Datenbanksysteme

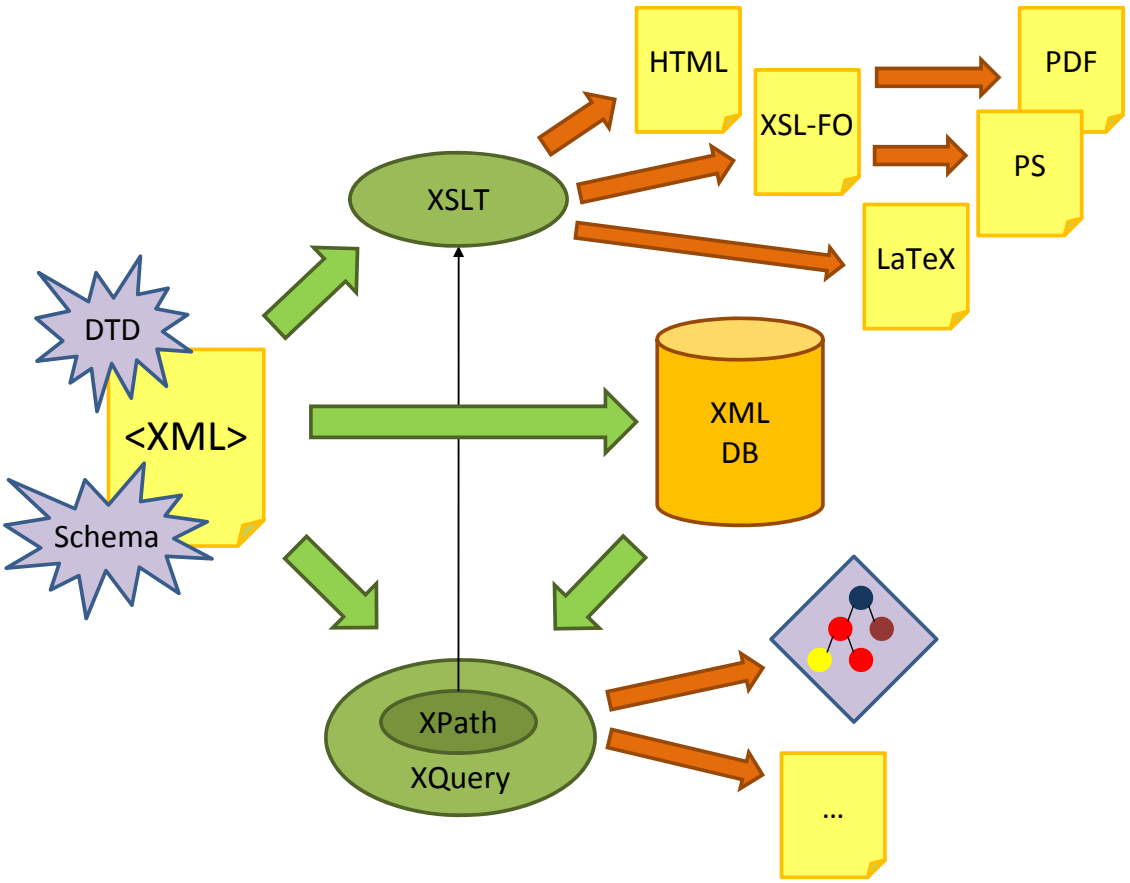
XML-Technologien

XML, XPath, XQuery und XSLT

22.6.2009

Dr. Martin Gieseking

Überblick



Speicherung von Daten

- seit Beginn des Computerzeitalters wurden unzählige Formate zur Speicherung von Daten entworfen
  - Binärformate
  - ASCII-basierte Formate
  - Mischformen
- das Spektrum reicht dabei von Formaten zur Beschreibung von sequenziellen Daten bis hin zu hoch strukturierten Dokumenten
- die wenigsten dieser Formate sind zueinander kompatibel
  - man benötigt auf das jeweilige Format zugeschnittene Module zum Schreiben und Lesen der Daten
    - komplexe Formate erfordern in aller Regel aufwändige Parser
    - die Entwicklung stabiler Parser ist zeitintensiv und fehleranfällig
  - selbst neuere Versionen eines Formats müssen nicht kompatibel zu einer Vorgängerversion sein
    - ältere Dateien können unter Umständen nicht mehr korrekt eingelesen werden
    - jede neue Version erfordert sehr oft einen neuen Parser

