06472 Abstracts Collection
XQuery Implementation Paradigms
— Dagstuhl Seminar —

Peter A. Boncz[1], Torsten Grust[2], Jérome Siméon[3] and Maurice van Keulen[4]

[1] CWI - Amsterdam, NL
boncz@cwi.nl
[2] TU München, DE
grust@in.tum.de
[3] IBM TJ Watson Research Center - Hawthorne, US
[4] Univ. of Twente, NL
m.vankeulen@utwente.nl

Abstract. From 19.11.2006 to 22.11.2006, the Dagstuhl Seminar 06472 "XQuery Implementation Paradigms" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. XQuery, XPath, XML, XQuery Benchmarking, XQuery Optimization, XQuery Interoperability, XQuery Hard Nut, Compilation, Benchmarking, XMark, Recursion, Database System, Functional Programming Language, Transaction Management, Distributed Query Processing

06472 Executive Summary – XQuery Implementation Paradigms

Only a couple of weeks after the participants of seminar No. 06472 met in Dagstuhl, the W3C published the Final Recommendation documents that fix the XQuery 1.0 syntax, data model, formal semantics, built-in function library and the interaction with the XML Schema Recommendations (see W3C's XQuery web site at http://www.w3.org/XML/Query/). With the language's standardization nearing its end and now finally in place, the many efforts to construct correct, complete, and efficient implementations of XQuery finally got rid of the hindering "moving target" syndrome. This Dagstuhl seminar on the different XQuery implementation paradigms that have emerged in the recent past, thus was as timely as it could have possibly been.

Keywords: XQuery, XML, Database, Compiler, Functional Programming, Transactions

Joint work of: Boncz, Peter A.; Grust, Torsten; Siméon, Jérome; van Keulen, Maurice

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2007/1022

An analysis of the current XQuery benchmarks

Loredana Afanasiev (University of Amsterdam, NL)

I will present an extensive survey of the currently publicly available XQuery benchmarks — XMach-1, XMark, X007, the Michigan benchmark, and XBench — from different perspectives.

We address three simple questions about these benchmarks: How are they used? What do they measure? What can one learn from using them? Our conclusions are based on an usage analysis, on an in-depth analysis of the benchmark queries, and on experiments run on four XQuery engines: Galax, SaxonB, Qizx/Open, and MonetDB/ XQuery.

We hope the lessons we have learned in this survey will serve as useful tips for future XQuery benchmarks.

Keywords: XQuery XPath benchmarking

See also: L. Afanasiev, and M. Marx. An analysis of the current XQuery benchmarks. In proceedings of ExpDB 2006. Chicago, Illinois, USA, June, 2006.

XPathMark: functional and performance tests for XPath

Massimo Franceschet (Università di Udine, I)

XPathMark consists of a Functional Test (XPath-FT) and a Performance Test (XPath-PT) for XPath 1.0 language. More specifically:

– XPath-FT contains several groups of queries each covering a different functional aspect of the language like navigational axes, filters, node tests, operators and functions. The queries are interpreted over a small educational document and each query is accompanied with the correct answer. The main goals of XPath-FT are testing completeness (which features of the language are supported?) and correctness (which features of the language are correctly implemented?) of an XML query processing system with respect to XPath 1.0.

– XPath-PT aims at investigating the performance of an XML query processor with respect to XPath 1.0 in terms of time spent to execute a query assuming that the processor correctly implements the XPath language. Queries of the test are divided into groups according to the intrinsic computational complexity of the corresponding evaluation problem. They are defined to challenge both data and query scalability and include some instances of transitive closure of path expressions.

More information about XPathMark is available at `http://www.dimi.uniud.it/~francesc/xpathmark`. In the talk I will discuss XPathMark and argue the central role that XPath should have in a mature XQuery benchmark.

Keywords: XML, XPath, benchmark

XPathMark: Functional and Performance Tests for XPath

Massimo Franceschet (Università di Udine, I)

We present a major revision of the XPath benchmark known as XPathMark. The new version splits into a functional test over a small educational document and a more elaborated performance test over XMark documents. We conclude by sharing with the reader our experience on running XPathMark on some popular XSLT/XQuery processors.

Keywords: XML, XPath, Benchmarking

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/892

Push or Pull (does it make a difference)

Michael Kay (Saxonica - Reading, GB)

We all know that XQuery implementations should use streaming or pipelining internally as much as possible. But I'd like to discuss the question of when it's better to use a push pipeline, and when it's better to use pull. Relational databases traditionally use a pull approach, whereas push pipelines are often seen in XML applications. XSLT 1.0 processors (as far as one can tell) generally use pull to access the input document, and push to write the output document - partly because that's the way the spec was written. In XQuery (and to a lesser extent in XSLT 2.0) input and output aren't so clearly separated, as the language is more composable and expressions can be arbitrarily nested. Nevertheless, I think there is still some value in making this separation internally: but I'm keen to hear the views of others.

