproduced here for space-saving purposes.

The optimization of this query is done by eliminating the second declaration for all the variables, for which we have an equality restriction in the “where” clause and renaming correspondingly in the reminder of the “for” clause the variables’ xpaths. The optimized query is shown below.

```
for $d in doc("univ.xml")/univ/dept,
    $f in $d/division,
    $dn1 in $f/faculty/fname,
    $dn2 in $f/faculty/fname
where $dn1 << $dn2
return <result> {$d/dept_name}{ $dn1 } { $dn2 }</result>
```

The result is again the same as above and will not be printed here for the same reasons.

The examples above show us that the XQuery query ran with and without the condition over the department ($d1/dept = $d2/dept) returns the same result. While the SQL syntax cannot always ignore it, in the xquery statement, it can be skipped because the division node equality implies that all nodes in the tree that are its ancestors (division) are equal also.

Sometimes the equality restrictions appear inlined in the “for” clause of xquery. Although the translation algorithm does not output such queries yet, we can account for them and briefly described how the optimization can be done in their presence. If the equality is based on a unique value (i.e. an ID attribute), then the element nodes corresponding to the variable declarations can be reduced to one.

Suppose that we add an attribute of type ID to the division element in the univ.xml document.

The DTD presented above will be enriched with the following statement:

<ATTLIST division id ID #REQUIRED>

For the query:

```
for $d in doc("univ.xml")/univ/dept,
    $f1 in $d/division,
    $f2 in $d/division[@did=$f1/@did],
    $dn1 in $f1/faculty/fname,
    $dn2 in $f2/faculty/fname
where $dn1 << $dn2
```

```
return <result> {$d/dept_name}{ $dn1 } { $dn2 }</result>
```

This query returns the same answer as the previous ones, and its equivalent, optimized query is:

```
for $d in doc("univ.xml")/univ/dept,
    $f in $d/division,
    $dn1 in $f/faculty/fname,
    $dn2 in $f/faculty/fname
where $dn1 << $dn2
return <result> {$d/dept_name}{ $dn1 } { $dn2 }</result>
```

Definition

We say that a variable declaration is *redundant* if two variables that refer to the same node exist simultaneously in the "for" clause of an XQuery query.

The redundancy can be eliminated if the variables are "required" to be equal. The equality restriction can appear in the "where" clause or it can be inlined in the "for" part statement of the query through the equality of the corresponding attributes of type ID.

Observation

The equality of their attributes of type ID can be stated in the "where" clause also, and the case is similar to what was stated in the above examples.

Theorem 2 (Redundant variable elimination)

Consider a query Q which contains redundant variables in its declarations. The redundancy of a variable that refers to an internal node can be eliminated from Q in the following particular cases:

1. If the two variables, $v1 and $v2, are restricted to being equal in the "where" clause: $v1=$v2;

2. If the two variables, $v1 and $v2, have an attribute (id) of type ID and
    - in the "where" clause: $v1/@id=$v2/@id;
    - in the "for" clause: [@id=$v1/@id] is appended to the declaration of $v2;
3. In the presence of a redundant variable conforming to cases 1 or 2 described above, it can be inferred that the redundancy propages to all its ancestors; the elimination process applies successively to all the redundant ancestors.

The elimination removes one of the variable declarations as well as the equality restriction, and replaces the deleted variable with the one that remains in the declaration of the variables that were previously bound to it.

After the elimination completes the resulted query (Q') is equivalent to the initial one, Q.

Proof

- *Case 1*. The two variables, $v1 and $v2, restricted to be equal in the "where" clause or their ID attributes required to be equal;

Let Q be of the following format:

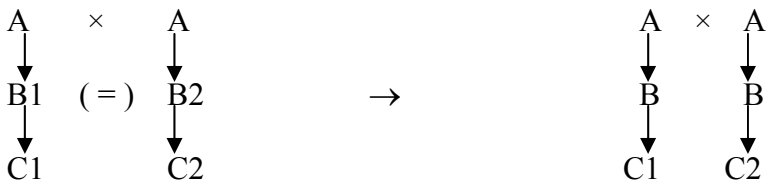
```
for     $B1 in doc(…)//A/B,
        $B2 in doc(…)//A/B,
        $C1 in $B1/C,
        $C2 in $B2/C
where $B1=$B2 (:$B1/@id=$B2/@id:)
return {$C1}{$C2}
```

Then Q' is expected to be of the following format:

```
for     $B in doc(…)//A/B,
        $C1 in $B/C,
        $C2 in $B/C
return {$C1}{$C2}
```

The "for" statement translated in plain words says that for each copy of the document doc(…), find two elements of type B on the path doc(…)//A/B and for each one of them find a type C element, right below it.

While the schema graph is similar to A $\xrightarrow{*}$ B $\xrightarrow{*}$ C, the schema trees translate to:



If we go further with our analysis, the "where" clause imposes that the two elements of type B be equal. So B1 = B2 in our tree (or $B1/@id = $B2/@id) ⇔ type B nodes will refer to exactly the same node element.

Going back to our interpretation we can reformulate the query to: for each copy of the document doc(…) find a node of type B on the path doc(…)//A/B and for each element of type B find two elements of type C (return a pair of elements of type C). Putting this in XQuery format:

```
for     $B1 in doc(…)//A/B,
        $C1 in $B1/C,
        $C2 in $B1/C
return {$C1}{$C2}
```

Which is equivalent to Q'.

- *Case 2*. The two variables, $v1 and $v2, have an attribute (id) of type ID and the equality restriction of form [@did=$v1/@did] is appended to the declaration of $v2 in the "for" clause.

The steps described above will take place here also with the difference that the binding to the same B type node will happen before the declaration of the C type variable. The proof follows then as before.

It is obvious that after establishing the redundancy, and binding the variables to the same node element, as we go upwards in the tree, towards the root, all the elements must be also equal. That means that we can safely remove redundant ancestors declarations.

3.2.2 Analysis of Time Complexity

In this section, we will try to compare the time complexity of the queries defined above as Q and Q'. Suppose that our document has n nodes of type B: $B_1, B_2, \ldots B_n$ and each B_i node has c_i descendent nodes (in our example, children).

Let's define an indicator b_{ij} as follows:

$$b_{ij} = \begin{cases} 1 \text{ if } B_i, B_j \text{ are selected} \\ 0 \text{ otherwise} \end{cases}$$

Looking back at the four declarations in the "for" statement of query Q, we see that for each module {$B_i in doc(…)//B, $C_i in $B_i/C}, we declare a variable B_i for which we define c_i variables of type C. Having *n* variables of type B, we conclude that there will be $n*c_i$ declarations per module.

Lets define a matrix $P = (p_{ij})_{i,j=\overline{1,n}}$ where $p_{ij} = b_{ij}c_ic_j$.

The two modules described above are basically constructing this matrix. The time complexity is then the sum of the elements of P: $O(\sum_{i,j=1}^{i,j=n} b_{ij}c_ic_j)$.

If the maximum number of descendants of a type B node is m, then the running time of the query is at most $O(n^2*m^2)$.

Remark: the actual output contains only the diagonal elements of the P matrix, so it displays only $\sum_{i=1}^{i=n} b_{ii}c_ic_i$ pairs.

The Q' xquery,

```
for     $B in doc(…)//B,
        $C1 in $B/C,
        $C2 in $B/C
return {$C1}{$C2}
```

by its declaration rules defines only the diagonal of matrix P, that means a sum of n elements. Again, if we allow that the maximum number of type B node descendants is m, then we get a running time of at most $O(n*m^2)$. The number of pairs in the output coincides this time to the number of elements visited if there are no other restrictions in the "where" clause of the query (i.e. ordering conditions or different elements pair).

