

Summary for Quilt

Michael Orlov, orlovm@cs.bgu.ac.il

November 2000

Abstract

Summary for *Quilt*, an XML Query Language.

1 Introduction

In recent past, several query languages for XML have been developed, each well suited for some particular applications, and not so well suited for others. *Quilt*[1] comes in an attempt to provide a general purpose query/construct tool for XML, which unifies the once separate world of *documents* and *relational databases*.

To begin with, let us consider consider a simple XML holding book contents, such as the one in App. A.1. The DTD which is included in this XML is not necessary for running the query, but is useful for constructing the query itself. Lets say that we want to list all the titles which contain the substring “data”:

```
/* List all the titles with the word "data" */
<queryresult>
  FOR $title IN document("book.xml")//title
  WHERE contains($title,"Data")
  RETURN
    <datatitle>
      $title/text()
    </datatitle>
</queryresult>
```

This query results in the output shown in App. A.2. Here we see, that the basic query block consists of FOR clause (which binds variables to XML nodes), followed by WHERE clause (that prunes some of the variables tuples), terminated with RETURN clause (that instantiates an XML node).

The string which follows an IN keyword is a *path expression*, which produces an ordered forest of all the `title` nodes in the document.

In the query above, we also wrapped the result in `queryresult`, so that the output will be a valid XML document. This also summarizes the features that make *Quilt* a powerful XML query language:

1. Path expressions for navigating in hierarchical documents (XPath and XQL);
2. Notion of binding variables (XML-QL);
3. FLWR (For-Let-Where-Return) restructuring data patterns (SQL);

4. Functional language — full expressions nesting (OQL). *This one is not evident in the basic example.*

2 Path Expressions

A Quilt path expression is based on XML Path Language, XPath[2], a syntax for addressing parts of XML documents. Such expression consists of a series of *steps*, where each step represents movement through a document, and each step can also apply a predicate to prune some nodes.

An example:

```
document("book.xml")//figure/[@width=400]
```

This path expression returns the ordered forest of all `figure` nodes in `book.xml` document, whose `width` attribute equals 400.

In general, *unabbreviated* location step in path expression has the following format: `axis::nodetest([predicate])*`, where

1. `axis` specifies the tree relationship between the nodes selected by the location step and the context node (current position in the tree). For example, `axis` can be `child`, `descendant`, `parent`, `preceding-sibling`, and even `self`, `attribute` or `namespace`;
2. `nodetest` is basically the element (or attribute) name against which we want to match, but can be, for example, `node()`, specifying any node element, or `text()`, specifying the text inside the element (the text node);
3. `predicates` are additional predicates which filter the node-set that results from the path; `predicate` can be any expression resulting in a number or a boolean (when the result is number, it is equivalent to the boolean expression `position()=n`).

For instance, the unabbreviated path for the example above is

```
document("book.xml")/descendant-or-self::node()  
/child::figure[attribute::width=400]
```

Absolute paths can also be specified, by prepending the path with `/`. Also, parentheses can be used for precedence. For example, `//section[1]` means any node that is a first `section` node of its parent, and `(//section)[1]` means the first `section` node in the document.

Before going on explaining *abbreviated paths*, let us take a look on a query that uses only unabbreviated location steps:

```
<result>  
FOR $sec IN document("book.xml")/descendant-or-self::section  
RETURN  
  IF EXISTS($sec/attribute::difficulty)  
  THEN  
    <dif id=$sec/attribute::id  
      prev_id=$sec/preceding-sibling::section/attribute::id  
      next_id=$sec/following-sibling::section/attribute::id>
```

```
    $sec/attribute::difficulty
  </dif>
ELSE
  <dif type="unknown" />
</result>
```

The result for this (not very meaningful query) is shown in App. A.3.

Usually, only abbreviated paths are used. For them, `child::` is the default axis, and the following abbreviations in Tab. 1 hold:

```
.   self:: axis
..  parent:: axis
//  descendant-or-self::node() (closure of /)
@   attribute:: axis
*   node() (unrestricted name)
```

Table 1: Path abbreviations.

Quilt also introduces the *dereference operator*, `->`. When `->` follows an IDREF-type attribute, it returns the elements that are referenced by this attribute (i.e., elements that have ID-type attribute with the same value as the IDREF attribute). For example:

```
chapter[title="Frogs"]//figref/@refid->/caption
```

Here, if there is a `chapter` element node (which has a child `title` node with value “Frogs”) with `figref` descendant which has an attribute `refid` (of type IDREF) with a value, say, “abc123”, and there is a `caption` element node somewhere else in the document with an attribute of type ID that also has the value “abc123”, then that `caption` element will be one of the nodes in the result of the expression above.

