Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

The XML Benchmark Project

A.R. Schmidt, F. Waas, M.L. Kersten, D. Florescu, I. Manolescu, M.J. Carey, R. Busse

Information Systems (INS)

INS-R0103 April 30, 2001

Report INS-R0103
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

The XML Benchmark Project

Albrecht Schmidt[1]

Florian Waas[1]

Martin Kersten[1]

Daniela Florescu[2]

Ioana Manolescu[3]

Michael J. Carey[2]

Ralph Busse[4]

[1] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, *firstname.lastname*@cwi.nl
[2] Propel, San Jose (CA), USA, DanaF@propel.com, mike.carey@propel.com
[3] INRIA-Rocquencourt, 78153 Le Chesnay Cedex, France, Ioana.Manolescu@inria.fr
[4] GMD-IPSI, Dolivostr. 15, 64293 Darmstadt, Germany, busse@darmstadt.gmd.de

ABSTRACT
With standardization efforts of a query language for XML documents drawing to a close, researchers and users increasingly focus their attention on the database technology that has to deliver on the new challenges that the sheer amount of XML documents produced by applications pose to data management: validation, performance evaluation and optimization of XML query processors are the upcoming issues. Following a long tradition in database research, the XML Store Benchmark Project provides a framework to assess an XML database's abilities to cope with a broad spectrum of different queries, typically posed in real-world application scenarios. The benchmark is intended to help both implementors and users to compare XML databases independent of their own, specific application scenario. To this end, the benchmark offers a set queries each of which is intended to challenge a particular primitive of the query processor or storage engine.

The overall workload we propose consists of a scalable document database and a concise, yet comprehensive set of queries, which covers the major aspects of query processing. The queries' challenges range from stressing the textual character of the document to data analysis queries, but include also typical ad-hoc queries.

We complement our research with results obtained from running the benchmark on our XML database platform. They are intended to give a first baseline, illustrating the state of the art.

1998 ACM Computing Classification System: [H.2.1] Logical Design (Data Models), [H.2.3] Languages (Query Languages),) [H.2.4] Database Systems, (Query Processing)
Keywords and Phrases: Benchmarking, Database Performance, Semi-structured Databases

1. Introduction

XML as a data interchange format has been penetrating virtually all areas of Internet application programming, and has been bringing about massive amounts of data. Thus content providers are increasingly interested in deploying advanced data management systems for sites whose data volume exceeds toy sizes. The complexity of the challenge has also attracted the attention of the database research community. Early efforts mainly concentrated on schema issues and the theory of organizing data lacking a fixed schema, which seemed incompatible with existing technology on first sight. However, as XML gained more and more momentum and commercial products began to appear on the market – many more being under development – the focus of research shifted; specific technical issues like physical data break-down and query performance are starting to determine the success or failure of the solutions under development.

Currently all major and minor database vendors (see [Bou00a] for an ever growing list) are scrambling to leverage their existing products – well beyond the rudimentary XML support like conversion of purely relational data to XML documents which most products already provide – with whatever one may need to meet the new requirements. However, these new requirements are still somewhat sketchy and though the differences between XML and relational or object-relational data are easy to grasp, the implications on the underlying data store are not fully understood yet.

XML, by definition is a textual markup language which means that unlike in the case of (O)RDBMS, data elements are ordered by nature; *string* is the core data type, from which richer data types, *e.g.* integers, floats and even user-defined abstract data types are derived; externally provided schema information, which may or may not be present, helps to avoid excessive and expensive coercions between data types. Additionally, to cope with the tree structure of XML documents and the resulting intricate hierarchical relationships between data, regular path expressions are an essential ingredient of query languages and need to be evaluated efficiently. References may be used to model relationships that exceed the limitations of tree structures and require further mapping logics like logical OIDs for efficient management.

Earlier work on how to map this wealth of new properties to existing data models provides undoubtedly helpful guidance but since almost all of these prototypes were implemented *on top* of data or object stores, *i.e.* using standard APIs but without direct access to the internal workings of a product, the conclusions drawn are only valid to a certain extent and the effectiveness of a particular mapping remains unclear. Often, simple extensions to the product could have caused significant performance improvements [RPNK00]. Due to their complexity, interaction, and interdependencies with various system components, most of the designs, with their obvious advantages and disadvantages, are hard to assess without putting them to the only conclusive test: a comprehensive quantitative assessment, or in short the right benchmark.

Dealing with design decisions as outlined above is certainly no new issue and so, over the past years the database community developed a rich tradition in performance assessment of systems ranging from research developments like the Hypermodel [ABM$^+$90], OO7-Benchmark [CDN93] or the BUCKY benchmark [CDN$^+$97] to industrial standards like the family of TPC benchmarks [Gra93] just to mention some of the better know examples. However, none of the available benchmarks offers the coverage needed: all of them are geared towards a certain data model but the flexibility and possibilities of semi-structured query languages exceed existing systems' limitations by far.

