

XPath (and XQuery)

Patryk Czarnik

XML and Applications 2015/2016
Lecture 8 - 09.05.2016

Models of XML processing

- Text level processing
 - possible but inconvenient and error-prone
- Custom applications using standardised API (DOM, SAX, JAXB, etc.)
 - flexible and (relatively) efficient
 - requires some work
- XML-related standards with high-level view on documents
 - XPath, XQuery, XSLT
 - XML-oriented and (usually) more convenient than above
 - sometimes not flexible enough
- “Off the shelf” tools and solutions

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
- Extension of XPath
- 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 3.0
 - Several W3C Recommendations, IV 2014

Version numbering

- Subsequent generations of related standards.

When	XPath	XSLT	XQuery
1999	1.0	1.0	-
2007	2.0	2.0	1.0
2014	3.0	3.0 (WD)	3.0

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 here...

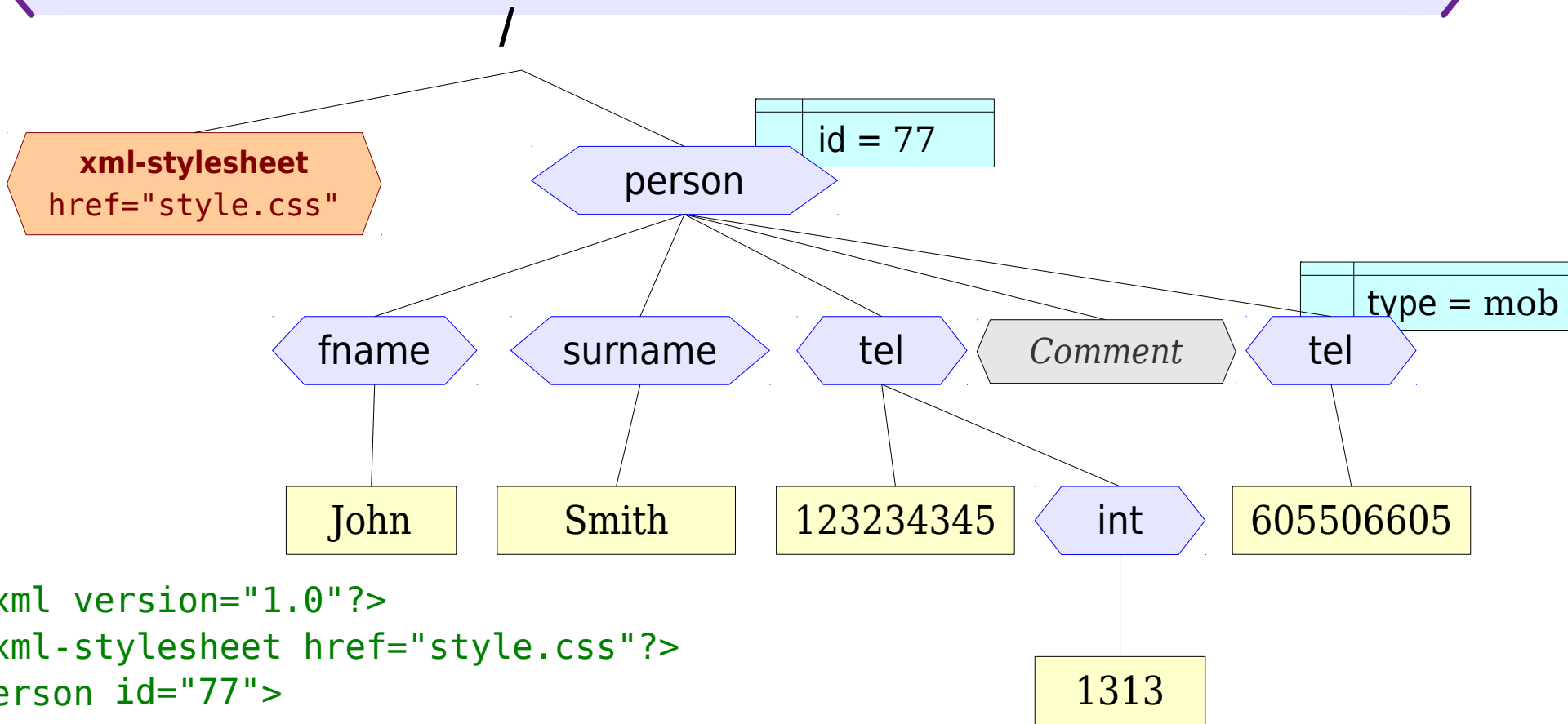
XPath (and XQuery) Data Model

- Theoretical base of XPath, XSLT, and XQuery
- XML document tree
- Structures and simple data types
- Basic operations (type conversions etc.)
- Model different in different versions of XPath
 - 1.0 – 4 value types, sets of nodes
 - 2.0 & 3.0 – XML Schema types, sequences of nodes and other values

