

Chapter 5

Query Languages: XPath

- Network Data Model: no query language; only some specific commands extending the host language
- SQL – only for a flat data model, but a “nice” language (easy to learn, descriptive, relational algebra as foundation, clean theory, optimizations)
- OQL: SQL with object-orientation and path expressions
- Lorel (OEM): extension of OQL
- F-Logic: navigation in a graph by path expressions with additional conditions descriptive, complex.

192

REQUIREMENTS ON AN XML QUERY LANGUAGE

- suitable both for databases and for documents
- declarative: binding variables and using them
 - rule-based, or
 - SQL-style clause-based (which is in fact only syntactic sugar)
- binding variables in the rule body/selection clause: suitable for complex objects
 - navigation by path expressions, or
 - patterns
- generation of structure in the rule head/generating clause

193

EVOLUTION OF XPATH

- when defining a query language, constructs are needed for addressing and accessing individual elements/attributes or sets of elements/attributes.
- based on this *addressing mechanism*, a clause-based language is defined.

Early times of XML (1998)

different navigation formalisms of that kind:

- XSL Patterns (inside the stylesheet language)
- XQL (XML Query Language)
- XPointer (referencing of nodes/areas in an XML document)

used all the same basic idea with slight differences in the details:

- paths in UNIX notation
- conditions on the path

[/mondial/country\[@car_code="D"\]/city\[population > 100000\]/name](#)

194

5.1 XPath – the Basics

1999: specification of the navigation formalism as *W3C XPath*.

- Base: UNIX directory notation
 - in a UNIX directory tree: [/home/dbis/Mondial/mondial.xml](#)
 - in an XML tree: [/mondial/country/city/name](#)

Straightforward extension of the URL specification:

[http://.../dbis/Mondial/mondial.xml#mondial/country/city/name](#) [XPointer until 2002]

[http://.../dbis/Mondial/mondial.xml#xpointer\(mondial/country/city/name\)](#) [XPointer now]

- W3C: XML Path Language (XPath), Version 1.0 (W3C Recommendation 16. 11. 1999)
[http://www.w3.org/TR/xpath](#)
- W3C: XPath 2.0 and XQuery 1.0 (W3C Recommendation 23. 1. 2007)
[http://www.w3.org/TR/xquery](#)
- Tools: see Web page
 - XML (XQuery) database system “eXist”
 - lightweight tool “saxonXQ” (XQuery)

195

XPATH: NAVIGATION, SIMPLE EXAMPLES

XPath is based on the UNIX directory notation:

- `/mondial/country`
addresses all country elements in `MONDIAL`,
the result is a set of elements of the form
`<country code="..."> ... </country>`
- `/mondial/country/city`
addresses all city elements, that are direct subelements of country elements.
- `/mondial/country//city`
addresses all city elements that are subelements (in any depth) of country elements.
- `//city`
addresses all city elements in the current document.
- wildcards for element names:
`/mondial/*/name`
addresses all name elements that are grandchildren of the mondial elements
(different from `/mondial//name` which goes to arbitrary depth!)

196

... and now systematically:

XPATH: ACCESS PATHS IN XML DOCUMENTS

- Navigation paths
`/step/step.../step`
are composed by individual navigation steps,
- the result of each step is a sequence of nodes, that serve as input for the next step.
- each step consists of
`axis::nodetest[condition]*`
 - an axis (optional),
 - a test on the type and the name of the nodes,
 - (optional) predicates that are evaluated for the current node.
- paths are combined by the “/”-operator
- additionally, there are function applications
- the result of each XPath expression is a *sequence* of nodes or literals.

197

XPATH: AXES

Starting with a *current node* it is possible to navigate in an XML tree to several “directions” (cf. xmllint’s “cd”-command).

In each navigation step

path/axis::nodetest[condition]/path

the *axis* specifies in which direction the navigation takes place. Given the sequence of nodes that is addressed by *path*, for *each* node, the step is evaluated.

- Default: child axis: `child::country` \equiv `country`.
- Descendant axis: all sub-, subsub-, ... elements:
`country/ancestor::city`
selects all city elements, that are contained (in arbitrary depth) in a country element.
Note: `path//city` actually also addresses all these city elements, but “//” is *not* the exact abbreviation for “/descendant::” (see later).

198

XPATH: AXES

... another important axis:

- attribute axis:
`attribute::car_code` \equiv `@car_code`
wildcard for attributes: `attribute::*` selects all attributes of the current context node.
- and a less important:
self axis: `self::city` \equiv `./city`
selects the current element, *if* it is of the element type city.

for the above-mentioned axes there are the presented abbreviations. This is important for *XSL patterns* (see Slide 351):

XSL (match) patterns are those XPath expressions, that are built *without* the use of “axis::” (the abbreviations are allowed).