The XMark Benchmark Version 1.0 described in this paper takes on the challenge and features a tool kit for evaluating the retrieval performance of XML stores and query processors: a workload specification, a scalable document database and a comprehensive set of queries. To help users, each of the queries is intended to challenge the query processor with an important primitive of the query language. This is useful in a number of ways: In the first place, a feature-wise examination of the query processors proves beneficial as a query processor can operate on a variety of architectures each of which tends to be suited for different application workloads and exhibits special characteristics. For instance, XML stores have been derived from relational, object-relational, main-memory and object-oriented database technology as well from textual information retrieval data structures and persistent object stores. Therefore, different products can be expected to display diverging behavior in performance and stress tests according to their system architecture. Second, the benchmark document and the queries can aid in the verification of query processors which has been a challenging problem since high level query languages were introduced [Slu98]. In the world of mark-up languages, the problem of equivalence of XML documents, which XML query processors are expected to output, aggravates. The degrees of freedom the different possible physical representations of the document (see [Boy01] for attempt to tackle it) introduce, are combined with the degrees of freedom in query execution regarding order of set-valued attributes, different character encodings, namespaces *etc.* In our experience, there is research to be done as to when regard the output of XML query processors equivalent. Third, executing the benchmark query set exhibits details of the workflow required to incorporate the query processor into an application scenario. Consequently, the doing of the benchmark also helps users to estimate the costs of actually deploying such a system in their application scenario and answer the question what systems fits best their needs.

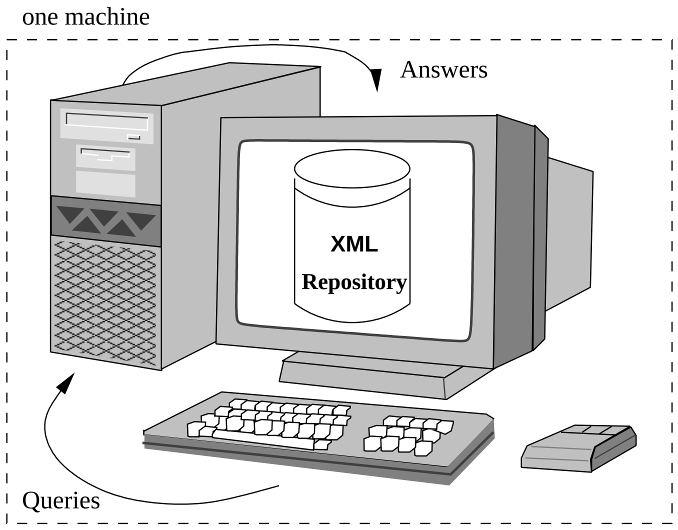

Figure 1: Workflow Scenario

XML processing systems usually consist of various logical layers and can be physically distributed over a network. To make the benchmark results interpretable we abstract from the systems engineering issues and concentrate only on the core ingredients: the query processor and its interaction with the data store. We do not consider network overhead, communication costs (*e.g.*, RMI, HTTP, CORBA, Sockets, RPC, Java Beans, *etc.*) or transformations (*e.g.*, XSL) of the output. Our test scenario is as depicted in Figure 1. All applications should be run on the same machine. As for the choice of language, we use XQuery [CFR$^+$01], an amalgamation of many research languages for semi-structured data or XML (see [BC00] for an overview). It is in the process of standardization as the language of choice of the major competitors in the field. We do not consider updates other than bulkload as there is little agreement on semantics and a standard is yet to be defined.

The target audience of this paper can be divided into three groups. First, the framework presented here can help database vendors to verify, refine their query processors by comparing them to other implementations. Second, customers can be assisted in choosing between products by our providing a simple case study or pilot project that yet provides essential ingredients of the targeted system. For researchers, lastly, we provide example data and a framework for helping to tailor existing technology for use in XML settings and for refinement or design of algorithms.

The rest of the paper is structured as follows: First, we motivate the necessity of a benchmark for XML query processors, then we introduce the structure of the document database. After presenting the queries we give some preliminary results obtained by running the queries in our test environment.

2. Motivation

Existing database benchmarks cover a plethora of aspects of traditional data management ranging from query optimization to transaction processing. But even if we make use of established techniques to store and process XML, it is not clear if the semi-structured nature of the data has repercussions on performance and engineering issues which impede on the effectiveness that these techniques have in their original area. In the sequel, we argue that there is a need for a new benchmark specifically for XML query processors.

The evolution of XML differs significantly from the evolution of relational databases in that there was a agreed standard at an early stage which was accepted and supported by a large community. This imposes a top-down perspective for the benchmark designer resulting in a kind of thematic benchmark, in the sense that it should provide challenges for typical query primitives. Thus, we think that the combination of traditional and new features present in XML processing systems results in the need for a new quality of systems engineering and, hence, a new benchmark.

While it has been shown that many 'data-centric' documents, *i.e.* documents which logically represent data structures [Bou00a], map very nicely to relational databases (*e.g.*, see [STZ$^+$99, FK99, SKWW00]) or object-relational databases [KM00], it is less clear how the same systems can handle efficiently documents that are 'text-oriented', *i.e.* natural language with mark-up interspersed (for example, compare Google's [BP98] full text search with that of your favorite RDBMS + XML mapping combination). Therefore we want to give hints how different DBMS architectures respond to the XML challenge which can be summarized as follows:

1. The textual *order* of XML structures as in the original document can be incorporated into queries: a feature that can make simple look-up queries dear in systems that are not prepared for this challenge (see Queries 2 and 3 in Section 5).

2. Strings are the basic data type; they can vary greatly in length posing additional storage problems. *Type problems* can also arise as the typing rules of query languages tend to clash the generic string tokens of XML.

