

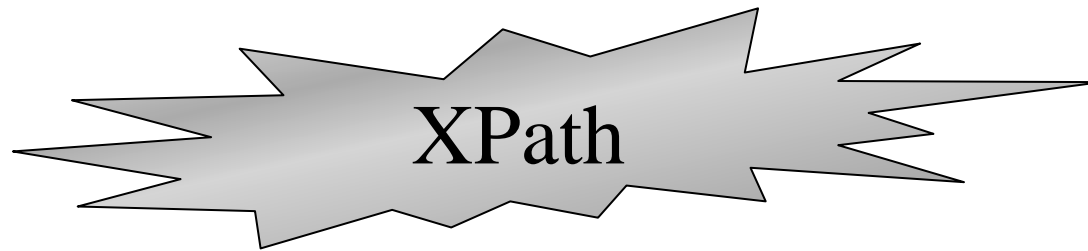


181135 VU Semistrukturierte Daten 1

XPath

20.10.2005

Reinhard Pichler

The logo for XPath, featuring the text "XPPath" in a serif font, centered within a grey, jagged, starburst-like shape with a black outline.

XPPath

- XPath ist die Basis für viele XML-related Standards:
 - insbesondere für XSLT
 - aber auch für XPointer und XQuery
 - (in eingeschränkter Form) auch in XML Schema
- Nachteil: ist selbst nicht in XML Notation
- Ziel: Navigieren im Dokumentenbaum
 - in der Art wie Dateisystem
 - Das Abfrageergebnis ist meistens eine Knotenmenge
 - Selektion von Knoten nach Kriterien wie Elementnamen, Attributnamen, Typ, Inhalt, Wert, Beziehungen
 - Suchergebnis einschränken mittels Filtern möglich
 - weitere Datentypen: String, Number, Boolean
- Version 1.0 Recommendation, Version 2.0 Working Draft
 - XPath 2.0 insbesondere für XQuery

Inhalt

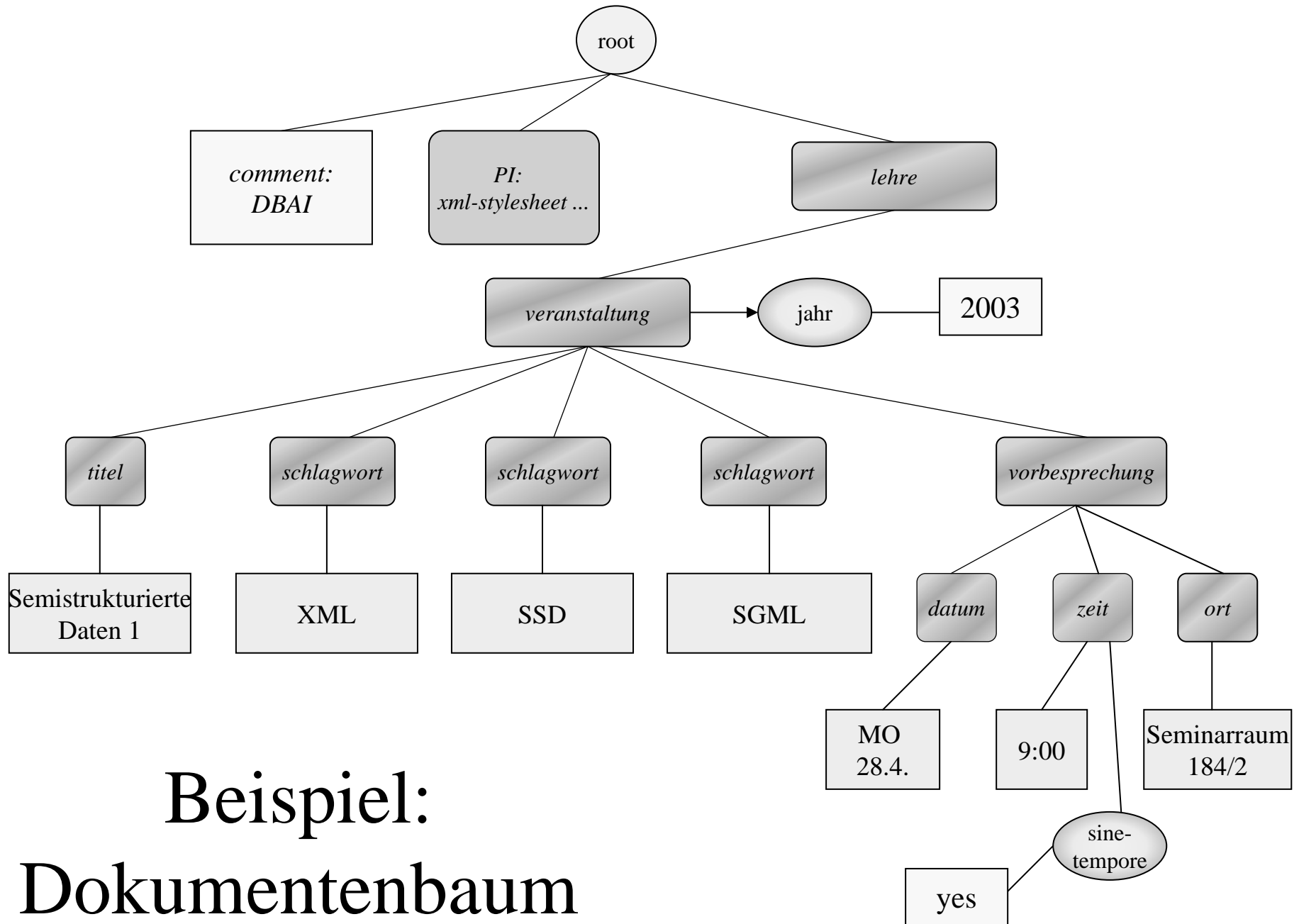
- Datenmodell (Dokumentenbaum)
- XPath Expressions
- Pfade (= location paths)
- location steps
- Operatoren
- Built-in Funktionen
- Tool: XPathTester