199

XPATH: AXES

Additionally, there are axes that do not have an abbreviation:

- parent axis: `//city[name="Berlin"]/parent::country`
selects the parent element of the city element that represents Berlin, *if* this is of the element type country.
(*only* the parent element, not all ancestors!)
- ancestor: all ancestors:
`//city[name="Berlin"]/ancestor::country` selects all country elements that are ancestors of the city element that represents Berlin (which results in the Germany element).
- siblings: `following-sibling::...`, `preceding-sibling::...`
for selecting nodes on the same level (especially in ordered documents).
- straightforward: “descendant-or-self” and “ancestor-or-self”.
Note: The popular short form `country//city` is defined as `country/descendant-or-self::node()/city`.
This makes a difference only in case of *context functions* (see Slide 221).

200

XPATH: AXES FOR USE IN DOCUMENT-ORIENTED XML

- following: all nodes after the context node in document order, excluding any descendants and excluding attribute nodes
- preceding: all nodes that are before the context node in document order, excluding any ancestors and excluding attribute nodes and namespace nodes

Note: For each element node x , the ancestor, descendant, following, preceding and self axes *partition* a document (ignoring attribute nodes): they do not overlap and together they contain all the nodes in the document.

Example:

Hamlet: what is the next speech of Lord Polonius after Hamlet said “To be, or not to be”?
(note: this can be in a subsequent scene or even act)

Exercise:

Provide equivalent characterizations of “following” and “preceding”

- i) in terms of “preorder” and “postorder”,
- ii) in terms of other axes.

201

XPATH: NODETEST

- The *nodetest* constrains the node type and/or the names of the selected nodes
- test if something is a node: `//city[name="Berlin"]/descendant::node()`
returns all descendant nodes.
- test if something is an element node: `//city[name="Berlin"]/descendant::element()`
returns all descendant *elements* (i.e., not the text nodes).
- test if something is a text node: `//city[name="Berlin"]/descendant::text()`
returns all descendant *text nodes*.
`//city[name="Berlin"]/population/text()`
returns the text contents of all population child elements (as a sequence of text nodes).
- test for a given element name:
`//country[name="Germany"]/descendant::element(population)`
or short form:
`//country[name="Germany"]/descendant::population`
returns all descendant *population elements*.
- "*" as wildcard: `//city[name="Berlin"]/child::*`
returns all child *elements* of any element name (analogously for `attribute::*` and `@*`).

202

XPATH: TESTS

In each step

`path/axis::nodetest[condition]/path`

condition is a predicate over XPath expressions.

- The expression selects only those nodes from the result of `path/axis::nodetest` that satisfy *condition*. *condition* contains XPath expressions that are evaluated relative to the current *context node* of the respective step.

`//country[@car_code="D"]`

returns the country element whose `car_code` attribute has the value "D"

- When comparing an element with something, the `string()` method is applied implicitly:

`//country[name = "Germany"]` is equivalent to

`//country[name/string() = "Germany"]`

- If the right hand side of the comparison is a number, the comparison is automatically evaluated on numbers:

`//country[population > 1000000]`

203

XPATH: TESTS (CONT'D)

- boolean connectives “and” and “or” in *condition*:

`//country[population > 100000000 and @area > 5000000]`

`//country[population > 100000000 or @area > 5000000]`

- boolean “not” is a *function*:

`//country[not (population > 100000000)]`

- XPath expressions in *condition* have existential semantics:

The *truth value* associated with an XPath expression is *true*, if its result set is non-empty:

`//country[inflation]`

selects those countries that have a subelement of type inflation.

⇒ formal semantics: a path expression has

- a semantics as a result set, and
- a truth value!

XPATH: TESTS (CONT'D)

- XPath expressions in *condition* are not only “simple properties of an object”, but are path expressions that are evaluated wrt. the current context node:

`//city[population/@year='1995']/name`

- Such comparisons also have existential semantics, when one comparand is a node sequence:

`//country[./city/name='Cordoba']/name`

returns the names of all countries, in which *some* city with name Cordoba is located.

`//country[not (./city/name='Cordoba')]/name`

returns the names of those countries where no city with name Cordoba is located.

XPATH: EVALUATION STRATEGY

- Input for each navigation step: A *sequence* of nodes (*context*)
- each of these nodes is considered separately for evaluation of the current step
- and returns zero or more nodes as (intermediate) result.
This intermediate result serves as context for the next step.
- finally, all partial results are collected and returned.

Example

- conditions can be applied to multiple steps

```
//country[population > 10000000]
  //city[located_on and population > 1000000]
    /name/text()
```

returns the names of all cities that have more than 1,000,000 inhabitants and are located (at least partially) on an island and in a country that has more than 10,000,000 inhabitants.

206

ABSOLUTE AND RELATIVE PATHS

So far, conditions were always evaluated only “local” to the current element on the main navigation path.

- Paths that start with a name are *relative* paths that are evaluated against the current context node (used in conditions):

```
//city[name = “Berlin”]
```

- Semijoins: comparison with results of independent “subqueries”:
Paths that start with “/” or “//” are absolute paths:

```
//country[number(@area) > //country[@car_code='B']/@area]/name
```

returns the names of all countries are bigger than Belgium.

- automatically, the string values of the attributes are taken,
- casting to number must be applied on (at least) one side.
- conflict between “/” for absolute paths and for descendant axis:

```
//country[.//city/name=“Berlin”]
(equivalent: //country[descendant::city/name=“Berlin”])
```

can be used for starting a relative path.