Keywords: XQuery implementation pipeline streaming

Snapshot-based Concurrency Control for XML

Bettina Kemme (McGill University - Montreal, CA)

Currently, only few XML data management systems support concurrent access to an XML document, and if they do, they typically apply variations of hierarchical locking to handle XML's nested structure. However, advanced query processing techniques use a wide range of indexes and allow for arbitrary navigation through the XML documents, making lock acquisition complex and potentially leading to high blocking rates. In this paper, we suggest two concurrency control protocols that avoid any read locks by providing transactions a committed snapshot of the data. OptiX enhances traditional optimistic concurrency control to work on XML while SnaX offers snapshot isolation as provided by Oracle and PostgreSQL. We evaluate the performance of these two protocols on XML documents of different structure and on the XMark benchmark (which was extended by several update operations). We also compared our solution to locking-based protocols. The results show that snapshot based concurrency control is a viable alternative. In particular Snax shows very good performance due to the fact that it does not need to keep track of read operations at all but only considers write/write conflicts. which currently supports only queries.

Keywords: XML, snapshot-based concurrency control, snapshot isolation

Joint work of: Kemme, Bettina; Sardar, Zeeshan

See also: Int. Conference on Data Engineering, 2006

XML Cardinality Estimation - A Request for Research

Norman May (Universität Mannheim, D)

To evaluate different plan alternatives for a query execution plan one has to compare these plans based on costs. Cardinality Estimates are the most important parameter for cost functions. To date, these estimates can be computed precisely only for a small fragment of XPath. This leaves many open issues for future research.

Keywords: Query optimization, XPath, XQuery, cost functions, cardinality estimation

Joint work of: May, Norman; Moerkotte, Guido

Put a tree pattern in your algebra

Philippe Michiels (University of Antwerp, B)

To address the needs of data intensive XML applications, a number of efficient tree pattern algorithms have been proposed.

Still, most XQuery compilers do not support those algorithms. This is due in part to the lack of support for tree patterns in XML algebras, but also because deciding which part of a query plan should be evaluated as a tree pattern is a hard problem. In this paper, we extend a tuple algebra for XQuery with a tree pattern operator, and present rewritings suitable to introduce that operator in query plans. We demonstrate the robustness of the proposed rewritings under syntactic variations commonly found in queries. The proposed tree pattern operator can be implemented using popular algorithms such as Twig joins and Staircase joins. Our experiments yield useful information to decide which algorithm should be used in a given plan.

Keywords: Tree patterns xquery compilation xpath

Joint work of: Michiels, Philippe; Mihăilă, George; Siméon, Jérôme; Hidders, Jan; Vercammen, Roel

Full Paper:
http://www.adrem.ua.ac.be/~michiels

See also: To appear at ICDE 2007

An Application-oriented XML Transaction Processing Benchmark

Matthias Nicola (IBM Silicon Valley Lab., USA)

XML database functionality has been emerging in “XML-only” databases as well as in the major relational database products. Yet, there is no industry standard XML database benchmark to evaluate alternative implementations. The research community has proposed several benchmarks which are all useful in their respective scope, such as evaluating XQuery processors. However, they do not aim to evaluate a database system in its entirety and do not represent all relevant characteristics of a real-world XML application. Often they only define read-only single-user tests on a single XML document. We propose an application-oriented and domain-specific benchmark that exercises all aspects of XML databases, including storage, indexing, logging, transaction processing, and concurrency control. Based on our analysis of real-world XML applications, the benchmark simulates a financial multi-user workload with XML data conforming to the FIXML industry standard. I’ll describe the benchmark and present early measurement results.

Key features of our benchmark:

– multi-user read/write workload
– scalable to billions of (small) XML documents
– supports XQuery and SQL/XML
– includes XML updates based on the XQuery Update Facility
– high concurrency transaction processing

– scalable and parallelized data generation
– implementation of a very flexible workload driver
– use of multiple namespaces, schema validation, joins across document types, etc.

