XPath

Web Data Management and Distribution

Serge Abiteboul  Ioana Manolescu  Philippe Rigaux
Marie-Christine Rousset  Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

June 23, 2010
Outline

1. Introduction
2. Path Expressions
3. Operators and Functions
4. XPath examples
5. XPath 2.0
6. Reference Information
7. Exercise
**XPath**

- An **expression language** to be used in another host language (e.g., XSLT, XQuery).
- Allows the description of **paths** in an XML tree, and the retrieval of nodes that match these paths.
- Can also be used for performing some (limited) operations on XML data.

**Example**

\[2 \times 3\] is an XPath **literal expression**.

\[//[@msg="Hello world"]\] is an XPath **path expression**, retrieving all elements with a `msg` attribute set to “Hello world”.

**Content of this presentation**

Mostly XPath 1.0: a W3C recommendation published in 1999, widely used. Also a **basic** introduction to XPath 2.0, published in 2007.
XPath Data Model

XPath expressions operate over XML trees, which consist of the following node types:

- **Document**: the root node of the XML document;
- **Element**: element nodes;
- **Attribute**: attribute nodes, represented as children of an Element node;
- **Text**: text nodes, i.e., leaves of the XML tree.

**Remark**

**Remark 1** The XPath data model features also **ProcessingInstruction** and **Comment** node types.

**Remark 2** Syntactic features specific to serialized representation (e.g., entities, literal section) are ignored by XPath.
Introduction

From serialized representation to XML trees

```xml
<?xml version="1.0" encoding="utf-8"?>
<A>
  <B att1='1'>
    <D>Text 1</D>
    <D>Text 2</D>
  </B>
  <B att1='2'>
    <D>Text 3</D>
  </B>
  <C att2="a" att3="b"/>
</A>
```
XPath Data Model (cont.)

- The root node of an XML tree is the (unique) Document node;
- The root element is the (unique) Element child of the root node;
- A node has a name, or a value, or both
  - an Element node has a name, but no value;
  - a Text node has a value (a character string), but no name;
  - an Attribute node has both a name and a value.
- Attributes are special! Attributes are not considered as first-class nodes in an XML tree. They must be addressed specifically, when needed.

Remark

The expression “textual value of an Element N” denotes the concatenation of all the Text node values which are descendant of N, taken in the document order.
Outline

1. Introduction

2. Path Expressions
   - Steps and expressions
   - Axes and node tests
   - Predicates

3. Operators and Functions

4. XPath examples

5. XPath 2.0

6. Reference Information
XPath Context

A step is evaluated in a specific context \([< N_1, N_2, \ldots, N_n >, N_c]\) which consists of:

a context list \(< N_1, N_2, \ldots, N_n >\) of nodes from the XML tree;

a context node \(N_c\) belonging to the context list.

Information on the context

- The context length \(n\) is a positive integer indicating the size of a contextual list of nodes; it can be known by using the function \texttt{last()};

- The context node position \(c \in [1, n]\) is a positive integer indicating the position of the context node in the context list of nodes; it can be known by using the function \texttt{position()}.  

WebDam (INRIA)
XPath
June 23, 2010
XPath steps
The basic component of XPath expression are steps, of the form:

\[ \text{axis} :: \text{node-test} [P_1] [P_2] \ldots [P_n] \]

- **axis** is an **axis name** indicating what the direction of the step in the XML tree is (*child* is the default).
- **node-test** is a **node test**, indicating the kind of nodes to select.
- **P_i** is a **predicate**, that is, any XPath expression, evaluated as a boolean, indicating an additional condition. There may be no predicates at all.

Interpretation of a step
A step is evaluated with respect to a **context**, and returns a **node list**.

Example
\[ \text{descendant} :: C[@att1='1'] \] is a step which denotes all the **Element** nodes named \( C \) having an **Attribute** node \( \text{att1} \) with value 1.
Path Expressions

A path expression is of the form: \[/\]step\(_1\) / step\(_2\) / \ldots / step\(_n\)

A path that begins with / is an **absolute** path expression;
A path that does not begin with / is a **relative** path expression.

