XPath and XQuery

Patryk Czarnik

Institute of Informatics University of Warsaw

XML and Modern Techniques of Content Management – 2010/11

Introduction
- Status
- XPath Data Model

XPath language
- Basic constructs
- XPath 2.0 extras
- Paths

XQuery
- XQuery query structure
- Constructors
- Functions
XPath and XQuery
Querying XML documents

Common properties

- Expression languages designed to query XML documents
- Convenient access to document nodes
- Intuitive syntax analogous to filesystem paths
- Comparison and arithmetic operators, functions, etc.

XPath
Used within other standards:
- XSLT
- XML Schema
- XPointer
- DOM

XQuery
Standalone standard. Main applications:
- XML data access and processing
- XML databases

XPath — status

XPath 1.0
- W3C Recommendation, XI 1999
- used within XSLT 1.0, XML Schema, XPointer

XPath 2.0
- Several W3C Recommendations, I 2007:
  - XML Path Language (XPath) 2.0
  - XQuery 1.0 and XPath 2.0 Data Model
  - XQuery 1.0 and XPath 2.0 Functions and Operators
  - XQuery 1.0 and XPath 2.0 Formal Semantics
- Used within XSLT 2.0
- Related to XQuery 1.0
XPath and XQuery Data Model

- Theoretical basis of XPath, XSLT, and XQuery
- XML document tree
- Structures and simple data types
- Basic operations (type conversions etc.)

### Differences between 1.0 and 2.0

<table>
<thead>
<tr>
<th></th>
<th>XPath 1.0</th>
<th>XPath 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple data types</td>
<td>boolean, number, string</td>
<td>all XML Schema simple types</td>
</tr>
<tr>
<td>structures</td>
<td>node sets</td>
<td>sequences of nodes and simple values</td>
</tr>
</tbody>
</table>

### XML document in XPath model

- Document tree
- Physical representation level fully expanded
  - CDATA, references to characters and entities
- No adjacent text nodes
- Namespaces applied and accessible
- XML Schema applied and accessible
  - XPath 2.0 “schema aware” processors only
- Attribute nodes as element “properties”
  - formally, attribute is not child of element
  - however, element is parent of its attributes
- Root of tree — **document node**
  - main element (aka document element) is not the root
XPath node kinds

Seven **kinds** of nodes:

- document node (root)
- element
- attribute
- text node
- processing instruction
- comment
- namespace node

Sequences

- Values in XPath 2.0 — **sequences**
- Sequence consists of zero or more **items**
  - nodes
  - atomic values

Sequences properties

- Items order and number of occurrence meaningful
- Singleton sequence equivalent to its item:
  \[ 3.14 = (3.14) \]
- Nested sequences implicitly flattened to canonical representation:
  \[ (3.14, (1, 2, 3), 'Alice') = (3.14, 1, 2, 3, 'Alice') \]
Data model in XPath 1.0

- Four types:
  - boolean
  - string
  - number
  - node set

- No collection of simple values
- Sets (and not sequences) of nodes
Effective Boolean Value

- Treating any value as boolean
- Motivation: convenience in condition writing, e.g. \(\text{if ( person[@passport] ) } \ldots\)

Conversion rules

- empty sequence \(\rightarrow\) false
- sequence starting with node \(\rightarrow\) true
- single boolean value \(\rightarrow\) that value
- single empty string \(\rightarrow\) false
- single non-empty string \(\rightarrow\) true
- single number equal to 0 or NaN \(\rightarrow\) false
- other single number \(\rightarrow\) true
- other value \(\rightarrow\) error

Atomization

- Treating any sequence as sequence of atomic values
- Motivation: sequences comparison, arithmetic, type casting

Conversion rules (for each item)

- atomic value \(\rightarrow\) that value
- node of declared atomic type \(\rightarrow\) node value
- node of list type \(\rightarrow\) sequence of list elements
- node of unknown simple type or one of \(\text{xs:untypedAtomic, xs:anySimpleType}\) \(\rightarrow\) text content as single item
- node with mixed content \(\rightarrow\) text content as single item
- node with element content \(\rightarrow\) error
Paths — typical XPath application

- /company/department/person
- //person
- /company/department[name = 'accountancy']
- /company/department[@id = 'D07']/person[3]
- ./surname
- surname
- ../person[position = 'manager']/surname

But there is much more to learn about XPath :)

Literals and variables

Literals

- strings: '12.5', "He said, ""I don’t like it.""
- numbers: 12, 12.5, 1.13e-8

Variables

- $x — reference to variable x,
- Variables introduced through:
  - XPath 2.0 (for, some, every)
  - XQuery (FLWOR, some, every, function parameters)
  - XSLT 1.0 and 2.0 (variable, param)
Type casting

Type constructors

- `xs:date("2010-08-25")`
- `xs:float("NaN")`
- `adresy:kod-pocztowy("48-200")` (schema aware processing)
- `string(//obiekt[4])` (valid in XPath 1.0 too)

**cast as operator**

- "2010-08-25" cast as `xs:date`
- ...