In addition to feature-specific micro-benchmarks that focus on core XML processing operations, application-level workloads are important to assess the performance of an entire system as a whole. The research community has proposed various XQuery and XML database benchmarks, e.g. XMach-1, XMark, XPathMark, XOO7, XBench, MBench, and MemBeR. Some are predominantly application oriented, such as XMach-1 and XBench, while others are designed as abstract micro-benchmarks, e.g. MBench and MemBeR. XMark, XPathMark and X007 can be viewed as a blend because their data and queries represent a fictitious application scenario but they also try to exercise most relevant aspects of the XQuery and XPath languages.

Except for XMach-1, all of these benchmarks focus predominantly on XQuery processing rather than on evaluating a complete database system. Indeed, most of the benchmarks define queries only despite real-world requirements for insert, update and delete operations. Many of them are also designed as single-user tests on a single large XML document. Such tests can be very valuable to investigate design alternatives and optimizations in an XQuery processing engine. However, these benchmarks are not sufficient to stress all performance-relevant components of a full-fledged XML database system.

We find that none of the existing benchmarks entirely matches our goal to comprehensively represent a realistic application scenario. We carefully looked at real-world XML applications, and devised a list of requirements for an application-oriented XML database benchmark.

Supporting financial companies in their adoption of XML helped us understand their data and processing characteristics. For example, we have worked with multiple brokerage and securities processing companies on storing and querying FpML, FIXML and other financial data. We decided to design a benchmark that is relevant to this application domain. It simulates an online trading scenario and uses FIXML to model some of its data. We will make our benchmark implementation available as open source (very soon). This does not remove the need for XML micro-benchmarking which has a different and equally important purpose.

Keywords: XQuery, SQL/XML, XML, XML database benchmark, micro-benchmark, application-oriented benchmark

Searching for XQuery

Paul Pedersen (FLWOR Foundation - Palo Alto, USA)

Search engine indexs can be effective for optimizing XQuery applications that need large repositories of relatively static documents.

Keywords: XQuery, search, repositories, optimization

Functional Optimizations of XQuery using Higher Order Rewriting

Kristoffer Rose (IBM TJ Watson Research Center, USA)

We show how higher-order rewriting can be used in practice to describe and implement XQuery optimizations as commonly used for functional programming.

Keywords: XQuery, optimizations, higher-order rewriting, combinatory reduction systems (CRS)

Demonstration of Virtual XML for File Systems

Kristoffer Rose (IBM TJ Watson Research Center, USA)

The “Virtual XML Garden” combines an XQuery engine with “functions” that create XML views on foreign data structures. In this demonstration, I will show how this can be used to navigate a local file system with XQuery.

Keywords: XQuery, Virtual XML, file system navigation

Further Material:
http://www.research.ibm.com/virtualxml

File Systems are Obsolete (!?)

Marc H. Scholl (Universität Konstanz, D)

Traditionally, database systems (DBMSs) sit on top of operating systems (OSs), largely bypassing the data management functionality of the OS file system, though. The reasons for this are performance-oriented: since the DBMS “knows” more about a transaction’s semantics (e.g., which SQL statement is currently being executed?), better policy decisions can be made inside the DBMS than by the OS.

On the other hand, quite a few problems (and solutions!) for efficient process management, synchronization, data allocation and access, recovery, and so on, are shared between DBMSs and OS file systems. In fact, Microsoft’s WinFS, which had been planned for the Vista version of Windows originally, was and attempt at turning this traditional software stack upside down: use a DBMS for OS file management. Also, Jim Gray claimed exactly this to be the right to do.

We present a few motivations, give examples where typical OS tools already use database systems instead of files, and highlight the potential that the ubiqituous XML file format offers for database support for OS file systems. Since XML database systems are hierarchy-aware, they lend themselves towards OS

file system support much easier then relational systems with their strict tabular data format. XQuery's search functionality (including fulltext retrieval) would automatically be offered at the file system interface, making desktop search very powerful, especially since search criteria could cross the border between file system attribute considitions (such as, file names, directory paths, modifications dates) and file content (such as, meta data, mpeg7 descriptions, or actual file content) in a single query.

A prototypical XML DBMS, BaseX, developed at U Konstanz has been used as a testbed to assess the viability of such an approach, namely to check whether the DBMS performance is sufficient as a file system replacement. We can report on our results and experiences for interactive use of the BaseX-DBMS "file sytem", which is more than sufficient. In fact, we will show a graphical file browser demo, that uses an advanced visual representation of disk space usage. Further work will implement BaseX as a VFS (virtual file system) layer, such that also low-level file access primitives can be evaluated, since these pose stronger performance requirements than the interactive interface.