Markup-Sprachen

- eine beliebte Methode zur Beschreibung strukturierter Daten ist die Verwendung von *Markup-Sprachen*
- dabei handelt es sich um textbasierte Dokumente, deren Bestandteile sich einer von zwei Kategorien zuordnen lassen
  - Content-Elemente
    - Zeichenfolgen, welche die atomaren Daten repräsentieren
  - Markup-Elemente
    - spezielle Zeichenfolgen, die die zugehörigen Datenelemente in besonderer Weise auszeichnen
- es existieren zahlreiche Markup-Sprachen, wie z.B. RTF, LaTeX und HTML

```
{\rtf\ansi\paperw11907\paperh16840
{\pard\plain Dies ist {\b mein} Text.}}
```

```
\documentclass{article}
\begin{document}
Dies ist \textbf{mein} Text.
\end{document}
```

```
<html>
  <body>
    Dies ist <b>mein</b> Text.
  </body>
</html>
```

XML – Extensible Markup Language

- Was ist XML?
  - XML ist eine Spezifikation des *World Wide Web Konsortiums* (W3C)
  - XML ist eine *Meta-Markup-Sprache,* auf deren Grundlage konkrete Markup-Sprachen definiert werden können
    - als X*ML-Dokumente* bezeichnet man alle Dokumente, die auf einer dieser konkreten Markup-Sprachen basieren
    - alle XML-Dokumente lassen sich prinzipiell unabhängig von ihrem Anwendungsbereich auf die gleiche Weise einlesen und speichern
    - Beispiele für XML-Sprachen: MathML, XHTML, XSLT
  - XML ist ein standardisiertes Meta-Format zur Beschreibung strukturierter Daten

- Was ist XML nicht?
  - XML ist keine Programmiersprache
  - XML ist kein HTML-Nachfolger oder -Ersatz
    - HTML und XML sind Untermengen von SGML

XML – Extensible Markup Language

- XML wurde als Mechanismus zur Beschreibung strukturierter Dokumente entwickelt
- wird heute zur Beschreibung nahezu beliebiger strukturierter Daten verwendet
  - u.a. zum plattformunabhängigen Austausch von Daten zwischen verschiedenen Applikationen und Plattformen
- XML-Dokumente implizieren keine festgelegte Visualisierung der Daten
  - anders als z.B. bei HTML
- XML definiert kein fest vorgegebenes Markup
  - Markup-Elemente können für beliebige Anwendungsbereiche frei definiert werden
  - die XML-Spezifikation legt allerdings fest, wie das Markup aufgebaut sein muss und wie die Dokumentelemente syntaktisch miteinander zu verknüpfen sind

XML-Beispiel

```xml
<?xml version="1.0"?>
<!-- Vorlesungsverzeichnis Sommersemester 2009 -->
<vv>
    <abschnitt titel="Mathematik &amp; Informatik">
        <abschnitt titel="Grundstudium">
            <v nr="1.234" typ="v">
                <doz>
                    <titel>Prof. Dr.</titel>
                    <vname>Klaus</vname>
                    <nname>Meier</nname>
                </doz>
                <titel>Einführung in die monotone Algebra I</titel>
                <zeit>Mo 10:00-12:00</zeit>
                <raum>12/13</raum>
            </v>
            <v nr="1.247" typ="ü">
                <doz>
                    <vname>Sandra</vname>
                    <nname>Schmidt</nname>
                </doz>
                <titel>Übung zur Einführung in die monotone Algebra I</titel>
                <zeit>Mi 8:00-12:00</zeit>
                <raum>97/E9</raum>
            </v>
        </abschnitt>
    </abschnitt>
</vv>
```


