XPath

Arantza Irastorza Goñi

eman ta zabal zazu

Informazioaren Kudeaketa Aurreratua

Gradua Ingeniaritza Informatikoan

Esp. Software Ingeniaritza

Lengoaiak eta Sistema Informatikoak saila

informatika fakultatea

facultad de informática

2017 urtarrila

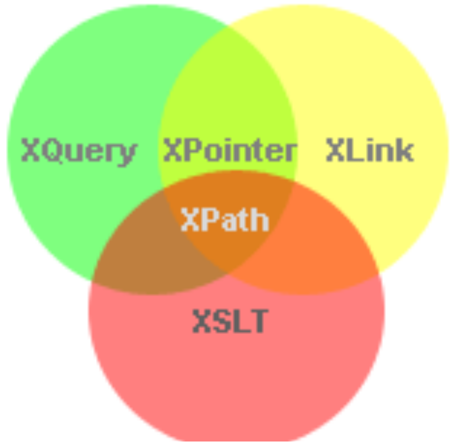
Contents

What is XPath?

- A language for addressing parts of an XML document

- A query language whose syntax uses path expressions on the document

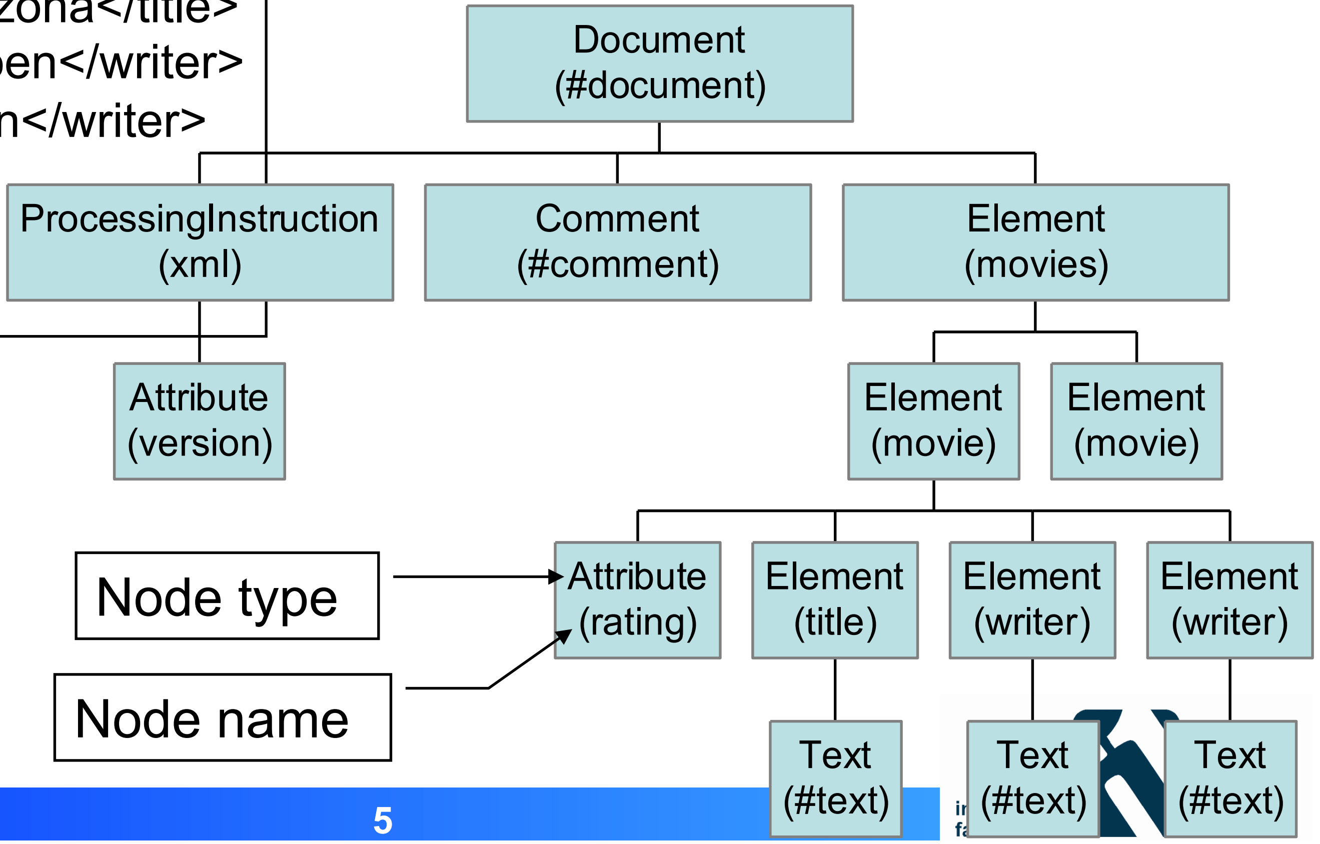
- XPath is used in other W3C specifications
  - *XSLT*
  - *XLink*
  - *XQuery*



