Semistrukturierte Daten

Sommersemester 2008

Teil 7: XPath 1.0

7.1. Einführung
7.2. XPath Ausdrücke (expressions)
7.3. Pfadangaben (location paths)
7.4. Operatoren
7.5. Vordefinierte Funktionen

7.1. Einführung

- XPath Recommendation
- Datenmodell
- Document Order
- String-Wert

XPath Recommendation

- 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
  - XPath ist selbst nicht in XML Notation
- Hauptaufgabe:
  - Navigation im Dokumentenbaum
  - Selektion von Knotenmengen und Inhalten
  - einfache Operationen auf Inhalten
    => weitere Datentypen: String, Number, Boolean
- Versionen:
  - XPath 1.0: Recommendation seit 1999
  - XPath 2.0: Recommendation seit 2007: entspricht im wesentlichen XQuery ohne Benutzer-definierte Funktionen

Datenmodell (1)

- 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 Dokumentelements (für PIs und comments im Prolog)
  - **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.
  - **Comment nodes**: für jeden Kommentar
  - **Text nodes**: Zeichendaten werden in möglichst große text nodes zusammengefasst

Datenmodell (2)

- Knoten liefern die folgenden Informationen:
  - **Name**: qualifizierter Name bei element und attribute nodes, Präfix bei namespace nodes
  - **URI**: Namespace eines element bzw. attribute nodes
  - **Elternknoten**: jeder Knoten außer der root node hat genau einen Elternknoten.
  - **Liste von Kindknoten**: nur bei root und element node
  - **Wert**: (siehe nächste Folie)
  - **Sammlung von attribute und namespace nodes**: nur bei element nodes; die namespace nodes enthalten die aktuellen Präfix-Bindungen; die attribute nodes enthalten die Attribute des Elements (außer xmlns und xmlns:*präfix*).