XML document in XPath model

- Document as a tree
- Physical representation level fully expanded
 - CDATA, references to characters and entities
 - No adjacent text nodes
- Namespaces resolved 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

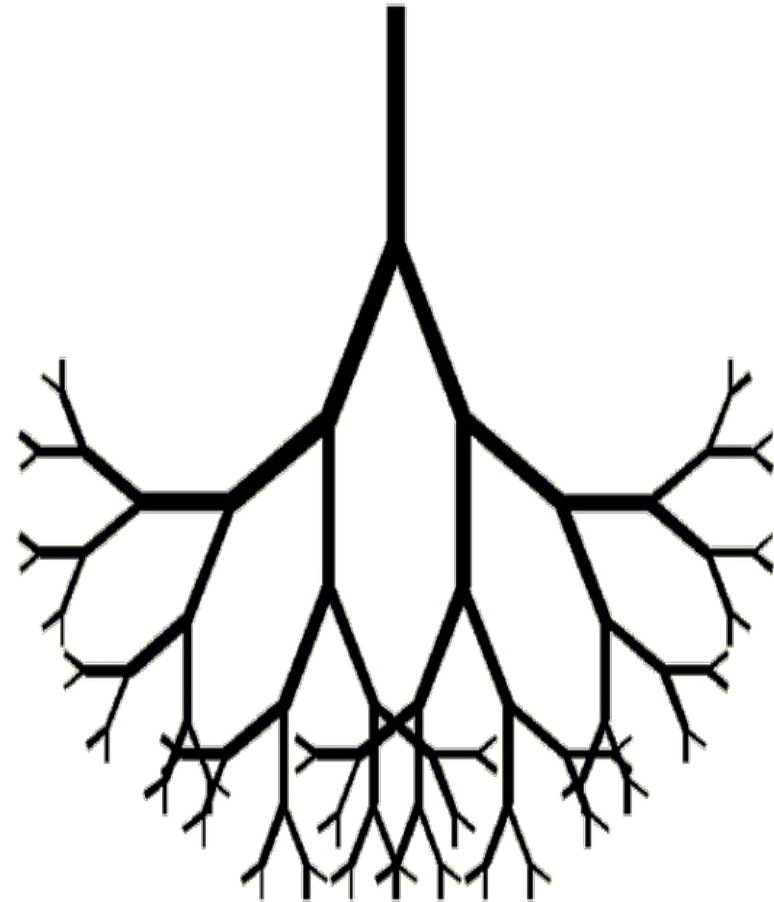
Document tree - example



```
<?xml version="1.0"?>
<?xml-stylesheet href="style.css"?>
<person id="77">
  <fname>John</fname>
  <surname>Smith</surname>
  <tel>123234345<int>1313</int></tel>
  <!-- Comment -->
  <tel type="mob">605506605</tel>
</person>
```

XPath node kinds

- Seven kinds of nodes:
 - document node (root)
 - element
 - attribute
 - text node
 - processing instruction
 - comment
 - namespace node
- Missing ones
(e.g. when compared to DOM):
 - CDATA
 - entity
 - entity reference