3 Element Constructors and Using Bound Variables

XML tags can be used to construct new nodes, and bound variables that were defined in preceding FOR/LET clauses (which will be discussed in Sec. 4 can be used anywhere inside them:

```
/* Convert some attributes to tags */
<reversed>
  FOR $parent IN document("book.xml")// *,
    $attr IN $parent/@*
  LET $aname := name($attr)
  RETURN
    <$aname parent=name($parent)>
      $attr/text()
    </$aname>
</reversed>
```

The result of the query above is shown in App. A.4

4 FLWR Expressions

FLWR is the acronym that denotes a generic sequence of FOR, LET, WHERE and RETURN clauses, which are used whenever it is necessary to iterate over the elements of a collection. We've already seen examples of FLWR expressions, and now it is time to explain in more detail the purpose of each element.

1. FOR *var* in *Expression* iteratively binds nodes in *Expression* (which must evaluate to list of nodes) to the variable *var*;
2. LET *var* *Expression* just binds the value of *Expression* to the variable *var* (like `let` clause in Lisp, for example);
3. WHERE *Expression* prunes the list of tuples of bound variables (which resulted from preceding FOR/LET expressions) so that only tuples satisfying *Expression* remain;
4. RETURN *Expression* evaluates the *Expression* and returns it to whoever waits for the result.

The *Expressions* above can be any Quilt expressions combined using various arithmetic, string, conditional and database operators; for example the following query could be used to find potential lawsuits in a surgery database:

```
<illegal-procedures>
  FOR $p in //procedure
  WHERE empty($p//anesthesia BEFORE ($p//incision)[1])
  RETURN $p
</illegal-procedures>
```

The flow control in this query is as follows:

1. Iteratively bind each `procedure` element node in the tree formed by the implicit root node to variable `p`;
2. Leave only those `p`'s that satisfy the expression in the WHERE clause, namely, that `p` has *no* `anesthesia` descendant that occurs before *the first* `incision` descendant of `p`; note that if there were no parentheses in the right side of the expression, the meaning would be *any* `incision` descendant of `p` that is *the first* child of its parent;
3. Return each validated `p` to whoever asked for it.

Thus, the query returns all `procedures` in which surgery operation started without anesthesia (with implicit assumption that each `procedure` had at least one `incision`).

Allowed database operators in expressions are UNION, INTERSECT, EXCEPT, BEFORE, AFTER and FILTER.

Another useful database operator is SORTBY. Here's an example:

```
FOR $author IN distinct(document("book.xml")//author)
RETURN
  <graphomaniac>
    $author/text()
  </graphomaniac> SORTBY (.)
```

This results in output which is shown in App. A.5. In this query, the `grafomaniac` elements that are returned by FOR iteration loop are sorted by the original author element nodes (that are implicitly converted to text strings before comparisons). Here, the elements are essentially sorted by themselves, but it is possible to perform the sort by other keys, such as children element nodes or attribute nodes.

5 Functions

Quilt allows user defined functions (although currently there is no precise specification for the typing of parameters).

The following query produces a kind of table of contents for the book XML in App. A.1.

```
/* Produce table of contents */
FUNCTION GET_TOC ($section) {
  <get_toc_result>
    $section/title,
    LET $subs := $section/section
    RETURN
      IF EXISTS($subs)
      THEN
        <nest>
          FOR $subsec IN $subs
          RETURN
            GET_TOC($subsec)/ *
        </nest>
      ELSE false
  </get_toc_result>
}

FOR $book IN document("book.xml")
RETURN
  <toc booktitle=$book/title/text()>
    GET_TOC($book)/ *
  </toc>
```

The result of this query can be seen in App. A.6. What happens here is that `GET_TOC` is called recursively for each `section`, and the result is embedded in a nest element node, which is constructed “on the fly”.

6 Quantifiers

Another interesting feature in Quilt is the availability of existential and universal quantifiers, `SOME` and `EVERY`. The following query produces titles of all sections in which all figures satisfy given size requirements.

```
<bigfigtitles>
  FOR $sec IN document("book.xml")//section
```

```
WHERE EVERY $fig IN $sec//figure SATISFIES
    ($fig/@height GEQ 250 AND $fig/@width GEQ 300)
RETURN
    <bigfigtitle>
        $sec/title
    </bigfigtitle>
</bigfigtitles>
```

Unfortunately, Kweelt engine does not yet implement quantifiers feature, so there no output for this query is given.