207

XPATH: FUNCTIONS

Input: a node/value or a set of nodes/values.

Result: in most cases a value; sometimes one or more nodes.

- dereferencing (see Slide 210)
- access to text value and node name (see Slide 213)
- aggregate functions `count(node_set)`, `sum(node_set)`

`count(/mondial/country)`

returns the number of countries.

- context functions (see Slide 220)
- access to documents on the Web:

`doc("file or url")/path`

`doc('http://www.dbis.informatik.uni-goettingen.de/index.html')//text()`

(for querying external HTML documents, consider use of namespaces as described on Slide 239 - nodetests work only with namespace!)

- see W3C document *XPath/XQuery Functions and Operators*

208

IDREF ATTRIBUTES

- ID/IDREF attributes serve for expressing cross-references
- SQL-style: (single-IDREF) references can be resolved by semi-joins: (similar to foreign keys in SQL)

`//city[@id = //organization[abbrev="EU"]/@headq]`

SQL equivalent (uncorrelated subquery):

```
SELECT *
FROM city
WHERE (name, country, province) IN
      (SELECT city, country, province
       FROM organization
       WHERE abbrev = 'EU')
```

... not a really elegant way in a graph-based data model ...
and would not work for IDREFS (white-space-separated tokens)

209

XPATH: DEREFERENCING

Access via “keys”/identifiers

The function `id(string*)` returns all elements (of the current document) whose id's are enumerated in `string*`:

- `id("D")` selects the element that represents Germany
(`country/@car_code` is declared as ID)
- `id(//country[car_code="D"]/@capital)`
yields the element node of type city that represents Berlin.

This notation is hard to read if multiple dereferencing is applied, e.g.

```
id(id( id(//organization[abbrev='IOC']/@headq)/@country)/@capital)/name
```

Alternative syntaxes:

```
//organization[abbrev='IOC']/id(@headq)/id(@country)/id(@capital)/name  
//organization[abbrev='IOC']/@headq/id(./)/@country/id(./)/@capital/id(./)/name
```

210

XPath: Dereferencing (Cont'd)

Analogously for multi-valued reference attributes (IDREFS):

- `//country[@car_code="D"]/@memberships`
returns “org-EU org-NATO ...”
- `id(//country[@car_code="D"]/@memberships)`
`//country[@car_code="D"]/id(@memberships)`
returns the set of all elements that represent an organisation where Germany is a member.
- `id(//organization[abbrev="EU"]/members/@country)`
`//organization[abbrev="EU"]/members/id(@country)`
returns all countries that are members (of some kind) in the EU.

211

Aside: Dereferencing by Navigation [Currently not supported]

Syntax:

attribute::nodetest⇒*elementtype*

Examples:

- `//country[car_code="D"]/@capital⇒city/name`
yields the element node of type city that represents Berlin.
- `//country[car_code="D"]/@memberships⇒organization`
yields elements of type organization.
- Remark: this syntax is not supported by all XPath Working Drafts:
 - XPath 1.0: no
 - has originally been introduced by Quilt (2000; predecessor of XQuery)
 - XPath 2.0: early drafts yes, later no
 - announced to be re-introduced later ...

212

XPATH: STRING() FUNCTION

The *function string()* returns the string value of a node:

- straightforward for elements with text-only contents:
`string(//country[name='Germany']/population[1])`
Note: for these (and only for these!) nodes, `text()` and `string()` have the same semantics.
- for attributes: `//country[name='Germany']/string(@area)`
Note: an attribute node is a name-value pair, not only a string (will be illustrated when constructing elements later in XQuery)!
free-standing attribute nodes as result cannot be printed!
- the `string()` function can also be appended to a path; then the argument is each of the context nodes: `//country[name='Germany']//name/string()`
- the string value of a subtree is the concatenation of all its text nodes:
`//country[@name='Germany']/string()`
Note: compare with `//country[@name='Germany']//text()` which lists all text nodes.
- `string()` *cannot* be applied to node sequences: `string(//country[name='Germany']//name)`
results in an error message.
(see W3C XPath and XQuery Functions and Operators).

213

XPATH: SOME MORE DETAILS ON COMPARISONS

- in the above examples, all predicate expressions like `[name="Berlin"]` or `[@car_code="D"]` always *implicitly* compare the string value of nodes, e.g., here the string values of `<name>Berlin</name>` or `attribute: (car_code, "D")`.

Usage of Numbers

- comparisons using `>` and `<` and a number literal given in the query implicitly cast the string values as *numeric* values.

`//city[population > 200000]`

returns the all cities with a population higher than 200,000.

`//city[population > '200000']`

returns the all cities with a population *alphabetically* "bigger" than 200,000, e.g., 3500, but not 1,000,000!

`//city[population > //city[name="Munich"]/population]`

does *not* recognize that numerical values are meant:

All cities with population alphanumerically bigger than "1244676" are returned.

`//city[population > //city[name="Munich"]/population/number()]`

It is sufficient to apply the `number()` casting function (see later) to one of the operands.