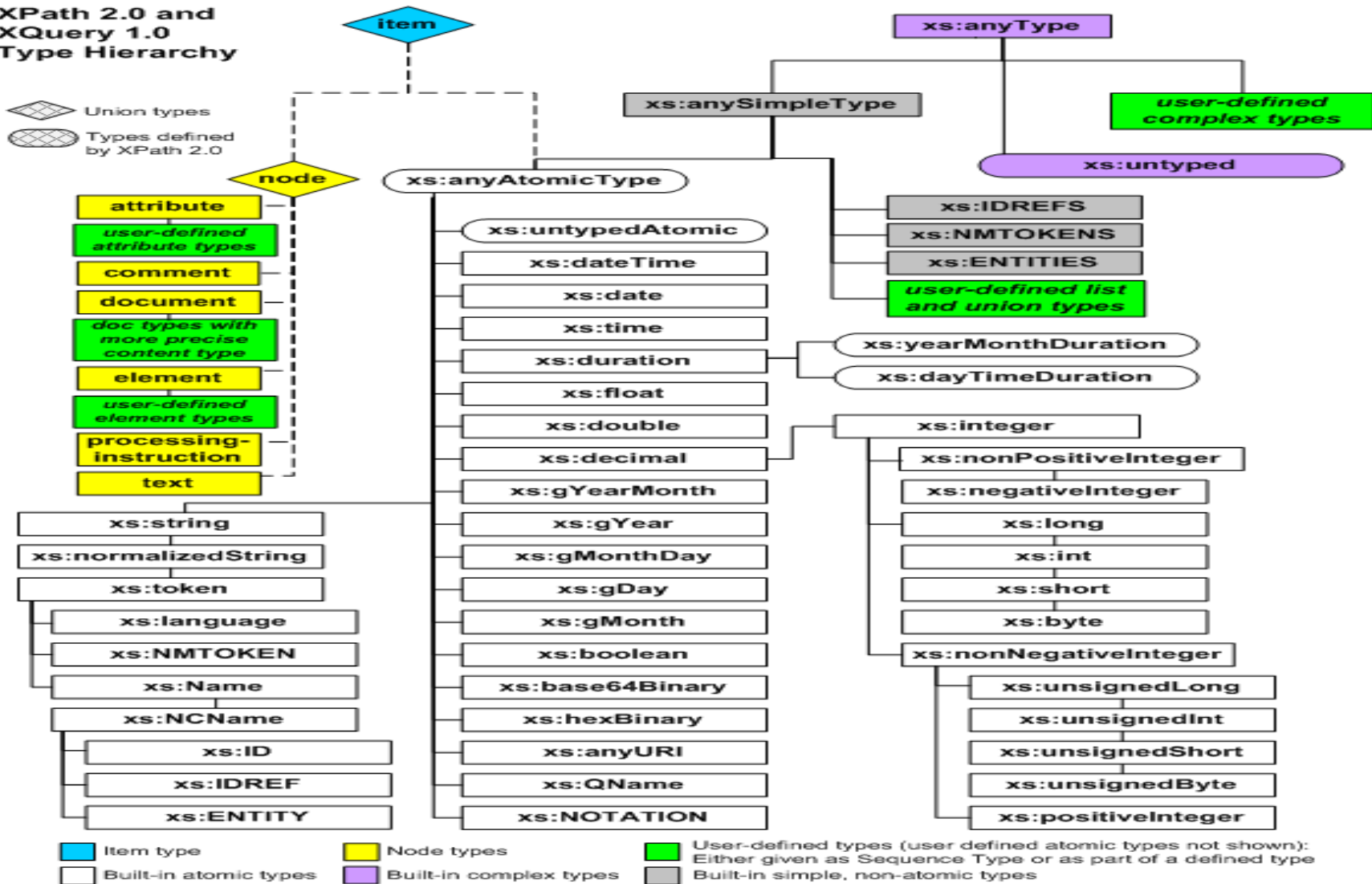


Sequences

- 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:
 $(3.14, (1, 2, 3), 'Ala') = (3.14, 1, 2, 3, 'Ala')$

Type system

XPath 2.0 and XQuery 1.0 Type Hierarchy



Data model in XPath 1.0

- Four types:
 - `boolean`
 - `string`
 - `number`
 - `node set`
- No collections of simple values
- Sets (and not sequences) of nodes

Effective Boolean Value

- Treating any value as boolean
- Motivation: convenience in condition writing, e.g. `if (customer[@passport]) then`
- Conversion rules
 - empty sequence → false
 - sequence starting with a node → true
 - single boolean value → that value
 - single empty string → false
 - single non-empty string → true
 - single number equal to 0 or NaN → false
 - other single number → true
 - other value → error

Atomization

- Treating any sequence as a sequence of atomic values
 - often with an intention to get a singleton sequence
- Motivation: comparison, arithmetic, type casting

Conversion rules (for each item)