The addition of imperative lanaguage features to XQuery has benefits but also introduces new riscs, especially when it is compared to the usual approach where XQuery is embedded in an imperative programming language. The benefits seem clear: a single powerful integrated programming environment with no impedance mismatch in which all data manipulation can be done. The riscs are that the declarative nature of XQuery is compromised and that users will often use the easier-to-progam imperative features instead of the easier-to-optimize declarative features of the language. In the embedded case there is a clear distinction between these features and it might be easier for programmers to maintain a coding discipline where as much as possible is done in the declarative data manipulation sublanguage. On the other it could be argued that in the integrated approach it will be easier to transform nondeclarative code to more declarative code, either by hand, supported by the system or entirely automatically. The group did not come to a definitive conclusion on whether the benefits outweigh the riscs or not.

Keywords: Database systems, operating systems, file systems, desktop search

An Algebraic Compiler for An Expressive XQuery Extension

Jérome Siméon (IBM TJ Watson Research Center - Hawthorne, USA)

As XQuery nears standardization, professional developers are eager to apply XQuery to complex XML applications. We present an algebra and optimization techniques that are suitable for building an XQuery processor that is both complete and efficient. We present the compilation rules for the complete XQuery language into that algebra and present new logical optimizations that are more effective on complex queries than existing techniques. We also propose novel

join and group-by algorithms that account for all of XQuery's complex semantics, including type assertions, existential quantification, and type promotion. In addition, we show how that algebra can be used for both index XML documents and streaming documents. The algebra and optimizations are implemented in our Galax XQuery engine.

Keywords: XQuery, XML, Optimization, Compilation, Algebra

Distributed XQuery

Jérome Siméon (IBM TJ Watson Research Center - Hawthorne, USA)

The goal of the DXQ project is to support development of reliable, extensible, and efficient distributed resource-management protocols. Our strategy to meet these requirements is to provide a high-level, distributed, and optimizable query language for implementing distributed resource-management protocols. By using a high-level language, a protocol's semantics is transparent, not hidden, in the implementation, which supports the reliability requirement. In addition, the implementation is optimizable by general query optimization techniques. Automating optimization supports the efficiency requirement and permits implementors to focus more on functionality and less on performance.

Keywords: XQuery, Distribution, Network application, Web Services

Further Material:
http://db.ucsd.edu/dxq/

A Purely Relational Approach to XQuery

Jens Teubner (TU München, D)

Out of the different implementation paradigms, the Pathfinder XQuery compiler is a representative of the relational "camp".

The key insight to this compilation procedure is the functional style of iteration in the XQuery language, which allows the evaluation of FLWOR expression in a parallel, bulk-oriented fashion. To this end, Pathfinder translates incoming XQuery expressions into purely relational query plans.

This design has several intersting implications to current fields of research within the joint project. To name some of the most intersting ones,

- Pathfinder's relational ("loop-lifted") sequence encoding naturally embraces features such as dynamic typing or full-text retrieval,
- in a distributed XQuery environment, we are able to send data for multiple XQuery iterations within a single network transfer,
- to optimize relational query plans, we can rely on established techniques from the relational domain; similar arguments hold for the estimation of (intermediate) result sizes in relational plans,

– though the compliant implementation of order constraints is a significant challenge in an inherently unordered relational environment, the system will particularly benefit whenever order is not significant, and
– since relational algebra is quite restricted with respect to recursion, the handling of recursive XQuery functions turns out to be quite hard (though Pathfinder is able to handle an important subset of recursive queries).

The Pathfinder XQuery compiler is part of the MonetDB/XQuery distribution. Its upcoming release includes an implementation of the XQuery Update Facility and support for mulit-hierarchical XML (StandOff annotations). Current work includes the support for alternative back-ends (SQL:1999, X100, MonetDB 5, Idefix).

Keywords: Relational XQuery, Pathfinder, XQuery Compilation

Joint work of: Teubner, Jens; Afanasiev, Loredana; Boncz, Peter; Grust, Torsten; van Keulen, Maurice; Manegold, Stefan; Rittinger, Jan; Zhang Ying

Further Material:
http://www.pathfinder-xquery.org/

Loop-lifted XQuery RPC with Deterministic Updates

Ying Zhang (CWI - Amsterdam, NL)

XRPC is a minimal XQuery extension that enables distributed query execution, combining the Remote Procedure Call (RPC) paradigm with the existing concept of XQuery functions. By calling out of a `for`-loop to multiple destinations, and by calling functions that themselves perform XRPC calls, complex P2P communication patterns can be achieved. We further propose the use of SOAP as the protocol for XRPC, which allows seamless integration with web services and Service Oriented Architectures (SOA).