214

XPATH: COMPARISON BETWEEN NODES

Usage of Node Identity

- as seen above, the `"="` predicate uses the string values of nodes.

In most cases, this is implicitly correct:

Consider the following query: "Give all countries whose capital is the headquarter of an organization":

`//country[id(@capital)=//organization/id(@headq)]/name`

Compares the overall string values of city elements, e.g., "Brussels 4.35 50.8 951580".

- but for empty nodes, the result is not as intended ...

215

Comparison of Nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mondial-simple SYSTEM "mondial-simple.dtd">
<mondial-simple>
  <country car_code="D" capital="Berlin"/>    <city name="Berlin"/>
  <country capital="Brussels" car_code="B"/>  <city name="Brussels"/>
  <organization name="EU" headq="Brussels"/>
</mondial-simple>
```

[Filename: XPath/node-comparison.xml]

- the query `//country[id(@capital)=//organization/id(@headq)]/string(@car_code)` yields both “D” and “B” (city@name is the id attribute).
- Test for node identity see Slide 223 (since XPath 2.0).
- “deep equality” of nodes can be tested with the predicate `deep-equal(x, y)`. (by this, two subtrees are checked to have the same structure+contents (including (unordered) attribute sets))
- the query `//country[deep-equal(id(@capital), //organization/id(@headq))]/string(@car_code)` yields only “B”.

216

XPATH: PREDICATES AND OPERATIONS ON STRINGS

- `concat(string, string, string*)`
also the SQL-like infix operator `||` is allowed (since XQuery 3.0)
- `startswith(string, string)`
`//city[starts-with(name,'St.')] /name`
- `contains(string, string)`
`//city[contains(name,'bla')] /name`
- `substring-before(string, string, int?)`
- `substring-after(string, string, int?)`
- `substring(string, int, int)`: the substring consisting of i_2 characters starting with the i_1 th position.

217

XPATH: NAME FUNCTION

- the function `name()` returns the element name of the current node:
 - `name(//country[@car_code='D'])` or `//country[@car_code='D']/name()`
 - `//*[name='Monaco' and not (name()='country')]` yields only the city element for Monaco.

XPATH: IDREF FUNCTION

- the function `idref(string*)` returns all nodes that have an IDREF value that refers to one of the given strings (note that the results are attribute nodes):
`idref('D')/parent::* /name` yields the name elements of all “things” that reference Germany.

218

FUNCTIONS ON NODESETS

- Aggregation: `count(nodeset)`, `sum(nodeset)`, analogously `min`, `max`, `sum`, `avg`
`sum(//country[encompassed/id(@continent)/name="Africa"]/@area)`
`count(//country)`
all numeric functions implicitly cast to numeric values (double).
- removal of duplicates:
 - recall that the XPath strategy works on *sets of nodes* in each step - duplicate *nodes* are automatically removed:
`//country/encompassed/id(@continent)`
Starting with 244 countries, yielding a *set* of five continent nodes
 - function `distinct-values(nodeset)`:
takes the *string values* of the nodes and removes duplicates:
`doc('hamlet.xml')//SPEAKER`
returns lots of `<SPEAKER>... </SPEAKER>` *nodes*.
`distinct-values(doc('hamlet.xml')//SPEAKER)`
returns only the different (text) *values*.
- and many more (see W3C XPath/XQuery Functions and Operators).

219

XPATH: CONTEXT FUNCTIONS

- All functions retain the order of elements from the XML document (document order).
- the `position()` function yields the position of the current node in the *current result set*.

```
/mondial/country[position()=6]
```

Abbreviation: `[x]` instead of `[position()=x]`; `[last()]` yields the last node:

```
/mondial/country[population > 1000000][6]
```

selects the 6th country that has more than 1,000,000 inhabitants (in document order, not the one with the 6th highest population!)

```
/mondial/country[6][population > 1000000]
```

selects the 6th country, if it has more than 1,000,000 inhabitants.

- the `last()` function returns the position of the last elements of the current sub-results, i.e., the size of the result.

```
//country[@car_code='D']/population[position()=last()] or
```

```
//country[@car_code='D']/population[last()]
```

for the most recent (last) population count.

220

XPATH: CONTEXT FUNCTIONS (CONT'D)

- consider again the `//` abbreviation (cf. Slide 200):
 - `/mondial/descendant::city[18]` selects the 18th city in the document,
 - `/mondial/descendant-or-self::node()/city[18]` selects each city which is the 18th child of its parent (country or province).
(note that some implementations are buggy in this point ...)

- Note: above, `position()` is used for *filtering*.
Selecting and displaying the position is more complex:

```
//country[@car_code='D']/position()
```

yields “1” (for every country), since after evaluating `//country[@car_code='D']`, the context contains only one element – and its position in the context is 1.

See Slides 304 (XQuery) and Slides 393 (XSLT).

- Example queries against `mondial.xml` and `hamlet.xml`.

221

XPATH: FORWARD- AND BACKWARD AXES

- the result of each query is a *sequence of nodes*
- document order (and final results): forward
- context functions: forward or backward
- all axes enumerate results starting from the current node.
 - forward axes: child, descendant, following, following-sibling
 - backward axes: ancestor, preceding, preceding-sibling
 - `//SPEECH[contains(.,'To be, or not to be')]/preceding-sibling::SPEECH`
selects all preceding speeches.
The result is -as always- output in document order.
 - `//SPEECH[contains(.,'To be, or not to be')]/preceding-sibling::SPEECH[1]`
selects the **last preceding** speech (context function on **backward** axis)
 - undirected: self, parent, attribute.
- only relevant for queries against document-oriented XML.

222

EXTENSIONS WITH XPATH 2.0

- first draft already in 2001 after first XQuery drafts; W3C Recommendation since 2007
- more complex path constructs (alternatives, parentheses)
 - `(//city//country)[name='Monaco']`
 - `/mondial/country/(city|(province/city))/name`
- constructor “,” for sequences, e.g., to be used in (item-wise!) comparisons:
 - `/mondial/country[@car_code = ('D', 'B', 'F')]`
 - `/mondial/country[position() = (1, 5 to 9, 64)]`
yields the first, the 5th to 9th, and the 64th country
- Comparison wrt. *node identity* is done by “is”
 - recall from Slide 216: node comparison only by string value comparison or deep-equality in XPath 1.0
 - “is” requires both comparands to be single nodes; not node sequences (cf. Slide 224)
 - `//country[id(@capital) is //organization[abbrev='EU']/id(@headq)]/name`
- alignment of the whole XML world (XPath, XQuery) with datatypes (data model and XML Schema)

223

EXTENSIONS WITH XPATH 2.0: EVERY AND SOME – LOGICAL QUANTIFIERS

- logical \forall and \exists semantics for conditions:
countries where all/at least one city has more than 1000000 inhabitants:
`//country[every $p in ./city/population[last()] satisfies $p > 1000000]`
`//country[some $p in ./city/population[last()] satisfies $p > 1000000]`

Quantifiers extend the language to more than navigation

- the usage and syntax of variables is inherited from XQuery 1.0 (2001),
- quantifiers motivated by the relational calculus (recall also EXISTS from SQL),
- break with the simplicity of XPath,
- “some”? – the XPath 1.0 comparisons have existential semantics
... when sequences are allowed in the comparison; otherwise the explicit “some” has to be used:
`//country[some $org in //organization satisfies $org/id(@headq) is id(@capital)]/name`
- “every” is obviously useful (remember the usage of relational division in SQL)

224

XPath with XPath 2.0's logical quantifiers

Compare with relational algebra, relational calculus:

- **inside of “[...]”**, variables and (even nested) quantifiers are allowed:
 - selection: filters
 - projection: not supported (but inside conditions everything where a projection is used can be replaced by variables and “and”)
 - join: `some $x1 in expr1 satisfies (...(some $xn in exprn satisfies subexpr($x1...$xn))...)`
 - union: “|”, “or”
 - non-atomic negation/set difference: not
 - universal quantification: “every” or like in SQL via “not some ... not”

⇒ wrt. boolean queries (yes/no) and unary (i.e. result has a single column) queries, relational completeness is obtained.

- missing: recombination of results (joins, generation of XML structures)
- complex queries are hard to write (and to test)

Exercise

- Give the names of all organizations that have at least one member on each continent.

225

5.2 Aside: Namespaces

The names in an XML instance (i.e., tag names and the attribute names) actually consist of two parts:

- localpart + namespace (which can be empty, as in the previous examples)

Use of Namespaces

- a namespace is similar to a language: defining a set of names and sometimes having a DTD (if intended as an XML vocabulary).
- e.g. “mondial:city”, “bib:book”, “xhtml:tr” “dc:author”, “xsl:template” etc.
- used for distinguishing coinciding element names in different application areas.
- each namespace is associated with a URI (which can be a “real” URL), and abbreviated by a *namespace prefix* in the document.
- e.g., associate the namespace prefix `xhtml` with url `http://www.w3.org/1999/xhtml`. these things will become clearer when investigating the RDF, RDFS, and Semantic Web Data Models.

226

USAGE OF NAMESPACES IN XML DOCUMENTS

- each element can have (or can be in the scope of) multiple *namespace declarations* (represented by a node in the data model, similar to an attribute node).
- namespace declarations are inherited to subelements
- the element/tag name and the attribute names can then use one of the declared namespaces.
By that, every element can have one *primary namespace* and “knows” several others.

Alternatives:

1. the elements have no namespace (e.g. mondial),
 2. the document declares a default namespace (for all elements (not the attributes!) that do not get an explicit one (often in XHTML pages)),
 3. elements have an explicit namespace (multiple namespaces allowed in a document; e.g. an XSL document that operates with XHTML markup and “mondial:” nodes).
- (2) and (3) are semantically equivalent.

... see next slides.

227

EXPLICIT NAMESPACE IN AN XML DOCUMENT

```
<xh:html xmlns:xh="http://www.w3.org/1999/xhtml">
  <xh:body>
    <xh:h3>Header</xh:h3>
    <xh:a href="http://www.informatik.uni-goettingen.de">IFI</xh:a>
  </xh:body>
</xh:html>
```

[Filename: XML-DTD/xhtml-expl-namespace.xml]

- Note: attribute is not in the HTML namespace!

This is actually already not XPath, but a simple XQuery query:

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
/ht:html//ht:a/string(@href)
```

[Filename: XPath/xhtml-query.xq]

- Note: the namespace *must* be used in the query, i.e., “ht:html” is different from just “html”
- more accurate, it means something like `<{http://www.w3.org/1999/xhtml}html>...</...>` since not the chosen namespace prefix matters, but only the URI assigned to it.