In light of this result, it is obvious that our claim that Q' is an optimized version of Q holds.

3.2.3 Why internal nodes?

The kind of queries that can be optimized this way usually require a pair of elements for the output, and for this purpose a duplicate declaration for the document is needed.

Our conclusion is that removal of the redundancy for *internal* nodes is possible, and we claimed and proved that this improves the running time of the query.

When we deal with leaf nodes, we actually have values that need to be compared. Let's look at some examples. Consider the following schema tree with the associated DTD:

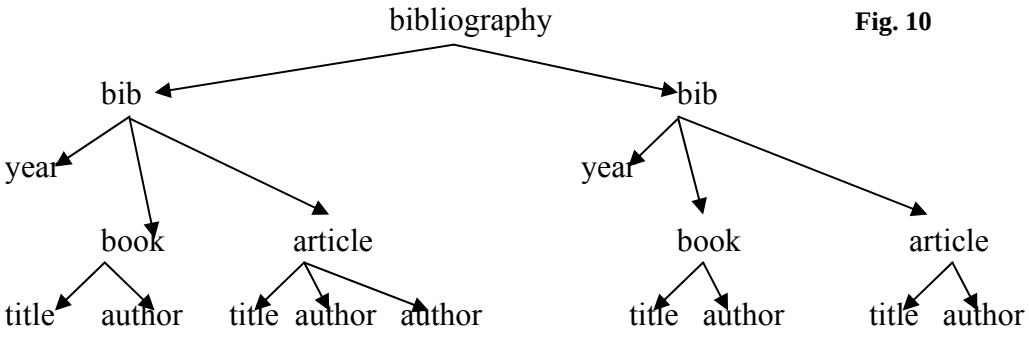


Fig. 10

```
<!ELEMENT bibliography(bib*)>
<!ELEMENT bib(year, book*,article*)>
<!ELEMENT book(title, author+)>
<!ELEMENT article(title, author+)>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT author  (#PCDATA )>
<!ATTLIST bib  year CDATA  #REQUIRED >
```

Let's attach values to the leaf nodes in the above tree and express the query that lists pairs of first author for books with the same titles.

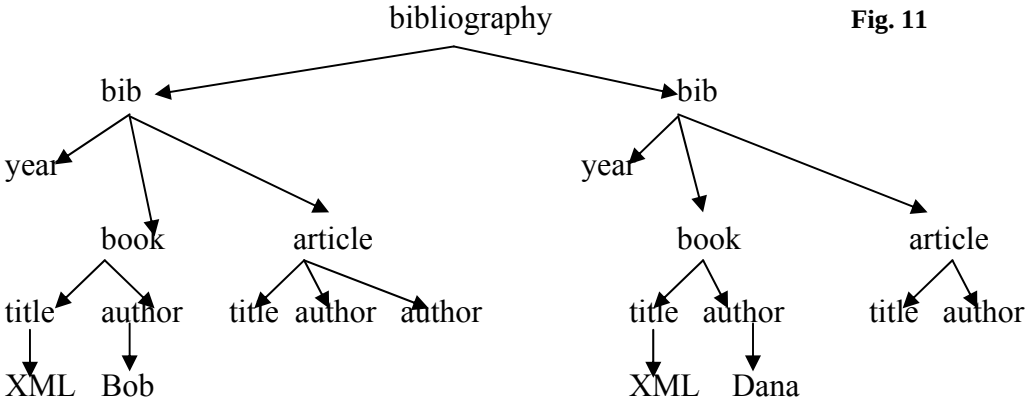


Fig. 11

```
Query Q1:
for  $D1 in doc("…")//book,
     $F1 in $D1/title,
     $D2 in doc("…")//book,
     $F2 in $D2/title
where $F1=$F2 and $D1/author[1]<<$D2/author[1]
return <result>{$D1/author[1]} {$D2/author[1]}</result>
```

Optimized query based on Theorem 2:

```
Query Q2:
for  $D1 in doc("…")//book,
     $F1 in $D1/title
return <result>{$D1/author[1]} {$D1/author[1]}</result>
```

It is obvious that the answer contains the first author of each book in the document listed twice.

Corollary (Leaf nodes redundancy)

The result of Theorem 2 cannot be applied to leaf nodes. That means that queries Q1 and Q2 mentioned above are not equivalent.

Proof

Suppose we try to apply the optimization principle described above to the title elements, which are leaf nodes in the schema tree (as in Q2). That means that the two variables referring to title elements for which we have an equality imposed in the "where" part of the query can be reduced to one. By the observation mentioned above, all the ascendants of the title nodes are also equal. That means that the two variables indicating book elements refer to the same book. But we were looking for different books with the same title. So we get a contradiction.

3.2.4 Unique values of XML Schemas

When the XML document has an XML Schema behind it, Theorem 2 can cover more cases. These are not particular situations, but rather extensions of the uniqueness property from DTD to Schema. Instead of considering only the case when the equality is based on the attributes of type ID, we can allow to have elements of type *unique* or *key* and carefully check for the coincidence of their scope with the xpaths defined for the variables being evaluated. At this point, the wildcard expansion is a must since the same element can be appear under different paths, with different features.

CHAPTER IV

SUMMARY OF FINDINGS – THE OPTIMIZATION ALGORITHM

Consider a query Q produced by the translation algorithm from [1] and Q_i, $0 \leq i \leq k$, $k \in \mathbb{N}^*$, nested queries for Q, such that if k = 0, then $Q_0 = Q$. The optimization algorithm takes Q as an input and produces Q' – an optimized query, with unneeded *distinct-values()* or redundant variables eliminated.

Optimization Algorithm

For each Q_i, $0 \leq i \leq k$, $k \in \mathbb{N}^*$, repeat the following steps:

1. check for the presence of distinct-values() function; if it does not exist go to *2*, otherwise:
    a. look at the XML Schema for the declaration of the element/attribute associated with the distinct-value() function; If the variable corresponds to a *unique* or *key* field xpath value, with a single field declaration, and its xpath is identical to the selector xpath in the same declaration of the schema, return true and go to *c*
    b. if the variable corresponds to an attribute of type ID in the XML Schema/DTD declaration return true

c. if true remove the distinct-value() method;

2. check for redundant variables in the "for" loop of Q_i; if they don't exist then the performance of the query execution in terms of time complexity depends only on the XQuery processor; otherwise:

   a. check the "where" clause for equality constraints on the two variables; if found return true and go to *c*

   b. check the DTD/ Schema for attributes of type ID; if found:

      i. check the "for" clause for equality restrictions of type [@id=$v/@id] appended to one of the redundant variable, where id is the ID attribute and $v is the other redundant variable; return true if found and go to *c*

      ii. check the "where" clause for equality restrictions over the id attributes; return true if found

   c. if true, then eliminate one of the variable declarations and refer only to the remaining one throughout the query being evaluated, Q_i; check for redundant ancestors; if found, repeat the elimination process; otherwise optimization of Q_i is completed.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

While the XQuery is being developed by the XML Query working group at the World Wide Web Consortium (W3C) in an effort to provide users with a powerful means to query and transform XML, the increased popularity of the latter in the database community merged to a focus in the design of query languages and storage methods to select data from vast amounts of XML databases efficiently.