Datenmodell

- XPath betrachtet ein XML-Dokument als Baum
 - Dieser Baum ist leicht abweichend vom „DOM“
- 7 Arten von Knoten im Dokumentenbaum:
 - **Root node:** Wurzelknoten des Baums als parent des Dokuments (für PIs und comments im Prolog, d.h.: vor dem Dokuments)
 - **Element nodes:** für jedes Element im Dokument
 - **Attribute nodes:** assoziiert mit entsprechendem element node
 - **Namespace nodes:** für alle NS-Präfixe plus einen etwaigen Default-NS, die für ein Element gültig sind
 - **Processing instruction nodes:** für jede PI (außerhalb einer DOCTYPE-Definition). Die xml-Deklaration gilt nicht als PI.
 - **Comment nodes:** für jeden Kommentar (außerhalb einer DOCTYPE-Definition)
 - **Text nodes:** Zeichendaten werden in möglichst große text nodes zusammengefasst (d.h.: Ein text node hat keine text nodes als Nachbarn).

Beispiel: XML Dokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DBAI -->
<?xml-stylesheet type="text/css" href="lehre.css"?>
<lehre>
  <veranstaltung jahr="2003">
    <titel>Semistrukturierte Daten 1</titel>
    <schlagwort>XML</schlagwort>
    <schlagwort>SSD</schlagwort>
    <schlagwort>SGML</schlagwort>
    <vorbesprechung>
      <datum>Mo 28.4.</datum>
      <zeit sine_tempore="yes">9:00</zeit>
      <ort>Seminarraum 184/2</ort>
    </vorbesprechung>
  </veranstaltung>
</lehre>
```



Beispiel: Dokumententbaum

Document Order