228

TWO EXPLICIT NAMESPACE IN AN XML DOCUMENT

- “Dublin Core” defines a vocabulary for metadata description of resources (here: of XML documents); cf. <http://dublincore.org/documents/dces/>

```
<xh:html xmlns:xh="http://www.w3.org/1999/xhtml"
        xmlns:dc="http://purl.org/dc/elements/1.1/">
  <xh:head> <dc:creator>John Doe</dc:creator>
           <dc:date>1.1.2000</dc:date> </xh:head>
  <xh:body> ... </xh:body> </xh:html>
```

[Filename: XML-DTD/xhtml-expl-namespaces.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
declare namespace dc = "http://purl.org/dc/elements/1.1/";
/ht:html//dc:creator/text()
```

[Filename: XPath/xhtml-dc-query.xq]

- the document is *not* valid wrt. the XHTML DTD since it contains additional “alien” elements.
(combination of languages is a problem in XML – this is better solved in RDF/RDFS)
- in RDF, dc:creator from above expands to the URI <http://purl.org/dc/elements/1.1/creator>.

229

(DIFFERENT) DEFAULT NAMESPACES IN AN XML DOCUMENT

- a Default Namespace can be assigned to an element (and inherited to all its subelements where it is not overwritten):

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
  <head> <dc:creator>John Doe</dc:creator>
        <date xmlns="http://purl.org/dc/elements/1.1/">1.1.2000</date> </head>
  <body> ... </body> </html>
```

[Filename: XML-DTD/xhtml-def-namespaces.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
declare namespace dc = "http://purl.org/dc/elements/1.1/";
/ht:html/ht:head/dc:date/text()
```

[Filename: XPath/xhtml-dc-def-query.xq]

230

NAMESPACES AND ATTRIBUTES

- Namespaces are *not* inherited to attributes in any case. If an attribute should be associated with a namespace, this *must* be done explicitly:

```
<ht:html xmlns:ht="http://www.w3.org/1999/xhtml">
  <ht:body>
    <ht:a href="1+" ht:href="2-">IFI</ht:a>
    <x:a xmlns:x="http://www.w3.org/1999/xhtml" href="3+" x:href="4-">IFI</x:a>
    <a xmlns="http://www.w3.org/1999/xhtml" href="5+" ht:href="6-">IFI</a>
  </ht:body> </ht:html>
```

[Filename: XML-DTD/namespaces-attr.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
/ht:html//ht:a/@href/string()
```

[Filename: XPath/namespaces-attr-query.xq]

- the “HTML-correct” attributes “1+”, “3+”, and “5+” are returned,
- the query `/ht:html//ht:a/@href/string()` returns the “wrong” attributes “2-”, “4-”, and “6-”.

231

DECLARING NAMESPACES IN THE DTD DOCUMENT

- introduce default namespace in the DTD as attribute of the root element (e.g. in the W3C XHTML DTD) described at

https://www.w3.org/TR/xhtml1/dtds.html#a_dtd_XHTML-1.0-Strict

```
<!ELEMENT html (head, body)>
<!ATTLIST html
  xmlns %URI; #FIXED 'http://www.w3.org/1999/xhtml' >
```

- an XHTML instance (or subtree, cf. Slide 240) loading this DTD automatically extends to:

```
<html xmlns="http://www.w3.org/1999/xhtml"> <body> ... </body></html>
```

- introduce explicit namespaces as attribute of the root element:

```
<!ELEMENT html (head, body)>
<!ATTLIST html xmlns:xh %URI; #FIXED 'http://www.w3.org/1999/xhtml' >
```

This is extensively used with RDF/XML in the Semantic Web.

232

DECLARING A DEFAULT NAMESPACE IN XQUERY

XQuery allows to declare default namespaces for elements and for functions:

- are then added to each element and function step, respectively;
- not for attributes (recall that namespaces from elements are not inherited to attributes). (cf. Slide 231)

```
declare default element namespace "http://www.w3.org/1999/xhtml";
/html//a/@href/string()
```

[Filename: XPath/namespaces-default-query.xq]

- the “HTML-correct” attributes “1+”, “3+”, and “5+” are returned,
- the equivalent query is `/h:html//h:a/@href/string()`.

