Progress Report on XQuery

Don Chamberlin

Almaden Research Center

May 24, 2002

History

- Dec. '98: W3C sponsors workshop on XML Query
- Oct. '99: W3C charters XML Query working group
  - Chair: Paul Cotton
  - About 50 members from about 35 companies
  - Weekly conference calls, meetings every 6-8 weeks
- 2000: WG publishes req'ts, use cases, data model
- June 2000: Quilt proposal presented at WebDB
- Feb. 2001: First working draft of XQuery language

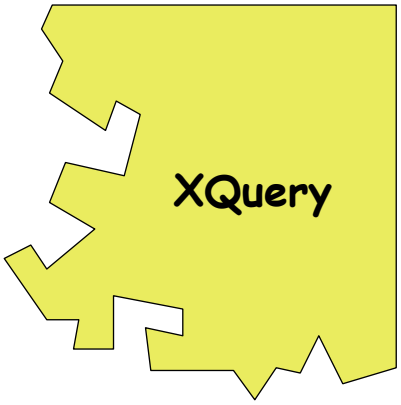
Useful websites

- Public website: www.w3.org/XML/Query
- Public comments (before May 2002):
  - Post to: www-xml-query-comments@w3.org
  - Archived at lists.w3.org/Archives/Public/www-xml-query-comments
- Public comments (after May 2002):
  - Post to: public-qt-comments@w3.org
  - Archived at lists.w3.org/Archives/Public/public-qt-comments

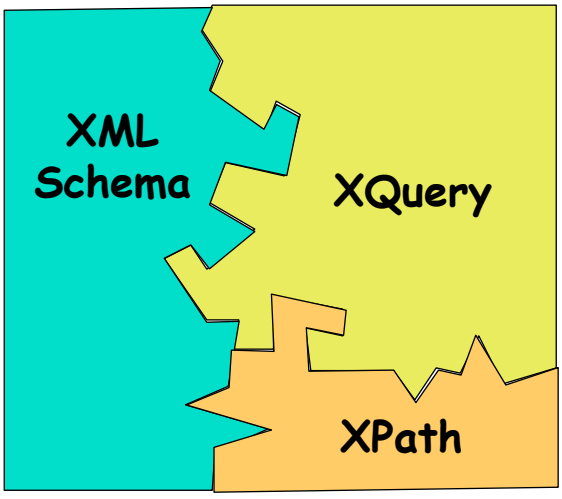
Working Drafts

- Linked from the XML Query WG homepage:
  - XQuery 1.0: An XML Query Language
  - XML Path Language (XPath) 2.0
  - XQuery and XPath Data Model
  - XQuery and XPath Functions and Operators
  - XQuery Formal Semantics
  - XML Query Requirements
  - XML Query Use Cases
  - XML Syntax for XQuery
- 17 reference implementations (many downloadable)

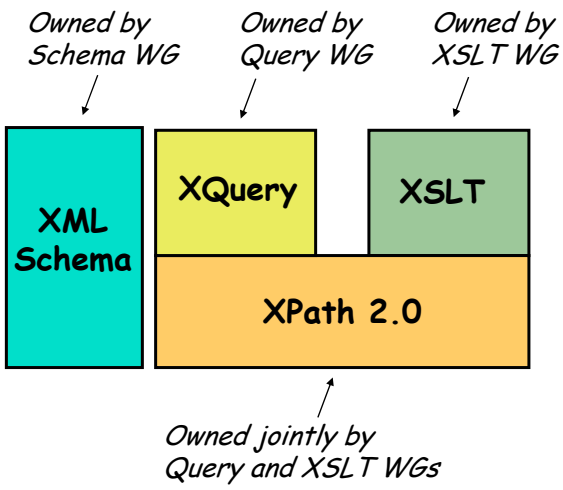
Why does XQuery look like this?