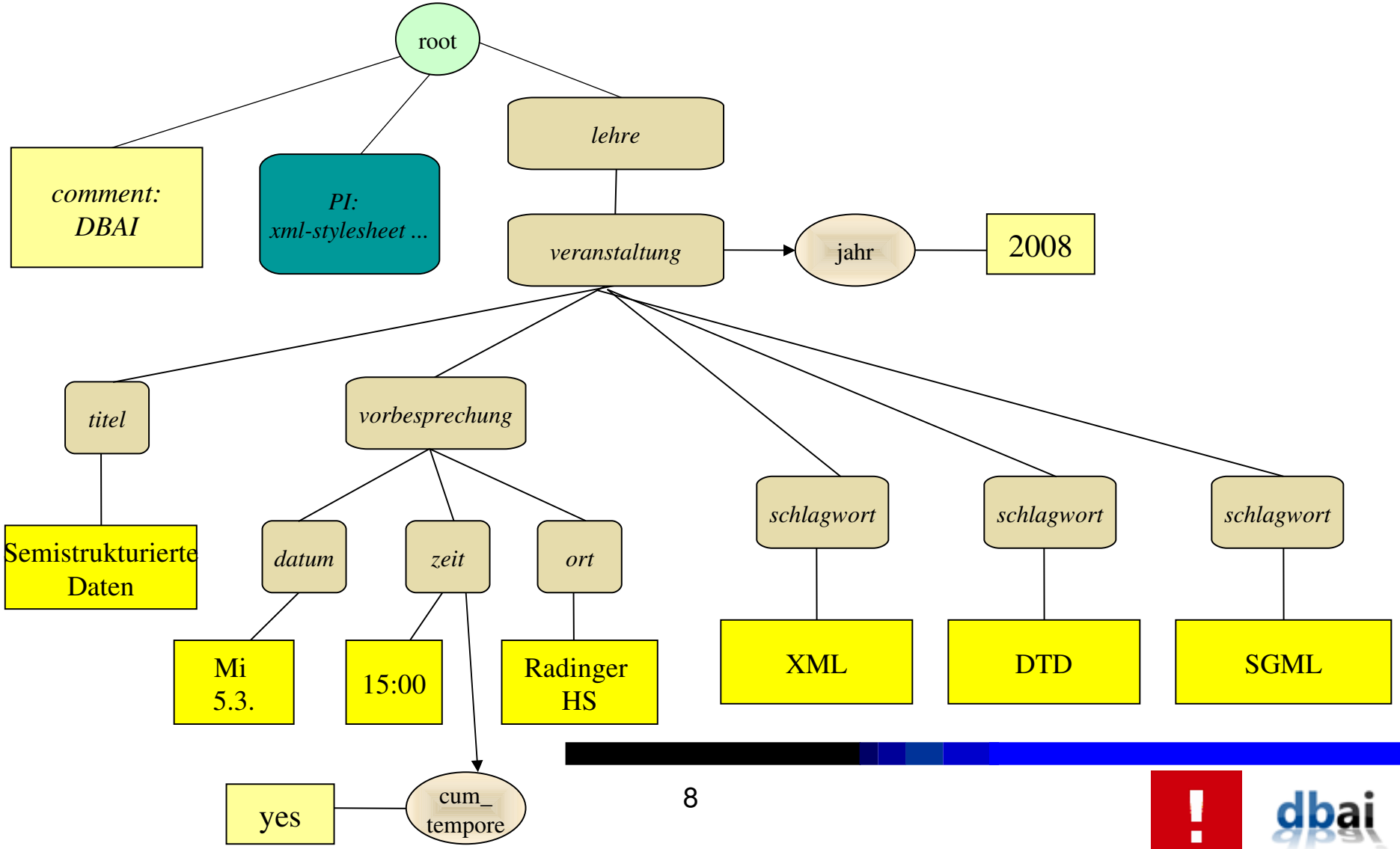
String-Wert eines Knoten

- **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

Beispiel: XML Dokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DBAI -->
<?xml-stylesheet type="text/css" href="lehre.css"?>
<lehre>
  <veranstaltung jahr="2008">
    <titel>Semistrukturierte Daten</titel>
     <vorbesprechung>
      <datum>Mi 5.3.</datum>
      <zeit cum_tempore="yes">15:00</zeit>
      <ort>Radinger HS</ort>
    </vorbesprechung>
    <schlagwort>XML</schlagwort>
    <schlagwort>DTD</schlagwort>
    <schlagwort>SGML</schlagwort>
</veranstaltung>
</lehre>
```

Beispiel: Dokumentenbaum



Document Order

- Alle Knoten im Baum sind geordnet.