XRPC is implemented in the open source MonetDB/XQuery system. We show that the technique of *loop-lifting*, that executes all expressions inside a `for`-loop in a single bulk operator – pervasively applied in MonetDB/XQuery to obtain efficient relational query plans – also benefits XRPC.

Loop-lifting enables us to send *bulk* RPC requests, dramatically reducing the number of SOAP messages, and thus the performance impact of network latency.

The XRPC extension is orthogonal to all XQuery language features, including the XQuery Update Facility (XUF). The XUF W3C Draft proposal does not define the order in which multiple update actions to the same node must be applied. We instead choose to make this order deterministic, and show how distributed updates can be made deterministic using a small protocol extension.

Keywords: Distributed query processing, XQuery, Loop-lifting, XML, RPC

Joint work of: Zhang, Ying; Boncz, Peter A.

XQuery Hard Nut: Recursive Functions in XQuery

Massimo Franceschet (Università di Udine, I)

An implementation of transitive closure of a location path via recursion functions in XQuery . To be interpreted on XMark.

```
(: Categories that are reachable from a given category
   through an arbitrary path in the category graph :)

declare namespace fun = 'have.more.fun';

declare function fun:closure($input as node()*,
                             $result as node()*) as node()*
{
   let $current := $input/idref(.)[name() = "from"]/../@to
   let $new := $current except $result
   let $all := ($result,$new)

   return
      if(exists($new))
      then ($new, fun:closure($new,$all))
      else ()
};

doc("auction.xml")//category[@id="category0"]
                             /@id/fun:closure(.,())/id(.)/name
```

Keywords: XQuery optimization, Hard Nut, Recursion

XQuery Hard Nut: Memoization (and Other FP Techniques) in XQuery

Torsten Grust (TU München, D)

The XQuery semantics are—largely—functional. Does this open the door for optimizations that have been developed in the domain of compilers for functional programming languages?

One such "functional gem" could be the *memoization* of expression values at runtime. The savings could be significant. In the simple XQuery expression below

```
for $x in (1,2,2)
  for $y in (3,4,3)
    return e($x,$y)
```

the nested FLWOR will lead to a nine-fold evaluation of expression e (in which variables `$x` and `$y` occur free). Alas, only four distinct variable binding pairs will be generated by the nested `for`-constructs. Can we evaluate (the potentially costly) expression e only four times and then later "compensate" for this optimization to ensure that the resulting sequence will be of length nine?

Of course, one challenge is the "largely" above... If e contains side-effecting constructors, for example, memoization (re-using the results of e) might be unsound or at least complicated to achieve. Interestingly, memoization *should* go well together with the current formulation of the XQuery Update Draft, if the compensation action is performed in a sensible fashion.

Feedback at the Seminar

Guido (Moerkotte) suggested that Goetz Graefe's BTW 2003 paper *Executing Nested Queries* contains a useful discussion of memoization techniques for correlated nested SQL queries. I had a look at this work and it is certainly highly relevant here (see Section 4.1, *Caching results of the inner query*). This is especially true for its discussion of the trade-offs of memoization: when does it pay off to instrument the compiled query with the necessary code and data structures to realize memoization (this is related to the evaluation cost of e)? Goetz' work, however, takes place in a query-only SQL context—which is purely functional and thus avoids the hard core of this nut, I believe.

Keywords: XQuery, Hard Nut, Memoization, Functional Programming Languages

XQuery Hard Nut: Recursive query with Join

Jérome Siméon (IBM TJ Watson Research Center - Hawthorne, USA)

The proposed XQuery "hard nut" presents an important use case where a join is performed over two documents. The first document is a hierarchical structure, and the second document a flat structure. The query builds a new hierarchy parallel to the hierarchy of the first document, but inludes information from the other document. This is a hard query as existing join optimization techniques do not operate accross function call boundaries.

The example query provided illustrates that case on data following the schemas of the XQuery Test Suite.

```
declare namespace res
        = "http://www.w3.org/2005/02/query-test-XQTSResult";
declare namespace cat
        = "http://www.w3.org/2005/02/query-test-XQTSCatalog";

declare variable $cat := doc("/mnt/e/Nuts/XQTSCatalog.xml");
declare variable $res := doc("/mnt/e/Nuts/testresults.xml");
declare variable $tests := $res//res:testcase;

declare function local:format($group) {
  <group name="{$group/@name}">
    { fn:count(
        for $x in $group/cat:test-case
        for $y in $tests
        where $x/@name = $y/@name
        return $x) }
    { local:format($group/cat:test-group) }
  </group>
};

local:format($cat/cat:test-suite/cat:test-group[1]$)
```

Keywords: XQuery, Recursion, Join

XQuery Hard Nut: Streaming better than sliced bread?