- atomic value → that value
- node of declared atomic type → node value
- node of list type → sequence of list elements
- node of unknown simple type or one of `xs:untypedAtomic`, `xs:anySimpleType` → text content as single item
- node with mixed content → text content as single item
- node with element content → error

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 with:
 - XPath 2.0 constructs (`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 operator

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

Functions

- Function invocation:
 - `concat('Mrs ', name, ' ', surname)`
 - `count(//person)`
 - `my:factorial(12)`
- 150 built-in functions in XPath 2.0, 27 in XPath 1.0
- Abilities to define custom functions
 - 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)`

- Nodes:

`name(n?)` `local-name(n?)` `namespace-uri(n?)`

- Sequences (*some only since XPath 2.0*):

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

- Context:

`current()` `position()` `last()`

Operators

- Arithmetic
 - + - * div idiv mod
 - + - also on date/time and duration
- Logical values
 - and or
 - true(), false(), and not() are functions
- Node sets / sequences
 - union | intersect except
 - not nodes found - type error
 - result without repeats, document order preserved
- Nodes
 - is << >>

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 sequence, for which the corresponding atomic comparison holds. (Argument sequences atomized on entry.)

Typical usage

```
books/price > 100
```

“At least one of the books has price greater than 100”

General comparison – nonobvious behaviour

- Equality operator does not check the real equality
 - $(1, 2) \neq (1, 2) \rightarrow \text{true}$
 - $(1, 2) = (2, 3) \rightarrow \text{true}$
- “Equality” is not transitive
 - $(1, 2) = (2, 3) \rightarrow \text{true}$
 - $(2, 3) = (3, 4) \rightarrow \text{true}$
 - $(1, 2) = (3, 4) \rightarrow \text{false}$
- Inequality is not negation of equality
 - $(1, 2) = (1, 2) \rightarrow \text{true}$
 - $(1, 2) \neq (1, 2) \rightarrow \text{true}$
 - $() = () \rightarrow \text{false}$
 - $() \neq () \rightarrow \text{false}$

Conditional expression (XPath 2.0)

- `if (CONDITION)`
 then `RESULT1` else `RESULT2`
- Using Effective Boolean Value of `CONDITION`
- One branch evaluated

Example

```
if(details/price)
then
  if(details/price >= 1000)
  then 'Insured mail'
  else 'Ordinary mail'
else 'No data'
```

Iteration through sequence (XPath 2.0)

- `for $VAR in SEQUENCE`
 `return RESULT`
 - `VAR` takes subsequent values from `SEQUENCE`
 - `RESULT` computed that many times
 - in context where `VAR` is assigned the given value
 - overall result - (flattened) sequence of partial results

Example

```
for $i in (1 to 10)  
  return $i * $i
```

```
for $o in //obiekt  
  return concat('Nazwa obiektu:', $o/@nazwa)
```


Sequence quantifiers (XPath 2.0)

- some $\$VAR$ in *SEQUENCE* satisfies *CONDITION*
- every $\$VAR$ in *SEQUENCE* satisfies *CONDITION*
 - Using Effective Boolean Value of *CONDITION*
 - Lazy evaluation allowed
 - Evaluation order not specified

Example

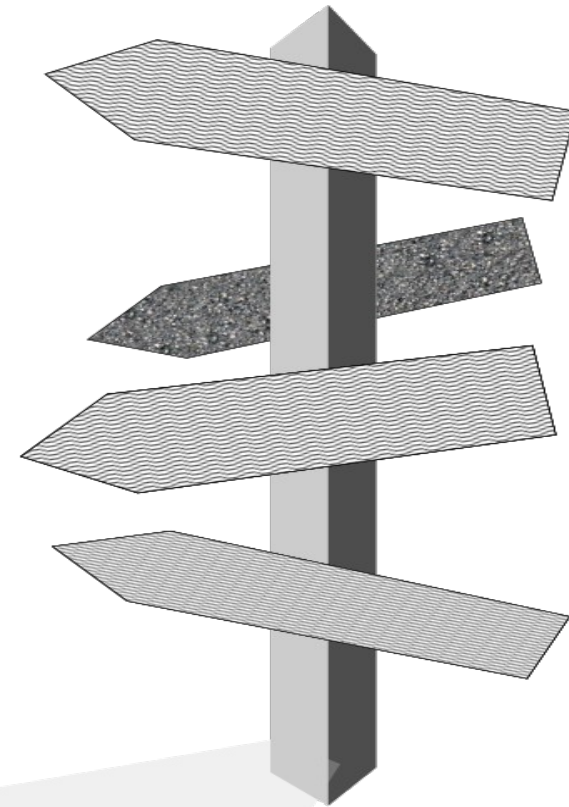
```
some $i in (1 to 10) satisfies $i > 7  
every $p in //person satisfies $p/surname
```