- Die Ordnung der element nodes im Baum ist top-down, left-to-right (d.h.: Reihenfolge der Start-Tags ist entscheidend)
- Reihenfolge innerhalb der namespace nodes eines Elements bzw. innerhalb der attribute nodes eines Elements ist implementierungsabhängig.
- Wenn Resultat eines XPath-Ausdrucks eine Knotenmenge ist, dann sind diese Knoten geordnet:
  - normalerweise in "document order"
  - bei Navigation in umgekehrter Richtung: "reverse document order"

7.2. XPath Ausdrücke (expressions)

- XPath Auswertung
- Datentypen
- Typ-Konvertierungen

XPath Auswertung

- Expression = zentrales syntaktisches Konstrukt in XPath
- Die Auswertung einer XPath Expression geschieht immer relativ zu einem 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.)

Datentypen

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

Typ-Konvertierungen (1)

- 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`**)

Typ-Konvertierungen (2)

- Konvertierung number -> string/boolean
  - **`string(x)`** = Dezimaldarstellung von x bzw. **`"NaN" (= not a number)`**, **`"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`)

7.3. Pfadangaben (location paths)

- Überblick
- Location steps
- XPath Achsen
- Node tests
- Abkürzungen
- Filter (= "predicates")
- Filterlisten
- Auswertung von Location steps

Überblick

- wichtigste Form von XPath Expressions: Pfadangaben (= location paths)