**Example**

/\textit{A}/B is an **absolute** path expression denoting the **Element** nodes with name \textit{B}, children of the root named \textit{A};

./\textit{B}/descendant::text() is a **relative** path expression which denotes all the **Text** nodes descendant of an **Element** \textit{B}, itself child of the context node;

/\textit{A}/\textit{B}/@\textit{att1}[. > 2] denotes all the **Attribute** nodes @\textit{att1} whose value is greater than 2.

. is a special step, which refers to the context node. Thus, ./\textit{toto} means the same thing as \textit{toto}.
Evaluation of Path Expressions

Each step $step_i$ is interpreted with respect to a context; its result is a node list.

A step $step_i$ is evaluated with respect to the context of $step_{i-1}$. More precisely:

For $i = 1$ (first step) if the path is absolute, the context is a singleton, the root of the XML tree; else (relative paths) the context is defined by the environment;

For $i > 1$ if $\mathcal{N} = \langle N_1, N_2, \ldots, N_n \rangle$ is the result of step $step_{i-1}$, $step_i$ is successively evaluated with respect to the context $[\mathcal{N}, N_j]$, for each $j \in [1, n]$.

The result of the path expression is the node set obtained after evaluating the last step.
Evaluation of /A/B/@att1

The path expression is absolute: the context consists of the root node of the tree.

The first step, A, is evaluated with respect to this context.
Evaluation of \texttt{/A/B/@att1} 

The result is A, the root element.

A is the context for the evaluation of the second step, B.
Evaluation of \texttt{/A/B/@att1}

The result is a node list with two nodes \texttt{B[1]}, \texttt{B[2]}.

\texttt{@att1} is first evaluated with the context node \texttt{B[1]}. 
Evaluation of /A/B/@att1

The result is the attribute node of B[1].
Evaluation of /A/B/@att1

@att1 is also evaluated with the context node B[2].
Evaluation of \( /A/B/@\text{att1} \)

The result is the attribute node of \( B[2] \).
Evaluation of /A/B/@att1
Final result: the node set union of all the results of the last step, @att1.
Axes

An axis = a set of nodes determined from the context node, and an ordering of the sequence.

- child (default axis).
- parent Parent node.
- attribute Attribute nodes.
- descendant Descendants, excluding the node itself.
- descendant-or-self Descendants, including the node itself.
- ancestor Ancestors, excluding the node itself.
- ancestor-or-self Ancestors, including the node itself.
- following Following nodes in document order.
- following-sibling Following siblings in document order.
- preceding Preceding nodes in document order.
- preceding-sibling Preceding siblings in document order.
- self The context node itself.
Examples of axis interpretation

**Child axis**: denotes the **Element** or **Text** children of the context node.

**Important**: An **Attribute** node has a parent (the element on which it is located), but an attribute node is *not* one of the children of its parent.

**Result of child::D (equivalent to D)**

![Diagram showing the result of child::D](image)
Examples of axis interpretation

Parent axis: denotes the parent of the context node.
The node test is either an element name, or * which matches all names, node() which matches all node types. Always a Element or Document node, or an empty node-set (if the parent does not match the node test or does not satisfy a predicate).

.. is an abbreviation for parent::node(): the parent of the context node.

Result of parent::node() (may be abbreviated to ..)

```
Document
  Element A
    Element B
      Element D
        Text
          Text 1
        Text
          Text 2
      Element D
        Text
          Text 3
    Element B
      Element D
        Attr
          att1
          2
        Attr
          att2
          3
```
Examples of axis interpretation

**Attribute axis**: denotes the attributes of the context node. The node test is either the attribute name, or \* which matches all the names.

**Result of attribute::* (equiv. to @*)**

```
Document
  +-- Element A
     +-- Element B
        +-- Element D
            +-- Text - Text 1
        +-- Element D
            +-- Text - Text 2
          +-- Element D
            +-- Text - Text 3
        +-- Element B
          +-- Element D
            +-- Attr att1 2
          +-- Element D
            +-- Attr att2 3
        +-- Element C
```
Examples of axis interpretation

**Descendant axis:** all the descendant nodes, except the **Attribute** nodes.

The node test is either the node name (for **Element** nodes), or `*` (any **Element** node) or `text()` (any **Text** node) or `node()` (all nodes).