Paths – more formally

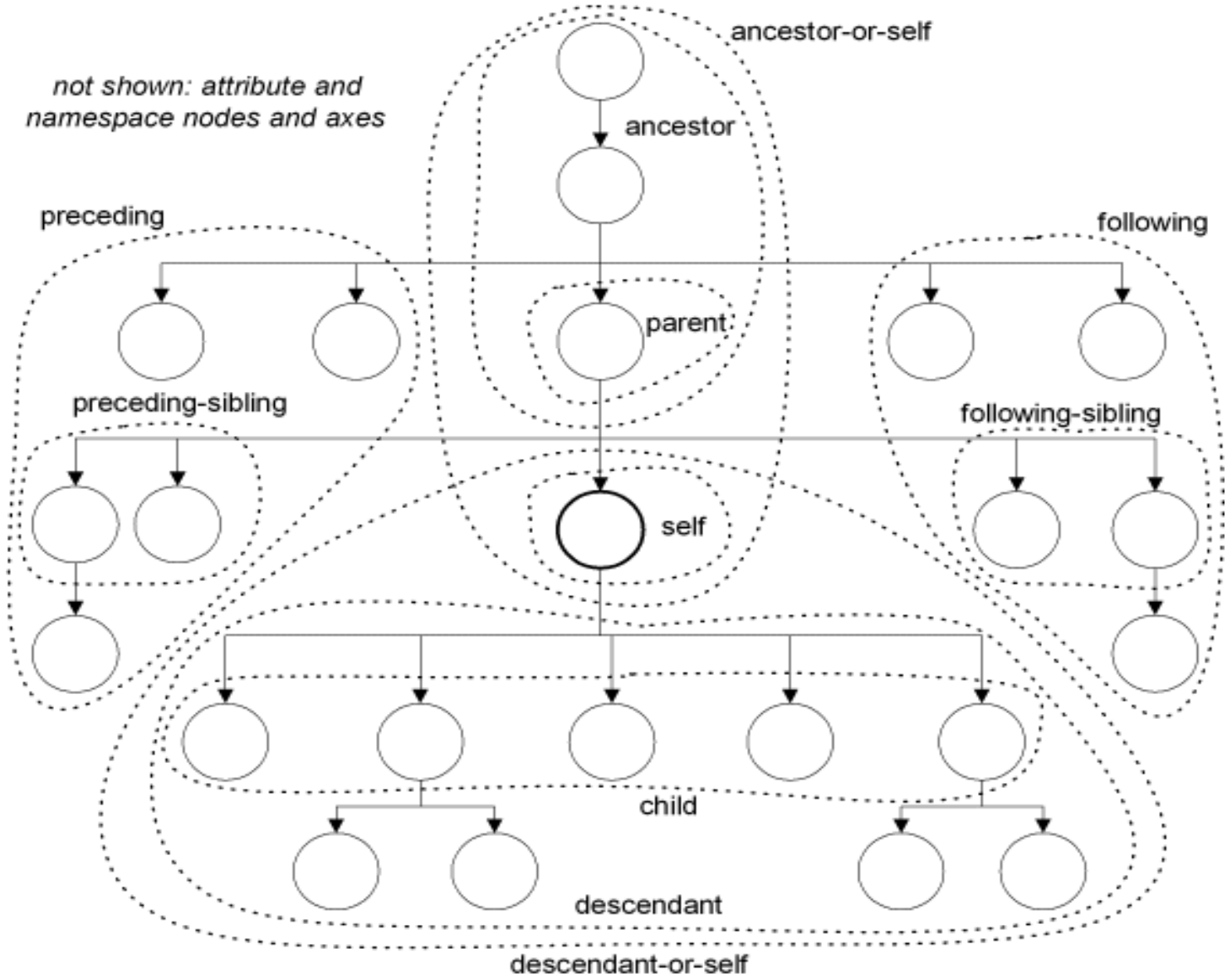
- Absolute path: `/step/step ...`
- Relative path: `step/step ...`
- Step – full syntax:
`axis::node-set [predicate1] [predicate2] ...`
 - **axis** – direction in document tree
 - **node-test** – selecting nodes by kind, name, or type
 - **predicates** – (0 or more) additional logical conditions for filtering