Jérome Siméon (IBM TJ Watson Research Center - Hawthorne, USA)

The proposed XQuery "hard nut" is an example of a simple recursive transformation over a data set corresponding to a genealogy tree. The query itself has been proposed by Alain Frisch. When he compares the performances of a a streaming implementation against standard XQuery implementations, he shows that streaming outperforms (by far!) all other existing approaches. More information about the use case can be found at: http://yquem.inria.fr/~frisch/xstream/bench.html.

```
declare function local:split($x as node()) as node() {
 let $name := $x/name/text() return
 let $sons :=
   for $p in $x/children/person[@gender="M"]
   return local:split($p)
 return
 let $daughters :=
   for $p in $x/children/person[@gender="F"]
   return local:split($p)
 return
   if ($x/@gender = "M") then
     <man name="{$name}">
       <sons>{$sons}</sons>
       <daughters>{$daughters}</daughters>
```

```
      </man>
    else
      <woman name="{$name}">
        <sons>{$sons}</sons>
        <daughters>{$daughters}</daughters>
      </woman>
}
;

<doc>{for $p in ./doc/person return local:split($p)}</doc>
```

Keywords: XQuery, recursion, streaming

XQuery Hard Nut: Compiler's robustness.

Jérome Siméon (IBM TJ Watson Research Center - Hawthorne, USA)

The proposed XQuery "hard nut" talks about the current difficulty of XQuery compilers to be robust in the face of very basic syntactic changes. The observation is that very simple changes in the query seem to affect greatly the kind of optimization that is applied to that query. The set of queries attached is the form of a challenge. All the queries proposed in that file are equivalent and are obtained through very simple rewritings. How many of those queries your compilers will successfully detect as simple nested queries which can be decorrelated into a join-group-by plan?

```
(: Q1 :)
(: Standard nested query :)

for $x in $auction//open_auctions
let $a :=
  count(
    for $y in $auction/site/people/person
    where $x/@id = $y/bid/@id
    return $y/name
  )
return
  <auc id="{$x/@id}">
    { $a }
  </auc>

(: Q2 :)
(: Query nested in the return clause :)
```

```
for $x in $auction//open_auctions
return
  <auc id="{$x/@id}">
    { count(
        for $y in $auction/site/people/person
        where $x/@id = $y/bid/@id
        return $y/name) }
  </auc>

(: Q3 :)
(: Query nested in the return clause, with XPath expanded
   in FLWORs :)

for $x in
  for $x1 in $auction return descendant::open_auctions
return
  <auc id="{$x/@id}">
    { count(
        for $y in
          for $x4 in
            for $x3 in
              for $x2 in $auction return $x2/child::site
            return $x3/child::people
          return $x4/child::person
        where $x/@id = $y/bid/@id
        return $y/name) }
  </auc>

(: Q4 :)
(: Query nested in the return clause, with XPath expanded
   in FLWORs, where clause expanded in conditional
   expression :)

for $x in
  for $x1 in $auction return descendant::open_auctions
return
  <auc id="{$x/@id}">
    { count(
        for $y in
          for $x4 in
            for $x3 in
              for $x2 in $auction return $x2/child::site
            return $x3/child::people
          return $x4/child::person
        if ($x/@id = $y/bid/@id) then $y/name else ()) }
  </auc>

(: Q5 :)
(: Query nested in the return clause, with XPath expanded
   in FLWORs, where clause expanded in conditional expression,
   and a function used for the join condition. :)

declare function cond($x1,$x2) {
  $x1/@id = $x2/bid/@id
};

for $x in
  for $x1 in $auction return descendant::open_auctions
return
  <auc id="{$x/@id}">
    { count(
        for $y in
          for $x4 in
            for $x3 in
              for $x2 in $auction return $x2/child::site
            return $x3/child::people
          return $x4/child::person
        if (cond($x,$y)) then $y/name else ()) }
  </auc>

(: Q6 :)

declare function cond($x1,$x2) {
  $x1/@id = $x2/bid/@id
};

for $x in
  for $x1 in $auction return descendant::open_auctions
return
  <auc id="{$x/@id}">
    { count(
        for $y in
          for $x4 in
            for $x3 in
              for $x2 in $auction return $x2/child::site
            return $x3/child::people
          return $x4/child::person
        if (cond($x,$y)) then $y/name else ()) }
  </auc>
```

Keywords: XQuery, Compilation, Robustness, Join Detection

XQuery Hard Nut: Eliding intermediary structures

Maurice van Keulen (University of Twente, NL)