3. Queries involving *hierarchical structures* in the form of complicated path expressions, especially $1 : n$ relationships in connection with order being queried tend to require expensive join and aggregation operations on relational systems.

4. To compound matters, the *loose schema* of many XML documents can make query formulation tedious from a user's point of view. Technically, NULL values can blow up the size of the database. From a user's viewpoint, specifying long and complicated path expressions is error-prone. (The authors know too well what they are talking about ...)

Activities around XML Schema [W3C01] try to allay the problem by making data-centric documents more accessible for (O)RDBMS by reformulating concepts integrity constraints in an XML context. This can indeed solve many problems but requires additional engineering efforts and thus sacrifices some of the quick-and-easy-to-use appeal that helped XML gain popularity so quickly. The benchmark queries have been designed to address these matters specifically.

3. Database Description

The design of an XML Benchmark requires a carefully modeled example database to make the behavior of queries predictable and to allow for the formulation of queries that both feel natural and present concise challenges. Let us first outline the characteristics of the document and then have a closer look at the technical issues of generating such documents.

3.1 Hierarchical Element Structure

The nesting of elements renders the overall tree structure of XML documents. In this subsection we describe the structure of the document which is modeled after a database as deployed by an Internet auction site. The main entities are: person, open auction, closed auction, item, and category. The relationships between them are expressed through references with the exception of *annotations* and *descriptions* which take after natural language text and are document-centric element structures embedded into the sub-trees to which they semantically belong. The hierarchical schema is depicted in Figure 2. The semantics of the entities just mentioned is as follows:

1. *Items* are the objects that are on for sale or that already have been sold. Each 'item' carries a unique identifier and bears properties like payment (credit card, money order, ...), a reference to the seller, a description *etc.*, all encoded as elements. Each item is assigned a world region represented by the item's parent.

2. *Open auctions* are auctions in progress. Their properties are the privacy status, the bid history (*i.e.* increases over time) along with references to the bidders and the seller, the current bid, the default increase, the type of auction (Dutch, Featured, Regular), the time interval within which bids are accepted, the status of the transaction and a reference to the item being sold.

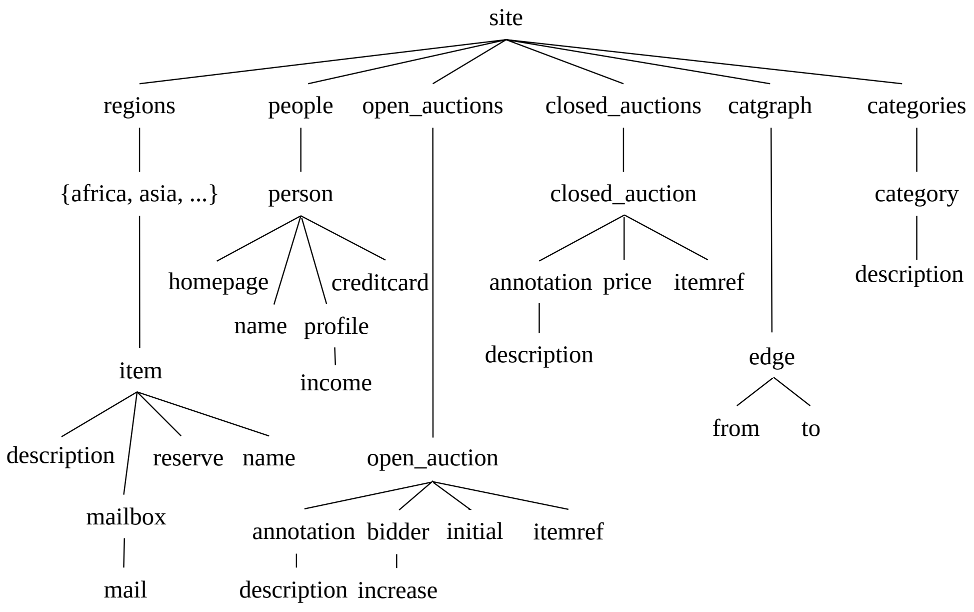

Figure 2: Element relationships between most of the queried elements

3. *Closed auctions* are auctions that are finished. Their properties are the seller (a reference to a person), the buyer (a reference to a person), a reference to the respective item, the price, the amount of items sold, the date when the transaction was closed, the type of transaction, and the annotations that were made before, during and after the bidding process.

4. *Persons* are characterized by name, email address, phone number, mail address, homepage URL, credit card number, profile of their interests, a set of open auctions they watch.

5. *Categories* feature a name and a description; they are used to implement classification of items. A *category_graph* links categories into a network.

These entities constitute the relatively structured and data-oriented part of the document: the schema is regular on a per entity basis and exceptions, such as that not every person has a homepage, are predictable. Apart from occasional lists such as bidding histories, the order of the input is not particularly relevant. This is in stark contrast to the offspring of *annotation* and *description* elements which make up for the dual nature, document-centric side, of XML. Here the length of strings and the internal structure of sub-elements varies greatly. The markup now comprises itemized lists, keywords, and even formatting instructions and character data, imitating the characteristics of natural language texts. This ensures that the database covers the full range of XML instance incarnations, from marked-up data structures to traditional prose. The appendix gives an impression of the document by showing some snippets.