Lakshmanan and Sadri [1] developed a simpler user interface which captures the information content of an XML document. Sophisticated XQuery queries can be formulated as simple SQL queries upon this view. A simple translation algorithm will facilitate the transition to XQuery.

In this thesis, I propose an algorithm that will optimize the translated XQuery queries in terms of running time. The results presented here are applicable to XQuery optimization in general, although they were initially intended to the HICT method.

There are some open issues that are still being reviewed by authors like: handling the internal nodes of the XML database and DTDs with conjunctions. The internal nodes problem arises from the lack of values to be inputted into the HICT: how do SQL treat nodes without values? An apparent simple solution is to assign numbers for all the nodes in the XML document tree and let SQL use numbers for internal nodes and values for leaf nodes. This method works in most cases, but there are certain situations when it fails, and they correspond to particular DTDs that allow an internal node to have subelements and

text value of type PCDATA. My suggestion, for these particular types of nodes, is the use of both number and text value in the HICT.

As I already stated, XQuery is a young standard and many issues with respect to its optimization are unresolved, but improvements in these areas will contribute to our optimization results implicitly. As the usage of XML shifts towards the data-oriented paradigm, more efforts are done to allow the efficient evaluation of XQuery.

BIBLIOGRAPHY

[1] "**On the information content of an XML Database**" Laks V.S. Laksmanan, Fereidon Sadri.

[2] XML Schema Part 0: Primer

http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[3] **"Keys for XML"** Peter Buneman, Susan Davidson, Wenfei Fany, Carmem Haraz, Wang-Chiew Tan.

[4] "**The XML Benchmark Project**" Ralph Busse, Mike Carey, Daniela Florescu, Martin Kersten, Ioana Manolescu, Albrecht Schmidt, Florian Waas; *April 30,2001*.

[5] "**Generating Relations from XML Documents**" Sara Cohen, Yaron Kanza, Yehoshua Sagiv; *Proc 9th International Conference on Database Theory*, *Jan.2003*, Springer-Verlag.

[6] **"XSEarch: A Semantic Search Engine for XML"** Sara Cohen, Jonathan Mamou, Yaron Kanza, Yehoshua Sagiv; *Proceedings of the 29th VLDB Conference 2003*, Berlin, Germany.

[7] "**Schema-Free XQuery**" Yunyao Li, Cong Yu, H.V. Jagadish; *Proceedings of the 30th VLDB Conference 2003*, Toronto, Canada.

[8] **http://www.w3schools.com/schema/default.asp**

[9] **http://monetdb.cwi.nl/xml/downloads.html - The Data Generation Tool – xmlgen**

[10] **http://www.relaxer.org/**

[11] **http://www.lumrix.net/dtd2xs.php**

[12] "**The Guru's Guide to SQL Server Stored Procedures, XML, and HTML**", Ken Henderson, Addison-Wesley, December 2002.

[13] "**Essential XML – Quick Reference**", Aaron Skonnard, Martin Gudgin, Addison-Wesley, January 2003.

[14] "**XML Unleashed**", Michael Morrison, et al.,SAMS Publishing, December 1999.

[15] "**Data on the web**", Serge Abiteboul, Peter Buneman, Dan Suciu, Morgan Kaufmann Publishers, San Francisco, California, 2000.

Appendix A GETTING THE DTD

An interesting thing I learned is that there exist few tools, available for free download, which will generate the associated DTD of a XML document. I used *relaxer* with the command *C:\usr\local\bin>relaxer -dtd auction.xml,* where the auction.xml was a 5.6 MB file. The resulted DTD is displayed below.

```
<!-- Generated by Relaxer 1.1b -->
<!-- Tue Dec 21 14:26:48 EST 2004 -->

<!ELEMENT phone (#PCDATA)>
<!ELEMENT start (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT province (#PCDATA)>
<!ELEMENT personref EMPTY>
<!ATTLIST personref person CDATA #REQUIRED>
<!ELEMENT date (#PCDATA)>
<!ELEMENT buyer EMPTY>
<!ATTLIST buyer person CDATA #REQUIRED>
<!ELEMENT education (#PCDATA)>
<!ELEMENT profile (interest*, education?, gender?, business, age?)>
<!ATTLIST profile income CDATA #REQUIRED>
<!ELEMENT interest EMPTY>
<!ATTLIST interest category CDATA #REQUIRED>
<!ELEMENT listitem (text | parlist)>
<!ELEMENT open_auctions (open_auction+)>
<!ELEMENT regions (africa, asia, australia, europe, namerica,
samerica)>
<!ELEMENT watches (watch+)>
<!ELEMENT keyword (bold | emph | (bold, emph) | (emph*, bold+))>
<!ELEMENT price (#PCDATA)>
<!ELEMENT emailaddress (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT europe (item+)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT increase (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT asia (item+)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT mail (from, to, date, text)>
<!ELEMENT africa (item+)>
<!ELEMENT itemref EMPTY>
<!ATTLIST itemref item CDATA #REQUIRED>
<!ELEMENT text ((emph+, bold, emph, bold+, keyword, bold, emph+) |
(emph+, bold, emph, bold+, keyword, bold, emph+) | (emph+, bold,
emph, bold+, keyword, bold, emph+) | (emph+, bold, emph, bold+, key
```

```
word, bold, emph+) | (emph+, bold, emph, bold+, keyword, bold, emph+)
| (emph+, bold, emph, bold+, keyword, bold, emph+) | (emph+, bold,
emph, bold+, keyword, bold, emph+) | (emph+, bold, emph, bold+,
keyword, bold, emph+) | (emph+, bold, emph, bold+, keyword, bold,
emph+) | (emph+, bold, emph, bold+, keyword, bold, emph+) | (emph+,
bold, emph, bold+, keyword, bold, emph+) | (emph+, bold, emph, bold+,
keyword, bold, emph+) | (emph+, bold, emph, bold+, keyword, bold,
emph+) | (emph+, bold, emph, bold+, keyword, bold, emph+) | (emph+,
bold, emph, bold+, keyword, bold, emph+) | (emph+, bold, emph, bold+,
keyword, bold, emph+) | (emph+, bold, emph, bold+, keyword, bold,
emph+) | (emph+, bold, emph, bold+, keyword, bold, emph+) | (emph+,
bold, emph, bold+, keyword, bold, emph+) | (emph+, bold, emph, bold+,
keyword, bold, emph+) | (emph+, bold, emph, bold+, keyword, bold,
emph+) | (emph+, bold, emph, bold+, keyword, bold, emph+) | (emph+,
bold, emph, bold+, keyword, bold, emph+) | (emph+, bold, emph, bold+,
keyword, bold, emph+) | (emph+, bold, emph, bold+, keyword, bold,
emph+) | (emph, keyword+, emph, bold, keyword, emph))>
<!ELEMENT mailbox (mail+)>
<!ELEMENT samerica (item+)>
<!ELEMENT watch EMPTY>
<!ATTLIST watch open_auction CDATA #REQUIRED>
<!ELEMENT happiness (#PCDATA)>
<!ELEMENT author EMPTY>
<!ATTLIST author person CDATA #REQUIRED>
<!ELEMENT zipcode (#PCDATA)>
<!ELEMENT business (#PCDATA)>
<!ELEMENT shipping (#PCDATA)>
<!ELEMENT item (location, quantity, name, payment, description,
shipping, incategory+, mailbox)>
<!ATTLIST item id CDATA #REQUIRED>
<!ATTLIST item featured CDATA #IMPLIED>
<!ELEMENT creditcard (#PCDATA)>
<!ELEMENT seller EMPTY>
<!ATTLIST seller person CDATA #REQUIRED>
<!ELEMENT australia (item+)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT address (street, city, country, province?, zipcode)>
<!ELEMENT categories (category+)>
<!ELEMENT homepage (#PCDATA)>
<!ELEMENT person (name, emailaddress, phone?, address?, homepage?,
creditcard?, profile?, watches?)>
<!ATTLIST person id CDATA #REQUIRED>
<!ELEMENT open_auction (initial, reserve?, bidder*, current,
privacy?, itemref, seller, annotation, quantity, type, interval)>
<!ATTLIST open_auction id CDATA #REQUIRED>
<!ELEMENT age (#PCDATA)>
<!ELEMENT annotation (author, description, happiness)>
<!ELEMENT namerica (item+)>
<!ELEMENT initial (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT parlist (listitem+)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT closed_auctions (closed_auction+)>
```

```
<!ELEMENT bidder (date, time, personref, increase)>
<!ELEMENT current (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT incategory EMPTY>
<!ATTLIST incategory category CDATA #REQUIRED>
<!ELEMENT edge EMPTY>
<!ATTLIST edge from CDATA #REQUIRED>
<!ATTLIST edge to CDATA #REQUIRED>
<!ELEMENT category (name, description)>
<!ATTLIST category id CDATA #REQUIRED>
<!ELEMENT interval (start, end)>
<!ELEMENT payment (#PCDATA)>
<!ELEMENT closed_auction (seller, buyer, itemref, price, date,
quantity, type, annotation)>
<!ELEMENT privacy (#PCDATA)>
<!ELEMENT catgraph (edge+)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT site (regions, categories, catgraph, people, open_auctions,
closed_auctions)>
<!ELEMENT bold ((emph, keyword) | (keyword+, emph*) | (keyword+,
emph*))>
<!ELEMENT description (text | parlist)>
<!ELEMENT reserve (#PCDATA)>
<!ELEMENT emph ((bold, keyword) | (keyword?, bold) | (bold,
keyword))>
<!ELEMENT people (person+)>
```