7 Querying Relational Data

Since a lot of data is currently stored in relational databases, it is of vital importance for an XML query language to provide access to such data (with relations converted to XML documents in some straightforward way).

Let us consider two relations, `bases` and `aircrafts`, which contain some IAF bases and aircrafts names, together with unique ids, as shown in App. A.7 and App. A.8.

Suppose we want to find all aircrafts made by McDonnell. In SQL, the query would look like this:

```
SELECT aircraft-no
FROM aircrafts
WHERE aircraft-name like 'McDonnell'
ORDER BY aircraft-no
```

The Quilt version of the query above is

```
FOR $craft IN document("rel_crafts.xml")/aircraft-tuple
WHERE contains($craft/aircraft-name,"McDonnell")
RETURN $craft/aircraft-no SORTBY(.)
```

The query result is shown below.

```
<?xml version="1.0"?>
<aircraft-no>
  3
</aircraft-no>
<aircraft-no>
  4
</aircraft-no>
<aircraft-no>
  5
</aircraft-no>
<!-- end of document -->
```

The result is not wrapped in a root element so that Quilt query will resemble SQL query as much as possible.

Joins are among the most important forms of relational queries. Here we show how to form joins in Quilt. Assume there's a third relation, `locations`, that expresses the relation between bases ids and aircrafts ids, and includes

the year at which the first aircraft of given type arrived to the base, such as locations relation, shown in App. A.9.

Then, we can perform an “inner” join between bases and aircrafts, using locations, as follows:

```
FOR $loc IN document("rel_locs.xml")/location-tuple,
   $craft IN document("rel_crafts.xml")/aircraft-tuple
           [aircraft-no=$loc/aircraft-no],
   $base IN document("rel_bases.xml")/base-tuple
           [base-no=$loc/base-no]
RETURN
  <aircraft-base-pair>
    <aircraft>
      $craft/aircraft-name/text()
    </aircraft>,
    <base>
      $base/base-name/text()
    </base>,
    $loc/year
  </aircraft-base-pair>
```

The result is shown in App. A.10.

8 Summary

Quilt appears to be a very nice XML query language, incorporating features from a broad range of preceding query languages. However, current Quilt specification leaves a lot of clarifications for the future, which suggests using more robust and established engines like XSLT[3] for documents’ tree preprocessing and transformation, and leaving the job of querying per se to a Quilt implementation, such as Kweelt[5].

Efficiency of Quilt is another important issue, since current implementations using DOM (representing XML document as a tree) appear to be very inefficient for large documents. Currently there is a research going for translation of Quilt queries into relational database query language, such as SQL (with appropriate representation of XML document as a set of relations), as to utilize the existing experience in optimizing traditional database queries. See, for example, AGORA[4] project.

A Data Sheets

A.1 Book XML data

```
<?xml version="1.0"?>
<!DOCTYPE book [
  <!ELEMENT book (title, author+, section+)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT section (title, (p | figure | section)* )>
]
```

```

    <!ATTLIST section
      id ID #IMPLIED
      difficulty CDATA #IMPLIED>
    <!ELEMENT p (#PCDATA)>
    <!ELEMENT figure (title, image)>
    <!ATTLIST figure
      width CDATA #REQUIRED
      height CDATA #REQUIRED >
    <!ELEMENT image EMPTY>
    <!ATTLIST image
      source CDATA #REQUIRED >
  ]>

<book>
  <title>Data on the Web</title>
  <author>Serge Abiteboul</author>
  <author>Peter Buneman</author>
  <author>Dan Suciu</author>
  <section id="intro" difficulty="easy" >
    <title>Introduction</title>
    <p>Text 1 ... </p>
    <section>
      <title>Audience</title>
      <p>Text 2 ... </p>
    </section>
    <section>
      <title>Web Data and the Two Cultures</title>
      <p>Text 3 ... </p>
      <figure height="400" width="400">
        <title>Traditional client/server architecture</title>
        <image source="csarch.gif"/>
      </figure>
      <p>Text 4 ... </p>
    </section>
  </section>
  <section id="syntax" difficulty="medium" >
    <title>A Syntax For Data</title>
    <p>Text ... </p>
    <figure height="200" width="500">
      <title>Graph representations of structures</title>
      <image source="graphs.gif"/>
    </figure>
    <p>Text 5 ... </p>
    <section>
      <title>Base Types</title>
      <p>Text 6 ... </p>
    </section>
    <section>
      <title>Representing Relational Databases</title>
      <p>Text 7 ... </p>
    </section>
  </section>