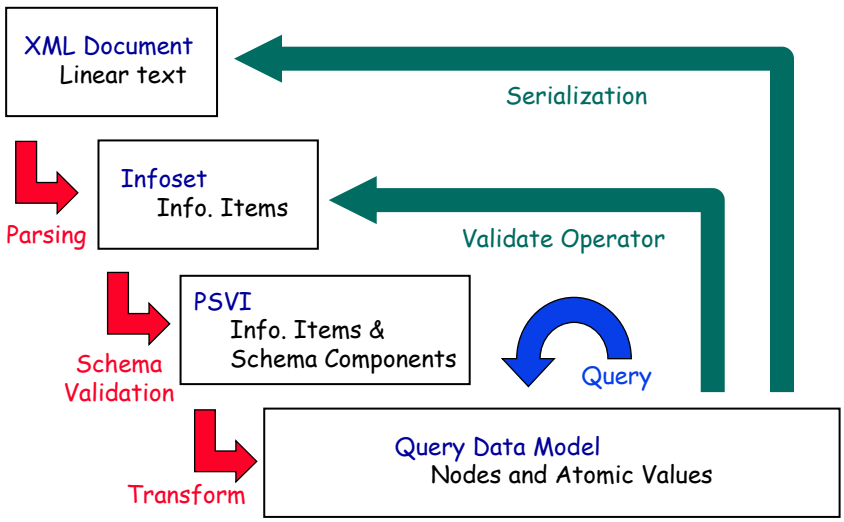
...because it has to fit into the XML world



XQuery and its close relatives



XML and the Query Data Model



Why does XQuery need a data model?

What does this mean?

```
/emp[salary > 10000]
```


The Query Data Model

- A value is either the error value, or an ordered sequence of zero or more items.
- An item is a node or an atomic value.
- There are seven kinds of nodes:
  - Document Node
  - Element Node
  - Attribute Node
  - Text Node
  - Comment Node
  - Processing Instruction Node
  - Namespace Node

Examples of values

- 47
- <goldfish/>
- (1, 2, 3)
- (47, <goldfish/>, "Hello")
- ( )
- An XML document
- An attribute standing by itself
- ERROR

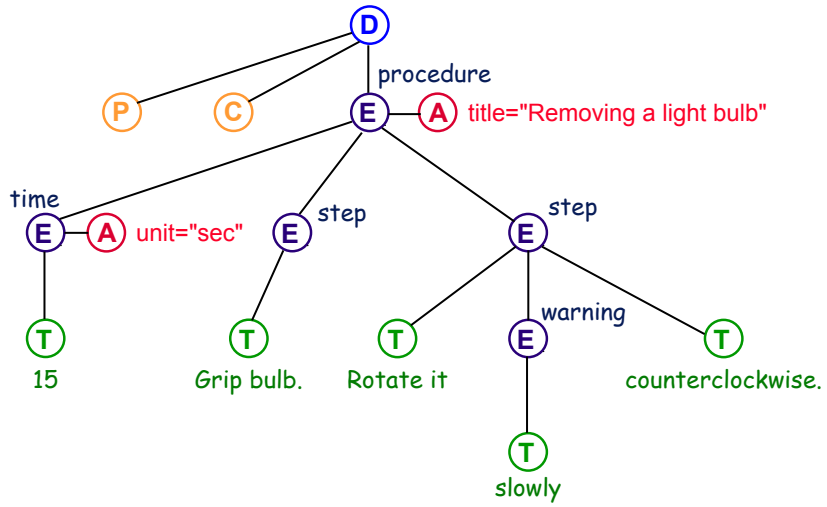
Facts about values

- There is no distinction between an item and a sequence of length one
- There are no nested sequences
- There is no null value
- A sequence can be empty
- Sequences can contain heterogeneous values
- All sequences are ordered

An XML Document ...

```
<?xml version = "1.0"?>
<!-- Requires one trained person -->
<procedure title = "Removing a light bulb">
   <time unit = "sec">15</time>
   <step>Grip bulb.</step>
   <step>
      Rotate it
      <warning>slowly</warning>
      counterclockwise.
   </step>
</procedure>
```


... and its Data Model Representation



Facts about nodes

- Nodes have identity (atomic values don't)
- Element and attribute nodes have a type annotation
  - Generated by validating the node
  - May be a complex type such as PurchaseOrder
  - Type may be unknown ("anyType")
- Each node has a typed value:
  - a sequence of atomic values (or ERROR)
  - Type may be unknown ("anySimpleType")
- There is a document order among nodes
  - Ordering among documents and constructed nodes is implementation-defined but stable