Bausteine einer XML-Datei

- XML-Dateien können sich aus folgenden Bausteinen zusammensetzen:
  - Elemente
  - Elementattribute
  - Text und Entities
  - Kommentare
  - Verarbeitungsanweisungen (processing instructions)
  - CDATA-Abschnitte
  - DOCTYPE-Angabe

```xml
<?xml version="1.0"?>
<!-- Vorlesungsverzeichnis Sommersemester 2009 -->
<vv>
    <abschnitt titel="Mathematik &amp; Informatik">
        <abschnitt titel="Grundstudium">
            <v nr="1.234" typ="v">
                <doz>
                    <titel>Prof. Dr.</titel>
                    <vname>Klaus</vname>
                    <nname>Meier</nname>
                </doz>
                <titel>Einführung in die ...</titel>
                <zeit>Mo 10:00-12:00</zeit>
                <raum>12/13</raum>
            </v>
        </abschnitt>
    </abschnitt>
</vv>
```

Syntax der Markup-Elemente

- die strukturierenden Bausteine werden *Elemente* genannt
- sie bestehen aus einem öffnenden und einem schließenden *Tag* sowie dem *Elementinhalt*
  - öffnende Tags haben immer die Form <name attr1="val1" attr2="val2" ...>
    - der Name kann aus Buchstaben, Ziffern und den Zeichen **_ . - ·** bestehen
      - erstes Zeichen muss ein Buchstabe sein
      - darf nicht mit *xml, Xml, xMl, xmL, XMl, XmL, xML* oder *XML* beginnen
    - zwischen „<“ und *name* darf sich kein Leerzeichen befinden
    - Attribute werden in der angegebenen Form durch Whitespaces getrennt hinter dem Markup-Namen aufgelistet
    - die Reihenfolge der Attribute ist nicht signifikant
    - Element- und Attributnamen sind case-sensitive
  - schließende Tags haben immer die Form </name>
    - der Name muss mit dem des zugehörigen öffnenden Tag übereinstimmen
    - schließende Tags enthalten neben dem Namen keine weiteren Informationen
  - alles, was sich zwischen öffnendem und schließendem Tag befindet, bildet den Elementinhalt

Syntax der Markup-Elemente

- zwischen öffnendem und schließendem Tag befindet sich der Elementrumpf (body), auf den das Markup bezogen ist

```
<aufgabe punkte="3">
  Erklären Sie Ihrem Tutor die wesentlichen Unterschiede
  zwischen HTML und XML.
</aufgabe>
```

- für den Fall, dass der Elementinhalt leer ist, gibt es die Kurzschreibweise <name attr1="val1" attr2="val2" ... />

```
<uhrzeit zeitzone="GMT"></uhrzeit>
```
ist identisch mit
```
<uhrzeit zeitzone="GMT"/>
```

- Elemente können sowohl Text als auch weitere Elemente und Kommentare enthalten
  - Text und Elemente können innerhalb des Rumpfes beliebig aneinander gereiht werden
  - Elemente können beliebig tief geschachtelt werden

Kommentare und Entities

- neben Markup- und Textelementen spezifiziert XML noch weitere Konstrukte
  - Kommentare
    <!-- dies ist ein Kommentar -->
    - Kommentare können nicht geschachtelt werden
  - Entities
    &name; &#dez; &#xhex
    - spezielle Zeichenfolgen, die ein Textelement beschreiben
    - einige Zeichen haben in der XML-Syntax eine besondere Bedeutung (Metazeichen), d.h. sie können nicht als normaler Textbestandteil verwendet werden
    - zahlreiche Zeichen sind nicht im normalen Zeichenvorrat eines Zeichensatzes enthalten, so dass sie nicht direkt eingegeben werden können (z.B. mathematische oder musikalische Zeichen, Ligaturen usw.)