XPath is a functional language: input: tree → output: node set

```
<?xml version="1.0" ?>
<!-- Nire pelikula gogozkoenak -->
<movies>
   <movie rating="PG-13" >
      <title>Raising Arizona</title>
      <writer>Ethan Coen</writer>
      <writer>Joel Coen</writer>
   </movie>
<movie> .... </movie>
</movies>
```

Relationship of nodes: parent, child, sibling, ancestor, descendant



XPath Expressions

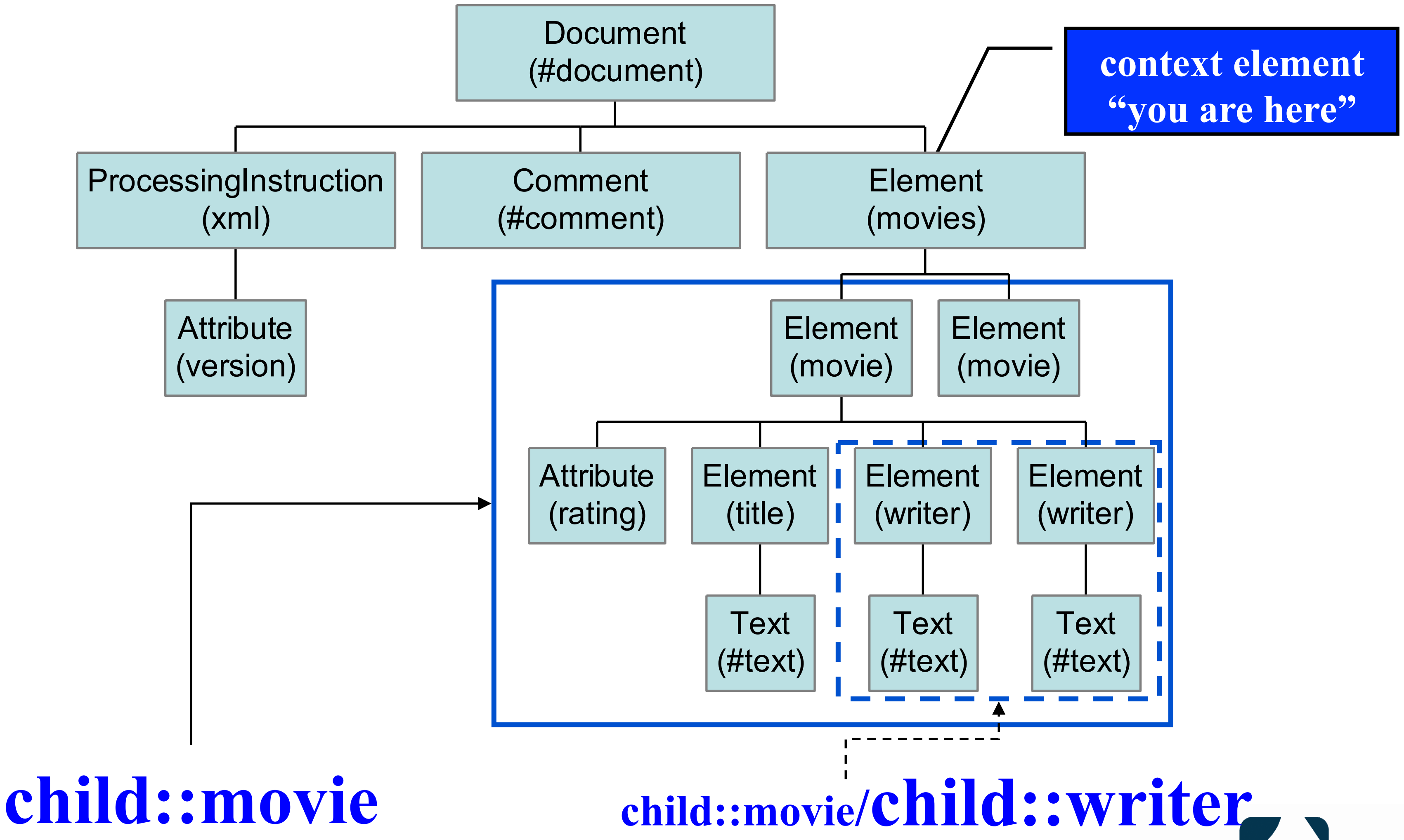
- An XPath expression can be seen as function composition
  - Each location step is evaluated relative to the context node