233

EXCLUSIVE CANONICAL XML

- Required for some applications (e.g., usage of XMLLiteral values in the “Jena” Semantic Web Framework)
- XML fragments/subtrees must be processable without their context – thus, namespaces must be present at appropriate levels in the tree.
- Details: <http://www.w3.org/TR/xml-exc-c14n/>
- in case you ever need it: can be obtained with `xmllint -exc-c14n x.xml > y.xml` (and analogously by other tools)

234

5.3 Aside: XML Catalogs

(cf. introductory note at Slide 163)

Accessing an XHTML document that contains a reference to W3Cs XHTML DTD at <http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd> via software (other than a browser) fails since the DTD is not accessible.

- an XML catalog is a dictionary *uri*→*accessible_url*:
- whenever the resource identified by *uri* is referenced, take the resource that is actually *accessible* at *accessible_url* (usually a local copy of the item).
 - DTDs
 - entity references (cf. Slide 185),
 - a graphics for an HTML ``, e.g. a company's logo
 - anything for an XML Inclusion (XInclude; cf. Slide 491)
- Software then uses a *Resolver* instance.

235

XML Catalog

- XML catalogs are XML documents themselves
- a catalog contains different subelements
- default catalog at /etc/xml/catalog (only root can change it),
- usage from several tools: put it in a central place (e.g., ~/teaching/ssd/XMLCatalog),
- if a tool or a servlet uses an own catalog (e.g., the XQuery Web interface) it can have an own, local one.
- put the DTDs (etc.) that should be made accessible somewhere, e.g., next to the catalog in a "DTD" subdirectory.

```
<?xml version="1.0"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.0//EN"
  "file:///usr/share/xml/schema/xml-core/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <system systemId="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
    uri="DTD/xhtml1-strict.dtd"/>
  <system systemId="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
    uri="DTD/xhtml1-transitional.dtd"/>
</catalog> [Filename: ~dbis/XMLTools/XMLCatalog/catalog]
```

236

Required files for XHTML

- xhtml1-strict.dtd, xhtml1-transitional.dtd,
- xhtml-lat1.ent, xhtml-symbol.ent, xhtml-special.ent

Using the XML Catalog

- software comes with a resolver, or
- get the XML Commons Resolver (resolver.jar) from Apache, put it somewhere (e.g. also below the XMLCatalog directory).
- since version 9.4 (Dec. 2011), saxon uses local copies of the W3C DTDs automatically.
- when (non-XHTML) XML documents with public DTD references are used frequently, copying them and using a catalog entry saves time and Web traffic.
- Technical description for using catalogs in saxon can be found at http://sourceforge.net/apps/mediawiki/saxon/index.php?title=XML_Catalogs and <http://saxonica.com/documentation/sourcedocs/xml-catalogs.xml>.

237

Saxon Call with Catalog until 9.3

- Java -D: set environment variable for java
- saxon -r,-x allows to refer to appropriate classes explicitly

```
java -cp $DBIS/XML-Tools/saxon/saxon9.jar:$DBIS/XML-Tools/XMLCatalog/resolver.jar \  
-Dxml.catalog.files=$DBIS/XML-Tools/XMLCatalog/catalog \  
net.sf.saxon.Query \  
-r:org.apache.xml.resolver.tools.CatalogResolver \  
-x:org.apache.xml.resolver.tools.ResolvingXMLReader \  
catalogtest.xq [Filename: XMLCatalog/saxon.call.old]
```

- (for saxonXSL: -r, -x, -y)

Shorter with -catalog (Saxon 9.4)

```
java -cp $DBIS/XML-Tools/saxon/saxon9.jar:$DBIS/XML-Tools/XMLCatalog/resolver.jar \  
net.sf.saxon.Query \  
-catalog:$DBIS/XML-Tools/XMLCatalog/catalog \  
catalogtest.xq [Filename: XMLCatalog/saxon.call]
```

```
doc('http://www.dbis.informatik.uni-goettingen.de/')
```

```
[Filename: XMLCatalog/catalogtest.xq]
```

238

EXAMPLE: QUERYING XHTML IN PRESENCE OF NAMESPACES

XHTML DTD at <http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd> contains:

```
<!ELEMENT html (head, body)>  
<!ATTLIST html id ID #IMPLIED  
xmlns %URI; #FIXED 'http://www.w3.org/1999/xhtml'>
```

Sample XHTML files:

- DBIS Web pages:

```
declare namespace h = "http://www.w3.org/1999/xhtml";  
doc('http://www.dbis.informatik.uni-goettingen.de/')//h:li/h:a/@href/string()
```

```
[Filename: XPath/web-queries.xq]
```

239

5.4 Use Case: Mondial with Embedded HTML Fragments

- define a default namespace for Mondial
 - is then inherited to all subelements (except they overwrite it, as the html element will do ...)
 - could also be declared as FIXED in the DTD (cf. the XHTML DTD)
- just use (XHTML-valid) html subelements (their default namespace URL will be FIXED in the original W3C XHTML DTD), inside the city elements,
- non-default namespace prefixes can be used, but then prefix:localname acts in the DTD like a name that contains a ":" – the prefix cannot be changed.