<	<
>	>
'	'
"	&quote;
&	&

```
<aufgabe punkte="1">
  <!-- das sollten eigentlich alle beantworten können -->
  Wofür steht in XML die Zeichenfolge &lt;!-- ... -->?
</aufgabe>
```

Wohlgeformte XML-Dokumente

- ein XML-Dokument heißt *wohlgeformt*, wenn es die Syntax- und Strukturvorgaben der XML-Spezifikation einhält
  - über 100 Regeln, die größtenteils intuitiv aus den bisher beschriebenen Aspekten hervorgehen
- die Wohlgeformtheit kann ohne Kenntnis der in einem Dokument zulässigen Markup-Elemente überprüft werden

nicht wohlgeformt

```
<?xml version="1.0"?>
<blatt nr="1">
  <aufgabe punkte="4">
    <list>
      <li>Punkt 1</li>
      <li>Punkt 2
  </aufgabe>
      </li>
    </list>
</Blatt>

<blatt nr="2">
  <aufgabe Punkte="6"/>
</blatt>
```

wohlgeformt

```
<?xml version="1.0"?>
<blattsammlung>
  <blatt nr="1">
    <aufgabe punkte="4">
      <list>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
      </list>
    </aufgabe>
  </blatt>

  <blatt nr="2">
    <aufgabe PUNKTE="6"/>
  </blatt>
</blattsammlung>
```

Document Type Definitions

- die genaue Struktur der im XML-Format beschriebenen Daten kann mit einer Document Type Definition (DTD) festgelegt werden
- eine DTD legt fest,
  - welche Markup-Elemente erlaubt sind,
  - welche und wie viele Markup-Elemente in welcher Reihenfolge in anderen Markup-Elementen enthalten sein dürfen,
  - welche Attribute ein Markup-Element besitzt,
  - ob Attribute optional oder verpflichtend sind,
  - welche zusätzlichen Entities erlaubt sind,
  - ...
- eine DTD kann direkt in die zugehörige XML-Datei integriert oder in einer separaten Datei abgelegt werden
- DTDs verwenden ein eigenes, ebenfalls von SGML abgeleitetes Format
- eine DTD besteht aus einer Folge von Regeln

DTD-Beispiel

```
<!ELEMENT vv (abschnitt+)>
<!ATTLIST vv semester CDATA #IMPLIED>

<!ELEMENT abschnitt (abschnitt+ | v+)>
<!ATTLIST abschnitt titel CDATA #REQUIRED>

<!ELEMENT v (doz+, titel, zeit, raum)>
<!ATTLIST v nr CDATA #REQUIRED>
<!ATTLIST v typ (v | s | ü) #REQUIRED>

<!ELEMENT doz (titel?, vname, nname)>

<!ELEMENT titel (#PCDATA)>
<!ELEMENT zeit (#PCDATA)>
<!ELEMENT raum (#PCDATA)>
<!ELEMENT nname (#PCDATA)>
<!ELEMENT vname (#PCDATA)>
```

```
<?xml version="1.0"?>
<!DOCTYPE vv SYSTEM "vv.dtd">
<!-- Vorlesungsverzeichnis Sommersemester 2009 -->
<vv>
    <abschnitt titel="Mathematik &amp; Informatik">
        <abschnitt titel="Grundstudium">
            <v nr="1.234" typ="v">
                <doz>
                    <titel>Prof. Dr.</titel>
                    <vname>Klaus</vname>
                    <nname>Meier</nname>
                </doz>
                <titel>Einführung in die ...</titel>
                <zeit>Mo 10:00-12:00</zeit>
                <raum>12/13</raum>
            </v>
        </abschnitt>
    </abschnitt>
</vv>
```