- Alle Knoten im Baum sind geordnet.
- Die Ordnung der element nodes ist top-down, left-to-right (d.h.: Reihenfolge der Start-Tags ist entscheidend)
- NS-nodes kommen vor den attribute nodes.
- Attribute nodes kommen vor den (Element-)Kindknoten
- Reihenfolge innerhalb der NS-nodes sowie innerhalb der attribute nodes ist implementierungsabhängig.
- Wenn Resultat eines XPath-Ausdrucks eine Knotenmenge ist, dann sind diese Knoten geordnet:
 - Normalerweise in „document order“
 - Bei Navigation in umgekehrter Richtung: „inverse document order“

String-Wert eines Knoten

- **Root node:** Konkatenation aller Textknoten im Dokument
- **Element node:** Konkatenation aller Textknoten unterhalb eines Elementknotens, d.h.: `descendant::text()`
- **Attribute node:** normalisierter Attributwert
- **Namespace node:** Namespace-URI
- **Processing instruction node:** String hinter dem (lokalen) Namen der PI, z.B. `'type="text/css" href="lehre.css"'`
- **Comment node:** Inhalt des Kommentars (ohne `<!--` und `-->`)
- **Text nodes:** Zeicheninhalt

Expressions

- Expression = zentrales syntaktisches Konstrukt in XPath
- Auswertung einer XPath expression für einen Kontext:
 - context-node Knoten im Dokumentenbaum
 - context-position pos. Integer
 - context-size pos. Integer
 - variable bindings für alle Variablen einer XPath expression
Variablennotation: \$x

context-node/position/size können sich während der Auswertung einer XPath expression ändern (s.u.)