- 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/@cum_tempore` bzw. `child::zeit/attribute::cum_tempore`

- Verwendung von Filtern:

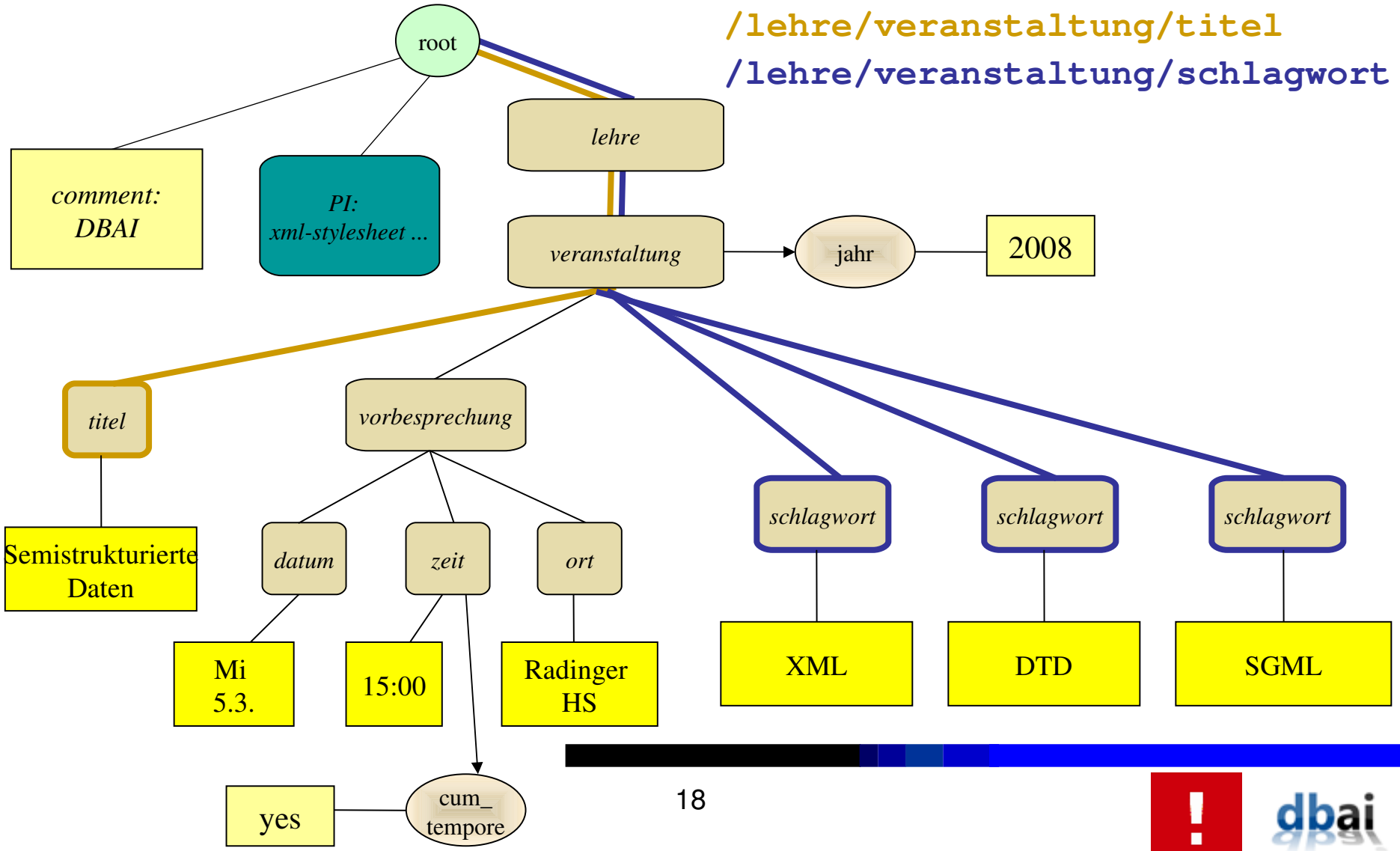
  `//zeit[@cum_tempore="yes"]` bzw.

  `/descendant-or-self::zeit[attribute::cum_tempore="yes"]`

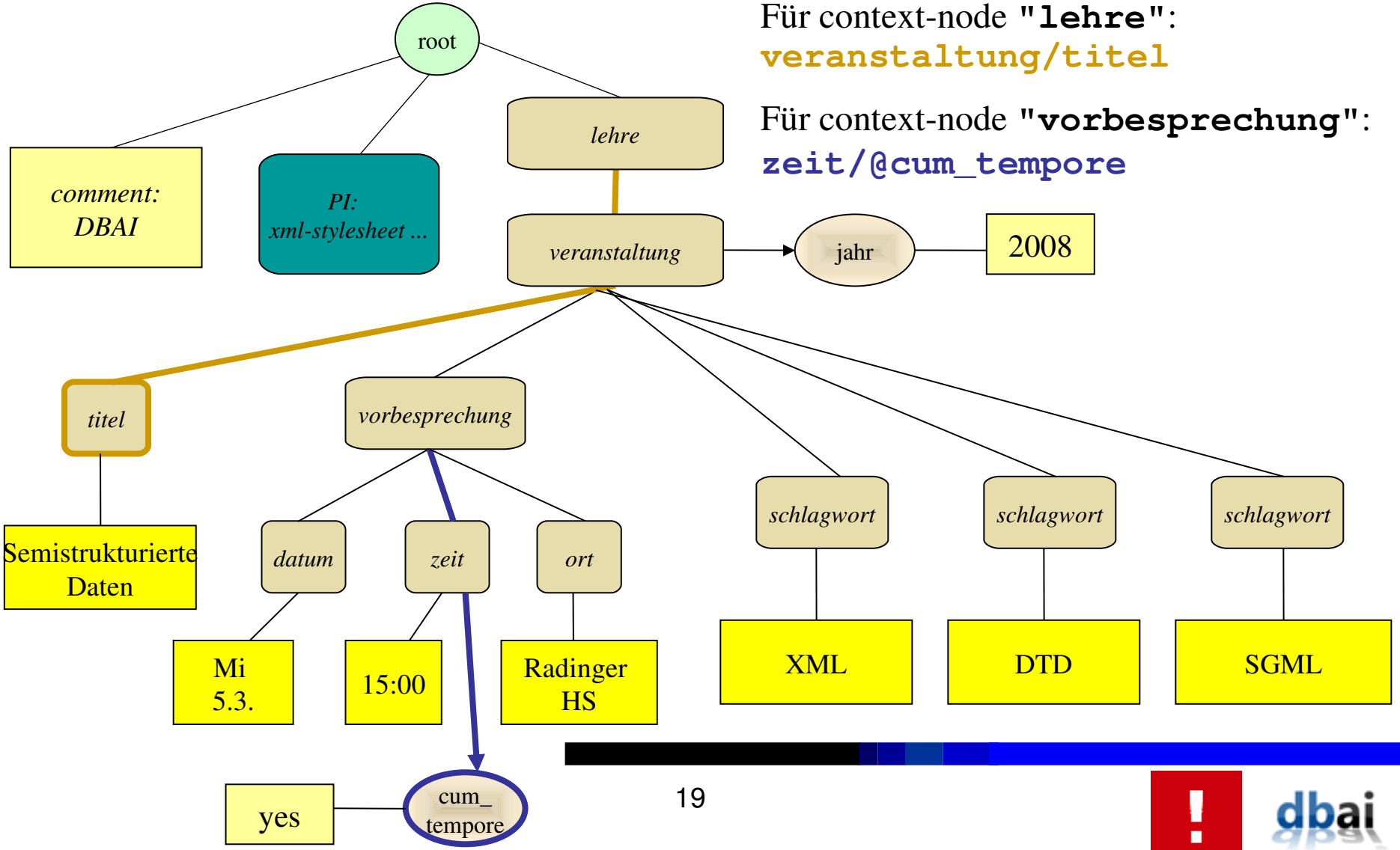
  `schlagwort[2]` bzw. `child::schlagwort[position()=2]`

  `schlagwort[last()]` bzw. `child::schlagwort[position()=last()]`

Beispiel: absolute Pfade



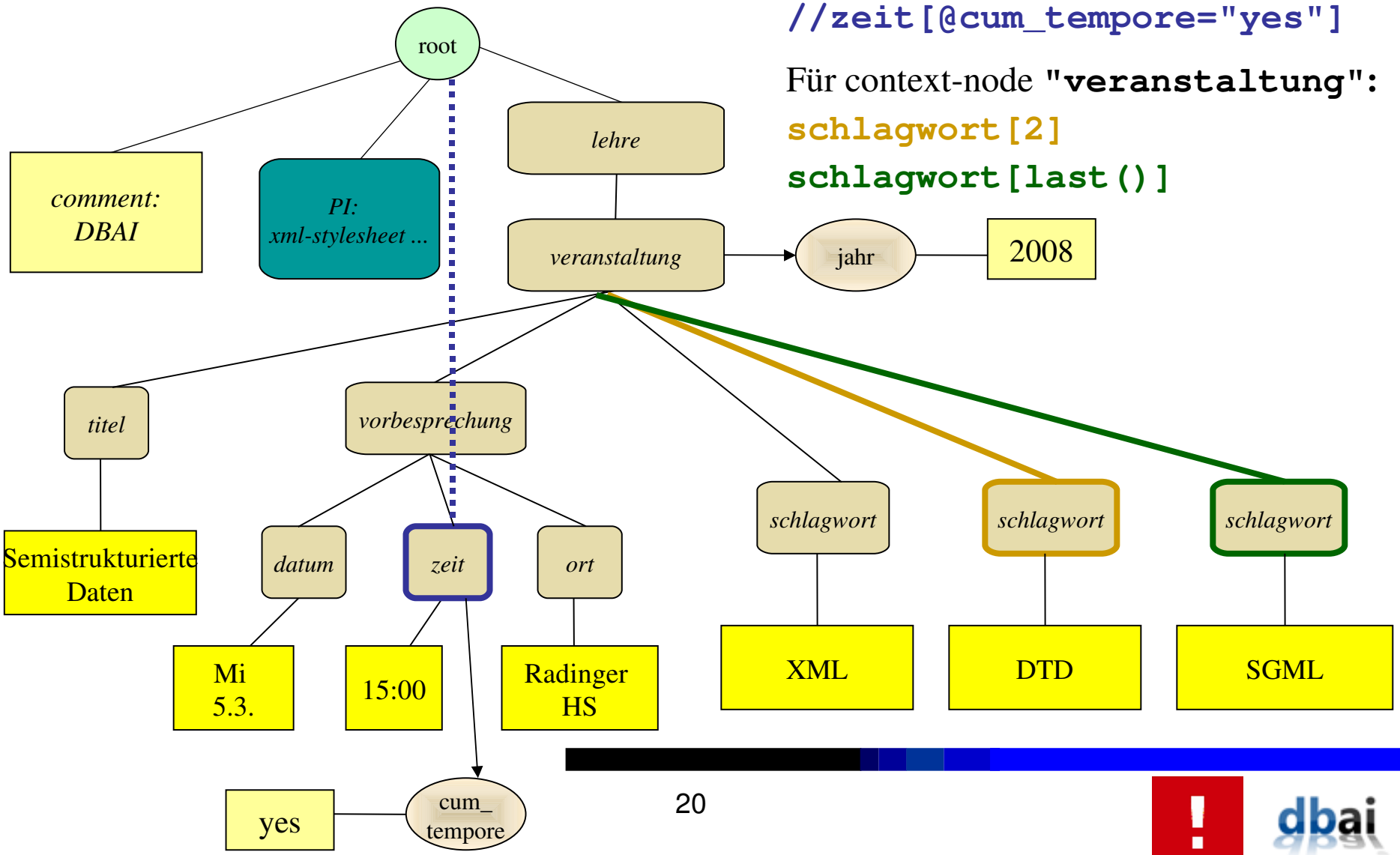
Beispiel: relative Pfade



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

Für context-node **"vorbesprechung"**:
zeit/@cum_tempore

Beispiel: Verwendung von Filtern