Schemasprachen

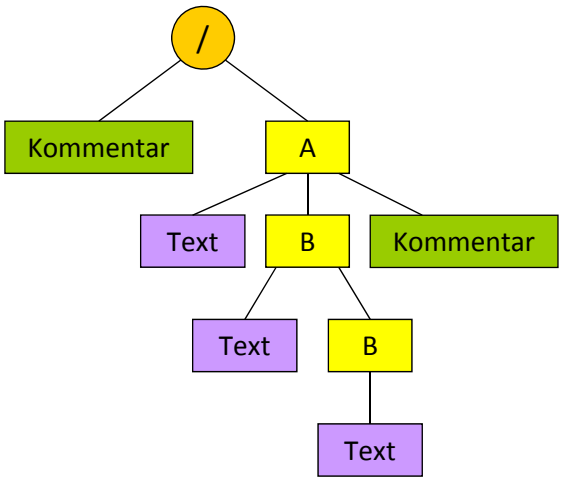
- DTDs werden für neue Spezifikationen kaum noch verwendet
  - Ausnahme: einige Dokumentformate
- Nachteile:
  - kein XML, aber Bestandteil der XML-Spezifikation
  - kennt keine Namensräume
  - kennt keine Datentypen
- Alternative: Schemasprachen
  - XML Schema (W3C-Standard)
  - Relax NG (ISO-Standard)
  - Schematron (ISO-Standard)

- XML-Dateien, die den Strukturvorgaben einer DTD oder eines Schemas genügen, werden *valide* genannt.

Struktur von XML-Dokumenten

- XML-Dokumente besitzen eine Baumstruktur
  - Wurzelknoten ist ein „unsichtbarer“ Knoten vom Typ *Document*
    - wird im Folgenden durch einen Slash (/) gekennzeichnet
  - bis auf den Dokumentknoten haben alle Elemente ein eindeutiges Elternelement
- der Dokumentknoten muss genau einen Elementknoten, das *Wurzelelement*, enthalten

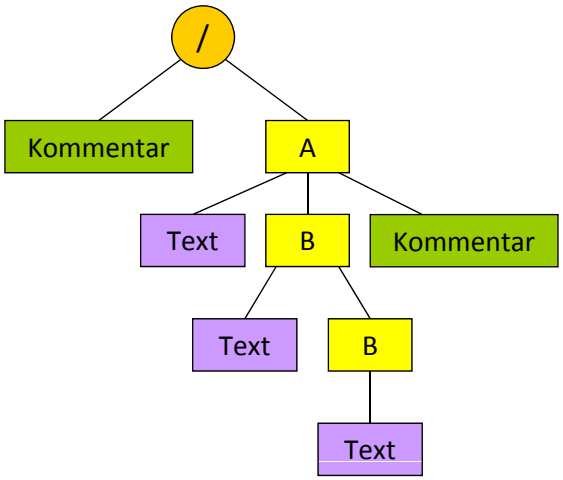
```
<?xml version="1.0"?>
<!- erster Kommentar -->
<A>
  erster Textteil
  <B>
    zweiter Textteil
    <B>
      dritter Textteil
    </B>
  </B>
  <!- zweiter Kommentar -->
</A>
```