- 4 Datentypen als mögliches Resultat einer XPath Expression:
 - node-set
 - boolean
 - number
 - string

Typ-Konvertierungen

- Explizite vs. implizite Typkonvertierungen in XPath
 - implizit: Operanden von Operatoren, die einen bestimmten Typ erwarten, z.B.: Operanden von `and` müssen boolean sein.
 - explizit: durch Aufruf der XPath-Konvertierungsfunktionen `string(x)`, `number(x)`, `boolean(x)`
 - Konvertierung in ein node-set ist nicht möglich!
- Konvertierung node-set -> string/number/boolean
 - `string(x)` = Stringwert des ersten Knotens von x
bzw. leerer String (bei leerem node-set x)
 - `number(x)` = `number(string(x))`
 - `boolean(x)` = `true`, falls x nicht leer ist (sonst `false`)

- Konvertierung string -> number/boolean
 - **number(x)** = von x dargestellte Zahl bzw. **NaN** falls x keine Zahl darstellt.
 - **boolean(x)** = **true**, falls x ein nicht-leerer String ist (sonst **false**)
- Konvertierung number -> string/boolean
 - **string(x)** = Dezimaldarstellung von x bzw. **"NaN"**, **"Infinity"**, **"-Infinity"**
 - **boolean(x)** = **true**, falls x weder 0 noch **NaN** (sonst **false**)
- Konvertierung boolean -> string/number
 - **string(x)** = **"true"**, falls x = true (sonst **"false"**)
 - **number(x)** = 1 falls x = true (sonst 0)

Pfade

- Pfade (location paths) = wichtigste Form von XPath Expressions