3.2 References

An overview over the references that connect sub-trees is given in Figure 3. Care has been taken that the references feature diverse distributions, derived from uniformly and normally distributed variables. Also note that all references are typed, *i.e.* all instances of an XML element point to the same type of XML element; for example, references of interest object always refer to categories.

3.3 Generated Text

To generate text that bears the characteristics of natural language text, we analyzed Shakespeare's plays and determined statistic characteristics of the text. The generator mimicks this distribution using the 17000 most frequently occurring words of Shakespeare's plays. We did not incorporate additional characteristics like punctutation as it is only of little relevance for the performance assessment. We believe,

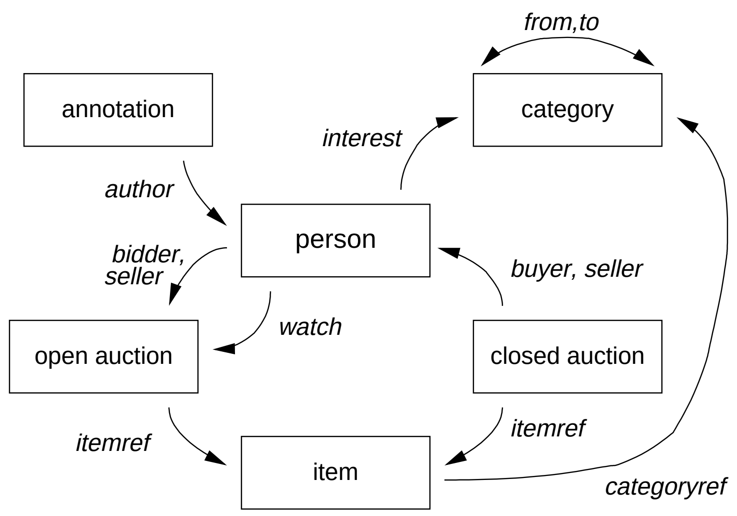

Figure 3: References

tokenization and other text compression methods commonly used can be sufficiently well assessed with the text we provide.

For names email addresses *etc.* we used various sources like electronically available phone directories *etc.* Care has been taken to preserve important characteristics of the original material, and at the same time to scramble data like names by combining different first and last names. Similar techniques were used to avoid clashes with existing email addresses and so forth.

3.4 XML Constructs

The XML Standard [BPSMM00] defines constructs that are useful for producing flexible markup but do not justify the definition of queries to challenge them directly. Therefore, we only made use of a restricted set of XML features in the data generator which we consider essential to XML processing. We do no generate documents with Entities or Notations. Neither do we distinguish between Parsed Character Data and Character Data as we assume that both are just string types from the viewpoint of the storage engine. Furthermore, we don't include Namespaces into the queries. We also restrict ourselves to the seven bit ASCII character set.

We provide a DTD and XML Schema information to allow for more efficient mappings. However, we stress that this is additional information that *may* be exploited.

3.5 `xmlgen` – Or How to Generate a Document

We designed and implemented a document generator, called `xmlgen`, to provide for a scalable XML document database. Besides the obvious requirement to be capable of producing the XML document specified above we were eager to meet the following additional demands. The generation of the XML document should be

- platform independent so that any user interested in running the benchmark is able to download the binary and generate exactly the same document no matter what hardware or operating system is used;
- accurately scalable ranging from a minimal document (scaling factor zero) to any arbitrary size limited only by the capacity of the system;
- both time and resource efficient, *i.e.* elapsed time ideally scales linearly whereas the resource allocation is constant – independent of the size of the generated document

First, in order to be able to reproduce the document independently of the platform, we incorporated a random number generator rather then relying on the system's built-in random number generators.

Name	Scaling Factor	Document Size
tiny	0.1	10 MB
standard	1	100 MB
large	10	1 GB
huge	100	10 GB

Figure 4: Scaling the benchmark document

Together with basic algorithms which can be found in statistics textbooks this unit implements uniform, exponential, and normal distributions of fairly high quality. We assigned each of the elements in the DTD a plausible distribution of its children and its references, observing consistency among referencing elements, that is, the number of items organized by continents equals the sum of open and closed auctions, *etc.* Second, to provide for accurate scaling we scale selected sets like the number of items and persons with the user defined factor. Moreover, we calibrated the numbers to match a total document size of slightly more than 100 MB for scaling factor 1.0 (*cf.* Fig. 4). In our tests, we found the resulting document size to deviate from the anticipated one usually by less than 1 percent. Finally, meeting the efficiency goals is an interesting challenge as references get created at different places and the straightforward solution of keeping some sort of log, which id has already been referenced is infeasible for large documents. We solved this problem by modifying the random number generation to produce several identical streams of random numbers. That way, we are able to implement a partitioning of sets like the item IDs that are referenced from both open and closed auctions. In its current version, `xmlgen` requires less than 2 MB of main-memory, and produces documents of sizes of 100 MB and 1 GB in 33.4 and 335.5 seconds, respectively (450MHz Pentium III). A more detailed description of the tool can be found at [SKF$^+$00].