I observed in a project where I use XQuery quite extensively, that I ended up programming functions that would construct intermediate XML structures containing parts of a document, and that other functions would "break down"' these intermediate structures again to construct the final result. For example, one function would construct a list of all combinations of children of two elements, a subsequent function would filter the combinations, and from that a final result is constructed. Observe that construction of all combinations means heavy duplication (due to copy semantics of element construction). Also observe that the intermediary structures are not really necessary: element construction and subsequent XPath navigation could cancel eachother out. How to find the possible cancellations is a hard nut to crack. Avoiding unnecessary duplication is another.
Some meaningless example:

```
declare function allCombinations($wlists as element(wlist)*)
                                 as element(wlist)*
{
   let $cnt := count($wlists)
   return
      if ($cnt eq 0) then ()
      else if ($cnt eq 1)
         then for $x in $wlists/child::w
              return <wlist>{$x}</wlist>
         else let $y := exactly-one($wlists[1])
                 ,$ys := $wlists[position() gt 1]
                 ,$as := allCombinations($ys)
              return
                 for $x in $y/child::w, $a in $as
                 return <wlist>{$x}{$a/child::w}</wlist>
};

declare function filter($wlists as element(wlist)*)
                        as element(wlist)*
{
   for $wl in $wlists
   where count($wl/*[1]/foo) eq 1
   return $wl
};
```

```
declare function result($wlists as element(wlist)*)
{
   for $wl in $wlists
   let $f := exactly-one($wl/*[1]/foo)
   return <foo>{$f/*}{$wl/*[position() gt 1]/*}</foo>
};

let $exA := (<wlist>
                <w><foo><bar/></foo></w><w><baz/></w>
             </wlist>,
             <wlist>
                <w><a><b/></a></w>
                <w><c/></w>
                <w><p><q><r/></q></p></w>
             </wlist>)
return
   result(filter(allCombinations($exA)))
```

Conclusions

An important observation that the example in the hard nut actually contained three hard nuts:

1. matching element construction to a subsequent navigation,
2. replication of structures because of the copying semantics of element construction, and
3. optimization across the boundaries of functions.

There were also more situations where hard nut (1) occurs often: when querying a view. A situation that currently occurs very often is that people have a relational table, they define an XML-view over it, and then query the XML-view. This is currently very inefficient. A possible source of techniques would be "deforestation" from the functional programming field.

Regarding hard nut (2), you could reformulate the XQuery so that only id's are stored in the intermediary functions. People feel that the programmer need not be burdened with this. The source of the example was actually a desire to construct a nested sequence. Many believe that we should not extend XQuery with nested sequences, but rather allow programmers to annote expressions with certain properties, e.g., node identity will not be observed for the elements constructed. In that case, the engine could choose an execution approach that shares structures instead of copying them. I observed in a project where I use XQuery quite extensively, that I ended up programming functions that would construct intermediate XML structures containing parts of a document, and that other functions would "break down" these intermediate structures again to construct the final result. For example, one function would construct a list of all combinations of children of two elements, a subsequent function would filter the combinations, and from that a final result is constructed.

Observe that construction of all combinations means heavy duplication (due to copy semantics of element construction). Also observe that the intermediary structures are not really necessary: element construction and subsequent XPath navigation could cancel eachother out. How to find the possible cancellations is a hard nut to crack. Avoiding unnecessary duplication is another.

Keywords: XQuery optimization, Hard Nut, Intermediary structures

XQuery Hard Nut: Recursion

Maurice van Keulen (University of Twente, NL)

XQuery allows recursion. For some approach (e.g., algebraic ones), it is hard to suport this fully. Some examples of functions I recently wrote:

```
declare function product($seq as xs:decimal*) as xs:decimal
{
   if (count($seq) gt 0)
   then exactly-one($seq[1]) * product($seq[position() gt 1])
   else 1.0
};

declare function allCombinations($wlists as element(wlist)*)
                                 as element(wlist)*
{
   let $cnt := count($wlists)
   return
      if ($cnt eq 0) then ()
      else if ($cnt eq 1)
           then for $x in $wlists/child::w return <wlist>{$x}</wlist>
           else let $y := exactly-one($wlists[1])
                   ,$ys := $wlists[position() gt 1]
                   ,$as := allCombinations($ys)
                return
                   for $x in $y/child::w, $a in $as
                   return
                      <wlist>{$x}{$a/child::w}</wlist>
};
```

Recursion usually “follows” the XML hierarchy (by descending into an XML tree), the order in sequences, some diminishing computation, or a combination thereof. Mutual recursion, etc. all are possible in XQuery.