```

```

    <figure height="250" width="400">
      <title>Examples of Relations</title>
      <image source="relations.gif"/>
    </figure>
  </section>
  <section>
    <title>Representing Object Databases</title>
    <p>Text 8 ... </p>
  </section>
</section>
</book>

```

A.2 Titles scan result

```

<?xml version="1.0"?>
<queryresult>
  <datatitle>
    Data on the Web
  </datatitle>
  <datatitle>
    Web Data and the Two Cultures
  </datatitle>
  <datatitle>
    A Syntax For Data
  </datatitle>
  <datatitle>
    Representing Relational Databases
  </datatitle>
  <datatitle>
    Representing Object Databases
  </datatitle>
</queryresult>
<!-- end of document -->

```

A.3 Unabbreviated query result

```

<?xml version="1.0"?>
<result>
  <dif id="intro" prev_id="" next_id="syntax">
    easy
  </dif>
  <dif type="unknown">
  </dif>
  <dif type="unknown">
  </dif>
  <dif id="syntax" prev_id="intro" next_id="">
    medium
  </dif>
  <dif type="unknown">

```

```
</dif>
<dif type="unknown">
</dif>
<dif type="unknown">
</dif>
</result>
<!-- end of document -->
```

A.4 Tags extraction query result

```
<?xml version="1.0"?>
<reversed>
  <difficulty parent="section">
    easy
  </difficulty>
  <id parent="section">
    intro
  </id>
  <height parent="figure">
    400
  </height>
  <width parent="figure">
    400
  </width>
  <source parent="image">
    csarch.gif
  </source>
  <difficulty parent="section">
    medium
  </difficulty>
  <id parent="section">
    syntax
  </id>
  <height parent="figure">
    200
  </height>
  <width parent="figure">
    500
  </width>
  <source parent="image">
    graphs.gif
  </source>
  <height parent="figure">
    250
  </height>
  <width parent="figure">
    400
  </width>
  <source parent="image">
    relations.gif
```

```
</source>
</reversed>
<!-- end of document -->
```

A.5 Authors sorting query result

```
<?xml version="1.0"?>
<graphomaniac>
  Dan Suciu
</graphomaniac>
<graphomaniac>
  Peter Buneman
</graphomaniac>
<graphomaniac>
  Serge Abiteboul
</graphomaniac>
<!-- end of document -->
```

A.6 TOC query result

```
<?xml version="1.0"?>
<toc booktitle="Data on the Web">
  <title>
    Data on the Web
  </title>
  <nest>
    <title>
      Introduction
    </title>
    <nest>
      <title>
        Audience
      </title>
      <title>
        Web Data and the Two Cultures
      </title>
    </nest>
    <title>
      A Syntax For Data
    </title>
    <nest>
      <title>
        Base Types
      </title>
      <title>
        Representing Relational Databases
      </title>
      <title>
        Representing Object Databases
      </title>
    </nest>
  </nest>
</toc>
```

```

        </title>
      </nest>
    </nest>
  </toc>
<!-- end of document -->

```

A.7 IAF Bases Relation

```

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE bases [
  <!ELEMENT bases (base-tuple*)>
  <!ELEMENT base-tuple (base-no,base-name)>
  <!ELEMENT base-no (#PCDATA)>
  <!ELEMENT base-name (#PCDATA)>
]>

<bases>
  <base-tuple>
    <base-no>1</base-no>
    <base-name>Haifa Airport</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>2</base-no>
    <base-name>Hatzerim</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>3</base-no>
    <base-name>Hatzor</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>4</base-no>
    <base-name>Lod Airport</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>5</base-no>
    <base-name>Nevatim</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>6</base-no>
    <base-name>Palmachim</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>7</base-no>
    <base-name>Ramat David</base-name>
  </base-tuple>
  <base-tuple>
    <base-no>8</base-no>
    <base-name>Ramon</base-name>
  </base-tuple>

```

```

<base-tuple>
  <base-no>9</base-no>
  <base-name>Sde Dov</base-name>
</base-tuple>
<base-tuple>
  <base-no>10</base-no>
  <base-name>Sdot Micha</base-name>
</base-tuple>
<base-tuple>
  <base-no>11</base-no>
  <base-name>Tel Nof</base-name>
</base-tuple>
</bases>