General XQuery Rules

- XQuery is a case-sensitive language
- Keywords are in lower-case
- XQuery is a functional language
- It consists of 21 kinds of expressions
- Every expression has a value and no side effects
- Expressions are fully composable
- Expressions propagate the error value
  - Exception: `and`, `or`, quantifiers have "early-out" semantics

XQuery Expressions

- Literals: `"Hello"  47  4.7  4.7E-2`
- Constructed values:
  `true()  false()  date("2002-03-15")`
- Variables: `$x`
- Constructed sequences
  - `$a, $b` is the same as `($a, $b)`
  - `(1, (2, 3), ( ), (4))` is the same as `1, 2, 3, 4`
  - `5 to 8` is the same as `5, 6, 7, 8`

Functions

- Function calls
  - `three-argument-function(1, 2, 3)`
  - `two-argument-function(1, (2, 3))`
- Functions are not overloaded (except certain built-ins)
- Evaluating a function call
  - Convert arguments to expected types and bind parameters
  - Evaluate function body
  - Convert result to expected result type
- Conversions (if needed):
  - Extract typed value from node
  - Cast "anySimpleType" argument to expected type
  - Promote numerics and derived types

Path Expressions

- Path expressions are inherited from XPath 1.0
- A path always returns a sequence of distinct nodes in document order
- A path consists of a series of steps: E1/E2/E3 . . .
- Each step can be any expression that returns a sequence of nodes
- Here's what E1/E2 means:
  - Evaluate E1—it must be a set of nodes
  - For each node N in E1, evaluate E2 with N as context node
  - Union together all the E2-values
  - Eliminate duplicate node-ids and sort in document order

Axis Steps

- A frequently-used kind of step is an axis step
- An axis step maps a node onto a sequence of related nodes
- An axis step has three parts:
  - The axis (defines the "direction of movement")
  - The node test (qualifies by name or kind of node)
  - Zero or more predicates
- Example of an axis step:

```
child::product[price > 100]
```

- Axis steps often use an abbreviated syntax:

```
product[price > 100]
```

Axes

XPath

Forward Axes:

- child
- descendant
- attribute
- self
- descendant-or-self
- following-sibling
- following
- namespace

Reverse axes:

- parent
- ancestor
- preceding-sibling
- preceding
- ancestor-or-self

XQuery

Forward Axes:

- child
- descendant
- attribute
- self
- descendant-or-self

(a growing list?)

Reverse axes:

- parent

Predicates

- Serve as a filter on a sequence (often used in paths)
- Meaning of E1[E2]:
- For each item e in the value of E1, evaluate E2 with:
  - Context item = e
  - Context position = position of e within the value of E1
- Retain those items in E1 for which the predicate truth value of E2 is true.

Predicates, continued

- The predicate truth value of an expression E:
  - If E has a Boolean value: use that value
    Example: `$emps[salary > 5000]`
  - If E has a numeric value: TRUE if e is equal to the context position, otherwise FALSE
    Example: `$emps[5]`
  - If E is an empty sequence: FALSE
    If E is a non-empty node sequence: TRUE
    Example: `$emps[secretary]`
  - Otherwise, return an error.

Expressions, continued

- Combining sequences: `union` `intersect` `except`
  - return sequences of distinct nodes in document order
- Arithmetic operators: `+` `-` `*` `div` `mod`
  - Extract typed value from node
  - Cast "anySimpleType" to double
  - Promote numeric operands to a common type
  - Multiple values => error
  - If operand is ( ), return ( )
  - Arithmetic supported for numeric and date/time types

Comparison Operators

Four kinds of comparison operators:

- `eq  ne  gt  ge  lt  le`
  Compare single atomic values
- `=  !=  >  >=  <  <=`
  Compare sequences of values, with existential semantics
- `is  isnot`
  Compare two nodes, based on node identity
- `<<  >>  precedes  follows`
  Compare two nodes, based on document order