Conclusions

The general agreement is that recursion is hard. People have been researching efficient evaluation of recursive programs for decades. Hence, we should not have the ambition to develop an engine that is able to recognize and optimize all forms of recursion. Rather, we should be able to handle certain forms of recursion, i.e., have techniques to efficiently evaluate certain forms of recursion and require of the programmer to formulate his queries using these forms. Magic sets and counting are promising techniques.

Keywords: XQuery optimization, Hard Nut, Recursion

Breakout Session: XQuery and Transaction Management

Bettina Kemme (McGill University - Montreal, CA)

In this session, we discussed the challenges XML poses to efficient transaction management.

We looked both at the programming interface and the internals of transaction management.

Questions posed that we posed:

- What kind of transaction model is appropriate for XML applications?
- Where should transaction boundaries be set?
- What isolation levels are appropriate for XML applications?
- How can the ACID properties be implemented in an XML engine?
- Do the techniques used to guarantee transactional properties differ from those used in relational systems?
- How can distributed transactions be handled?

Keywords: Transaction management; atomicity, concurrency control, isolation levels, consistency, distributed transactions, transaction boundaries

Joint work of: Kemme, Bettina; Boncz, Peter; Borkar, Vinayak; Helmer, Sven; Zhang, Ying

Breakout Session: Benchmarking

Norman May (Universität Mannheim, D)

The XMark benchmark has been used by the XML community for benchmarking their systems. The main reasons for this are its simplicity in generating data and the queries included in the benchmark. However, the process of running the benchmark is not well-defined. Thus the interpretation of benchmark results is difficult. Additionally, the set of queries does not reflect the state of the XQuery specification and queries formulated by users (see slides for more details).

Future benchmarks should preserve the strengths of XMark while overcoming its shortcomings. To this end we need to structure the process of defining both application or micro benchmarks. We propose a bottom-up approach: (1) define relevant data sets, (2) define statement sets (i.e. queries and updates), and (3) define the workload as a combination of statements.

To organize this process, we propose to use the MemBer repository to manage (1) datasets, (2) queries, (3) workloads, and (4) discussions. Moreover, the community can participate in all these tasks.

DISCUSSION

The participants agreed that a new benchmark must be a shared effort of the XML community. For this reason, we propose to assign the "Dagstuhl-label" to the new benchmark and, hence, call it DMark. Only this way, a wide adoption of the benchmark is possible.

Keywords: XQuery, benchmark, application benchmark, micro benchmark, XMark

Breakout Session: XQuery interoperability

Philippe Michiels (University of Antwerp, B)

Minutes from the break-out session on XQuery Interoperability

Keywords: XQuery interoperability

Breakout Session: XQuery and Applications

Marc H. Scholl (Universität Konstanz, D)

This breakout disussion focused on the functionality of XQuery from two perspectives: 1. Application view: The functionality is very rich (maybe too rich), making it difficult to understand all subtleties. On the other hand, there may even be the need to add some selected additional functionality (such as some imperative language features). 2. Implementation view: The functionality is very rich and some parts of the semantics make an (efficient!) implementation difficult.

When we look back at the history of SQL or, in particular, at OQL, we can observe, that rich languages have been intensively studied in research and discussed in standardization efforts for long periods, but finally, typical application make use of only a tiny fraction of the overall functionality. SQL's recursion feature is rarely used, even worse: still many applications don't use complex join queries, rather applications often program this functionality outside the DBMS in the application code. OQL is virtually dead, as is the whole object-oriented database systems area (with some exceptions in niche markets).

Possibly, one lesson from this could be to try to identify a small and simple subset of XQuery as a language kernel and package additional functionality as predefined sets of optional features. This could be proposed as an item on an XQuery 2.0 wish list.

Obviously, this "keep it simple" attitude towards XQuery is motivated by looking at XQuery as a Database Language, where the focus is on efficient execution of a small set of salient, performance-critical base operators for fast and scalable mass data processing. If XQuery is considered a full-flegded Functional Programming Language, however, completeness and richness play a more important role. There is an obvious tension between the two views, one that might potentially be resolved or at least ameliorated by a modularized definition of XQuery 2.0 with a small kernel leaning itself towards DBMS-based execution and extended packages offering computational completeness in a seamless "host language" for database operations. The discussion could not, obviously, identify this small "kernel" vs. the "extension packages". Two sample features played a role in the discussion: recursion and (imperative) variables with assignments (see XQueryP).

Keywords: XQuery, database programming language, computational completeness, optimization, language design