Axis

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



Axis



Node test

- By kind of node:
 - `node()`
 - `text()`
 - `comment()`
 - `processing-instruction()`
- By name (examples):
 - `person`
 - `pre:person`
 - `pre:*`
 - `*:person` (XPath 2.0 only)
 - `*`
- kind of node here: element or attribute, depending on axis

Node test in XPath 2.0

- In XPath 2.0 more tests, basing on kinds of nodes, and schema-provided types of nodes (“schema aware” only).
Examples:
 - `document-node()`
 - `processing-instruction(xml-styleSheet)`
 - `element()`
 - `element(person)`
 - `element(*, personType)`
 - `element(person, personType)`
 - `attribute()`
 - `attribute(id)`
 - `attribute(*, xs:integer)`
 - `attribute(id, xs:integer)`

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

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
- `.` stands for `self::node()`
- `..` stands for `parent::node()`
- `//` translated to `/descendant-or-self::node()/`
(textually, inside an expression)

Example

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

expands to

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


Evaluation order

- From left to right
- Step by step (predicate applied to the last 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
 - XQuery 3.0 – W3C Recommendation, IV 2014
 - Data model, functions and ops – shared with XPath 2.0 / 3.0
 - Syntax defined in a separate document
 - In practice: extension of XPath
- Main features:
 - Picking up data from XML documents
 - Constructing new result nodes
 - Sorting, grouping
 - Defining custom functions
 - Various output methods (XML, HTML, XHTML, text)
 - shared with XSLT

XQuery – query structure

- (Unexpectedly) XQuery is not an XML application
 - There exists a verbose XML syntax for XQuery, not intended to be written by hand
- Typical file extensions: .xquery, .xq, .xqm (for modules)
- Text format, header and body
 - header optional in normal queries
 - units declared as *modules* do not have body

Minimalistic example

```
2 + 2
```

XQuery headers

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

Example

```
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

- **F**or, **L**et, **W**here, **O**rder by, **R**eturn
 - More clauses in XQuery 3.0, but the acronym remains.
- Replaces **for** from XPath
- Obvious influence of SQL SELECT

Example

```
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

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

- Expressions nested within constructors – braces

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

Node constructors – computed

- The same example again to show the syntax

```
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

```
<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

- Simple example:

```
declare function local:factorial($n) {  
  $n * local:factorial($n - 1)  
};
```

- Example using type declarations:

```
declare function local:factorial($n as xs:integer)  
  as xs:integer {  
  $n * local:factorial($n - 1)  
};
```


Type constraints

- Type declarations possible (but not obligatory) for:
 - variables
 - function arguments and result
 - also in XSLT 2.0 (variables and parameters)
- Dynamic typing used in practical applications
 - `13 + if (aCondition) then 'not a valid number' else 1`
may fail or not depending on input data
 - `some $x in (1+1, xs:date('long long time ago'))`
`satisfies $x=2` fails or not depending on the processor
- Static typing discussed, but rarely deployed
("academic" solutions, for XQuery rather than XSLT)

Type declarations

- Capabilities:
 - type name
 - built-in - always available
 - user-defined - schema aware processors only
 - kind of node | `node()` | `item()`
 - occurrence modifier (`?`, `*`, `+`, exactly one occurrence by default).
- Examples:
 - `xs:double`
 - `element()`
 - `node()*`
 - `xs:integer?`
 - `item()+`

XQuery 3.0 – selected improvements (and XPath 3.0 to some extent)

- Minor features
 - string concatenation operator `||`
`'Her name is ' || name`
 - mapping operator `!`
`(1, 2, 3) ! (. + 5) → (6, 7, 8)`
 - switch
 - count clause in FLWOR
- More serious features
 - group-by clause and grouping “windows” in FLWOR
 - higher-level functions,
 - more predefined functions
 - in particular, more math available in separate namespace