Logical Expressions

- Operators: `and  or`
- Function: `not( )`
- Return TRUE or FALSE (2-valued logic)
- Result depends on effective boolean value of operands
  - If operand is of type boolean, it serves as its own EBV
  - If operand is ( ), EBV is FALSE
  - If operand is a non-empty node sequence, EBV is TRUE
  - In any other case, return an error
- "Early-out" semantics (need not evaluate both operands)

Constructors

- To construct an element with a known name and content, use XML syntax:

```
<book isbn="12345">
   <title>Huckleberry Finn</title>
</book>
```

- If the content of an element or attribute must be computed, use a nested expression enclosed in `{ }`

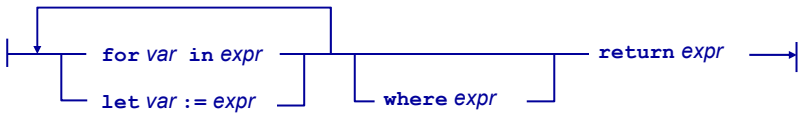
```
<book isbn="{$x}">
   { $b/title }
</book>
```

- If both the name and the content must be computed, use a computed constructor:

```
element { name-expr } { content-expr }
attribute { name-expr } { content-expr }
```


FLWR Expressions

- A FLWR expression binds some variables, applies a predicate, and constructs a new result.



FOR and LET clauses generate a list of tuples of bound variables, preserving document order.

WHERE clause applies a predicate, eliminating some of the tuples

RETURN clause is executed for each surviving tuple, generating an ordered list of outputs

An Example Query

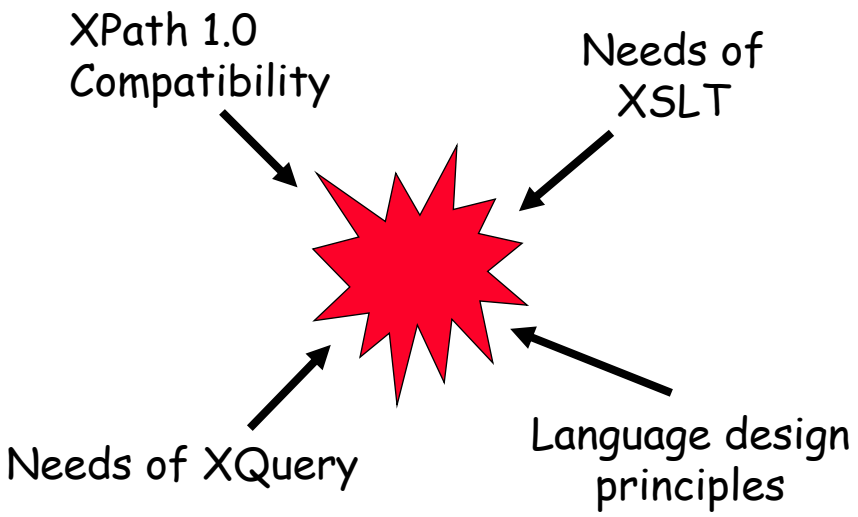
- "Find the description and average price of each red part that has at least 10 orders"

```
for $p in document("parts.xml")
              //part[color = "Red"]
let $o := document("orders.xml")
              //order[partno = $p/partno]
where count($o) >= 10
return
   <important_red_part>
      { $p/description }
      <avg_price> {avg($o/price)} </avg_price>
   </important_red_part>
```


Expressions, continued

- *expr1* **sortby** *expr2*, ...
  - For each item I in *expr1*, *expr2* is evaluated with I as focus
  - Resulting values used to reorder the items in *expr1*
- **unordered** *expr*
  - Indicates that the order of *expr* is not significant
- **if** (*expr1*) **then** *expr2* **else** *expr3*
  - Uses effective boolean value, like **and** and **or**
- { **some** / **every** } *var* **in** *expr1* **satisfies** *expr2*
  - Also based on effective boolean value
  - Allow early-out for errors

Issue: the future of XPath



31

Fun with XPath 1.0

- a[b = 5]
  returns a-elements that have ***any*** b-child with value 5
- a[b+0 = 5]
  returns a-elements whose ***first*** b-child has value 5
- a[b-0 = 5]
  returns a-elements that have a child named "b-0" with value 5