- A function *(“location step”)*
  - *Input: context node → Output: node set*
  - E.g. *“child::movie”* is a function that returns a “movie” set

- Function composition (“/”)
  - E.g.: *child::movie/child:writer returns “writer”* nodes
  - It evaluates from left to right
    - The output of a function provides the context node to evaluate the next function
      - o the output of *child:movie* becomes the input of *child:writer*

informatika fakultatea facultad de informática

XPath Expressions. Example

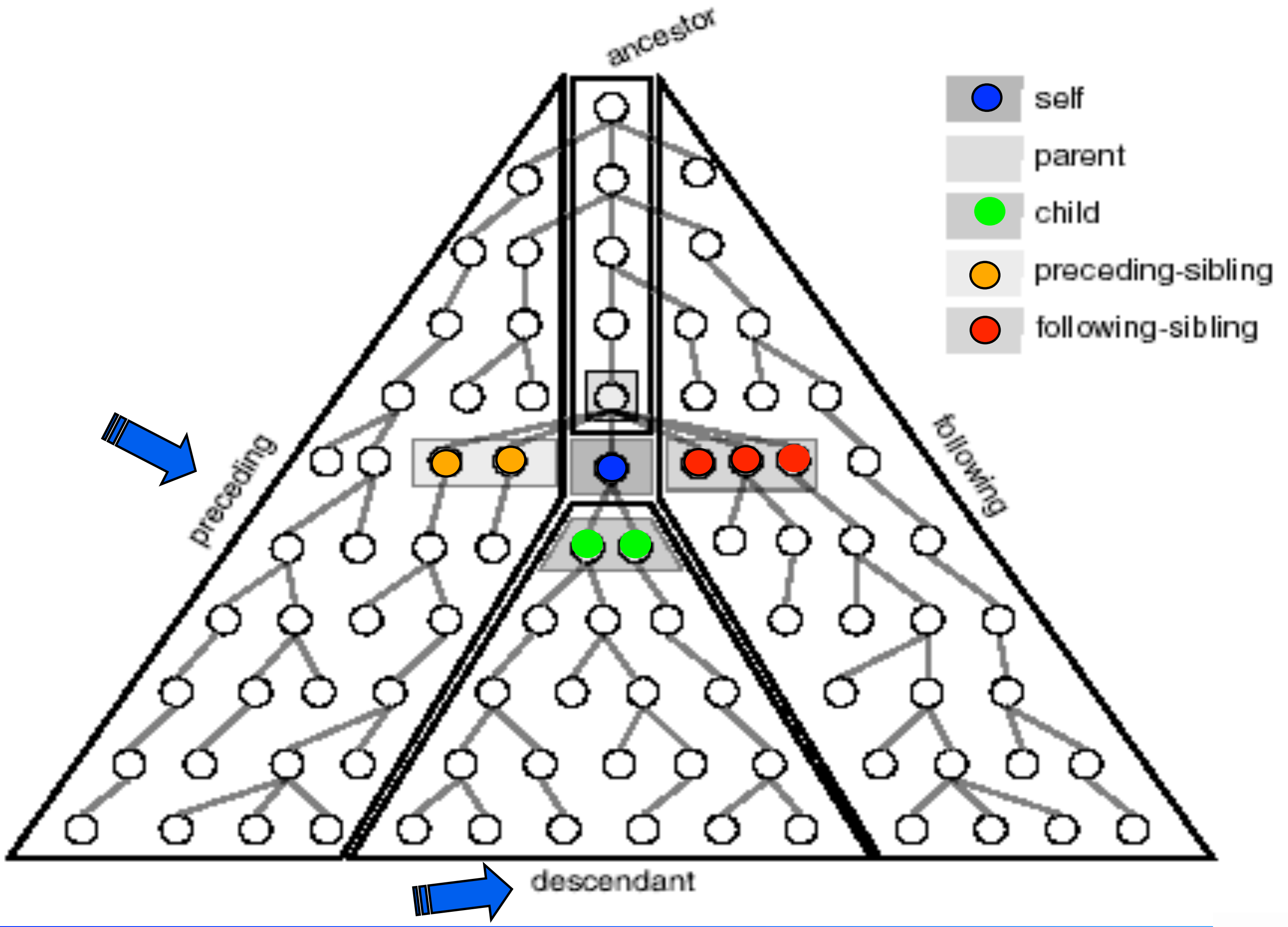


Location Path

- Consists of
  - axis
  - node test
  - predicates (optional)

- Syntax: **axis::nodeTest[predicate]**
  - E.g.: *child::movie[child::title='Raising Arizona']*

axis::nodeTest[predicate]



Axis types

- **parent**: selects the parent
- **ancestor**: selects all the ancestors
- **ancestor-or-self**: selects all the ancestors, including the current node

- **child**: selects all the child elements (DEFAULT AXIS)
- **descendant**: selects all the descendants
- **descendant-or-self**: selects all the descendants, including the current node

- **following**: selects all nodes that follow the current node, except ancestors, attribute nodes and namespace nodes
- **following-sibling**: selects the following siblings

- **preceding**: selects all nodes that appear before the current node, except ancestors, attribute nodes and namespace nodes
- **preceding-sibling**: select the preceding siblings