```
//zeit[@cum_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 bzw. Name der gewünschten Knoten
  - keine, eine, oder mehrere Filterbedingungen (= "predicates")
- Beispiele:
  - `child::schlagwort[2]`
  - `parent::*`
  - `preceding-sibling::text()[last()]`
  - `following::node()[@cum_tempore="yes"][1]`
  - `attribute::jahr`
  - `descendant::processing-instruction()`

XPath Achsen (1)

- child: Kind-Knoten
- descendant: alle Nachfahren
- descendant-or-self: alle Nachfahren und selbst
- parent: Elternelement
- ancestor: Vorfahren
- ancestor-or-self: alle Vorfahren und selbst
- self: der Knoten selbst

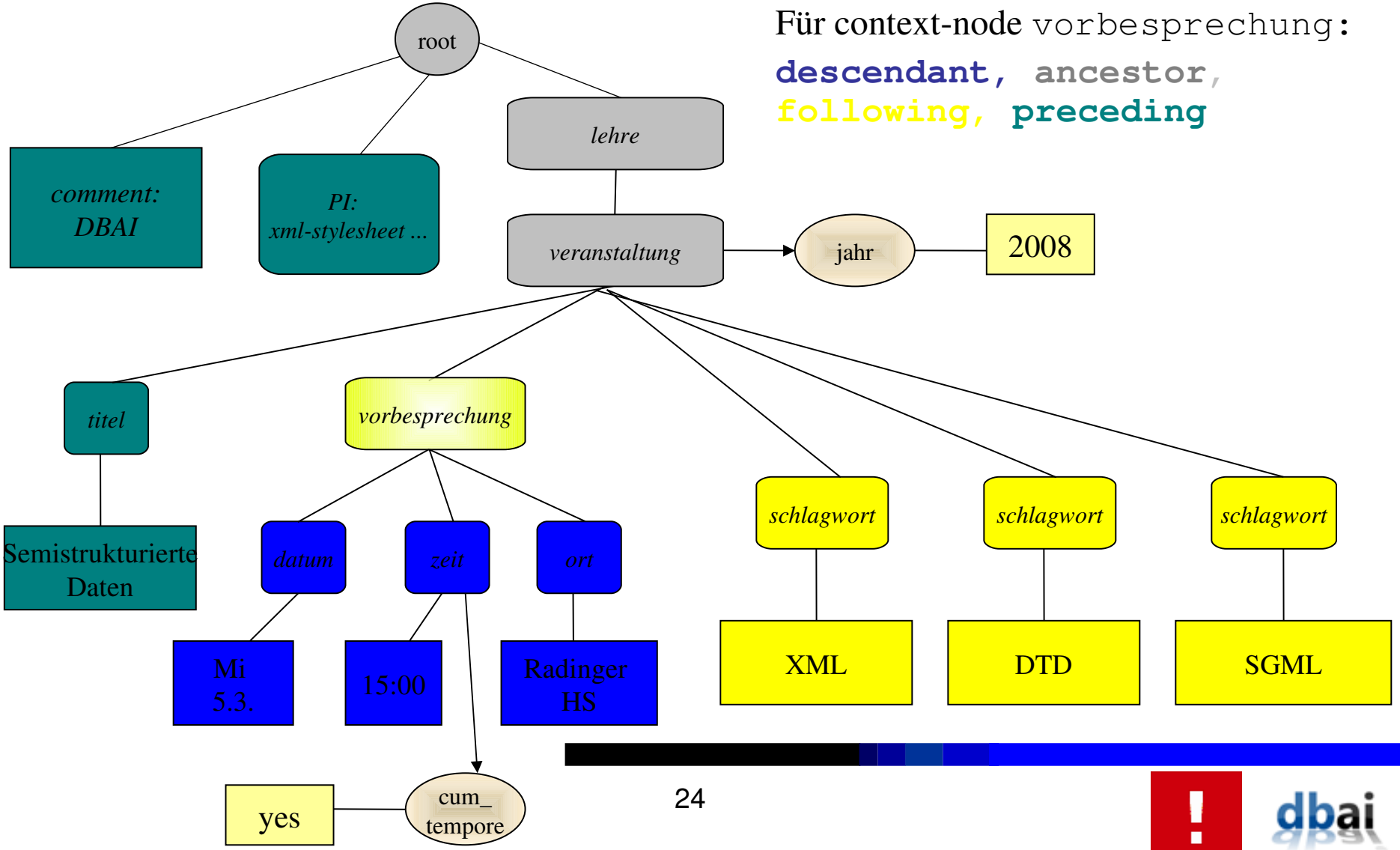
XPath Achsen (2)

