Data-centric XML

XPath

XPath Overview

- Non-XML language for identifying particular parts of XML documents
  - First "person" element of a document
  - Seventh child element of third person element
  - ID attribute of the first person element whose contents are "Fred Jones"
  - All "xml-stylesheet" processing instructions
  - …
- Originally developed for XSLT
  - Split off XSLT to support also Xpointer
- Also integrated into XML Schema, DOM, …
- http://www.w3.org/TR/xpath
- XPath 2.0 (23 January 2007, 2nd edition 14 December 2010):
  http://www.w3.org/TR/xpath20/

XML Tree Structure according to XPath

- Document made up of nodes containing other nodes
- Seven kinds of nodes:
  - Root node
    - Like DOM, different from document element
  - Element nodes
  - Text nodes
  - Attribute nodes
    - Excludes namespace attributes
  - Comment nodes
  - Processing instruction nodes
  - Namespace nodes
- Not included: CDATA section, entity references, DTD “things”
- XPath 2: model uses infoset (possibly PSVI)

Location Path

- Typical top-level expression
- Identifies a set of nodes in the document
- Consists of “location steps”
  - Each location step is evaluated in a “context”
- Root location path: /
  - Identifies root node of the document independent of context

Child Location Steps

- A child location step selects all immediate child elements of the context
- Consists just of the element name:
  - Relative location path, e.g. “body”
  - Must have context to resolve the step
- Can be combined to form compound location paths
  - With root location path: /html
  - With compound location path, using “/” as the separator (immediate children): /html/body
  - Using “//” as the separator (all descendents): /html//p
    - Starting with // denotes all descendents of the context: //a

Attribute Location Steps

- Selects named attributes from a context
- Consists of “@” followed by the attribute name
  - //a/@href selects the href attributes of all “a” elements
  - //@id selects all “id” attributes in the document
- Location step selects the attribute nodes of the tree, not the attribute values
  - Conversion to strings will cause attribute values to be retrieved

Other Location Steps

- comment() selects all comment nodes of the context
- text() selects all text nodes in the context
  - CDATA sections and entity references are resolved
  - Each text node is the maximum contiguous text block without intervening markup (like DOM normalize())
- processing-instruction() selects all PIs in the context
  - processing-instruction('name') selects PIs with target 'name'

Wildcards

- “*” selects all elements in the context regardless of element name
  – //* selects all elements in the document
  – Can be prefixed with a namespace:
    - svg:* selects all elements with the same namespace that the svg prefix maps to
- node() selects all nodes in the context
- @* selects all attributes in the context
  – Can be prefixed again, e.g. @xlink:*

Alternatives

- “|” forms the union of selections
  - “a | link” selects all elements named “a” or “link”
  - @id|@xlink:type selects all attributes of name “id” or “xlink:type”
  - *|@* matches all element and attribute nodes

Traversing the Axis

- “..” selects the parent node
  - //@id/.. selects all element nodes which have an ID attribute
- “.” selects the context node
  - Can be used to make “//” not start at the root:
    - .//p selects all p nodes nested in the context node
  - In XSLT, used to access the string value of the current node

Predicates

- Select subset of the selected node
- Evaluated in the context of each node
- Written in square brackets:
  - //profession[. = ‘physicist’] selects all profession nodes whose string value is ‘physicist’
    - String value of an element is the text content of the element
  - //p[@id = ‘foo’] selects all “p” nodes for which the string value of the ‘id’ attribute equals ‘foo’
    - The string value of an attribute is the attribute value

Predicates (2)

- Predicate subexpressions can have multiple data types:
  - Strings, numbers, booleans, node sets
- Various operators are available:
  - Arithmetic and relational operations on numbers
    - //person[@born < 1970]
  - Relational operations on strings
  - Logical operations on booleans
- Implicit conversions between data types
- If the result is a number, the predicate holds if the position of the context node equals the number
  - person[3] selects the third “person” in the context

Unabbreviated Location Paths

- Location step consists of three parts: axis, test, and predicates
- XPath defines 13 axes:
  - ancestor: selects all ancestor nodes of the context
  - ancestor-or-self: like ancestor, but includes the context
  - attribute: selects all attributes
  - child: selects immediate child nodes
  - descendant: selects all descendents
  - descendent-or-self: like descendant, but includes the context
  - following, preceding: all nodes before or after the context (in document order)
  - following-sibling, preceding-sibling: all sibling nodes
  - parent: select the parent node
  - namespace: selects all namespaces of the context
  - self: selects the context

Unabbreviated Location Paths (2)