- **self**: selects the current node
- **attribute**: selects all the attributes

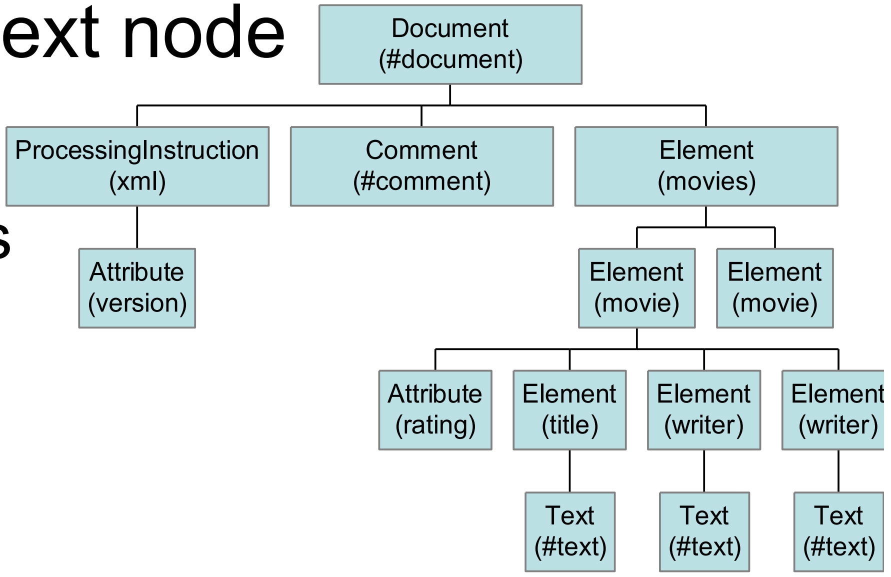
axis::nodeTest[predicate]

- An axis can potentially locate a lot of nodes

- The test is a predicate that is applied to each node of the axis and filters only nodes of a given type
  - If the test is fulfilled, the node is kept. Otherwise, it is discarded.

axis::nodeTest[predicate] Examples

Taking ‘*movies*’ as the context node



- *descendant::movie*
  - → retrieves the *movie* nodes
- *descendant::title*
  - → retrieves the *title* nodes
- *descendant::comment()*
  - → retrieves the comment nodes
- *descendant::movie/attribute::rating*
  - → retrieves the *rating* attribute of *movie* nodes

axis::nodeTest[predicate]

- The predicate acts as a filter: e[p]
  - for each element in the sequence e, if p is true, then propagate the element to the output, otherwise discard it

descendant::movie[child::title = “Airbag”]

- Existential semantics of predicates
  - [child::actor = “Barden”]   is true if at least one element returned by “actor” is string and equals to “Barden”
  - [child::actor = “Barden”] is false if “actor” returns the empty sequence or in the sequence there is no “Barden”
- The predicate may be a simple XPath
  - [actor] is true if “actor” returns a non-empty sequence

axis::nodeTest[predica
Examples

```xml
<?xml version="1.0" ?>
<movies>
  <movie rating="PG-13" type="mistery">
    <title>Raising Arizona</title>
    <writer>Ethan Coen</writer>
    <writer>Joel Coen</writer>
    <producer><name>Ethan</name>…</produ
    <director>Joel Coen</director>
  </movie>
  <movie type="comedy">
    <producer>Pete Smith</producer>
  </movie>
</movies>
```

- /child::movies/child::movie [child::producer/child::name]
- /child::movies/child::movie[child::producer/ child::name=‘Ethan’]
- /child::movies/ child::movie[attribute::type="mistery"]/ child::producer
- /child::movies/child::movie[child::producer/ child::name='Ethan']/child::director

informatika fakultatea facultad de informática

Path types

➢ Absolute

- evaluated from the root node
- start with /
- /child::movies/child::movie/child::title → would retrieve all the titles

➢ Relative

- evaluated from a context node
- start directly with the expression
- child::title → would select the title of the context node

Location path: abbreviated syntax

- The axis and the node test are combined
- Lets walk along the child, parent, self, attribute, and descendant-or-self axes

```
/child::movies/child::movie/attribute::rating
        /movies/movie/@rating

/child::movies/child::movie[contains(child::writer, ‘thom’)]
        /movies/movie[contains(writer, ‘thom’)]
```

