Chapter 5
Query Languages: XPath

- Network Data Model: no query 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.

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

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

XPATH

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:
  /home/dbis/Mondial/mondial.xml#mondial/country/city/name
  [XPointer; later]
- W3C: XML Path Language (XPath), Version 1.0
  `http://www.w3.org/TR/xpath`
- W3C: XPath 2.0 and XQuery 1.0
  `http://www.w3.org/TR/xquery`
- Tools: see Web page (2004 course)
  - XML (XQuery) database system "eXist"
  - lightweight tool "saxonXQ" (XQuery)
  - embedded in XSLT stylesheet (using any XSLT processors)

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
  adresses all city elements that are subelements of country elements.
- //city
  addresses all city elements in the current document.
- wildcards for element names:
  /mondial/country/*/capital
  addresses all capital elements that are grandchildren of country elements
  (different from /mondial/country//capital !)

... 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 set of nodes, that serve as input for the next step.
- each step consists of

  axis::nodetest[condition]

  - an axis (optional),
  - a nodetype-test,
  - a predicate (optional) that is 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.

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*

an *axis* can be specified, along which the navigation takes place. The *context node* for the step is always the node that is addressed by *path*.

- Default: child axis: child::country ≡ country.
- Self axis: self::city ≡ ./city
  selects the current element, *if* it is of the element type city.
  It can also be addressed by the current() function.
- Descendant axis: all sub-, subsub-, ... elements:
  country/descendant::city ≡ country//city
  selects all city elements, that are contained (in arbitrary depth) in a country element.
  Remark:
  note that descendant::city ≡ .//city ≢ //city

XPath: axes

... another important axis:

- attribute axis:
  attribute::car_code ≡ @car_code
  wildcard for attributes: attribute::* selects all attributes of the current context node.

for the above-mentioned axes there are the presented abbreviations. This is important for *XSL patterns* (see Slide 275):

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

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 "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 the country element that is the parent of the city element "Berlin".
  Analogously for descendant-or-self and ancestor-or-self.
- Siblings: following-sibling::..., preceding-sibling::...
  for selecting nodes on the same level (especially in ordered documents).

XPath: Nodetest

- The *nodetest* constrains the node type and the names of the selected nodes
- "*" as wildcard

XPath: Access to Text Contents and Node Names

- direct #PCDATA content of elements:
  the nodetest text() selects all text children of the current element.
  The result is a sequence of strings.
  /mondial/country/name/text()
- the function string() returns the whole (nested) text content of the first node the argument sequence:
  string(mondial/country[@car_code='D'])
- the function name() returns the element name of the current node:
  name(//city[name='Berlin'])
  //*[name='Monaco' and not (name()='country')]

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 text() method is applied implicitly:

  //country[name = "Germany"] is equivalent to
  //country[name/text() = "Germany"]

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

  //country[population > 1000000]

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='95']/name

- Such comparisons also have existential semantics:

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

  returns the names of all countries, in which a 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 set 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[@is_capital and population > 1000000]
  /name/text()

  returns the names of all cities that have more than 1,000,000 inhabitants and that are the capital of a country that has more than 10,000,000 inhabitants.

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[population > //country[@car_code=’B’]/population]/name

  returns the names of all countries that have more inhabitants than Belgium

- 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.

XPath: Functions

Input: a node/value or a set of nodes/values.
Result: in most cases a value; sometimes one or more nodes.

- access to text content and node name (see Slide 192)
- aggregate functions count(*node_set*), sum (*node_set*)

  count(/mondial/country)

  returns the number of countries.

- context functions (see Slide 199)
- dereferencing (see Slide 202)
- access to document on the Web:

  document(“*file or url*”)/*path* or doc(“*file or url*”)/*path*
  document(’http://www.dbis.informatik.uni-goettingen.de/index.html’)//text()

  (for querying external HTML documents, consider use of namespaces as described on Slide 209 - nodetests work only with namespace!)

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

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. Abbreviation: [position()=x] instead of [x]; [-1] yields the last node:

  //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!)

  //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[position()=last()]
- Example queries against hamlet.xml.

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

    //table/preceding-sibling::h4//text()

    selects all preceding h4 elements (section headers).

    The result is -as always- output in document order

    //table/preceding-sibling::h4[1]//text()

    selects the last preceding section header (context function on backward axis)
  - undirected: self, parent, attribute (and namespace)
- only relevant for queries against document-oriented XML.

IDREF ATTRIBUTES

- ID/IDREF attributes serve for expressing cross-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 ...