Functions

- **Function invocation:**
  - `concat(‘Mrs ’, name, ’ ’, surname)`
  - `count(//person)`
  - `my:fac(12)`

- **150 (XPath 2.0) built-in functions:**

- **Custom functions defining:**
  - XQuery
  - XSLT 2.0
  - execution environment

- **EXSLT — de-facto standard of additional XPath functions and extension mechanism for XSLT 1.0**
# Chosen built-in XPath functions

## Text

- `concat(s1, s2, ...)`
- `substring(s, pos, len)`
- `starts-with(s1, s2)`
- `contains(s1, s2)`
- `string-length(s)`
- `translate(s, t1, t2)`

## Numbers

- `floor(x)`
- `ceiling(x)`
- `round(x)`
- `abs(x)`

## Nodes

- `name(n?)`
- `local-name(n?)`
- `namespace-uri(n?)`
- `id(s)`
- `nilled(n?)`
- `document-uri(doc)`

## Context

- `current()`
- `position()`
- `last()`
- `current-time()`

## Sequences

- `count(S)`
- `sum(S)`
- `min(S)`
- `max(S)`
- `avg(S)`
- `empty(S)`
- `reverse(S)`
- `distinct-values(S)`

## Date and time

- `month-from-date(t)`
- `adjust-date-to-timezone(t, tz)`

## Operators

- 68 operators in XPath 2.0 (after overloading expansion)

### Arithmetic

- `+ - * div idiv mod` on numbers
- `+ and -` on date/time and duration

### Node sets / sequences

- union  |  intersect  except
- not nodes found — type error
- result without repeats, document order preserved

### Logical values

- and  or
Comparison operators

Atomic comparison (XPath 2.0 only)

- eq  ne  lt  le  gt  ge
- applied to singletons

General comparison (XPath 1.0 and 2.0)

- =  !=  <  <=  >  >=
- applied to sequences
- XPath 2.0 semantics:
  There exists a pair of items, one from each argument sequences, for which the corresponding atomic comparison holds.
  (Argument sequences atomized on entry.)

General comparison — nonobvious behaviour

Equality operator does not check the real equality

(1, 2) = (2, 3) – true
(1, 2) != (1, 2) – true

Equality is not transitive

(1, 2) = (2, 3) – true
(2, 3) = (3, 4) – true
(1, 2) = (3, 4) – false

Inequality is not just equality negation

(1, 2) = (1, 2) – true
(1, 2) != (1, 2) – true
() = () – false
() != () – false
Conditional expression (XPath 2.0)

if \textit{CONDITION} 
  then \textit{RESULT1} 
else \textit{RESULT2}

- Effective Boolean Value of \textit{CONDITION}
- One branch computed

Example

if details/price 
then 
  if details/price \geq 1000 
  then ‘Insured mail’ 
  else ‘Ordinary mail’ 
else ‘No data’

Iteration through sequence (XPath 2.0)

for $\textit{VAR}$ in \textit{SEQUENCE} 
return \textit{RESULT}

- \textit{VAR} assigned subsequent values from \textit{SEQUENCE}
- \textit{RESULT} computer in context where \textit{VAR} is assigned current value
- overall result — (flattened) sequence of subsequent partial results

Examples

for $i$ in (1 to 10) 
return $i \times i$

for $p$ in //person 
return concat($p$/name, ‘’, $p$/surname)
Sequence quantifiers (XPath 2.0)

some $VAR$ in $SEQUENCE$
satisfies $CONDITION$

every $VAR$ in $SEQUENCE$
satisfies $CONDITION$

▶ Effective Boolean Value of $CONDITION$
▶ Lazy evaluation allowed
▶ Arbitrary order of items checking

Examples

some $i$ in (1 to 10) satisfies $i > 7$
every $p$ in //person satisfies $p/surname$

XPath paths

Absolute path Relative path
/step/step ... step/step ...

Step — fully expanded syntax

axis::node-test [predicate1] ...[predicateN]

axis direction in document tree

node-test selecting nodes basing on kind, name, ...

predicates arbitrary filtering expressions

Example

/descendant::team[attribute::id = ‘3’]/child::person[1]
/child::surname/child::text()
Axis

- child
- descendant
- parent
- ancestor
- following-sibling
- preceding-sibling
- following
- preceding
- attribute
- namespace
- self
- descendant-or-self
- ancestor-or-self

Axis

not shown: attribute and namespace nodes and axes

self

ancestor

parent

preceding

following

child

descendant

descendant-or-self

ancestor-or-self
Node test

Kind of node

▶ node()
▶ text()
▶ document-node()
▶ element()
▶ attribute()
▶ processing-instruction(xml-stylesheet)

Kind and name of node

▶ element(person)
▶ attribute(id)
▶ processing-instruction(xml-stylesheet)

Node test (ctd.)