Appendix B THE *AUCTION.XML* DOCUMENT

The auction.xml document can be generated in different sizes ranging from 25KB to 100MB. The document generator, called xmlgen is available for free download at http://monetdb.cwi.nl/xml/downloads.html. It is part of the XML Benchmark project and it conforms to the DTD presented in Appendix A. It is a computer generated document and many part of it (refer to item names, text elements etc) are meaningless.

```
<?xml version="1.0" standalone="yes" ?>
- <site>
- <regions>
- <africa>
- <item id="item0">
<location>United States</location>
<quantity>1</quantity>
<name>duteous nine eighteen</name>
<payment>Creditcard</payment>
- <description>
- <parlist>
- <listitem>
- <text>
page rous lady idle authority capt professes stabs monster petition heave humbly removes rescue runs shady peace most piteous worser oak assembly holes patience but malice whoreson mirrors master tenants smocks yielded
<keyword>officer embrace such fears distinction attires</keyword>
</text> </listitem>
- <listitem>
<text>shepherd noble supposed dotage humble servilius bitch theirs venus dismal wounds gum merely raise red breaks earth god folds closet captain dying reek</text> </listitem> </parlist> </description>
<shipping>Will ship internationally, See description for charges</shipping>
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
- <mailbox>
- <mail>
<from>Dominic Takano mailto:Takano@yahoo.com</from>
<to>Mechthild Renear mailto:Renear@acm.org</to>
<date>10/12/1999</date>
```

```
- <text> asses scruple learned crowns preventions half whisper logotype
weapons doors factious already pestilent sacks dram atwain girdles
deserts flood park lest graves discomfort sinful conceiv therewithal
motion stained preventions greatly suit observe sinews enforcement
  <emph>armed</emph>
  gold gazing set almost catesby turned servilius cook doublet preventions
shrunk
  </text> </mail> </mailbox> </item> </africa>
- <asia>
- <item id="item1">
  <location>United States</location> <quantity>1</quantity>
  <name>great</name> <payment>Money order, Cash</payment>
- <description>
- <text> print deceit arming ros apes unjustly oregon spring hamlet
containing leaves italian turn
  <bold>spirit model favour disposition</bold>
  approach charg gold promotions despair flow assured terror assembly
marry concluded author debase get bourn openly gonzago wisest bane
continue cries </text> </description>
  <shipping>Will ship internationally</shipping>
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
- <mailbox>
- <mail>
  <from>Fumitaka Cenzer mailto:Cenzer@savera.com</from>
  <to>Lanju Takano mailto:Takano@itc.it</to>
  <date>02/24/2000</date>
- <text> entreaty hath fowl prescience bounds roof fiend intellect boughs
caught add jests feelingly doubt trojans wisdom greatness tune worship
doors fields reads canst france pay progeny wisdom stir mov impious
promis clothes hangman trebonius choose men fits preparation
  <keyword>benefit since eclipse gates</keyword> </text> </mail>
- <mail>
  <from>Papa Godskesen mailto:Godskesen@uwindsor.ca</from>
  <to>Ioana Blumberg mailto:Blumberg@conclusivestrategies.com</to>
  <date>08/02/2001</date>
- <text> jealousy back greg folded gauntlets conduct hardness across
sickness peter enough royal herb embrace piteous die servilius avoid
  <keyword>laying chance dungeons pleasant thyself fellow purse steward
heaven ambassador terrible doubtfully</keyword>
  milk sky clouds unbraced put sacrifices seas childish longer flout heavy
pitch rosalind orderly music delivery appease
</text> </mail> </mailbox> </item> </asia>
- <australia>
- <item id="item2"> <location>United States</location>
  <quantity>1</quantity> <name>scarce brook</name> <payment />
```

- <description>
- <parlist>
- <listitem>
- <text> senses concave valiant star further instruments bankrupts countrymen horrid costard youth necessity tend curiously waken witness navy there honest interest perceive defendant chief traffic where nuptial descent travel prepare agreed voices swears remember peerless doing
<keyword>preparation rejoice</keyword> </text> </listitem>
- <listitem>
- <text> swear canker barbarian parching coxcomb excess conspiring nobles sounded consider sayings fishified prime may spirit
<emph>untruths misgives choughs mew here garments tenfold</emph>
error discontent hung beatrice straight muse shame deep twice mann maecenas any conveyance fingers whereupon child case
<keyword>season presently victory women beating</keyword>
deprive almost wed dreams slew reveal </text> </listitem>
- <listitem>
- <text> spotted attend burden camillo field enlarge stead corporal ground tormenting
<bold>naturally sanctuary riddle exile coming awake senseless chance famous albans</bold>
service cricket limb from clouds amongst shore penker defend quantity dumb churlish uncover swung eros figur sulphur sky birth stare negligent unction shield instance ambition gate injury fort put infants find slavish hugh see afterwards slanders chides eyes minds alb loved endure combating voyage </text> </listitem>
- <listitem>
- <parlist>
- <listitem>
<text>maintained peril rivall suddenly finds studies weary truth indulgence anatomy assisted imminent may excepted yonder aches regal</text> </listitem>
- <listitem>
- <text>
<bold>friar prophetess</bold>
spirits delays turning cassio finding unpractis steel sweets promises credulity err nym complete star greatly mope sorry experience virtues been offending bed drives faction learnt hurl eleven huge
</text> </listitem>
- <listitem>
- <text> piece hours cruelly april league winged
<keyword>tract element sails course placed fouler four plac joint</keyword>
words blessing fortified loving forfeit doctor valiant crying wife county planet charge haughty precious alexander longboat bells lewd kingdoms knife giver frantic raz commend sit sovereignty engaged perceive its art alliance forge bestow perforce complete roof fie confident raging possible cassio teen crave park reign lords sounded our requite fourth confidence high </text> </listitem> </parlist> </listitem>
- <listitem>