4. Bulkloading The Document

In the context of XML, the rôle of bulkloading stands apart from its importance in other benchmarks. As (prospective) standards like XQuery [CFR$^+$01] are not explicitly designed as database but data integration languages, we can, strictly speaking, neither assume the need to bulkload documents nor the presence of a database. However, there should be little doubt that databases can help with managing large amounts of XML data. Whereas previous XML benchmarking efforts [BR01] are designed as application-centric multi-user workloads and include updates and bulk operations, we dispense with doing so because to date there is neither a standard nor an agreement on the exact semantics of updates. We therefore resort to the following interpretation: We take the XQuery Syntax for

```
    FOR $a in document("site.xml")/...
```

literally and formulate all queries with respect to a single large document without committing ourselves to a specific database scenario.

Although the authors themselves did not experience problems when bulkloading the document in our test environments, we are aware that the size of the document may be too large for some systems. Hence, the data generator `xmlgen` additionally offers a mode that outputs n entities per file where n can be chosen by the user.

Note that in this case, modifications to the one-document version of the benchmark may become necessary. For example, if the user chooses to make use of the DTD we supply, parser-controlled references, *i.e.* ID and IDREF declared attributes, should be converted to REQUIRED attributes. Otherwise a validating parser tries to check for uniqueness and existence of IDs respectively IDREFs. With respect to queries, changes to the path expressions, which as presented assume a single document, are necessary. Nevertheless, all changes remain local. However, we stress that this solution should be regarded as an work-around and that the semantics of the queries as defined in the following section should not differ no matter whether they are executed against a single document or a collection of document. The query semantics of the *one* document are normative.

5. Benchmark Queries

This section lists the queries of which the benchmark consists. We chose to express the queries in XQuery [CFR+01], the successor to Quilt [CRF00], which is about the be standardized. Note that the query set still is preliminary.

5.1 Exact Match

This simple query is mainly used to establish a simple performance primitive unit to help establish a 'metric' to interpret subsequent queries. It 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()
```

5.2 Ordered Access

These queries should help users to gain insight how the DBMS copes with the intrinsic order of XML documents and how efficient they can expect the DBMS to handle queries with order constraints.

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>
```

This query evaluates the cost of array look-ups. Note that this query may actually be harder to evaluate than it looks; especially relational back-ends may have to struggle with rather complex aggregations to select the bidder element with index 1.

Q3. Return the fist 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()>
```

This is a more complex application of index lookups. In the case of a relational DBMS, the query can take advantage of set-valued aggregates on the index attribute to accelerate the execution. Queries Q2 and Q3 are akin to aggregations in the TPCD [Gra93] benchmark.

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>
```

This time, we stress the textual nature of XML documents by querying the tag order in the source document.

5.3 Casting

Strings are the generic data type in XML documents. Queries that interpret strings will often need to cast strings to another data type that carries more semantics. This query challenges the DBMS in terms of the casting primitives it provides. Especially, if there is no additional schema information or just a DTD at hand, casts are likely to occur frequently.

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)
```

5.4 Regular Path Expressions

Regular path expressions are a fundamental building block of virtually every query language for XML or semi-structured data. These queries 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)
```

A good evaluation engine should realize that there is no need to traverse the complete document tree to evaluate such expressions.

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;
```

Also note that the `COUNT` aggregation does not require a complete traversal of the tree. Just the cardinality of the respective relation is queried. Note that the tag `<email>` does not exist in the database document.

5.5 Chasing References

References are an integral part of XML as they allow richer relationships than just hierarchical element structures. These queries define horizontal traversals with increasing complexity. A good query optimizer should take advantage of the cardinalities of the sets to be joined.

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

Constructing new elements may put the storage engine under stress especially in the context of creating materialized document views. The following query reverses the structure of person records by grouping them according to the interest profile of a person. Large parts of the person records are repeatedly reconstructed. To avoid simple copying of the original database we translate the mark-up into french.

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>
```

Fernandez *et al.* point out in [FMS01] that the cost of generating textual answers and creating materialized views may differ greatly. Note that we do not the require the system to construct a materialized view. However, comparing the relative costs of creating a view versus outputting text is certainly interesting.

5.7 Joins on Values

This query tests the database's ability to handle large (intermediate) results. This time, joins are on the basis of values. The difference between these queries and the reference chasing queries Q8 and Q9 is that references are specified in the DTD and may be optimized with logical OIDs for example. The two queries Q11 and Q12 cascade in the size of the result set and provide various optimization opportunities. We note that alternative query formulations with the '`->`' operator may be used.

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>
```

5.8 Reconstruction

A key design for XML→DBMS mappings is to determine the fragmentation criteria. The complementary action is to reconstruct the original document from its broken-down representation. Query 13 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>
```

5.9 Full Text

We continue to challenge the textual nature of XML documents; this time, we conduct a full-text search in the form of keyword search. Although full-text scanning could be studied in isolation we think that the interaction with structural mark-up is essential as the concepts are considered orthogonal; so query Q14 is restricted to a subset of the document by combining content and structure.