The context node does not belong to the result: use **descendant-or-self** instead.

Result of `descendant::node()`
Examples of axis interpretation

**Descendant axis:** all the descendant nodes, except the **Attribute** nodes.

The node test is either the node name (for **Element** nodes), or \(*\) (any **Element** node) or \(\text{text()}\) (any **Text** node) or \(\text{node()}\) (all nodes).

The context node does not belong to the result: use **descendant-or-self** instead.

**Result of descendant:: ***

![Diagram of XPath expressions]

- **Document**
- **Element A**
- **Element B**
- **Element C**
- **Element D**
- **Text:** Text 1, Text 2, Text 3
- **Attributes:** att1, att2
Examples of axis interpretation

**Ancestor axis:** all the ancestor nodes.
The node test is either the node name (for Element nodes), or node () (any Element node, and the Document root node).
The context node does not belong to the result: use ancestor-or-self instead.

**Result of ancestor::node()**

![Diagram of XPath ancestor axis example]
Examples of axis interpretation

**Following axis**: all the nodes that follows the context node in the document order.  
**Attribute** nodes are *not* selected.  
The node test is either the node name, *text()* or *node()*.

The axis **preceding** denotes all the nodes the precede the context node.

**Result of** following::node()

```
Document
  Element A
    Element B
      Element D
        Text
          Text 1
        Text
          Text 2
    Element D
      Text
        Text
          Text 3
  Element D
    Attr att1 2
  Element C
    Attr att2 3
```
Examples of axis interpretation

**Following sibling axis:**
all the nodes that follow the context node, and share the same parent node.

Same node tests as descendant or following.
The axis preceding-sibling denotes all the nodes the precede the context node.

Result of `following-sibling::node()`
Abbreviations (summary)

Summary of abbreviations:

<table>
<thead>
<tr>
<th>somename</th>
<th>child::somename</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>self::node()</td>
</tr>
<tr>
<td>..</td>
<td>parent::node()</td>
</tr>
<tr>
<td>@somen attr</td>
<td>attribute::somen attr</td>
</tr>
<tr>
<td>a//b</td>
<td>a/descendant-or-self::node()/b</td>
</tr>
<tr>
<td>//a</td>
<td>/descendant-or-self::node()/a</td>
</tr>
<tr>
<td>/</td>
<td>/self::node()</td>
</tr>
</tbody>
</table>

Examples

@b selects the b attribute of the context node.

../* selects all siblings of the context node, itself included (unless it is an attribute node).

//@somen attr selects all some attr attributes wherever their position in the document.
Node Tests (summary)

A node test has one of the following forms:

- `node()` any node.
- `text()` any text node.
- `*` any element (or any attribute for the attribute axis).
- `ns:*` any element or attribute in the namespace bound to the prefix `ns`.
- `ns:toto` any element or attribute in the namespace bound to the prefix `ns` and whose name is `toto`.

Examples

- `a/node()` selects all nodes which are children of a `a` node, itself child of the context node.
- `xsl:*` selects all elements whose namespace is `ns` and that are children of the context node.
- `/ *` selects the top-level element node.
XPath Predicates

- Boolean expression, built with tests and the Boolean connectors and and or (negation is expressed with the not() function);
- a test is
  - either an XPath expression, whose result is converted to a Boolean;
  - a comparison or a call to a Boolean function.

Important: predicate evaluation requires several rules for converting nodes and node sets to the appropriate type.

Example

- //B[@att1=1]: nodes B having an attribute att1 with value 1;
- //B[@att1]: all nodes B having an attributes named att1!
  ⇒ @att1 is an XPath expression whose result (a node set) is converted to a Boolean.
- //B/descendant::*text()[position()=1]: the first Text node descendant of each node B.
  Can be abbreviated to //B/descendant::*text()[1].
Predicate evaluation

A step is of the form

\texttt{axis::node-test[P]}.

- First
  \texttt{axis::node-test}
  is evaluated: one obtains an intermediate result \textit{I}.

- Second, for each node in \textit{I}, \textit{P} is evaluated: the step result consists of those nodes in \textit{I} for which \textit{P} is true.