- <parlist>
- <listitem>
- <text> sent fled bids oswald help answer artillery jealous hugh fingers gladly mows our craving
<emph>preventions spurr edmund drunk how faction quickly bolingbroke painfully</emph>
valorous line clasp cheek patchery encompassed honest after auspicious home engaged prompt mortimer bird dread jephthah prithee unfold deeds fifty goose either herald temperance coctus took sought fail each ado checking funeral thinks linger advantage bag ridiculous along accomplishment flower glittering school disguis portia beloved crown sheets garish rather forestall vaults doublet embassy ecstasy crimson rheum befall sin devout pedro little exquisite mote messenger lancaster hideous object arrows smites gently skins bora parting desdemona longing third throng character hat sov quit mounts true house field nearest lucrece tidings fought logotype eaten commanding treason censur ripe praises windsor temperate jealous made sleeve scorn throats fits uncape tended science preventions preventions high pipes reprieves
<bold>sold</bold>
marriage sampson safety distrust witch christianlike plague doubling visited with bleed offenders catching attendants
<emph>cars livery stand</emph>
denay
<keyword>cimber paper admittance tread character</keyword>
battlements seen dun irish throw redeem afflicts suspicion
</text> </listitem>
- <listitem>
- <text> traduc barks twenty secure pursuit believing necessities longs mental lack further observancy uncleanly understanding vault athens lucius sleeps nor safety evidence repay whensoever senses proudest restraint love mouths slaves water athenian willingly hot grieves delphos pavilion sword indeed lepidus taking disguised proffer salt before educational streets things osw rey stern lap studies finger doomsday pots bounty famous manhood observe hopes unless languish
<keyword>transformed nourish breeds north</keyword>
</text> </listitem> </parlist> </listitem> </parlist> </description>
<shipping>Will ship internationally</shipping>
<incategory category="category0" />
<incategory category="category0" />
- <mailbox>
- <mail>
<from>Aspi L'Ecuyer mailto:L'Ecuyer@intersys.com</from>
<to>Lesley Jeris mailto:Jeris@zambeel.com</to>
<date>10/09/1998</date>
<text>necessities chains rosencrantz house heed course lawn diest unvirtuous supposed sees chough swor numbers game roman soundest wrestler sky lodovico beast shivers desolate norfolk forgot paulina wars george while beggar sheath thursday capable presently his protector father orchard enemies believe drains tokens prison charge cloud stab york mild scene true devotion confidence hundred those guiltless pricks