- following: alle in document order nachfolgenden Knoten außer eigene Nachfahren
- following-sibling: alle nachfolgenden Geschwisterknoten
- preceding: alle in document order vorangehenden Knoten außer eigene Vorfahren
- preceding-sibling: alle vorangehenden Geschwisterknoten

preceding(-sibling) bzw. ancestor(-or-self) erzeugen node-sets in reverse document order.

- attribute: alle Attribute eines Elements
- namespace: alle Namespaces eines Elements

Dokumentenbaum: XPath Achsen (3)



Für context-node `vorbesprechung`:

descendant, ancestor, following, preceding

Node Tests

- Üblicherweise: Angabe eines Namen
  z.B. **`/child::lehre/descendant::zeit`**
- * steht für beliebigen Namen, z.B.:
  - **`child::*`** gibt alle Kind-Elementknoten zurück
  - **`attribute::*`** gibt alle Attributknoten zurück
- *praefix*:* steht für beliebige qual. Namen mit Präfix *praefix*
- **`comment()`**: alle Kommentar-Knoten
  z.B. **`/child::lehre/child::comment()`**
- **`processing-instruction(`*`ziel?`*`)`**: alle PI-Knoten
  z.B. **`/child::lehre/child::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::node()`**
- `//` = **`descendant-or-self::node()/`**
- `@` = Abkürzung für attribute-Achse

  z.B. `@*` entspricht **`attribute::*`**

Beispiele

`./titel`	alle titel-Elemente im momentanen Kontext (äquivalent zu `titel` bzw. `child::titel`)
`/titel`	selektiert titel, falls es das Dokument-Element ist.
`.//titel`	starte vom aktuellen context-node und selektiere alle titel Elemente, die tiefer liegen (also relativ): eigentlich `descendant-or-self::node()/titel` (vgl. `descendant::titel` !)
`//titel`	starte von Wurzel und selektiere alle titel Elemente, die tiefer liegen (also absolut)
`//zeit/../*`	liefert Element plus seine Geschwister

`/lehre/*/schlagwort` ist äquivalent zu `/child::lehre/child::*/child::schlagwort`

Filter (= "predicates")

- Angabe einer beliebigen XPath Expression in eckigen Klammern

  z.B. `//veranstaltung/schlagwort[position() >= 2]`

- Die Filter-Bedingung `[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]`

Beispiele

`veranstaltung[vorbesprechung]`

selektiert alle Elemente die ein Vorbesprechungselement enthalten
Bemerkung: Typumwandlung des Node-sets von "vorbesprechung" in boolean erforderlich.

`vorbesprechung[datum="Mi 7.3."]`

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

`zeit[@cum_tempore="yes"]`

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

`vorbesprechung[not(zeit/@cum_tempore="yes")]`

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

Beispiele

`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-Subelement 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[.="DTD"][2]`

    wählt unter den Schlagwörtern mit Wert DTD das zweite aus

    => selektiert in diesem Fall die leere Knotenmenge
  - `schlagwort[2][.="DTD"]`

    wählt das zweite Schlagwort aus, vorausgesetzt dass der Wert DTD ist.
    => selektiert in diesem Fall das zweite Schlagwort

Auswertung von Location Steps

Auswertung eines steps **`achse::node-test[x]`**

- Zuerst wird das node-set aufgrund von **`achse::node-test`** berechnet. => Ergebnis S = {s_1, ..., s_k}
- Nun wird für jeden Kandidaten s_i das predicate x bezüglich dem folgenden Context ausgewertet:
  - 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.