32

Fun with XPath 1.0, continued

- //person[8]
  returns the eighth person in the list of all persons
- //person[shoesize]
  returns all persons who have at least one shoesize
- //person[shoesize + 0]
  returns persons whose position in the list of persons is equal to their (first) shoesize
- //person[married = true( )]
  returns all persons that have a "married" subelement, regardless of its value

Fun with XPath 1.0, continued

- Comparisons:
  - "4" = 4.0 returns True
  - "4" = "4.0" returns False
  - "4" >= "4.0" returns True
  - "4" <= "4.0" returns True
  - "Apple" < "Banana" returns False (treated as NaN < NaN)
- Arithmetic:
  - 1 + 2 returns 3.0 (all arithmetic is floating point)
  - "1" + 2 returns 3.0
  - "1" + "2" returns 3.0
  - "Apple" + "Banana" returns NaN

Fun with XPath 1.0, continued

- The following two elements are "equal" (the XPath 1.0 "=" operator returns TRUE when comparing them):

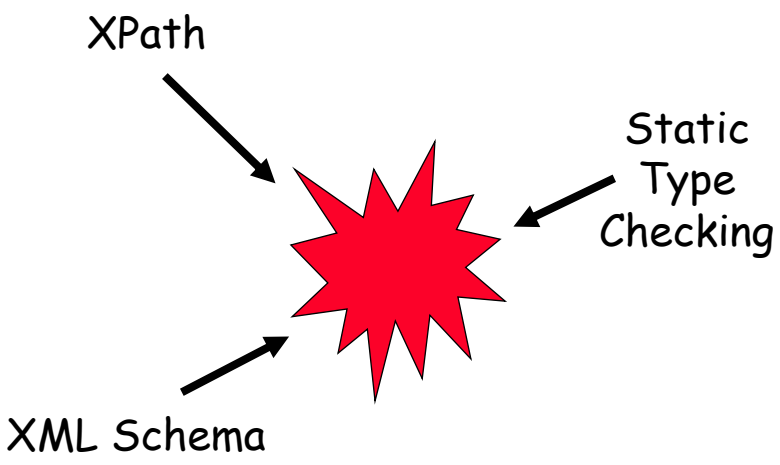
```
<book>
   <author> Mark Twain </author>
   <title> Huckleberry Finn </title>
</book>

<book>
   <title> Mark Twain </title>
   <author> Huckleberry Finn </author>
</book>
```


What to do about all this?

- A few incompatible changes to XPath
- A compromise: "type exceptions"
- Examples of type exceptions:
  - Arithmetic on a sequence of multiple values
  - Comparison of two elements by "="
- Type exceptions can be handled by the "host language"
  - XQuery treats all type exceptions as errors
  - XSLT handles type exceptions by "fallback conversions"
  - Mostly, these preserve the semantics of XPath 1.0

Issue: Types in XQuery



Types in XPath

- XPath 1.0 recognizes four basic types:
  - String
  - Float
  - Boolean
  - Node Set
- XPath has various rules for coercing any type into any other type without raising any run-time errors