sort himself mutiny officers directive wholesome edge acts dion ride draw brings custom chapless beside sex dowry casca goods priam blasphemy prick octavia brain curer thinkest idiot inward missing conspiracy tents scab inundation caesar officer dramatis</text>
</mail> </mailbox> </item> </australia>
- <europe>
- <item id="item3"> <location>Uzbekistan</location>
<quantity>1</quantity> <name>abhorr execution beckon rue</name>
<payment>Money order, Creditcard, Cash</payment>
- <description>
- <parlist>
- <listitem>
- <text> <keyword>perjur kills insanie unfortunate conjuration deeper confounded belied first guard</keyword>
pale profits height desir ashore france strength kept entrench poisons worth fought ignorance moody poniards speaks jack egg offspring victory food double emperor round jewel abbey apparel untainted lass protest start wings acquit lake lady battles further low thief try brook cake mounted officers dean shrunk lowness dew sandy prologue armies suspicion eighty advance thankfulness albany ended experience halt doubted wert kingdom fiend directed pair perhaps </text> </listitem>
- <listitem>
- <text>
prayer odds rend condemn conrade swearing dispos losses boar little from thought different couch respected human robe dictynna later pays edward babe distemper bards damned mayst sustain while self alcibiades listen weak soil <keyword>view presume loggets feed</keyword>
afoot yields erection balthasar fathers datchet thankless lear cause evil cousin reported porter beastly jade bark sex slack lear devil devoured amiable mason moss shoulders labour meanest feign eggs encount forbid enobarbus halters nam emilia fiends bearing food inheritor wiser
<emph>hedge</emph>
functions there capital greasy dark crush your sequest between devout thou strikes demand dost reverent conference least told ado modena nearness safer jacks shut dire mates wind unfortunate monsieur parcels sauced extremities ropes contrive story slain advise lecher ardea relics keeping treads buckingham defences lag neighbour ourself marshal disordered moderate venus afeard article rot hazards craft crowns
<emph>plainness patient</emph>
lying knowledge diseases meritorious medicine instead lid happy without them bands answer
</text> </listitem> </parlist> </description>
<shipping>Will ship only within country</shipping>
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />
<incategory category="category0" />

```
  <incategory category="category0" />
  <mailbox />  </item>  </europe>
- <namerica>
- <item id="item4">  <location>United States</location>
  <quantity>1</quantity>  <name>unsur brutish</name>
  <payment>Money order, Creditcard</payment>
- <description>
- <parlist>
- <listitem>
- <text>  prepar likelihood eagle body walk borachio month writing left speed patents coach through protectorship congruent confusion favours bring usurers stew beard longed creep hid pursuivant beholders senators son mercutio woo bestow trumpet excess muffler pick ugly felt causes remove adding tear often rounds underbearing tree purer kibes endless women benefit throw
- <emph>  claim firmness
  <keyword>arrived sees wrestled multitude repent preventions infamy reproof shalt hearted prais knave doubtless</keyword>
  deny  </emph>
  merely grave voluble late loath digest horn slave hunger stronger amazed salt killing ross cry dry tongue kiss yields auspicious quietness perpetual ways  </text>  </listitem>
- <listitem>
- <text>  court mean returning brook creatures appointed paunches henry sights west prunes flutes regiment seems bed musicians slumber post friendship prevention abreast wouldst words vexation builds unfelt holly walk inform moods deck bulk begin action school nobles antique people unkennel stomach into petitions jack assail yongrey ages betimes golden sink droop kernel hoppedance perfection weight
  <emph>whining safe english rod other featur</emph>
  betwixt orator across amiss mine guests guard yon willing remit longing goneril visitation honey  </text>  </listitem>  </parlist>  </description>
  <shipping>Will ship only within country, Buyer pays fixed shipping charges, See description for charges</shipping>
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
- <mailbox>
- <mail>
  <from>Honari Castan mailto:Castan@uni-muenchen.de</from>
  <to>Maz Lucky mailto:Lucky@washington.edu</to>
  <date>01/24/1998</date>
  <text>scene disposition substance prick counsel start temples</text>
  </mail>  </mailbox>  </item>  </namerica>
- <samerica>
- <item id="item5">  <location>United States</location>
  <quantity>1</quantity>  <name>nakedness</name>
  <payment>Creditcard, Personal Check, Cash</payment>
- <description>
```

```
- <text>  music sift kissing design airy office dismantled hope reconcil
combat wert quite translate overcome unthrifty
- <emph>  fell othello
  <bold>wolf entreat audaciously down sands sports pilgrimage duellist
league holiday cheek that tables merrily knot selves ionia impure</bold>
  prophet draw throwing solemn yonder  </emph>
  rightful foam worthless polack veronesa antony beget thereby carry
untread hales  </text>  </description>
  <shipping>Will ship only within country, Will ship
internationally</shipping>
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
  <incategory category="category0" />
  <mailbox />  </item>  </samerica>  </regions>
- <categories>
- <category id="category0">
  <name>dispatch reported dotard holofernes</name>
- <description>
- <parlist>
- <listitem>
  <text>shift carrion doubtful strangle sounding crowned troubled naked
yesterday owe silent recount waters derive sans four</text>  </listitem>
- <listitem>
- <parlist>
- <listitem>
  <text>fragment pamper arthur thrive wound fouler streets preventions
obey vow bawds myrtle said infinite montague fierce sense ride souls
commended gainsay profession labour intents alter</text>  </listitem>
- <listitem>
  <text>ord villain wore thunder congeal pawned alack customary deny
faithful top office spoken please neighbour office afternoon drum
embowell touch sue lifeless leapt called weary congregation yield</text>
  </listitem>
- <listitem>
- <text>  mental fatal hard ancient stands cor dishes therein gramercy
discipline farewell dire tricks protest cut horatio brother speech sleeping
rebellion afraid repented tree scald stopp wine advise undermine norfolk
betimes sight now for oaths vouchsafe particulars globe laertes afflictions
rouse once news humanity buck destroy military lucius lap
  <keyword>considered forc mourning verona</keyword>
  waters triumphing officer hastily
  <emph>resign subject figure hay thwart written signs gout bred distance
period glove players change folly</emph>
  going wat lost song hautboys pick business crocodile leading cave twice
frenzy sprightly dislike invite forbids morn devour ambassador seldom
speak tickling rejoice triumphant ascanius forward
  </text>  </listitem>  </parlist>  </listitem>  </parlist>  </description>
  </category>  </categories>
- <catgraph>  <edge from="category0" to="category0" />  </catgraph>
```

```
- <people>
- <person id="person0"> <name>Jaak Tempesti</name>
  <emailaddress>mailto:Tempesti@labs.com</emailaddress>
  <phone>+0 (873) 14873867</phone>
  <homepage>http://www.labs.com/~Tempesti</homepage>
  <creditcard>5048 5813 2703 8253</creditcard>
- <watches> <watch open_auction="open_auction0" />
  </watches> </person> </people>
- <open_auctions>
- <open_auction id="open_auction0">
  <initial>13.56</initial> <reserve>33.78</reserve>
- <bidder> <date>10/22/2001</date>
  <time>10:21:43</time> <personref person="person0" />
  <increase>55.50</increase> </bidder>
- <bidder> <date>07/27/2001</date>
  <time>12:36:50</time> <personref person="person0" />
  <increase>19.50</increase> </bidder>
- <bidder> <date>02/14/2000</date>
  <time>16:40:16</time> <personref person="person0" />
  <increase>19.50</increase> </bidder>
- <bidder> <date>05/09/2001</date>
  <time>11:39:57</time> <personref person="person0" />
  <increase>30.00</increase> </bidder>
- <bidder> <date>07/12/1999</date>
  <time>23:20:27</time> <personref person="person0" />
  <increase>13.50</increase> </bidder>
- <bidder> <date>10/21/2001</date>
   <time>01:19:47</time> <personref person="person0" />
  <increase>3.00</increase> </bidder>
- <bidder> <date>09/28/2001</date>
  <time>17:03:24</time> <personref person="person0" />
  <increase>6.00</increase> </bidder>
- <bidder> <date>11/15/1999</date>
  <time>14:23:15</time> <personref person="person0" />
  <increase>9.00</increase> </bidder>
- <bidder> <date>01/02/1998</date>
  <time>22:18:07</time> <personref person="person0" />
  <increase>1.50</increase> </bidder>
- <bidder> <date>12/24/2001</date>
  <time>16:46:32</time> <personref person="person0" />
  <increase>13.50</increase> </bidder>
- <bidder> <date>08/12/2000</date>
  <time>11:41:54</time> <personref person="person0" />
  <increase>3.00</increase> </bidder>
- <bidder> <date>11/15/2000</date>
  <time>15:53:40</time> <personref person="person0" />
  <increase>6.00</increase> </bidder>
- <bidder> <date>03/04/2000</date>
  <time>20:46:15</time> <personref person="person0" />
  <increase>16.50</increase> </bidder>
```

```
- <bidder>  <date>07/22/1998</date>
  <time>10:34:11</time>  <personref person="person0" />
  <increase>25.50</increase>  </bidder>
- <bidder>  <date>04/01/1998</date>
  <time>10:44:22</time>  <personref person="person0" />
  <increase>7.50</increase>  </bidder>
  <current>243.06</current>  <itemref item="item0" />
  <seller person="person0" />
- <annotation>  <author person="person0" />
- <description>
- <text>  debauch corpse canons domain night forsake yea satisfy
between fume were monsters ear players moreover ungentleness sorrows
prouder tonight favours rome bastard unshown excellence journey loves
needy feasted romeo rivers worser occupation brook stoops brooch plucks
level samp tent windsor rubs whereof beam signior built suff heavy dull
husbands roman favour urge spear gone wolf cheeks execute resolv such
horrid drives provide twice spoke trade friar taking pheasant sentenc
scarf corrections brothers charge spur ass agamemnon truepenny saves
roots practis impatient diest didest starv seeing beneath interpose gods
home black forgot snuff dress dozen napkins
  <emph>countess northumberland headlong angry pleading</emph>
  better joy  <emph>meagre</emph>
  reap enquire crab wales died violent rear past liberty
  <emph>braggart armour infer bankrupt winds teeth</emph>
  case wore pouch crows cognition
  <keyword>reports expedition free chief cressida hearsed</keyword>
  loath monuments silent congregation soon farm doct ross susan ready
empty dedicate shilling whole soul foot beseech higher lifeless hay
postmaster distress disposition  <bold>inherits</bold>
  marcus betters pitch betray beam corse player quality ros conduct
thersites greediness boast pilgrims startles contented belch hung thus
captain early blood par brook jul gain needs above ensign grapes revelling
glean thank  </text>  </description>
  <happiness>6</happiness>  </annotation>
  <quantity>1</quantity>  <type>Regular</type>
- <interval>  <start>06/16/1999</start>  <end>05/12/2001</end>
  </interval>  </open_auction>  </open_auctions>
- <closed_auctions>
- <closed_auction>
  <seller person="person0" />  <buyer person="person0" />
  <itemref item="item1" />  <price>113.87</price>
  <date>06/06/2000</date>  <quantity>1</quantity>
  <type>Regular</type>
- <annotation>
  <author person="person0" />
- <description>
- <parlist>
- <listitem>
- <parlist>
- <listitem>
```