- child::para selects all immediate child elements of type “para”
  - Abbreviated as “para”
- child::text() selects all text node children of the context
  - Abbreviated as “text()”
- attribute::name selects all “name” attributes
  - Abbreviated as “@name”
- child::chapter/descendant::para selects all “para” descendants of all “chapter” children
  - Abbreviated as “chapter//para”
- ‘//’ is short for /descendant-or-self::node()/
- .//para is short for self::node()/descendant-or-self::node()/ child::para
  - //para[3] is the set of all para elements which are third para children

Unabbreviated Location Paths (3)

- following-sibling::chapter[1] selects the next “chapter” sibling
  - No abbreviation possible
- self::para selects the current node if it is a “para” node, else selects nothing:
  - child::*[self::chapter or self::appendix] selects all “chapter” and “appendix” children of the context
  - child::*[self::chapter or self::appendix][position()=last()] selects the last such element
- Ordering of selected nodes depends on the axis
  - An axis containing only elements before the context is a reverse axis
  - The “proximity position” always follows the order on the axis, node numbers start with 1

Syntax: Location Paths

```
[1]  LocationPath  ::=  RelativeLocationPath
                        | AbsoluteLocationPath
[2]  AbsoluteLocationPath  ::=  '/' RelativeLocationPath?
                        | AbbreviatedAbsoluteLocationPath
[3]  RelativeLocationPath  ::=  Step
                        | RelativeLocationPath '/' Step
                        | AbbreviatedRelativeLocationPath
```

Syntax: Location Steps

```
[4]   Step   ::=   AxisSpecifier NodeTest Predicate*
                   | AbbreviatedStep
[5]   AxisSpecifier   ::=   AxisName '::'
                   | AbbreviatedAxisSpecifier
```

Syntax: Node Tests

```
[7]    NodeTest    ::=    NameTest
                              | NodeType '(' ')'
                              | 'processing-instruction' '(' Literal ')'
[38]   NodeType    ::=    'comment'
                              | 'text'
                              | 'processing-instruction'
                              | 'node'
```

Syntax: Predicates

[8] Predicate ::= '[' PredicateExpr ']'

[9] PredicateExpr ::= Expr

- PredicateExpr is evaluated in the context of the selected steps
- Result is converted to boolean
  - Numbers are converted to boolean by comparing them with position ()

Syntax: Abbreviations

```
[10]   AbbreviatedAbsoluteLocationPath   ::=
              '//' RelativeLocationPath
[11]   AbbreviatedRelativeLocationPath   ::=

              RelativeLocationPath '//' Step
[12]   AbbreviatedStep   ::=    '.'
                          |     '..'
[13]   AbbreviatedAxisSpecifier   ::=   '@'?
```

Syntax: Expressions

```
[14]  Expr  ::=  OrExpr
[15]  PrimaryExpr  ::=  VariableReference
                        | '(' Expr ')'
                        | Literal
                        | Number
                        | FunctionCall
[36]  VariableReference  ::=  '$' QName
```

- Variables are provided by the XPath application as part of the context

Syntax: Function Calls

```
[16]   FunctionCall   ::=
    FunctionName '(' ( Argument ( ',' Argument )* )? ')'
[17]   Argument   ::=   Expr
[35]   FunctionName   ::=   QName - NodeType
```

- Functions are built-in or provided by the XPath application
- Arguments are converted to their argument types
  - As if by calling string(), number(), boolean() built-ins

Syntax: Node Sets

```
[18]  UnionExpr   ::=  PathExpr
                       | UnionExpr '|' PathExpr
[19]  PathExpr    ::=  LocationPath
                       | FilterExpr
                       | FilterExpr '/' RelativeLocationPath
                       | FilterExpr '//' RelativeLocationPath
[20]  FilterExpr  ::=  PrimaryExpr
                       | FilterExpr Predicate
```

Syntax: Boolean Expressions

```
[21]  OrExpr  ::=  AndExpr
                   | OrExpr 'or' AndExpr
[22]  AndExpr  ::=  EqualityExpr
                   | AndExpr 'and' EqualityExpr
[23]  EqualityExpr  ::=  RelationalExpr
                   | EqualityExpr '=' RelationalExpr
                   | EqualityExpr '!=' RelationalExpr
[24]  RelationalExpr  ::=  AdditiveExpr
                   | RelationalExpr '<' AdditiveExpr
                   | RelationalExpr '>' AdditiveExpr
                   | RelationalExpr '<=' AdditiveExpr
                   | RelationalExpr '>=' AdditiveExpr
```