7.4. Operatoren

- Überblick
- Vergleichsoperatoren mit Knotenmengen

Überblick

- 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 eine built-in Funktion **`not()`**
    z.B. `//schlagwort[not(position()=2)]`
  - Operanden werden implizit in boolean konvertiert
    z.B. `//*[descendant::titel and descendant::datum]`
- Vergleichsoperatoren: **`<, >, <=, >=, =, !=`**
- Arithmetische Operatoren: **`+, -, *, mod, div`**
  (Bemerkung: **`div`** ist Dezimal-Division, weil "/" eine andere Bedeutung hat.)

Vergleichsoperationen mit Node-Sets

- Node-sets x in Ausdrücken der Form `x RelOp y` mit `RelOp` in {`<, <=, =, !=, >, >=`} haben "exists"-Semantik
- z.B.: `.//schlagwort = "XML"` liefert true, wenn es ein schlagwort-Element mit dem Textinhalt `"XML"` als Nachfahren des context-node gibt.
- 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 conext-node aus mittels `./veranstaltung/nummer` und b ist vom conext-node aus mittels `.//termin/lvanummer` erreichbar.
  - string(a) = string(b)

7.5. Vordefinierte Funktionen

- Überblick
- Node-set Funktionen
- String Funktionen
- Boolean Funktionen
- Number Funktionen

Überblick

- Klassifizierung (laut XPath Recommendation):
  - Node-set Funktionen
  - String Funktionen
  - Boolean Funktionen
  - Number Funktionen
- Beschreibung:
  result-type **function-name** (*arguments*)

Node-set Funktionen (1)

- *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

Node-set Funktionen (2)

- *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 Funktionen (1)

- *string* **string**(*object?*)
  Typkonvertierung in einen String. Falls Argument fehlt, wird der context-node in einen String konvertiert.
- *string* **concat**(*string,string,string**)
  Konkatenation von 2 oder mehr Strings
- *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"`**

String Funktionen (2)

- *string* **substring-before**(*string*,*string*)
  liefert substring des ersten Arguments vor dem ersten Auftreten des zweiten Arguments (sonst den leeren String).
- *string* **substring-after**(*string*,*string*)
  liefert substring des ersten Arguments nach dem ersten Auftreten des zweiten Arguments (sonst den leeren String).
- *number* **string-length** (*string?*)
  liefert String-Länge des Arguments (bzw. des String-Wertes des context-node)
- *string* **normalize-space** (*string?*)
  normalisiert Argument-String (bzw. des String-Wertes des context-node) bzgl. Whitespaces
- *string* **translate** (*string*,*string*,*string*)
  Zeichen-weise Transformation des ersten Arguments, z.B.:
  translate("---aaa---", "abc", "ABC") liefert **"---AAA---"**

Boolean Funktionen

- *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 Funktionen

- *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

Tools (1)

XpathTester:

- basiert auf XPath Engine von
  - Xalan XSLT Processor (XPathTester Version 1.3) bzw.
  - Saxon XSLT Processor (XPathTester Version 1.4)
- Start von der Kommandozeile aus:
  - java –jar xpathtester_1_3_xalan2.jar
- Bemerkung:
  - XPathTester ist nur ein "Spielzeug", um XPath kennen zu lernen.
  - XPath wird hauptsächlich als Teil von XSLT verwendet.

- http://www.dbai.tuwien.ac.at/education/ssd/SS08/uebung/xpathtester_1_3_xalan2.jar

Tools (2)

xpath (linux commandline zB auf minteka)

- Aufruf: xpath *file* "*xpath-query*"
- Achtung! unterstützt keine NS.

XPath Explorer:

- http://sourceforge.net/projects/xpe/

XPathTester und Xpath Explorer haben Probleme bei NS, insb. bei Redefinitionen von Präfices.