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()
```

5.10 Path Traversals

In contrast to Section 5.4 we now try to quantify the costs of long path traversals that don't include wildcards. We first descend deep into the tree (Query 15) and then return again (Query 16). Both queries only check for the existence of paths rather than selecting paths with predicates.

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 kweywords 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 />
```

5.11 Missing Elements

This is to test 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()/>
```

The fraction of people without a homepage is rather high so that this query also presents a challenging path traversal to non-clustering systems.

5.12 Function Application

This query puts the application of user defined functions (UDF) to the proof. In the XML world, UDFs are of particular importance because they allow the user to assign semantics to generic strings that go beyond type coercion.

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())
```

5.13 Sorting

Thanks to the lack of a schema, SORTBY clauses often play the role of the SQL-ish ORDER BY and GROUP BY. This query requires a 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 (.)
```

5.14 Aggregation

The following query computes a simple aggregation by assigning each person to a category. Note that the aggregation is truly semi-structured as it also includes those persons for whom the relevant data is not available.

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>
```

6. Experiments and Experiences

The benchmark has been a group-design activity and is being used to evaluate progress in both commercial and research settings. The evaluation presented here is meant as an illustrative case study because a full comparative analysis of various systems is definitely beyond the scope of this paper. The experience reported in this section is gathered from our work with Monet XML [SKWW00] to whose internal structure we have access and which is our research vehicle for XML processing. We also contrasted our findings with experiments made on disk-based relational systems which showed very similar behavior but did not add fundamentally new insights; therefore these results are not reported here. The semantics of the XQuery version of the queries were checked with Kweelt [Sah00].

The tools to run the benchmark document will be made available on the project web site [SKF+00]. They include the data generator and the query set along with a mapping tool to convert the benchmark document into a flat file that may be bulk-loaded into a (relational) DBMS; the mapping is very similar to the binary fragmentation described in [SKWW00]. It can be used to derive other mappings if that is desirable.

The experiments with Monet XML are based on the following set-up: we translated the queries of Section 5 into the Monet's low-level executable algebra MIL [BK99] using techniques such as described in [SABdB94]. Due to the highly fragmented binary data model of Monet XML, many MIL queries were quite lengthy in comparison to their rather compact XQuery representation. An extreme example is Query 14 where 136 binary relations have to be scanned for respective full-text string 'gold' which translated to nearly a thousand lines of MIL. We also took advantage of the Path Summary [SKWW00] that Monet XML provides to formulate queries, *i.e.*, we assumed it was available at query formulation time. The numbers presented in Figure 5 were measured on a warm database against the benchmark document at scaling factor 1.0; no DTD or schema information was used. The times were measured on a Silicon Graphics 1400 Server equipped with 1 GB of main memory, an 18 GB SCSI hard disk and

Query	RP	Monet XML
Query 1	1000	1217 ms
Query 2	1	2945 ms
Query 3	1	3891 ms
Query 4	1	153 ms
Query 5	1	161 ms
Query 6	1000	762 ms
Query 7	1000	2167 ms
Query 8	1	469 ms
Query 9	1	977 ms
Query 10	1	22292 ms
Query 11	1	8672 ms
Query 12	1	7464 ms
Query 13	1	25 min
Query 14	1	9223 ms
Query 15	100	762 ms
Query 16	100	2018 ms
Query 17	1000	250 ms
Query 18	100	7799 ms
Query 19	1	493 ms
Query 20	10	346 ms

Figure 5: Performance Impression: Monet XML at scaling factor 1.0

four Pentium III Katmai CPUs clocked at 550 MHz only one of which was used. The installed operating system was Redhat Linux 6.2, kernel version 2.2.14-12smp.

The benchmark document was scaled at factor 1.0 and generated without indentation. Bulk load of the 116.5 MB took 49.7 seconds. This time includes parsing the document, mapping it to relational tables and writing the tables to disk as a committed transaction. The mapping process of Monet XML works without DTDs and generates the database schema on the fly.

We now turn our attention to the running times as displayed in Figure 5 and give a preliminary interpretation. Note that for each query a repetition factor (RP) is listed to indicate how often the query was executed in the same environment. Query Q1 consists of a table scan and two look-ups. As Monet features a main-memory kernel, these operations are quite fast as they incur only little data volume. The query is mainly supposed to establish a performance baseline: The scan goes over 10000 tuples and is followed by two table look-ups (in other mapping schemes this probably varies).

Queries Q2 and Q3 are the first ones to provide surprises. It turned out that the parts of the query plans that compute the indices are quite complex TPCD-like aggregations: they require the computation of set-valued attributes to determine the bidder element with the least index with respect to the open auction ancestor. Therefore the complexity of the query plan is higher than the rather innocent looking XQuery representations of the queries might suggest. Consequently, running times are quite high. Q4 features the 'BEFORE' predicate which is implemented with a theta-join over the aforementioned attributes which store the global order.

We now come to Query Q5 which tries to quantify the cost of casting operations such as those necessary for the comparisons in Q3. For Monet XML, the cost of this coercion is rather low with respect to the relative complexity of Q3's query execution plan and given the 161 ms Q5 needs to execute. We note that all character data in the original document, including references, were stored as strings and cast at runtime to richer data types if that was necessary such as for Queries 3, 5, 11, 12, 18, 20. We did not apply any domain-specific knowledge, pre-calculation or caching of casting results.

Regular path expressions are the challenge presented by queries Q6 and Q7. Monet XML evaluates regular path expression by regular expression search on the *path summary*, which is Monet XML's equivalent of a schema and which contains all sequences of element and attribute tags that were encountered in the source document. Therefore, Q6 and Q7 are surprisingly fast; however, on systems without access to a path summary, which effectively plays the role of an index that comes for free, these queries can be much more expensive to execute. To complicate matters, Q7 actually looks for non-existing paths.

Queries Q8 and Q9 are implemented as joins. As reference attributes are stored as strings, chasing references in Monet XML amounts to executing equi-joins on strings. We were surprised how cheap Q8 and Q9 were in comparison to Q2 and Q3 as we would have deemed the individual elements similarly expensive.