Boolean Expressions

- Arguments of boolean operators (or, and) are converted to boolean first
- Comparing node sets in relational operations:
  - If both arguments are node sets:
    - True, if a node can be selected from each set so that their string values compare true
  - If one argument is a number:
    - True if a node can be converted to a string, then a number, so that it compares true
  - If one argument is a string:
    - True if a node can be converted to a string so that it compares true
  - If one argument is boolean:
    - True if the nodeset, when converted to boolean(), compares true

Boolean Expressions (2)

- Comparing other values for equality/inequality:
  - If one value is a boolean, convert the other to boolean
  - [Otherwise] If one value is a number, convert the other to a number
  - [Otherwise] convert both arguments to strings
- Comparing values for <, <=, >, >=:
  - Convert both arguments to numbers

Syntax: Numbers

```
[25]  AdditiveExpr  ::=  MultiplicativeExpr
                         | AdditiveExpr '+' MultiplicativeExpr
                         | AdditiveExpr '-' MultiplicativeExpr
[26]  MultiplicativeExpr  ::=  UnaryExpr
                  | MultiplicativeExpr MultiplyOperator UnaryExpr
                  | MultiplicativeExpr 'div' UnaryExpr
                  | MultiplicativeExpr 'mod' UnaryExpr
[27]  UnaryExpr  ::=  UnionExpr
                  | '-' UnaryExpr
[34]  MultiplyOperator  ::=  '*'
```

- Computations are floating-point normally; mod is the same as '%' in Java
- Whether "*" is a multiply operator or a wildcard depends on the lexical context

Core Functions

- Certain functions are provided built-in in XPath
  - XSLT adds more built-in functions on top of that
  - Applications may provide custom functions, in a proprietary fashion
    - Should use QNames, to scope extensions by XML namespace
- Each function defined with name, parameter types, return type, semantics

Node Functions

- *number* **last**()
- *number* **position**()
- *number* **count**(*node-set*)
- *node-set* **id**(*object*)
  - If argument is a node set, apply string() to each one, then id()
  - Otherwise: convert argument to string, split at whitespace boundaries, then find node with id
- *string* **local-name**(*node-set*?)
  - If nodeset is given, return local-name for first node, else for context node
- *string* **namespace-uri**(*node-set*?)
- *string* **name**(*node-set*?)

String Functions

- *string* **string**(*object*?)
  - Node-set: convert first node in document order into string
    - Empty string for empty node-set
  - Numbers: decimal, with sign, possibly “NaN”, “Infinity”
  - Booleans: “true”, “false”
  - Nodes: Depending on type
    - Root node/Element node: concatenation of all string values of all text node descendants
    - Attributes: attribute value
    - Namespace node: namespace URI
    - PI: PI contents
    - Comment: Comment text
    - Text: Text value (always non-empty)

String Functions (2)

- *string* **concat**(*string*, *string*, *string**)
- *boolean* **starts-with**(*string*, *string*)
- *boolean* **contains**(*string*, *string*)
- *string* **substring-before**(*string*, *string*)
- *string* **substring-after**(*string*, *string*)
- *string* **substring**(*string*, *number*, *number*?)
  - Character indices start at 1, indices are rounded
- *number* **string-length**(*string*?)
- *string* **normalize-space**(*string*?)
- *string* **translate**(*string*, *string*, *string*)

Boolean Functions

- *boolean* **boolean**(*object*)
  - Number: true if != +/-0, !=NaN
  - Node-set: true if non-empty
  - String: true if length is non-zero
- *boolean* **not**(*boolean*)
- *boolean* **true**()
- *boolean* **false**()
- *boolean* **lang**(*string*)
  - Looks for xml:lang in the context node
  - Case-insensitive, ignoring country separated by “-”

Number Functions

- *number* **number**(*object*?)
  - Strings: convert to nearest IEEE-754 number, or NaN
  - Boolean: true gives 1, false gives 0
  - Node-set: convert to string first
- *number* **sum**(*node-set*)
- *number* **floor**(*number*)
- *number* **ceiling**(*number*)
- *number* **round**(*number*)

XPath 2

- developed together with XQuery
  - XQuery is a superset of XPath2
- single value type: sequence of items (atomic value or node)
  - Data types integrated with XML Schema
  - syntax for complex types (e.g. element(a)+), to use in type tests and casts
- static vs. dynamic context
  - static: configuration of evaluator, e.g. XPath 1.0 compatibility mode, in-scope schema definitions, ...
  - dynamic: information at evaluation time, e.g. context item, current dateTime, ...
- comment syntax (: Houston, we have a problem :)
- syntax for sequence expressions (10, 1 to 4)
- predicate calculus operations (for, if, some, every)
  - for $a in ../@age return $a + 1