- <text> farewells religion fetch bells rage names valued exeunt soul albans ungently advised serving ratcliff braggarts knowest desp sheep died repeat toy corrupted michael help dunghill trembles pill reap office early secure desires hated garland carriage impatient deserts feel challenger evil
<bold>editions depart laur hereford richer</bold> </text> </listitem>
- <listitem>
<text>proudest lust approve rey should spectacles fiery perfect worshipp foul quod yes remorse young tyburn thrust attending spear shun doctor wild</text> </listitem> </parlist> </listitem>
- <listitem>
<text>throng grandam awak helpless ventidius tread defeat teem durst wonderful attaint chaste sees fulfill mortality arme expedient attendants themselves performed leading sing villain skill store mischief see consciences sail text speed sons spleen die oft girl atomies commodity honor fall stopp they</text> </listitem>
- <listitem>
- <text> rain pays spilling rancour reasons grieves camp bachelor crow can whom soldiers growth invite less for vaughan properties
<keyword>record penury herself reasons merits villainous whereupon wrote penny mar</keyword> preventions followed best eternity bestow blots rocks barnardine torn liege astonished suspicion unmannerd alexander crown soil committed god stately incensed trance oracle slowness fast princes damned corn grandsire change tender end fields
<bold>riggish</bold>
quirks shut thence beware jewels sland preventions has sells assails influences oppression pow maggot caught methought mechanical durst liker not seat
- <emph> assigns flesh made his third <keyword>seemeth</keyword>
peril gain they stroke forsworn scape full determin professes commons
</emph>
lordship clear operation practice pyrrhus earnest broke devil posterity company text misbegotten oregon strike saw arthur earnestly brow popilius ugly serves presentation commandment metal comparing thereon true secretly gallows horridly slack lieutenant hers stop clown rosalinde wed pretty wildly </text> </listitem> </parlist> </description>
<happiness>9</happiness> </annotation> </closed_auction>
- <closed_auction>
<seller person="person0" /> <buyer person="person0" />
<itemref item="item2" /> <price>96.92</price>
<date>12/05/2001</date> <quantity>1</quantity>
<type>Featured</type>
- <annotation>
<author person="person0" />
- <description>
- <text> hitherto queen painted seat fords clay recall countryman divided delicate mocking active bills filth pledge surrender madness sufficiency moved converse goot claw show edmundsbury torment tough fish mediators tarquin pyrrhus
<keyword>heathen</keyword> </text> </description>

```
<happiness>6</happiness> </annotation> </closed_auction>
- <closed_auction>
<seller person="person0" /> <buyer person="person0" />
<itemref item="item3" /> <price>53.85</price>
<date>05/11/1999</date> <quantity>1</quantity>
<type>Featured</type>
- <annotation>
<author person="person0" />
- <description>
<text>strives occasion question sticks shall ingenious sinews liquid ashy gentlewomen authority assay hole selves living near doting modest wiltshire mocker eton profess forgeries butt wade lawful maccabaeus wert forced succeeding becomes wayward got</text> </description>
<happiness>6</happiness> </annotation> </closed_auction>
- <closed_auction>
<seller person="person0" /> <buyer person="person0" />
<itemref item="item4" /> <price>123.52</price>
<date>02/11/1999</date> <quantity>1</quantity>
<type>Regular</type>
- <annotation>
<author person="person0" />
- <description>
- <text> vowed keys imperial were swinstead forsake cat aliena spies crave requite forfeit doctor
<emph>possess</emph>
aught demand ceremonies obscure engross hero restraint bolingbroke neighbour crimes dominions common turns conduct wav therewithal abandon yet hunger </text> </description>
<happiness>5</happiness> </annotation> </closed_auction>
- <closed_auction>
<seller person="person0" /> <buyer person="person0" />
<itemref item="item5" /> <price>96.06</price>
<date>04/24/1999</date> <quantity>2</quantity>
<type>Featured</type>
- <annotation>
<author person="person0" />
- <description>
- <text> pelican contrive paradoxes unmask desdemona weak pleases shame wisely cheek poison avoid ulysses exeunt answer smoothing punishment much anointed bloody worst putting repeal grounds bestrid commission crave mess tarries sport view freely lame done intend cast shun kills presented body landed question hem same burdens plenty esteem weak sigh sunday body preventions revenge horses cleomenes
<emph>ajax states mark parcels advertised utterly virtue flatter sleeping ope</emph>
lucilius tybalt glow killed account obdurate kindly
<bold>heart light bosom garden cog yet daughters tott</bold>
lifted offer </text> </description>
<happiness>4</happiness>
</annotation> </closed_auction> </closed_auctions> </site>
```

Appendix C GETTING THE XML SCHEMA

Using the dtd2xs program I was able to generate the XML Schema corresponding to the auction.xml's DTD.

The dtd2xs is located at http://www.lumrix.net/dtd2xs.php and it runs using the html file *dtd2xsd*. It takes the DTD file as an input and it generates the corresponding XML Schema. I have attached the resulted schema below.

I was also able to validate the schema against different xml files (generated with win32 of XMark) using the tool "XSD Validator" located at http://www.w3schools.com/schema/default.asp

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="site">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="regions" />
      <xs:element ref="categories" />
      <xs:element ref="catgraph" />
      <xs:element ref="people" />
      <xs:element ref="open_auctions" />
      <xs:element ref="closed_auctions" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="regions">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="africa" />
      <xs:element ref="asia" />
      <xs:element ref="australia" />
      <xs:element ref="europe" />
      <xs:element ref="namerica" />
      <xs:element ref="samerica" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="edge">
    <xs:complexType>
      <xs:attribute name="from" type="xs:IDREF" use="required" />
```

```
<xs:attribute name="to" type="xs:IDREF" use="required" />
   </xs:complexType>
 </xs:element>
 <xs:element name="categories">
   <xs:complexType>
     <xs:sequence>
     <xs:element maxOccurs="unbounded" ref="category" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="category">
   <xs:complexType>
     <xs:sequence>
     <xs:element ref="name" />
     <xs:element ref="description" />
     </xs:sequence>
     <xs:attribute name="id" type="xs:ID" use="required" />
   </xs:complexType>
 </xs:element>
 <xs:element name="name" type="xs:string" />
 <xs:element name="description">
   <xs:complexType>
     <xs:choice>
     <xs:element ref="text" />
     <xs:element ref="parlist" />
     </xs:choice>
   </xs:complexType>
 </xs:element>
 <xs:element name="text">
   <xs:complexType mixed="true">
     <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element ref="bold" />
     <xs:element ref="keyword" />
     <xs:element ref="emph" />
     </xs:choice>
   </xs:complexType>
 </xs:element>
 <xs:element name="bold">
   <xs:complexType mixed="true">
     <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element ref="bold" />
     <xs:element ref="keyword" />
     <xs:element ref="emph" />
     </xs:choice>
   </xs:complexType>
 </xs:element>
 <xs:element name="keyword">
   <xs:complexType mixed="true">
     <xs:choice minOccurs="0" maxOccurs="unbounded">
     <xs:element ref="bold" />
     <xs:element ref="keyword" />
     <xs:element ref="emph" />
     </xs:choice>
   </xs:complexType>
 </xs:element>
 <xs:element name="emph">
```

```
    <xs:complexType mixed="true">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="bold" />
      <xs:element ref="keyword" />
      <xs:element ref="emph" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="parlist">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="listitem" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="listitem">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="text" />
      <xs:element ref="parlist" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="catgraph">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="edge" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="africa">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="item" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="asia">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="item" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="australia">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="item" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="namerica">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="item" />
      </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
<xs:element name="samerica">
  <xs:complexType>
    <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="item" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="europe">
  <xs:complexType>
    <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="item" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
    <xs:element ref="location" />
    <xs:element ref="quantity" />
    <xs:element ref="name" />
    <xs:element ref="payment" />
    <xs:element ref="description" />
    <xs:element ref="shipping" />
    <xs:element maxOccurs="unbounded" ref="incategory" />
    <xs:element ref="mailbox" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
    <xs:attribute name="featured" type="xs:string" />
  </xs:complexType>
</xs:element>
<xs:element name="location" type="xs:string" />
<xs:element name="quantity" type="xs:string" />
<xs:element name="payment" type="xs:string" />
<xs:element name="shipping" type="xs:string" />
<xs:element name="reserve" type="xs:string" />
<xs:element name="incategory">
  <xs:complexType>
    <xs:attribute name="category" type="xs:IDREF" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="mailbox">
  <xs:complexType>
    <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="mail" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="mail">
  <xs:complexType>
    <xs:sequence>
    <xs:element ref="from" />
    <xs:element ref="to" />
    <xs:element ref="date" />
    <xs:element ref="text" />
```