Location steps

- **/** : the root element
- **comment()** : matches any comment node child of the context node
- **..** : indicates the parent of the current node
- **.** : indicates the context node
- **//** : selects from all descendants of the context node, as well as the context node itself
- ***@atrib*** : gets the value of the ‘*atrib*’ attribute
- **attribute()** : matches any attribute node child of the context node
- **@***: matches any attribute node child of the context node
- *****: matches any element node regardless of name
- **node()** : matches all node types: the element node and also the root node, text nodes, attribute nodes, comment nodes, …
- **text()** : matches any text node child of the context node
- **|** : matches any of the named elements

informatika fakultatea facultad de informática

Location steps. Examples

- child::movie/parent::.
  - movie/..
- child::movie/descendant::surname
  - movie//surname
- movie/director | movie/producer
- /movies/movie/node() <u>vs.</u> /movies/movie/*

Functions

- XPath provides more than 25 functions
- Functions with no argument operate on the context node
- Node Set Functions
  - **count()** – returns the number of nodes that the argument returns
- String Functions
  - **concat()** – returns the concatenation of its arguments
- Number Functions
  - **sum()** – returns the sum of the values of a node set
- Boolean Functions
  - **not()** – true if the argument is false, and vice versa
  - **contains(**string1, string2) – true if string1 contains string2
  - **starts-with(**string, pattern**)** – true if string starts with the pattern

http://www.w3.org/TR/xpath-functions/

Operators

- Boolean operators
  - *and, or, not*
- Arithmetic operators:
  - + - * *div mod*
- Relational operators:

<	>	=	>=	<=	<>	LIKE
&lt;	*&gt;*	=	*&gt;*=	*&lt;*=	!=	~=

Functions and Operators. Examples

- /movies/movie[title = “Airbag”]
- /movies/movie[count(actor) > 3]
- /movies/movie[contains(actor, ‘Marlon’)]
- /movies/movie[position() = 3]
- /movies/movie[(budget * 1,2) > 20000]/producer
- /movies/movie[budget < 1000 and matches(producer/name, "Coen“)]
- /movies/movie[@type != 'mistery']
- /movies/movie/producer[name   =   "Athen"]
- /movies/movie[matches(producer/name, "^Nico")]

Examples …

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
                <author> Serge Abiteboul </author>
                <author> <first-name> Rick </first-name>
                                <last-name> Hull </last-name>
                </author>
                <author> Victor Vianu </author>
                <title> Foundations of Databases </title>
                <year> 1995 </year>
    </book>
    <book price="55">
                <publisher> Freeman </publisher>
                <author> Jeffrey D. Ullman </author>
                <title> Principles of Database and Knowle
                <year> 1998 </year>
    </book>
</bib>
```

© Prof. Don Suciu

/bib/book/year

Result: ????

/bib/paper/year

Result: ????

Examples …

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowlec
            <year> 1998 </year>
    </book>
</bib>
```

//author (*Restricted Kleene Closure*)

Result: ????

/bib//first-name

Result: ????????

informatika fakultatea facultad de informática

Examples ...

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowledge
            <year> 1998 </year>
    </book>
</bib>
```

/bib/book/author/text()

Result: ????

Examples…

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowledge
            <year> 1998 </year>
    </book>
</bib>
```

//author/*

Result: ????

//author/node()

Result: ????

Examples …

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowledge
            <year> 1998 </year>
    </book>
</bib>
```

/bib/book/@price

Result: ?

Examples …

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowledge
            <year> 1998 </year>
    </book>
</bib>
```

/bib/book/author[first-name]

Result: ???

/bib/book/author[first-name]/last-name

Result: ???

Examples …

```
<bib>
    <book>  <publisher> Addison-Wesley </publisher>
            <author> Serge Abiteboul </author>
            <author> <first-name> Rick </first-name>
                     <last-name> Hull </last-name>
            </author>
            <author> Victor Vianu </author>
            <title> Foundations of Databases </title>
            <year> 1995 </year>
    </book>
    <book price="55">
            <publisher> Freeman </publisher>
            <author> Jeffrey D. Ullman </author>
            <title> Principles of Database and Knowledg
            <year> 1998 </year>
    </book>
</bib>
```

/bib/book[@price < "60"]

/bib/book[author/@age < "25"]/title

/bib/book[author/text()]