- Ein Pfad besteht aus 1 oder mehreren Schritten (location steps)
- Absoluter Pfad:
 - beginnt beim root node
 - Schreibweise, z.B.: /lehre/veranstaltung/schlagwort
- Relativer Pfad:
 - beginnt beim aktuellen context-node
 - Schreibweise, z.B.: veranstaltung/schlagwort

Beispiele

- Absoluter Pfad:

`/lehre/veranstaltung/titel` bzw.
`/child::lehre/child::veranstaltung/child::titel`

- Relativer Pfad:

`veranstaltung/titel` bzw. `child::veranstaltung/child::titel`
`zeit/@sine-tempore` bzw. `child::zeit/attribute::sine-tempore`

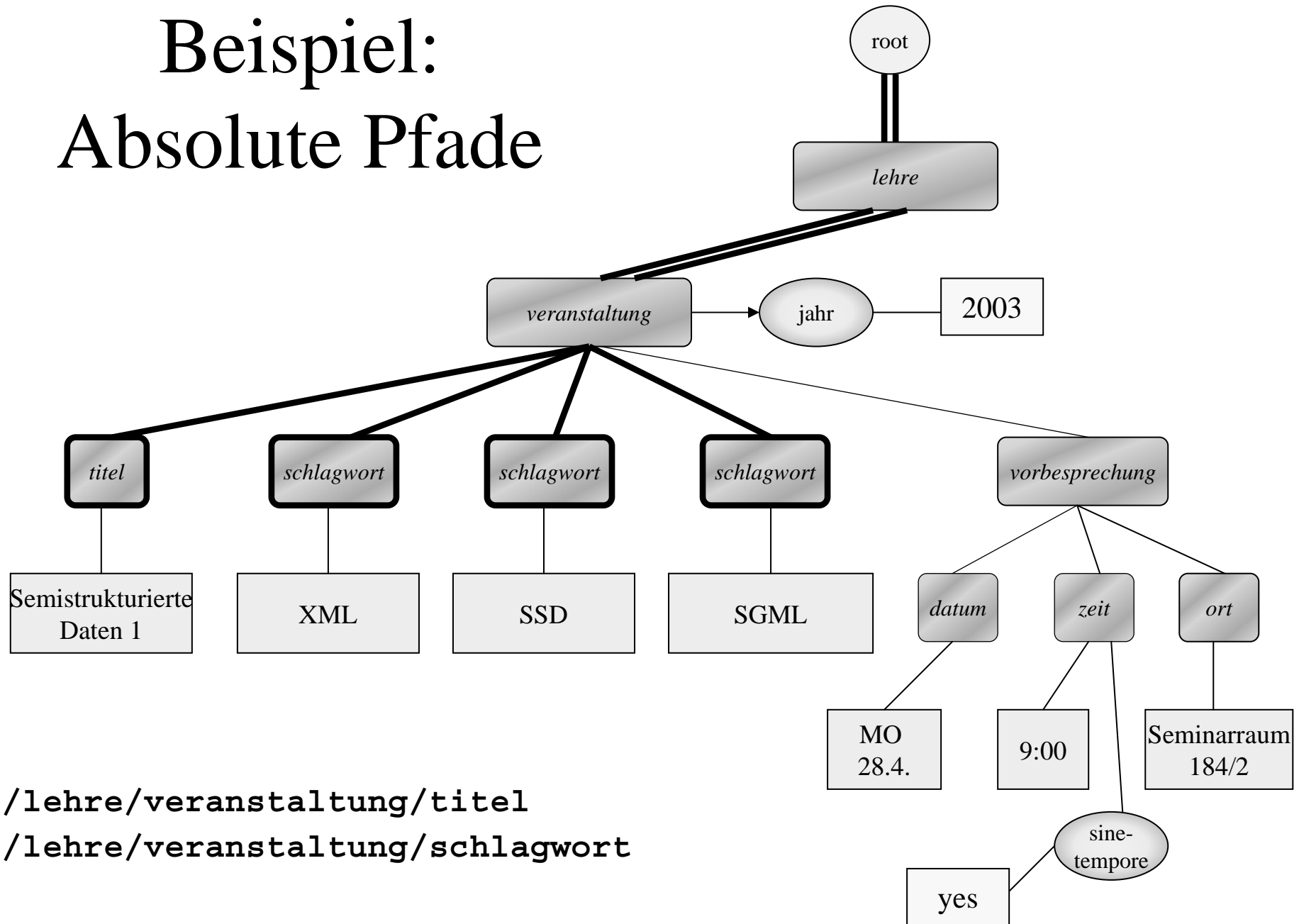
- „Unvollständiger“ Pfad:

`../zeit` bzw. `descendant::zeit`
`/lehre/*/schlagwort` bzw. `/child::lehre/child::* /child::schlagwort`
`//titel` bzw. `/descendant::titel`

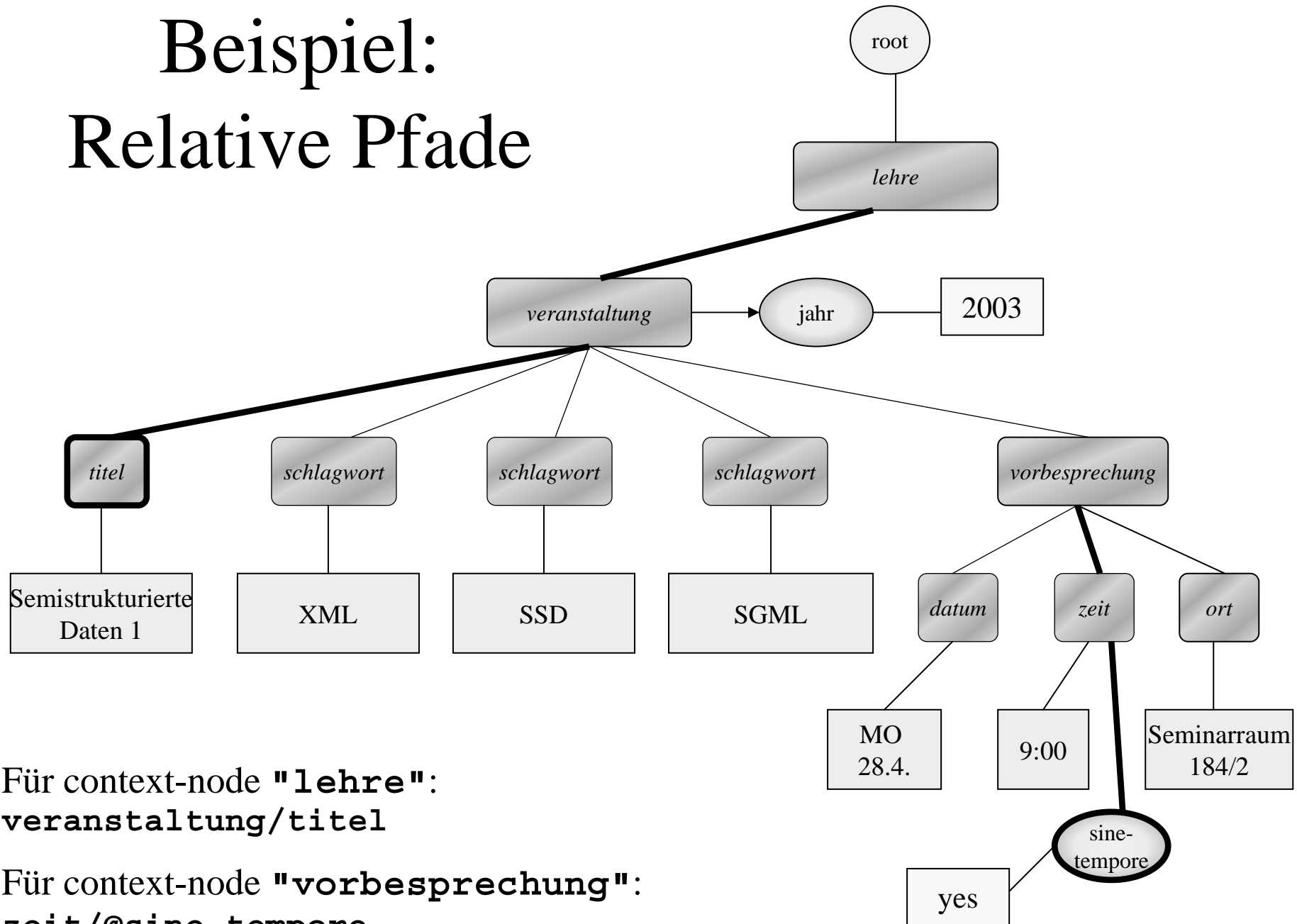
- Verwendung von Filtern:

`//zeit[@sine-tempore="yes"]` bzw.
`/descendant::zeit[attribute::sine-tempore="yes"]`
`schlagwort[2]` bzw. `child::schlagwort[position()=2]`
`schlagwort[last()]` bzw. `child::schlagwort[position()=last()]`

Beispiel: Absolute Pfade



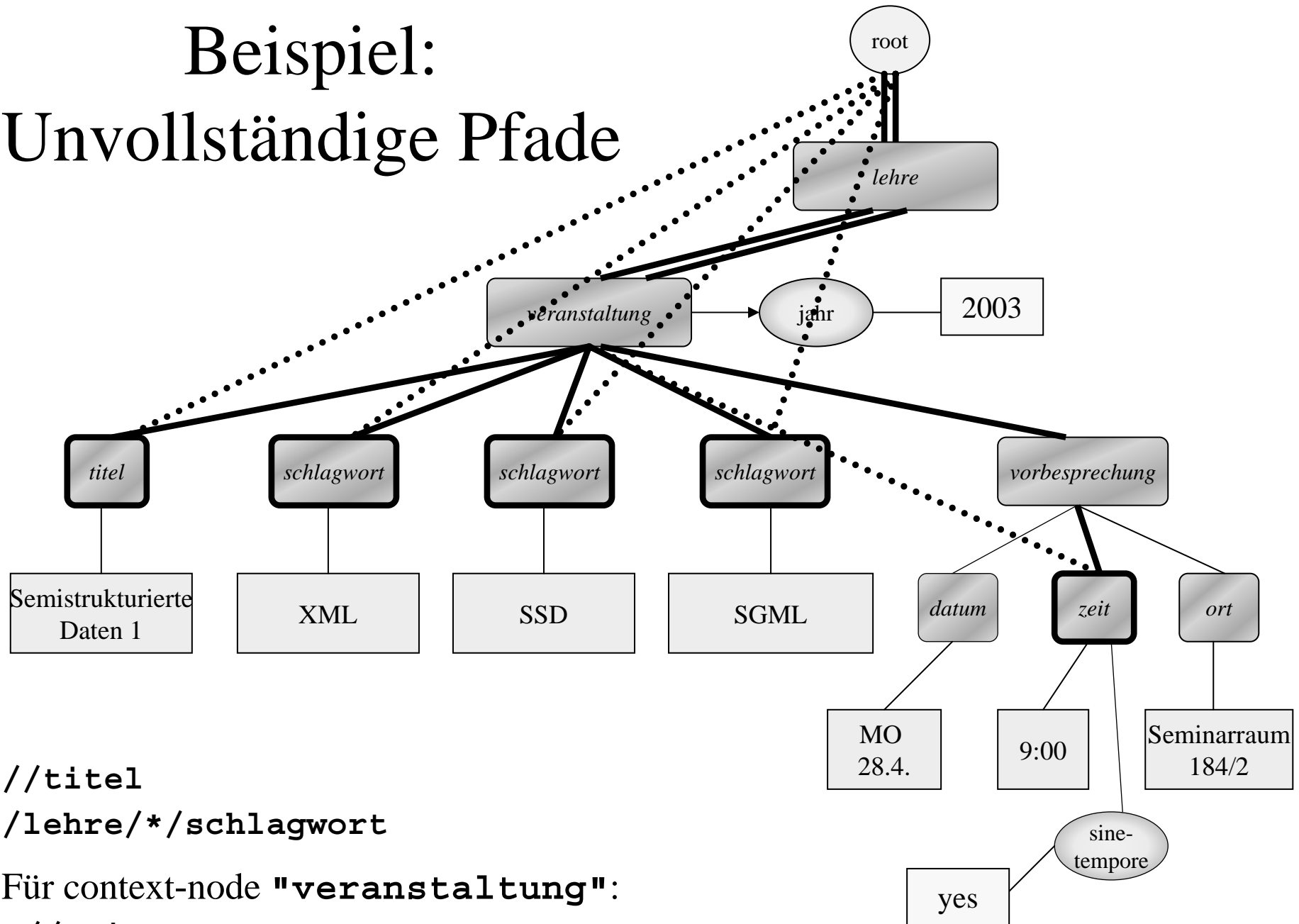
Beispiel: Relative Pfade



Für context-node "**lehre**":
veranstaltung/titel

Für context-node "**vorbesprechung**":
zeit/@sine-tempore

Beispiel: Unvollständige Pfade



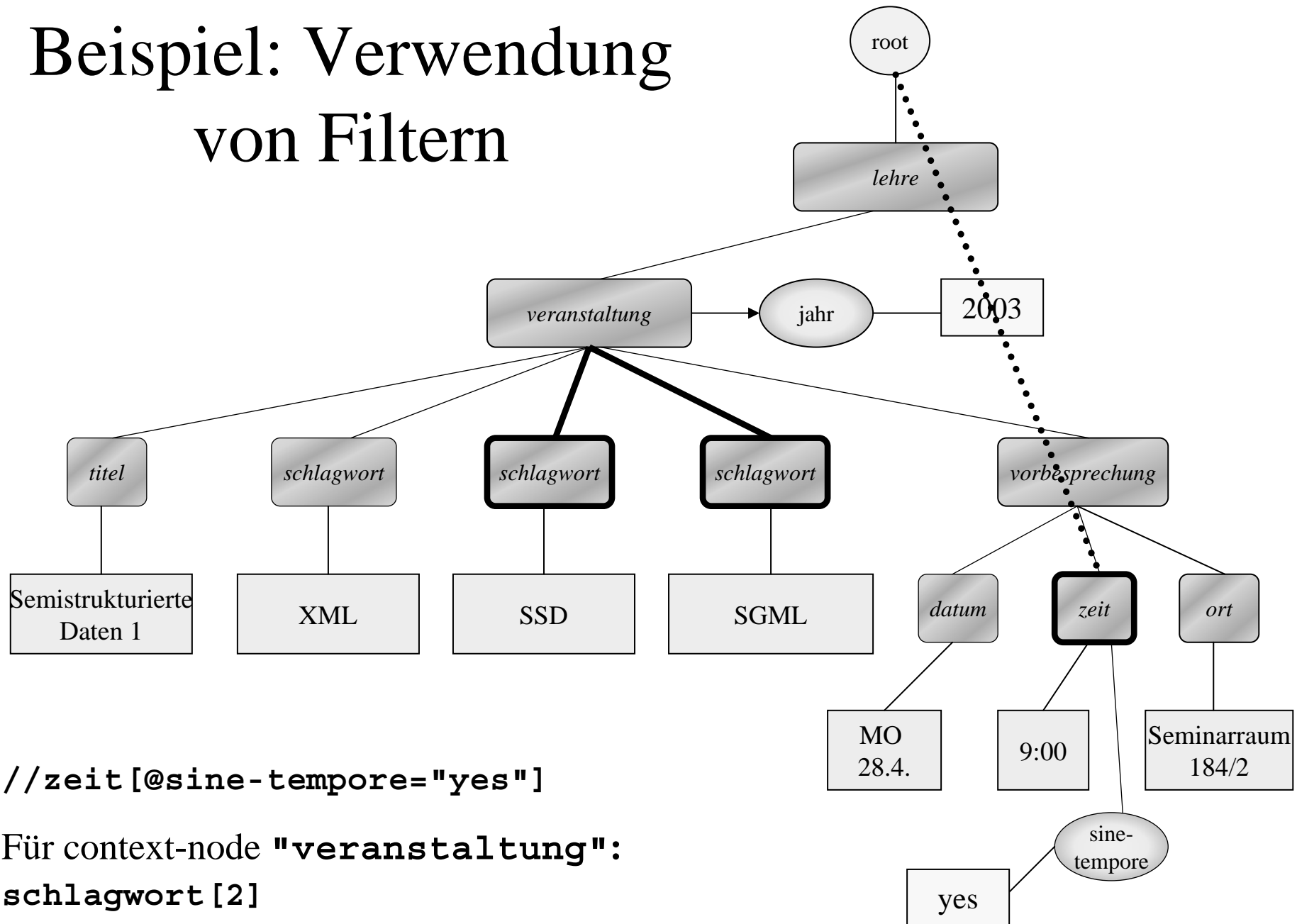
`//titel`

`/lehre/*/schlagwort`

Für context-node "**veranstaltung**":

`../zeit`

Beispiel: Verwendung von Filtern