```
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="from" type="xs:string" />
<xs:element name="to" type="xs:string" />
<xs:element name="date" type="xs:string" />
<xs:element name="itemref">
  <xs:complexType>
    <xs:attribute name="item" type="xs:IDREF" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="personref">
  <xs:complexType>
    <xs:attribute name="person" type="xs:IDREF" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="people">
  <xs:complexType>
    <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="person" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
    <xs:element ref="name" />
    <xs:element ref="emailaddress" />
    <xs:element minOccurs="0" ref="phone" />
    <xs:element minOccurs="0" ref="address" />
    <xs:element minOccurs="0" ref="homepage" />
    <xs:element minOccurs="0" ref="creditcard" />
    <xs:element minOccurs="0" ref="profile" />
    <xs:element minOccurs="0" ref="watches" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="emailaddress" type="xs:string" />
<xs:element name="phone" type="xs:string" />
<xs:element name="address">
  <xs:complexType>
    <xs:sequence>
    <xs:element ref="street" />
    <xs:element ref="city" />
    <xs:element ref="country" />
    <xs:element minOccurs="0" ref="province" />
    <xs:element ref="zipcode" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="street" type="xs:string" />
<xs:element name="city" type="xs:string" />
<xs:element name="province" type="xs:string" />
<xs:element name="zipcode" type="xs:string" />
<xs:element name="country" type="xs:string" />
```

```
  <xs:element name="homepage" type="xs:string" />
  <xs:element name="creditcard" type="xs:string" />
  <xs:element name="profile">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="interest" />
      <xs:element minOccurs="0" ref="education" />
      <xs:element minOccurs="0" ref="gender" />
      <xs:element ref="business" />
      <xs:element minOccurs="0" ref="age" />
      </xs:sequence>
      <xs:attribute name="income" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="interest">
    <xs:complexType>
      <xs:attribute name="category" type="xs:IDREF" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="education" type="xs:string" />
  <xs:element name="income" type="xs:string" />
  <xs:element name="gender" type="xs:string" />
  <xs:element name="business" type="xs:string" />
  <xs:element name="age" type="xs:string" />
  <xs:element name="watches">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="watch" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="watch">
    <xs:complexType>
      <xs:attribute name="open_auction" type="xs:IDREF" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="open_auctions">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="open_auction"
/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="open_auction">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="initial" />
      <xs:element minOccurs="0" ref="reserve" />
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="bidder" />
      <xs:element ref="current" />
      <xs:element minOccurs="0" ref="privacy" />
      <xs:element ref="itemref" />
      <xs:element ref="seller" />
      <xs:element ref="annotation" />
      <xs:element ref="quantity" />
```

```
      <xs:element ref="type" />
      <xs:element ref="interval" />
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="privacy" type="xs:string" />
  <xs:element name="initial" type="xs:string" />
  <xs:element name="bidder">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="date" />
      <xs:element ref="time" />
      <xs:element ref="personref" />
      <xs:element ref="increase" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="seller">
    <xs:complexType>
      <xs:attribute name="person" type="xs:IDREF" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="current" type="xs:string" />
  <xs:element name="increase" type="xs:string" />
  <xs:element name="type" type="xs:string" />
  <xs:element name="interval">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="start" />
      <xs:element ref="end" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="start" type="xs:string" />
  <xs:element name="end" type="xs:string" />
  <xs:element name="time" type="xs:string" />
  <xs:element name="status" type="xs:string" />
  <xs:element name="amount" type="xs:string" />
  <xs:element name="closed_auctions">
    <xs:complexType>
      <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded"
ref="closed_auction" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="closed_auction">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="seller" />
      <xs:element ref="buyer" />
      <xs:element ref="itemref" />
      <xs:element ref="price" />
      <xs:element ref="date" />
      <xs:element ref="quantity" />
```

```
      <xs:element ref="type" />
      <xs:element minOccurs="0" ref="annotation" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="buyer">
    <xs:complexType>
      <xs:attribute name="person" type="xs:IDREF" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="price" type="xs:string" />
  <xs:element name="annotation">
    <xs:complexType>
      <xs:sequence>
      <xs:element ref="author" />
      <xs:element minOccurs="0" ref="description" />
      <xs:element ref="happiness" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="author">
    <xs:complexType>
      <xs:attribute name="person" type="xs:IDREF" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="happiness" type="xs:string" />
</xs:schema>
```

Appendix D The MLCAS Algorithm

```
MLCAS (I1, I2, . . . , Im):
 0.  let the set of input nodes from I1, I2, . . . , Im be I
 1.  while (unprocessed input or stack is not empty)
 2.        let tmin(from Iindex)be the node with smallest StartPos in I
 3.        while (stack is not empty &&
 4.              tmin is not a descendant of current stack top)
 5.          /* pop the top element in the stack */
 6.          popped = stack→Pop(), top = stack→Top()
 7.          if (popped and its Elists contain MLCASs)
 8.              output popped /*no more MLCAS on current stack*/
 9.              while (stack is not Empty) stack→Pop()
10.          /* popped will not be a root of any MLCAS*/
11.          else if (popped→head is a child of top→head)
12.              mark all the non-empty Elists of popped as Related
13.            /* if popped qualifed to be part of an MLCAS */
14.            if (for any i, popped→Elist[i] or top→Elist[i] is empty)
15.              top→AppendLists(popped→GetLists()
16.            else if(for any i,j(i≠j),if top→ElistsRelated(i,j)=true
17.              then popped→ElistsRelated(i,j) = true)
18.              if(exists i,j(i≠j) that top→ElistsRelated(i,j)=false
19.                    && popped→ElistsRelated(i,j)=true)
20.                    /*delete nodes unqualifed to be in an MLCAS*/
21.                    delete all nodes from top→Elist[i], top→Elist[j]
22.              top→AppendLists(popped→GetLists())
23.          else let pt = popped→head→GetParent()
24.              mark all the non-empty Elists of popped as Related
25.              popped→ReplaceHead(pt) /*replace with parent*/
26.              stack→Push(popped)
27.        if (stack is empty)
28.              stack→Push(tmin), top = stack→Top()
29.              top→SetMaxID(0)
30.              /*set min of newnode be 0*/
31.              newnode=NewListNode(tmin, 0)
32.              top→Elist[index] →AppendNode(newnode)
33.        else
34.              oldtop = stack→Top(),stack→Push(tmin)
35.              top = stack→Top()
```

```
36.             top→SetMaxID(oldtop→GetMaxID())
37.             /*assign min to distinguish nodes of different MLCASs*/
38.             if (oldtop→Elist[index] is empty)
39.             newnode=NewListNode(t_min, oldtop→GetMaxID())
40.             else if(oldtop→Elist[index] not Related with other Elists)
41.                     newnode=NewListNode(t_min,oldtop→GetMaxID())
42.                 else
43.                     top→SetMaxID(oldtop→GetMaxID()+1)
44.                     newnode=NewListNode(tmin,top→GetMaxID())
45.             top→Elist[index]  →AppendNode(newnode)
46.         read I for the next t_min
```