The construction of complex query results is a point that really can put XML processing systems to the test as Q10 shows. The path expressions and join expression used in the query are kept simple so that the bulk of the work lies in the construction of the answer set which amount to more than 10 MB of (unindented) XML text. Monet XML spends about 90% of the total processing time just in constructing the answer set and comparatively time in extracting the variables from the original data.

Whereas Q10 produced massive amounts of output data, Q11 and Q12 test the ability to cope with large intermediate results by theta-joining potential buyers and items they might be of interest to them. The theta-join produces more than 12 million tuples. Q12 is also a challenge to the query optimizer to pick a good execution plan and allows insights into how the data volume influence query and output performance. For Monet XML the theta-join dominates the costs.

A very desirable but possibly expensive feature of XML stores is the ability to reconstruct a document that is at least isomorphic (if not equivalent) to the document that was bulk loaded. Q13 shows this. Monet XML needs 25 minutes to reconstruct a rather tiny fraction of the original documents. The basic action performed is a sort of the union of the order relations of the relevant sub-trees and a large number of lookups with unscoped OIDs. Obviously, Monet's performance is not impressive. We are experimenting with logical OIDs to improve performance.

Query Q14 continues to stress the textual side of XML by executing a full-text search. Monet XML does not provide full-text indices and has to scan a large part of the database. The scan for the word indeed makes up most of the execution time; the data volume involved in the remaining joins is negligible in terms of processing time.

Up to now, we have not tried to estimate the influence of long path expression versus short ones. This is exactly where Q15 and Q16 want to help. As the table displays, traversing a path itself is rather cheap but still longer paths take significantly more time than their short counterparts so that one should avoid unnecessary traversals.

Q17 again stresses the loose schema of most XML document by querying for non-existing data. The query execution plan computes the intersection of two sets. Queries Q18 and Q19 are primarily of interest to interest to establish the relative costs of function application and sorting operations when comparing two system architectures. Q17 to Q19 perform favorably as the binary fragmentation of the data model incurs only little data volume so that all computations easily fit into main memory. The aggregations of Q20 conclude the query set with a combination of three table scans and a set difference. Again, the data model allows for a plan with very little data volume.

In addition to the presented tests with relational backends, we are evaluating the benchmark on the persistent object store PDOM from GMD. A note on query formulation: we used XQuery [CFR$^+$01] as novices to formulate the benchmark queries. After a short phase of getting used to the language, we had no problems with the language itself. However, we would like to mention one point that would facilitate query formulation enormously. If a query processor was able to validation of path expressions online, *i.e.* tell the user whether a given sequence of tags actually exists in the database instance, it would often be of great help to users as quite regularly, simple typos in path names often evaluate to empty results. While the DBMS of course can't decide whether a given path expression contains a typo or not, it could well issue a warning if path expression contains non-existing tags. An approach like Query By Example could possibly lead to very helpful results.

We are also looking forward to a standardized query language being agreed upon which could tremendously facilitate the exchange of ideas between interested parties.

Finally, we want to propose some simple optimizations that we applied as a direct consequence of our analysis of the benchmark results to Monet XML to accelerate query execution without sacrificing generality. However, note that these optimizations are not included in our performance figures. If we are given schema information in the form of [LC00] or [W3C01] we often can simplify considerably the schema that in created on the fly in Monet XML to reduce storage requirements and query processing time; especially, inlining (collapsing) 1 : 1 relationships along path expression. The storage engine of Monet collects enough meta-information to make it possible to decide which paths can be collapsed without sacrificing the original semantics of the document. In other storage engines similar information can be collected during the creation of hash indexes. For achieving this, there hardly seems to be a need for novel technology. Another straight-forward optimization for chasing references is the materialization of destination nodes with logical OIDs; especially in the case of unscoped references we can avoid large scans. A third point is the replacement of generic strings with integers, where appropriate, helps to eliminate expensive casting operations; they same however can be achieved by integrating views or intermediate results into query optimization as it has been done the Monet Squeezer Project [MPK00].

We also think that the 'classic' gap between data-centric and document-centric documents [Bou00b] is still hard to come by in a generic way as the high costs of reconstructing parts of highly fragmented data show in Q13. We definitely consider the construction of specialized indexes and their costs a research topic.

7. Conclusion

In this paper, we presented the specifications developed in the XML Benchmarking Project, a set of queries and first lessons learned while executing the workload specification. The benchmark was designed top-down taking the standardization issues around XML as a starting point. The queries try to capture the essential primitives of XML processing in Data Management Systems such as path expression, querying for NULL values, full-text search, hierarchical data and ordered data and coercions between data types. We also discussed lessons learned by running the queries on a research vehicle, Monet XML. It demonstrated that there is no single best generic mapping from text documents to database instances and that the dividing line lies between queries that stress the textual nature of the document like order constraints or reconstruction queries and those queries that stress the data-centric perspective of associative retrievals.

Schema information is often useful in schema and query optimization, especially when it gives hints about document-centric characteristics such as unscoped references that do not go together well with databases in general. However it only alleviates the problem that there are often competing design rules a schema can adhere to depending on the application area.