240

Use Case: Mondial with Embedded HTML and Dublin Core Fragments

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mondial SYSTEM "monhtml.dtd">
<mondial xmlns="https://www.semwebtech.org/mondial/10/">
  <country car_code="D" capital="berlin">
    <name>Germany</name>
    <html><head><title>GERMANY</title></head><body><p>...</p></body></html>
    <city id="berlin"><name>Berlin</name>
      <dc:author>John Doe</dc:author>
      <html><head><title>BERLIN</title></head><body><p>...</p></body></html>
    </city>
    <city id="hambg"><name>Hamburg</name>
      <dc:author>Jane Doe</dc:author>
      <html><head><title>HAMBURG</title></head><body><p>...</p></body></html>
    </city>
  </country>
</mondial>
```

[Filename: XQuery/monhtml.xml]

- Next slide: DTD that defines the XHTML default namespace by a FIXED attribute for the <html> element

241

The Mondial+HTML DTD

- extend the Mondial DTD:
 - allow for an xmlns attribute (could be FIXED),
 - allow nested `html` elements where needed,
 - add the city's `dc:author` subelement declaration stuff (with hardcoded prefix),
 - reference the XHTML DTD (by its URL) using a SYSTEM external entity.

```
<!ELEMENT mondial (country*)>
  <!ATTLIST mondial xmlns CDATA #IMPLIED>
<!ELEMENT country (name+, html?, city+)>
<!ATTLIST country car_code ID #REQUIRED
                capital IDREF #IMPLIED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT city (name+, dc:author?, html?)>
  <!ATTLIST city id ID #REQUIRED>

<!ELEMENT dc:author (#PCDATA)>
  <!ATTLIST dc:author xmlns:dc CDATA #FIXED 'http://purl.org/dc/elements/1.1/'>

<!ENTITY % htmlDTD SYSTEM 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
%htmlDTD;                                     [Filename: XQuery/monhtml.dtd]
```

242

Validate Mondial+HTML wrt. both DTDs

- use saxon to validate – it will automatically use its internal XHTML DTD
- xmllint tries (unsuccessfully) to access the W3C DTD. For it, a dirty workaround is

```
<!ENTITY % htmlDTD SYSTEM 'myxhtml.dtd'>
%htmlDTD;
```

and copy the W3C DTD to this local file (and download or remove references at its beginning).

243

Querying Mondial+HTML

- The XHTML DTD defines (fixed)

```
<!ELEMENT html (head, body)>
<!ATTLIST html xmlns %URI; #FIXED 'http://www.w3.org/1999/xhtml'>
```

- in this example mondial also defines a namespace.
- note that the “dc” prefix cannot be changed since it is hardcoded in the monhtml.dtd.

```
declare namespace mon="https://www.semwebtech.org/mondial/10/";
declare namespace h="http://www.w3.org/1999/xhtml";
declare namespace dc="http://purl.org/dc/elements/1.1/";
/mon:mondial//mon:city[dc:author]/h:title [Filename: XQuery/monhtml.xq]
```

- recall that the element namespace is *not* applied to the attributes:

```
declare namespace mon="https://www.semwebtech.org/mondial/10/";
declare namespace h="http://www.w3.org/1999/xhtml";
/mon:mondial//mon:country/id(@capital)/h:html/h:head/h:title
[Filename: XQuery/monhtml2.xq]
```

244

5.5 XPath: Conclusion

XPath without variables

- simple (and cheap) navigation language
 - only following a “main path” for addressing sets of nodes (including semijoins)
 - not “give all pairs of ...”
 - selection/filtering: yes
 - projection/reduction: no. Only complete nodes can be selected
 - join/combination: no. Only semi-joins can be expressed in the conditions
 - subqueries: inside the conditions as semijoins
 - restructuring of the results: no
- ⇒ only a fragment of a query language for addressing nodes.
- compared with SQL, XPath allows only for “SELECT **” and a unary “FROM” clause
 - XQL (Software AG, 1998/1999) for some time followed (as one of the predecessors of XPath) an approach to add join variables and constructs for projection and restructuring/grouping to the path language (cf. Slides 251 ff).

245

XPath (3.0) with “some”/“every” and Variables (cf. Slide 224)

- relational completeness (cf. SQL, relational calculus) *inside* filters
- still no generation of structures/joins on the result level.

Exercise

Consider the following query that yields the highest mountain in Africa:
(without variables, using semijoins)

```
doc('mondial.xml')//mountain[
  id(id(located/@country)/encompassed/@continent)/name='Africa'
and
not (number(elevation) <
  //mountain[
    id(id(located/@country)/encompassed/@continent)/name='Africa']/elevation)]
/name
```

[Filename: XPath/highestmountain.xq]

Give the names of all mountains that are the highest ones on the continent where they are located.

(two properties of the same object (elevation, continent) must be compared independently → **requires variable binding**)

246

IMPORTANCE OF XPATH IN THE XML-WORLD

- addressing mechanism for nodes in XML documents
- navigation in the tree structure
- serves as base for different concepts:
 - XQuery
 - XSL/XSLT: stylesheets, transformation language
 - other query languages
 - XML Schema
 - [XPointer/XLink – rarely used]

247