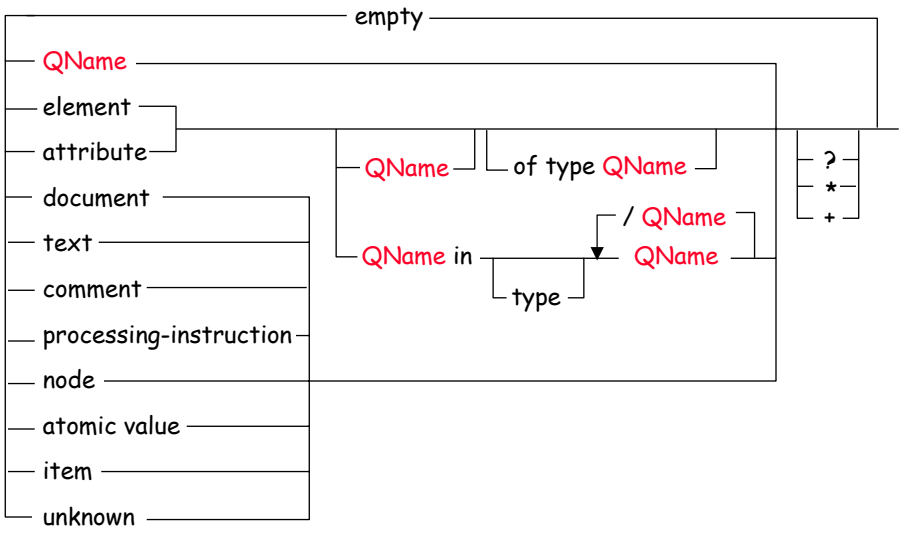
Types in XML Schema

- W3C Recommendation: 3 parts, 341 pages
- 19 primitive datatypes: string, decimal, etc.
- 25 built-in derived datatypes
- User-defined types, both simple and complex
- The type of an element is different from its name
- 2 different ways to define derived types
  - extension: adding to the content
  - restriction: placing constraints on the content

Types in XQuery

- Where do types occur in queries?
- Function signatures (parameter and return types)
- Other expressions that operate on types
  - cast
  - instanceof
  - typeswitch
  - treat
  - assert

SequenceType



validate Expression

- Syntax: `validate { expr }`
- Semantics: evaluate *expr*, then serialize its value as an XML string and invoke the schema validator on it
- Elements and attributes that are recognized by the validator receive type annotations.
  - `<a>{5}</a>` has annotation `anyType`
  - `validate {<a>{5}</a>}` might have annotation `hatsize`

Testing Types

- Instance Of expression returns TRUE or FALSE:

```
$animal instance of element dog
```

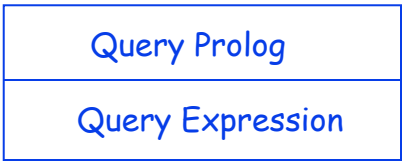
- Typeswitch expression executes one branch, based on the type of its operand:

```
typeswitch($animal)
   case element dog return woof($animal)
   case element duck return quack($animal)
   default return "No sound"
```


Tinkering with Types

- `cast as ST ( expr )`
  - Converts value to target type
  - Only for predefined type pairs and derived -> base type
  - May return error at run-time
- `treat as ST ( expr )`
  - Serves as a compile-time "promise"
  - At run-time, returns an error if type of expr is not ST
  - `treat as element of type USAddress ($myaddress)`
- `assert as ST ( expr )`
  - Serves as a compile-time assertion
  - Compile-time error if static type of expr is not ST
  - `assert as PurchaseOrder (query)`

Structure of an XQuery



- The Query Prolog contains:
  - Namespace declarations (bind namespace prefixes to URI's)
  - Schema imports (import namespaces and their schemas)
  - Function definitions (may be recursive)
- The Query Expression contains:
  - an expression that defines the result of the query

Formal Semantics of XQuery

- http://www.w3.org/TR/query-semantics/
- Defines static and dynamic semantics for every type of expression
- Static type-checking (compile-time)
  - Depends only on the query itself
  - Infers result type based on types of operands
  - Purpose: catch errors early, guarantee result type
  - May not be required at all conformance levels of XQuery
- Dynamic execution (run-time)
  - Depends on input data
  - Defines the result value based on the operand values

Formal Semantics, continued

- If a query passes static type checking, it may still return the error value
  - It may divide by zero
  - Casts may fail. Example:
    `cast as integer($x)` where value of $x is "garbage"
- If a query fails static type checking, it may still execute successfully and compute a useful result. Example (with no schema):

```
$emp/salary + 1000
```

  - Static semantics says this is a type error
  - Dynamic semantics executes it successfully if `$emp` has exactly one salary subelement with a numeric value

Beyond Version 1

- Updates
- View definitions
- Language bindings
- Full-text search
- Output serialization
- Importing function libraries
  - Defined in XQuery
  - Defined in host language

Summary: XQuery on one slide

- Query prolog: namespaces, schemas, function def'ns
- Composable expressions:
  - Literals & variables
  - Sequences
  - Function calls
  - Path expressions
  - Predicates
  - Constructors
  - Union, intersect, except
  - Comparisons
  - and, or
  - Arithmetic
  - FLWR expressions
  - sortby
  - unordered
  - if ... then ... else
  - some, every
  - instanceof
  - typeswitch
  - cast as
  - treat as
  - assert as
  - validate