Kind and type of node

! XPath 2.0 schema aware processing
▶ element(*, studentType)
▶ element(person, studentType)
▶ attribute(*, xs:integer)
▶ attribute(id, xs:integer)

Name of node

! Kind of node default for current axis (element or attribute)
▶ person
▶ *
▶ pre:*
▶ *:person
Predicates

- Evaluated for each node selected so far (node becomes the context node).
- Every predicate filters result sequence.
- Depending on result type:
  - number — compared to item position (counted from 1)
  - not number — Effective Boolean Value used
- **Filter expressions** — predicates outside paths (XSLT 2.0)

Examples

```
/child::staff/child::person[child::name = 'Patryk']
```

```
child::person[child::name = 'Patryk']/child::surname
```

```
//person[attribute::passport][3]
```

```
(1 to 10)[. mod 2 = 0]
```

Abbreviated Syntax

- child axis may be omitted
- @ before name indicates attribute axis
- . instead of self::node()
- .. instead of parent::node()
- // instead of /descendant-or-self::node()/

Example

```
./object[@id = 'E4']
```

```
self::node()/descendant-or-self::node()/
  child::object[attribute::id = 'E4']
```
Evaluation order

- From left to right
- Step by step
  - //department/person[1]
  - (/department/person)[1]
- Predicate by predicate
  - //person[@manages and position() = 5]
  - //person[@manages][position() = 5]

XQuery — the query language for XML

Status

- XQuery 1.0 — W3C Recommendation, I 2007
- Data model, functions and operators — shared with XPath 2.0
- Formally: syntax defined from scratch
- Practically: XPath syntax extension

Features

- Picking up data from XML documents
- Constructing new result nodes
- Sorting, grouping, …
- Custom functions definition
- Various output methods (XML, HTML, XHTML, text) — shared with XSLT
XQuery query structure

- Header and body
- Header consists of declarations:
  - version declaration
  - import
  - flags and options
  - namespace declaration
  - variable or query parameter
  - function

Example

```xquery
xquery version "1.0" encoding "utf-8";
declare namespace foo = "http://example.org";
declare variable $id as xs:string external;
declare variable $doc := doc("example.xml");

$doc//foo:object[@id = $id]
```

FLWOR expression

- Acronym of **For, Let, Where, Order by, Return**
- Replaces `for` from XPath
- Motivation: SQL SELECT

Example

```xquery
for $obj in doc("example.xml")/list/object
let $prev := $obj/preceding-sibling::element()
let $prev-name := $prev[1]/@name
where $obj/@name
order by $obj/@name
return
  <div class="result">
    Object named {xs:string($obj/@name)}
    has {count($prev)} predecessors.
    The nearest predecessor name is
    {xs:string($prev-name)}.
  </div>
```
Node constructors — direct

XML document fragment within query

```xml
for $el in doc("example.xml")// * return
  <p style="color: blue">I have found an element.
  <!-- Comments and PIs also taken to result-->
  </p>
```

Expressions nested within constructors — braces

```xml
<result>
  {
    for $el in doc("example.xml")// * return
      <elem depth="{count($el/ancestor::node())}">
        {name($el)}
      </elem>
  }
</result>
```

Node constructors — computed

Syntax

```xml
for $el in doc("example.xml")// * return
element p {
  attribute style {"color: blue"},
  text { "I have found an element."},
  processing-instruction pi { "bla Bla" }
  comment { "Comments and PIs also taken to result" }
}
```

Application example — dynamically computed name

```xml
<result>
  {
    for $el in doc("example.xml")// * return
      element {concat("elem-", name($el))} {
        attribute depth {count($el/ancestor::node())},
        text {name($el)}
      }
  }
</result>
```
Custom function definitions

Example

declare function
  local:twice($x)
{ 2 * $x }; 

Type declarations example

declare function
  local:twice($x as xs:double)
as xs:double
{ 2 * $x }; 

Type declarations

- Type declarations possible (not obligatory) for:
  - variables
  - function arguments and result
  - also in XSLT 2.0
- Capabilities:
  - type name
  - node kind | node() | item()
  - occurrence modifier (?, *, +, exactly one occurrence by default).
- Examples:
  - xs:double
  - element()
  - node() *
  - xs:integer?
  - item() +