We expect our work to continue and evolve in the future. Formally, important parts of complete application scenario are still missing: the lack of update specifications being the most prominent one. We hope that continuing standardization efforts will make it possible to incorporate updates in the future. After all, it is commonly agreed standards that benchmarks need to exist. Especially the XML world is a good example for such collaborative efforts.

References

[ABM$^+$90] T. L. Anderson, A. J. Berre, M. Mallison, H. H. Porter, and B. Schneider. The HyperModel Benchmark. In *Advances in Database Technology - EDBT'90. International Conference on Extending Database Technology, Venice, Italy, March 26-30, 1990, Proceedings*, volume 416 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 1990.

[BC00] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1):68–79, 2000.

[BK99] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *VLDB Journal*, 8(2):101–119, 1999.

[Bou00a] R. Bourett. XML Database Products. available at `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`, 2000.

[Bou00b] R. Bourret. XML and Databases. available at `http://www.rpbourret.com/xml/XMLAndDatabases.htm`, 2000.

[Boy01] J. Boyer. *Canonical XML Version 1.0*, 1 2001. available at `http://www.w3.org/TR/xml-c14n`.

[BP98] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1–7):107–117, 1998.

[BPSMM00] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). available at `http://www.w3.org/TR/REC-xml`, 2000.

[BR01] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *Proceedings of BTW2001*, Oldenburg, 2001. Springer, Berlin.

[CDN93] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The OO7 Benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 12–21. ACM Press, 1993.

[CDN$^+$97] M. J. Carey, D. J. DeWitt, J. F. Naughton, M. Asgarian, P. Brown, J. Gehrke, and D. Shah. The BUCKY Object-Relational Benchmark. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 135–146. ACM Press, 1997.

[CFR$^+$01] D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery: A Query Language for XML, February 2001. available at `http://www.w3.org/TR/xquery`.

[CRF00] D. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *International Workshop on the Web and Databases (WebDB)*, pages 53–62, 2000.

[FK99] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[FMS01] M. F. Fernandez, A. Morishima, and D. Suciu. Efficient Evaluation of XML Middleware Queries. In *Proceedings ACM SIGMOD International Conference on Management of Data*, 2001. to appear.

[Gra93] J. Gray. Database and Transaction Processing Performance Handbook. available at `http://www.benchmarkresources.com/handbook/contents.asp`, 1993.

[KM00] M. Klettke and H. Meyer. XML and Object-Relational Database Systems - Enhancing Structural Mappings Based on Statistics. In *International Workshop on the Web and Databases (In conjunction with ACM SIGMOD)*, pages 63–68, 2000.

[LC00] D. Lee and W. W. Chu. Comparative Analysis of Six XML Schema Languages. *SIGMOD Record*, 29(3):76–87, 2000.

[MPK00] S. Manegold, A. Pellenkoft, and M. L. Kersten. A Multi-Query Optimizer for Monet. In *Proceedings of the British National Conference on Databases (BNCOD)*, volume 1832 of *Lecture Notes in Computer Science (LNCS), Springer-Verlag*, pages 36–51, 2000.

[RPNK00] K. Ramasamy, J. M. Patel, J. F. Naughton, and R. Kaushik. Set Containment Joins: The Good, The Bad and The Ugly. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 351–362, 2000.

[SABdB94] H. J. Steenhagen, P. M. G. Apers, H. M. Blanken, and R. A. de By. From Nested-Loop to Join Queries in OODB. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile*, pages 618–629, 1994.

[Sah00] A. Sahuguet. Kweelt, the Making-of: Mistakes Made and Lessons Learned. Technical report, University of Pennsylvania, 2000.

[SKF$^+$00] A. Schmidt, M. Kersten, D. Florescu, M. Carey, I. Manolescu, and F. Waas. The XML Store Benchmark Project, 2000. `http://www.xmlperformance.org`.

[SKWW00] A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases (In conjunction with ACM SIGMOD)*, pages 47–52, Dallas, TX, USA, 2000.

[Slu98] D. R. Slutz. Massive Stochastic Testing of SQL. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases*, pages 618–622, 1998.

[STZ$^+$99] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB), September 7-10, 1999, Edinburgh, Scotland, UK*, pages 302–314, 1999.

[W3C01] W3C. W3C XML Schema. `http://www.w3.org/XML/Schema`, 2001.

Appendix: Some Snippets of the Benchmark Document

```
<?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> ...
          </parlist> ...
        </description>
        <shipping>Will ship internationally, See description for charges</shipping>
        <incategory category="category540"/>
        <incategory category="category418"/>
        <incategory category="category985"/> ...
      </item> ...
    </africa> ...
  </regions> ...
  <people>
    <person id="person0">
      <name>Kasidit Treweek</name>
      <emailaddress>mailto:Treweek@cohera.com</emailaddress>
      <phone>+44 (645) 96317722</phone>
      <homepage>http://www.cohera.com/˜Treweek</homepage>
      <creditcard>9941 9701 2489 4716</creditcard>
      <profile income="20186.59">
        <interest category="category251"/>
        <education>Graduate School</education>
        <business>No</business>
      </profile>
      <watches>
        <watch open_auction="open_auction8747"/>
        <watch open_auction="open_auction10177"/>
        <watch open_auction="open_auction7314"/>
        <watch open_auction="open_auction10552"/> ...
      </watches>
    </person> ...
  </people>...
</site>
```