XPATH: DEREFERENCING

Access via "keys"/identifiers

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

- 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

Many implementations support to write the dereferencing in the navigation step:

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

or even

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

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.

Dereferencing by Navigation

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 be introduced by Quilt (2000; predecessor of XQuery)
  - XPath 2.0: early Drafts yes, later no
  - announced to be re-introduced later ...

XPath: String Functions

- concat(string, string, string*)
- 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_1th position.

Extensions with XPath 2.0

- further string- and aggregate functions
- more complex path constructs (alternatives, parentheses)
  (//city|//country)[name='Monaco']
  /mondial/country/(city|(province/city))/name
- extended subscript operator:
  //country[population > 1000000][-3]
  //country[population > 1000000][5-10]
  //country[population > 1000000][1,5-10,-3]
- ANY and ALL semantics for *condition*:
  //country[ALL city/population > 1000000]
  //country[ANY city/population > 1000000]
  (countries where all/at least one city have more than 1000000 inhabitants)
- extending the language to more than usual navigation ...
- alignment of the whole XML world (XPath, XQuery) with datatypes (data model and XML Schema)

5.1 Aside: Namespaces

- “labeling” element/attribute names as e.g. “mondial:city”, “bib:book”, “xsl:template” etc.
- each element can be assigned to a namespace (represented by a node in the data model, similar to an attribute node)
- used for distinguishing coinciding element names in different application areas
- current situation: a namespace is associated with a URL (no further semantics)
- introduced as attribute of the root element (e.g. in XHTML):

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

With this, a (nameless) namespace is declared for `html` elements which is bound to the URL ’http://www.w3.org/1999/xhtml’.

... what is now done with it?

Namespaces in an XML Document

Given:

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

- An instance to this DTD could look as follows:

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

- Can be written less explicitly:
(cf. XHTML-Example-File)

<html with implicit xmlns:xx=“...”>
    <body> ... </body></html>

since all subelements also belong to the scope of the namespace (idea: they belong to the same application area)

Scope of Namespaces

... has to be considered when stating queries:

- XSL: in the surrounding <xsl:stylesheet> element

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:ht="http://www.w3.org/1999/xhtml"
                version="1.0">
```

  tells the XSL processor that the namespace bound to 'http://www.w3.org/1999/xhtml' is denoted by "ht:" in this document.

- XPath/XQuery

```
declare namespace xhtml = "http://www.w3.org/1999/xhtml";
doc('/db/xlink.htm')//xhtml:img
```

- Queries use the namespace in the nodetest, e.g. //ht:title
- default-namespaces are inherited to all subelements,
- but not to attributes.
- the important thing of a namespace is not its (local) name, but the url where it is "identified" with!


```
<!-- XSLT Stylesheet for XPath queries against remote documents
 call: xalan -xsl web-queries.xsl -in some-xml-file
       xsltproc web-queries.xsl some-xml-file -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:ht="http://www.w3.org/1999/xhtml"
                version="1.0">
<xsl:template match="*">
 <result>
 <!-- *** query follows here, must start with document('...') *** -->
 <xsl:copy-of select="
  document('http://www.dbis.informatik.uni-goettingen.de/index.html')//ht:li
   "/>
 <!-- *** query ends here *** -->
 </result>
</xsl:template>
</xsl:stylesheet>

<!--
Queries
  against a remote XML document:
  document('http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml')//country[@car_code='D']
  against an XHTML document (note the use of a namespace! - definition above)
  document('http://www.dbis.informatik.uni-goettingen.de/index.html')//ht:li
-->
```

[Filename: XPath/web-queries.xsl]

5.2 XPath: The Limits

- addressing only sets of nodes
- not “give all pairs of ...”
- the highest mountain in Africa:

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

[Filename: XPath/highestmountain.xq]

- for *each* continent, give the highest mountain?
  not possible: two properties of the same object (height, continent) must be compared independently → requires variable binding

5.3 XPath: Conclusion

What can XPath do?

Comparison with relational operators

- selection: yes (selection of values and of (sub)structures)
- projection/reduction: no. Only complete nodes can be selected
- join/combination: no. Only semi-joins can be expressed in the filters

Other functionality:

- correlated subqueries: inside the filters as semijoins
- restructuring of the results: no
- only following a “main path” for navigating to nodes (including semijoins)

⇒ only a fragment of a query language for addressing nodes.

  – 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.

IMPORTANCE OF XPATH IN THE XML-WORLD

- adressing 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