```
//zeit[@sine-tempore="yes"]
```

Für context-node "veranstaltung":

```
schlagwort [2]
```

```
schlagwort [last ()]
```

Location Steps

- Ein Pfad setzt sich aus beliebig vielen Schritten zusammen (= location steps)
- Bestandteile eines location steps:
 - Achse: Richtung, in die navigiert wird
 - Node-Test: Typ und Name der gewünschten Knoten
 - 0 oder mehr Filterbedingungen (= „predicates“)
- Beispiele:
 - `child::schlagwort [2]`
 - `parent::*`
 - `preceding-sibling::text () [last ()]`
 - `following::node () [@sine-tempore="yes"] [1]`
 - `attribute::jahr`
 - `descendant::processing-instruction ()`

XPath Achsen

- child: Kindelemente
- ancestor: Vorfahren
- self: der Knoten selbst
- ancestor-or-self: alle Vorfahren und selbst
- descendant: alle Nachfahren
- descendant-or-self: alle Nachfahren und selbst
- parent: Elternelement
- following: alle in document order nachfolgenden Knoten außer Nachfahren, Attribut-Knoten und NS-Knoten
- following-sibling: alle nachfolgenden Geschwisterknoten
- preceding: alle in document order vorangehenden Knoten außer Vorfahren, Attribut-Knoten und NS-Knoten
- preceding-sibling: alle vorangehenden Geschwisterknoten
- attribute: alle Attribute eines Elements
- namespace: alle Namespaces eines Elements

Node Tests

- Üblicherweise: Angabe eines Namen
 - z.B. `"/child::lehre/descendant::zeit"`
- * steht für beliebigen Namen
 - `child::*` gibt alle Elementknoten zurück
 - `attribute::*` gibt alle Attributknoten zurück
- `comment()`: alle Kommentar-Knoten
 - z.B. `"/child::lehre/child::comment()"` bzw. `"/lehre/comment()"`
- `processing-instruction()`: alle PI-Knoten
 - z.B. `"/child::lehre/child::processing-instruction()"` bzw. `"/lehre/processing-instruction()"`
- `text()`: alle Textknoten
- `node()`: alle Knoten

Abkürzungen

- Weglassen der Achse entspricht der child-Achse
 - z.B. `zeit` entspricht `child::zeit`
- `.` = context node: entspricht `self::node()`
- `..` = parent des context node: entspricht `parent::*`
- `//` = descendants des context node, d.h.: `descendant::*`
- `@` = Abkürzung für attribute-Achse
 - z.B. `@*` entspricht `attribute::*`
- Beispiele:

<code>./titel</code>	alle titel-Elemente im momentanen Kontext (in Verwendung äquivalent zu <code>titel</code>)
<code>/titel</code>	selektiert Titel wenn Top-Level Element
<code>././titel</code>	starte vom aktuellen context-node und selektiere alle Titel Elemente, die tiefer liegen (also relativ)
<code>//titel</code>	starte von Wurzel und selektiere alle Titel Elemente, die tiefer liegen (also absolut)
<code>/lehre/veranstaltung</code>	absoluter Pfad zu Veranstaltungen
<code>//zeit/../*</code>	liefert Element plus seine Geschwister

Filter (= „predicate“)

- Angabe einer beliebigen XPath Expression in eckigen Klammern
 - z.B. `//schlagwort[position() >= 2]`
- Die predicate expression `[x]` muss ein boolean Resultat liefern. Andernfalls wird der Typ auf folgende Weise konvertiert:
 - node-set, string: `[x]` entspricht `[boolean(x)]`
z.B.: `[.//datum]` entspricht `[boolean(.//datum)]`
 - number: `[x]` entspricht `[position() = x]`
z.B.: `[3]` entspricht `[position() = 3]`
- Die Auswertung eines steps `achse::node-test [x]`
 - Zuerst wird das node-set aufgrund von `achse::node-test` berechnet.
=> Ergebnis $S = \{s_1, \dots, s_k\}$
 - Nun wird für jeden Kandidaten s_i das predicate `x` ausgewertet bzgl. `context-node = s_i` `context-position = i` und `context-size = k`
 - Im Endergebnis liegen jene Knoten s_i für die `x` den Wert `true` liefert.

XPath Filter Beispiele

veranstaltung [vorbesprechung]

selektiert alle Elemente die ein Vorbesprechungselement enthalten

Bemerkung: Typumwandlung des Node-sets von „vorbesprechung“ in boolean erforderlich.

vorbesprechung [datum="MO 28.4."]

selektiert alle Vorbesprechungen, die ein datum-Subelement mit diesem Inhalt haben.

zeit [@sine-tempore="yes"]

selektiert alle Zeit-Elemente, die über das Attribut sine-tempore verfügen mit dem Wert "yes".

vorbesprechung [not (zeit/@sine-tempore="yes")]

selektiert alle Vorbesprechungs-Elemente, die kein Subelement **zeit** mit Attribut **@sine-tempore="yes"** haben.

vorbesprechung [datum and ort]