Navigation in XML-Bäumen mit XPath

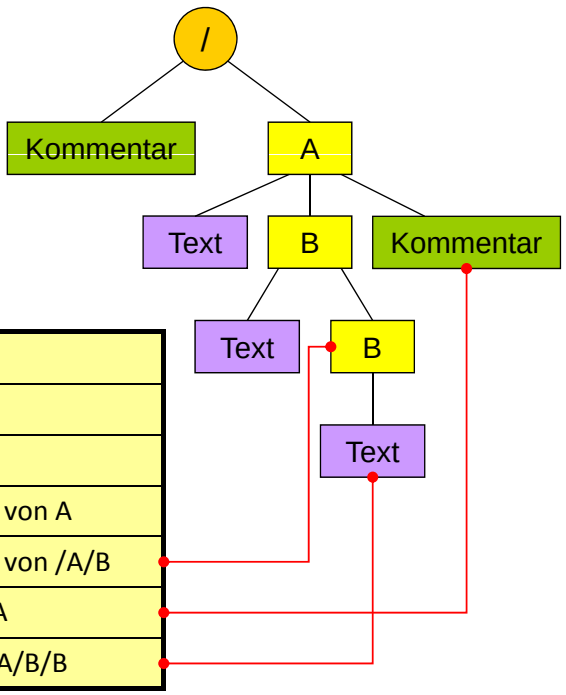
- Zur Navigation in XML-Dokumenten und zur gezielten Auswahl von Bestandteilen des Dokuments wurde **XPath** entwickelt
  - vom W3C entwickelte Lokatorsprache
  - XPath-Ausdrücke sind in aller Regel einzeilige Pfadangaben
  - XPath-Ausdrücke bestehen aus einer Folge von *Knotentests*, die einen Einzelknoten oder eine Knotenmenge/-sequenz beschreiben

```
<?xml version="1.0"?>
<!- erster Kommentar -->
<A>
  erster Textteil
  <B>
    zweiter Textteil
    <B>
      dritter Textteil
    </B>
  </B>
  <!- zweiter Kommentar -->
</A>
```



XPath: Einfache Pfadangaben

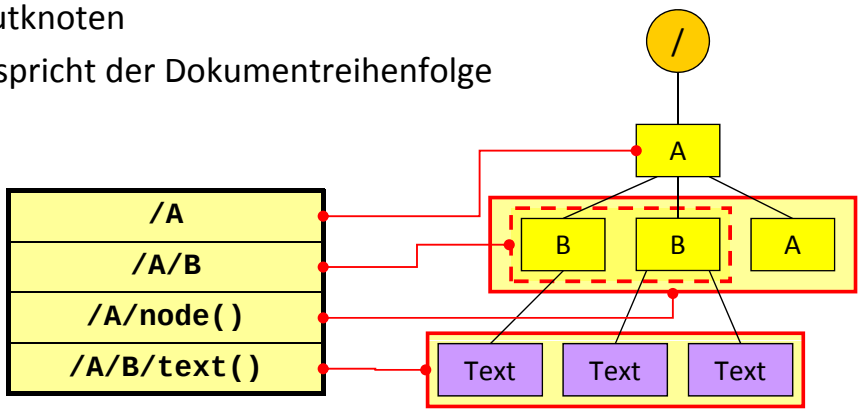
- XPath orientiert sich an der UNIX-Syntax zur Navigation im Dateisystem und erweitert diese um XML-spezifische Aspekte
  - Knoten können in Form von Pfaden getrennt durch Slashes (/) ausgewählt werden
  - Elementknoten werden durch ihren Namen selektiert
  - Text- und Kommentarknoten werden durch *text()* bzw. *comment()* selektiert



XPath-Ausdruck	Bedeutung
`/`	Dokumentknoten (Wurzel)
`/A`	Elementknoten A
`/A/B`	Elementknoten B unterhalb von A
`/A/B/B`	Elementknoten B unterhalb von /A/B
`/A/comment()`	Kommentar unterhalb von A
`/A/B/B/text()`	Textknoten unterhalb von /A/B/B