\textbf{Ex.}: \texttt{/A/B/descendant::text()[1]}
Predicate evaluation

Beware: an XPath step is always evaluated with respect to the context of the previous step. Here the result consists of those Text nodes, first descendant (in the document order) of a node B.
### XPath 1.0 Type System

Four primitive types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Literals</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Boolean values</td>
<td>none</td>
<td>true(), not($a=3)</td>
</tr>
<tr>
<td>number</td>
<td>Floating-point</td>
<td>12, 12.5</td>
<td>1 div 33</td>
</tr>
<tr>
<td>string</td>
<td>Ch. strings</td>
<td>&quot;to&quot;, 'ti'</td>
<td>concat('Hello', '!', '!'')</td>
</tr>
<tr>
<td>nodeset</td>
<td>Node set</td>
<td>none</td>
<td>/a/b[c=1 or @e]/d</td>
</tr>
</tbody>
</table>

The `boolean()`, `number()`, `string()` functions convert types into each other (no conversion to nodesets is defined), but this conversion is done in an implicit way most of the time.

Rules for converting to a boolean:

- A number is true if it is neither 0 nor NaN.
- A string is true if its length is not 0.
- A nodeset is true if it is not empty.
Rules for converting a nodeset to a string:

- The string value of a nodeset is the string value of its first item in document order.
- The string value of an element or document node is the concatenation of the character data in all text nodes below.
- The string value of a text node is its character data.
- The string value of an attribute node is the attribute value.

Examples (Whitespace-only text nodes removed)

```xml
<a toto="3">
  <b titi='tutu'><c /></b>
  <d>tata</d>
</a>
```

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>string(//)</td>
<td>&quot;tata&quot;</td>
</tr>
<tr>
<td>string(/a/@toto)</td>
<td>&quot;3&quot;</td>
</tr>
<tr>
<td>boolean(/a/b)</td>
<td>true()</td>
</tr>
<tr>
<td>boolean(/a/e)</td>
<td>false()</td>
</tr>
</tbody>
</table>
Outline

1 Introduction

2 Path Expressions

3 Operators and Functions

4 XPath examples

5 XPath 2.0

6 Reference Information

7 Exercise
Operators

The following operators can be used in XPath.

+, -, *, div, mod  standard arithmetic operators (Example: 1+2*-3).

Warning! div is used instead of the usual /.

or, and  boolean operators (Example: @a and c=3)

=, !=  equality operators. Can be used for strings, booleans or numbers. Warning! //a!=3 means: there is an a element in the document whose string value is different from 3.

<, <=, >=, >  relational operators (Example: ($a<2) and ($a>0)).

Warning! Can only be used to compare numbers, not strings. If an XPath expression is embedded in an XML document, < must be escaped as &lt;.

| union of nodesets (Example: node() | @*)

Remark

$a$ is a reference to the variable a. Variables can not be defined in XPath, they can only be referred to.
Node Functions

`count($s)` returns the number of items in the nodeset $s$

`local-name($s)` returns the name of the first item of the nodeset $s$ in document order, without the namespace prefix; if $s$ is omitted, it is taken to be the context item

`namespace-uri($s)` returns the namespace URI bound to the prefix of the name of the first item of the nodeset $s$ in document order; if $s$ is omitted, it is taken to be the context item

`name($s)` returns the name of the first item of the nodeset $s$ in document order, including its namespace prefix; if $s$ is omitted, it is taken to be the context item
String Functions

concat($s_1, \ldots, s_n)$ concatenates the strings $s_1, \ldots, s_n$

starts-with($a$, $b$) returns true() if the string $a$ starts with $b$

contains($a$, $b$) returns true() if the string $a$ contains $b$

substring-before($a$, $b$) returns the substring of $a$ before the first occurrence of $b$

substring-after($a$, $b$) returns the substring of $a$ after the first occurrence of $b$

substring($a$, $n$, $l$) returns the substring of $a$ of length $l$ starting at index $n$ (indexes start from 1). $l$ may be omitted.

string-length($a$) returns the length of the string $a$

normalize-space($a$) removes all leading and trailing whitespace from $a$, and collapse all whitespace to a single character

translate($a$, $b$, $c$) returns the string $a$, where all occurrences of a character from $b$ has been replaced by the character at the same place in $c$. 