selektiert alle Vorbesprechungs-Elemente, die mindestens ein Datum sowie einen Ort als Kinder haben.

schlagwort [1] bzw. schlagwort [position()=1]

findet das erste Schlagwort

schlagwort [last()] bzw. schlagwort [position()=last()]

findet das letzte Schlagwort

//schlagwort [2] /following::*

findet alle Elemente, die im XML-Dokument nach dem zweiten schlagwort vorkommen (aber nicht als Nachfahre von schlagwort).

Filterlisten


- Ein location step kann 0 oder beliebig viele „predicates“ haben.
- Wenn **[b]** weder **position()** noch **last()** enthält, dann sind **[a and b]** und **[a] [b]** identisch. Aber im allgemeinen sind die beiden Ausdrücke verschieden, da nach der Auswertung eines predicates der context neu ermittelt wird.
- Beispiel:
 - **schlagwort [.="SSD"] [2]**
wählt unter den Schlagwörtern mit Wert SSD das zweite aus
=> selektiert in diesem Fall die leere Knotenmenge
 - **schlagwort [2] [.="SSD"]**
wählt das zweite Schlagwort aus, vorausgesetzt dass der Wert SSD ist.
=> selektiert in diesem Fall das zweite Schlagwort

Operatoren

- Verknüpfung von node-sets:
 - | bildet die Vereinigung von 2 node-sets
 - z.B. `child::schlagwort | descendant::datum`
- Boole'sche Operatoren:
 - Operatoren `and` und `or`
 - Außerdem gibt es built-in Funktion `not ()`
 - z.B. `//schlagwort [not (position () =2)]`
 - Operanden werden implizit in boolean konvertiert
 - z.B. `//* [descendant::titel and descendant::datum]`
entspricht `//* [boolean (descendant::titel) and boolean (descendant::datum)]`
- Vergleichsoperatoren: `<, >, <=, >=, =, !=`
- Arithmetische Operatoren: `+, -, *, mod, div`
(Bemerkung: `div` ist Dezimal-Division, „/“ hat eine andere Bedeutung)

Vergleichsoperationen mit Node-Sets

- Node-sets x in Ausdrücken der Form $x \text{ RelOp } y$ mit $\text{RelOp} \in \{<, <=, =, !=, >, >=\}$ haben eine „exists“-Semantik
- z.B.: `./schlagwort = "SSD"` liefert true, wenn es einen Nachfahren des context-node gibt, der den Wert "SSD" hat.
- Wenn sowohl x als auch y ein node-set liefert, dann wird „exists“ auf beide node-sets angewendet.
- z.B.: `./veranstaltung/nummer = ./termin/lvanummer` liefert true, wenn es 2 Knoten a und b gibt, sodass gilt:
 - a ist vom context-node aus mittels `./veranstaltung/nummer` und
 - b ist vom context-node aus mittels `./termin/lvanummer` erreichbar.
 - `string(a) = string(b)`



Built-in Funktionen

- Built-in Funktionen von XPath (= „Core Library Functions“)
- Klassifizierung:
 - Node-set functions
 - String functions
 - Boolean functions
 - Number functions
- Beschreibung:
result-type **function-name** (*arguments*)

Node-set Functions

- *number* **position**()
Knotenposition in einer Knotenmenge (d.h.: context-position)
- *number* **last**()
Gesamtzahl der zuletzt selektierten Knoten (d.h.: context-size)
- *string* **local-name**(*node-set*?)
Lokaler Name des ersten Knoten (in document order) im node-set bzw. des context-node (falls node-set fehlt).
- *string* **namespace-uri**(*node-set*?)
Namespace URI des ersten Knoten (in document order) im node-set bzw. des context-node (falls node-set fehlt).
- *string* **name**(*node-set*?)
Qualifizierter Name(d.h.: NS-Präfix + local-name) des ersten Knoten (in document order) im node-set bzw. des context-node.
- *number* **count**(*node-set*)
Anzahl der Knoten im node-set

id-Funktion

node-set **id**(object?)

Fall 1: Object = string:

- Zerlege string in whitespace-separated Liste von Tokens
- Selektiere alle Knoten, deren ID-Attribut (laut ID-Definition in der DTD) den Wert eines solchen Token hat.

Fall 2: Object = node-set:

- Betrachte den string-Wert von jedem Knoten im node-set
- Zerlege jeden dieser strings in Liste von Tokens (wie Fall 1)
- Selektiere alle Knoten, deren ID-Attribut (laut ID-Definition in der DTD) den Wert eines solchen Token hat.

Beispiel: **id**("xy07 yz21") wählt die Elemente, die ein ID-Attribut mit dem Wert "xy07" oder "yz21" haben.

String Functions

- *string* **string**(*object?*)
Typkonvertierung in einen String. Falls Argument fehlt, wird der context-node in einen String konvertiert.
- *string* **concat**(*string, string, string**)
Konkatenation der Argumente
- *boolean* **starts-with**(*string, string*)