```

A.8 IAF Aircrafts Relation

```

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE aircrafts [
  <!ELEMENT aircrafts (aircraft-tuple*)>
  <!ELEMENT aircraft-tuple (aircraft-no,aircraft-name)>
  <!ELEMENT aircraft-no (#PCDATA)>
  <!ELEMENT aircraft-name (#PCDATA)>
]>

<aircrafts>
  <aircraft-tuple>
    <aircraft-no>1</aircraft-no>
    <aircraft-name>Boeing F-15I Thunder</aircraft-name>
  </aircraft-tuple>
  <aircraft-tuple>
    <aircraft-no>2</aircraft-no>
    <aircraft-name>Lockheed F-16 Fighting Falcon</aircraft-name>
  </aircraft-tuple>
  <aircraft-tuple>
    <aircraft-no>3</aircraft-no>
    <aircraft-name>McDonnell Douglas F-15 Eagle</aircraft-name>
  </aircraft-tuple>
  <aircraft-tuple>
    <aircraft-no>4</aircraft-no>
    <aircraft-name>McDonnell Douglas F-4 Phantom</aircraft-name>
  </aircraft-tuple>
  <aircraft-tuple>
    <aircraft-no>5</aircraft-no>
    <aircraft-name>McDonnell Douglas A-4 Skyhawk</aircraft-name>
  </aircraft-tuple>
  <aircraft-tuple>
    <aircraft-no>6</aircraft-no>
    <aircraft-name>Dassault Mirage IIIC</aircraft-name>
  </aircraft-tuple>

```

```

    <aircraft-tuple>
      <aircraft-no>7</aircraft-no>
      <aircraft-name>Avia S-199 Messerschmitt</aircraft-name>
    </aircraft-tuple>
  </aircrafts>

```

A.9 IAF Aircrafts/Bases Locations (fictional)

```

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE locations [
  <!ELEMENT locations (location-tuple*)>
  <!ELEMENT location-tuple (aircraft-no,base-no,year)>
  <!ELEMENT aircraft-no (#PCDATA)>
  <!ELEMENT base-no (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
]>

<locations>
  <location-tuple>
    <aircraft-no>2</aircraft-no>
    <base-no>9</base-no>
    <year>1994</year>
  </location-tuple>
  <location-tuple>
    <aircraft-no>1</aircraft-no>
    <base-no>2</base-no>
    <year>1997</year>
  </location-tuple>
  <location-tuple>
    <aircraft-no>4</aircraft-no>
    <base-no>2</base-no>
    <year>1988</year>
  </location-tuple>
  <location-tuple>
    <aircraft-no>4</aircraft-no>
    <base-no>4</base-no>
    <year>1992</year>
  </location-tuple>
  <location-tuple>
    <aircraft-no>6</aircraft-no>
    <base-no>7</base-no>
    <year>1965</year>
  </location-tuple>
</locations>

```

A.10 IAF Aircrafts/Bases Join

```

<?xml version="1.0"?>

```

```
<aircraft-base-pair>
  <aircraft>
    Lockheed F-16 Fighting Falcon
  </aircraft>
  <base>
    Sde Dov
  </base>
  <year>
    1994
  </year>
</aircraft-base-pair>
<aircraft-base-pair>
  <aircraft>
    Boeing F-15I Thunder
  </aircraft>
  <base>
    Hatzirim
  </base>
  <year>
    1997
  </year>
</aircraft-base-pair>
<aircraft-base-pair>
  <aircraft>
    McDonnell Douglas F-4 Phantom
  </aircraft>
  <base>
    Hatzirim
  </base>
  <year>
    1988
  </year>
</aircraft-base-pair>
<aircraft-base-pair>
  <aircraft>
    McDonnell Douglas F-4 Phantom
  </aircraft>
  <base>
    Lod Airport
  </base>
  <year>
    1992
  </year>
</aircraft-base-pair>
<aircraft-base-pair>
  <aircraft>
    Dassault Mirage IIIC
  </aircraft>
  <base>
    Ramat David
```

```
</base>
<year>
  1965
</year>
</aircraft-base-pair>
<!-- end of document -->
```

References

- [1] Don Chamberlin, Jonathan Robie, and Daniela Florescu. *Quilt: An XML Query Language for Heterogeneous Data Sources*. See http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf.
- [2] World Wide Web Consortium. *XML Path Language (XPath) Version 1.0*. See <http://www.w3.org/TR/xpath>. W3C Recommendation 16 November 1999.
- [3] World Wide Web Consortium. *XSL Transformations (XSLT) Version 1.0*. See <http://www.w3.org/TR/xslt>. W3C Recommendation 16 November 1999.
- [4] Daniela Florescu, Ioana Manolescu, Donald Kossmann, Dan Olteanu, and Florian Xhumari. *Agora: Living with XML and Relational*. See <http://www-caravel.inria.fr/ioana/AGORA/index.html>.
- [5] *Kweelt: Querying XML in the New Millennium*. See <http://db.cis.upenn.edu/Kweelt>.