Operators and Functions

Boolean and Number Functions

- `not($b)` returns the **logical negation** of the boolean `$b`
- `sum($s)` returns the **sum** of the values of the nodes in the nodeset `$s$
- `floor($n)` rounds the number `$n` to the **next lowest** integer
- `ceiling($n)` rounds the number `$n` to the **next greatest** integer
- `round($n)` rounds the number `$n` to the **closest** integer

Examples

- `count(//*)` returns the number of elements in the document
- `normalize-space('tititotototo')` returns the string “tititotototo”
- `translate('bababababababcdef', 'ABCDEF')` returns the string “BABA”
- `round(3.457)` returns the number 3
Outline

1. Introduction
2. Path Expressions
3. Operators and Functions
4. XPath examples
5. XPath 2.0
6. Reference Information
7. Exercise
Examples (1)

child::A/descendant::B : B elements, descendant of an A element, itself child of the context node;
Can be abbreviated to A://B.

child::*/child::B : all the B grand-children of the context node:

descendant-or-self::B : elements B descendants of the context node, plus the context node itself if its name is B.

child::B[position()=last()] : the last child named B of the context node.
Abbreviated to B[last()]

following-sibling::B[1] : the first sibling of type B (in the document order) of the context node,
Examples (2)

Not: the tenth element of the document, if its type is B!
child::*B[child::*C] : child elements B that have a child element C.
Abbreviated to B[C].
/descendant::*B[@att1 or @att2] : elements B that have an attribute att1 or an attribute att2;
Abbreviated to //B[@att1 or @att2]
*[self::*B or self::*C] : children elements named B or C
Outline

1. Introduction
2. Path Expressions
3. Operators and Functions
4. XPath examples
5. XPath 2.0
6. Reference Information
7. Exercise
XPath 2.0

An extension of XPath 1.0, backward compatible with XPath 1.0. Main differences:

**Improved data model** tightly associated with XML Schema.

- ⇒ a new **sequence** type, representing ordered set of nodes and/or values, with duplicates allowed.
- ⇒ XSD types can be used for node tests.

**More powerful** new operators (loops) and better control of the output (limited tree restructuring capabilities)

**Extensible** Many new built-in functions; possibility to add user-defined functions.

XPath 2.0 is also a subset of XQuery 1.0.
Path expressions in XPath 2.0

New node tests in XPath 2.0:

- `item()` any node or atomic value
- `element()` any element (eq. to `child::*` in XPath 1.0)
- `element(author)` any element named `author`
- `element(*, xs:person)` any element of type `xs:person`
- `attribute()` any attribute

Nested paths expressions:
Any expression that returns a sequence of nodes can be used as a step.

```
/book/(author | editor)/name
```
Outline

1. Introduction
2. Path Expressions
3. Operators and Functions
4. XPath examples
5. XPath 2.0
6. Reference Information
7. Exercise
XPath 1.0 Implementations

Large number of implementations.

- **libxml2**  Free C library for parsing XML documents, supporting XPath.
- **java.xml.xpath**  Java package, included with JDK versions starting from 1.5.
- **System.Xml.XPath**  .NET classes for XPath.
- **XML::XPath**  Free Perl module, includes a command-line tool.
- **DOMXPath**  PHP class for XPath, included in PHP5.
- **PyXML**  Free Python library for parsing XML documents, supporting XPath.
References

- http://www.w3.org/TR/xpath
- *XML in a nutshell*, Eliotte Rusty Harold & W. Scott Means, O’Reilly
Outline

1. Introduction
2. Path Expressions
3. Operators and Functions
4. XPath examples
5. XPath 2.0
6. Reference Information
7. Exercise
Exercise

We suppose that all text nodes containing only whitespace are removed from the tree.

- Give the result of the following XPath expressions:
  - //e/preceding::text()
  - count(//c|//b/node())

- Give an XPath expression for the following problems, and the corresponding result:
  - Sum of all attribute values
  - Text content of the document, where every “b” is replaced by a “c”
  - Name of the child of the last “c” element in the tree