liefert true, wenn das erste Argument mit dem zweiten beginnt
- *boolean* **contains**(*string, string*)
liefert true, wenn das erste Argument das zweite enthält
- *string* **substring**(*string, number, number?*)
liefert substring des ersten Arguments;
zweites Argument = Startposition (Zählung beginnt bei 1);
drittes Argument = Länge.
z.B.: **substring("abcdef", 2, 3)** liefert **"bcd"**
substring("abcdef", 2) liefert **"bcdef"**

- *string* **substring-before** (*string*, *string*)
liefert substring des ersten Arguments vor dem ersten Auftreten des zweiten Arguments (bzw. den leeren String, falls es kein solches Auftreten gibt).
- *string* **substring-after** (*string*, *string*)
liefert substring des ersten Arguments nach dem ersten Auftreten des zweiten Arguments (bzw. den leeren String, falls es kein solches Auftreten gibt).
- *string* **string-length** (*string*?)
liefert String-Länge des Arguments (bzw. den String-Wert des context-node)
- *string* **normalize-space** (*string*?)
normalisiert Argument-String (bzw. des String-Wertes des context-node) bzgl. Whitespaces
- *string* **translate** (*string*, *string*, *string*)
character-wise Transformation des ersten Arguments, z.B.:
`translate("---aaa---", "abc", "ABC")` liefert `---AAA---`

Boolean Functions

- *boolean* **boolean**(*object*)
Typkonvertierung in einen boolean Wert.
- *boolean* **not**(boolean)
logische Negation.
- *boolean* **true**()
liefert den Wahrheitswert true
- *boolean* **false**()
liefert den Wahrheitswert false
- *boolean* **lang**(string)
liefert true, wenn die Sprache des context-node (laut xml:lang-Attribut) dieselbe Sprache oder eine Subsprache des Input-Strings ist.

Number Functions

- *number* **number** (*object?*)
Typkonvertierung in eine Zahl. Falls Argument fehlt, wird der context-node in eine Zahl konvertiert.
- *number* **sum** (node-set)
Konvertiert den String-Wert jedes Knoten im node-set in eine Zahl und bildet von diesen Zahlen die Summe.
- *number* **round** (number)
rundet zur nächstgelegenen ganzen Zahl
- *number* **ceiling** (number)
rundet nach oben
- *number* **floor** (number)
rundet nach unten

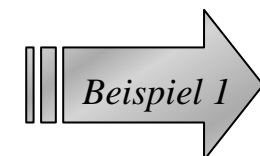
XPath Schwächen

- keine regulären Pfadausdrücke
 - wie `(part*)/name`
- keine regulären Ausdrücke für Textelemente
 - Diskussionen in XPath 2.0 und XSLT 2.0
- keine beliebige Kombinierbarkeit von Mengenoperationen und Navigation
 - z.B. `(child::a|preceding-sibling::b)/child::c` nicht erlaubt
- Fehlen nützlicher Funktionen (z.B.: min/max)
- eigenartige Typumwandlungen
 - z.B. `node-set+zahl` wird zu `number(node-set[1])+zahl`
- exists-Semantik bei Vergleichsoperationen mit node-sets
 - z.B. `not(node-set = a)` ist verschieden von `node-set != a`

The logo for XPPathTester is a grey, jagged, starburst-like shape with a black outline. The text "XPPathTester" is centered within this shape in a black, serif font.

XPPathTester

- <http://www.fivesight.com/downloads/xpathtester.asp>
- Graphische Testumgebung, um XPath Expressions auszuwerten
- Basiert auf XPath Engine von
 - Xalan XSLT Processor (XPPathTester Version 1.3) bzw.
 - Saxon XSLT Processor (XPPathTester Version 1.4)
- Start von der Kommandozeile aus:
 - `java -jar xpathtester_1_3_xalan2.jar`



Übungsbeispiel A

Erstellen Sie XPath Queries, um aus der Datei „Buchbestand.xml“ folgende Informationen zu extrahieren:

- Alle Autoren von allen Büchern
- Gesamtwert der vorhandenen Bücher
- Titel der Bücher, die den Substring „Harry Potter“ enthalten
- Preis des teuersten Buchs
- Die erste Hälfte aller Bücher in der Liste
bei ungerader Zahl: mittleres Buch zur ersten Hälfte dazurechnen
- Die restlichen Bücher
- Titel aller Bücher von Kafka (Vorsicht beim Vornamen)

Beispiel B

Erstellen Sie folgende XPath Queries für die Datei „Sammlung.xml“.
Schreiben Sie die ersten 3 Queries derart, dass Sie sich nicht auf ein bestimmtes NS-Präfix im XML-Dokument verlassen, d.h.: verwenden Sie die Funktionen `namespace-uri()` und `local-name()`!

- Die Titel aller CDs
- Gesamtwert der vorhandenen Bücher
- Titel aller CDs, die von Mozart komponiert sind
- Titel aller CDs, die nur von Jagger komponiert sind
- Gesamtzahl der CDs
- Titel aller CDs mit mindestens 2 Komponisten