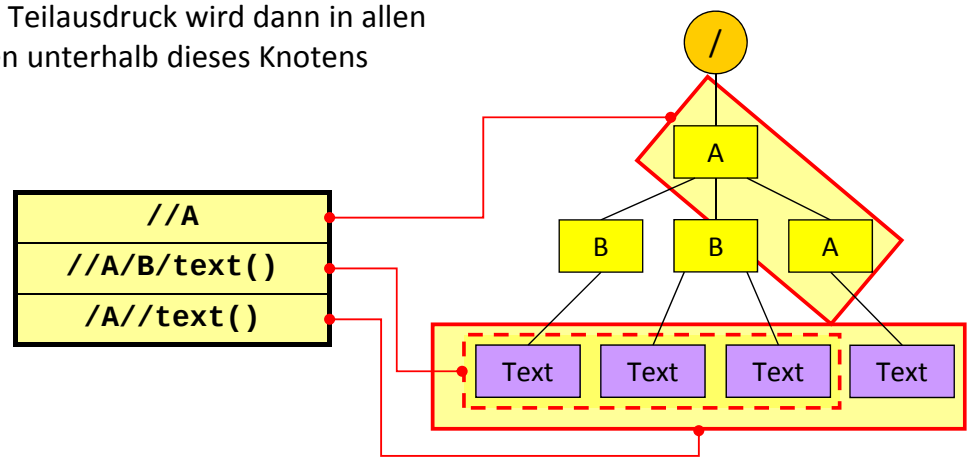
XPath: Knotensequenzen

- da die Knoten nicht durch eindeutige IDs sondern durch ggf. mehrfach vorkommende Bezeichner ausgewählt werden, kann ein XPath-Ausdruck mehrere Knoten, sog. *Knotensequenzen*, bezeichnen
  - Knotensequenzen können prinzipiell beliebige Knoten des Baums enthalten (keine Beschränkung auf gleiche Zweige oder Ebenen)
  - die Knotentests ***, *text()* und *comment()* liefern **alle** Knoten vom Typ Element, Text bzw. Kommentar, die auf die Pfadangabe passen
  - mit dem Knotentest *node()* erhält man unabhängig vom Typ alle Knoten, die zur Pfadangabe passen
    Ausnahme: Attributknoten
  - die Sortierung entspricht der Dokumentreihenfolge



XPath: rekursiver Abstieg

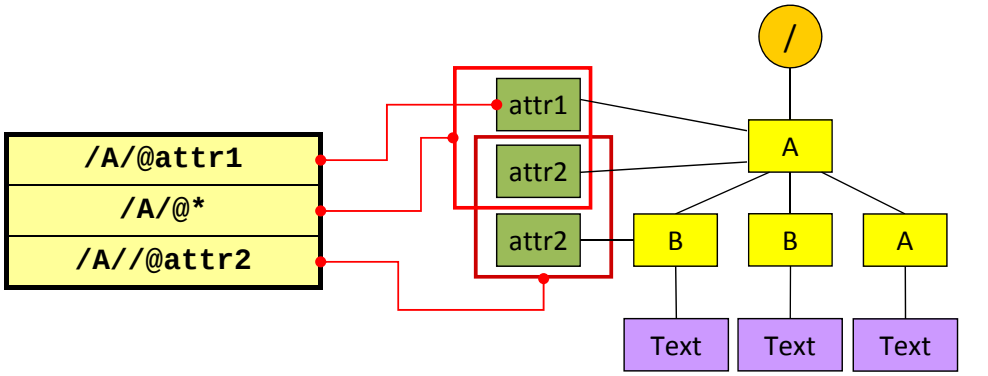
- ohne weitere Angaben bezieht sich ein Knotentest immer auf Kinder des aktuellen Knotens
- XPath bietet die Möglichkeit, automatisch in allen Teilbäumen unterhalb eines gegebenen Knotens nach passenden Knoten zu suchen
  - dies wird durch zwei direkt aufeinander folgende Slashes (//) kenntlich gemacht
    - zunächst wird zum Knoten navigiert, der auf der linken Seite von // steht (falls der XPath-Ausdruck mit // beginnt, startet die Suche bei der Wurzel (/))
    - der rechte Teilausdruck wird dann in allen Teilbäumen unterhalb dieses Knotens gesucht



XPath: Attributknoten

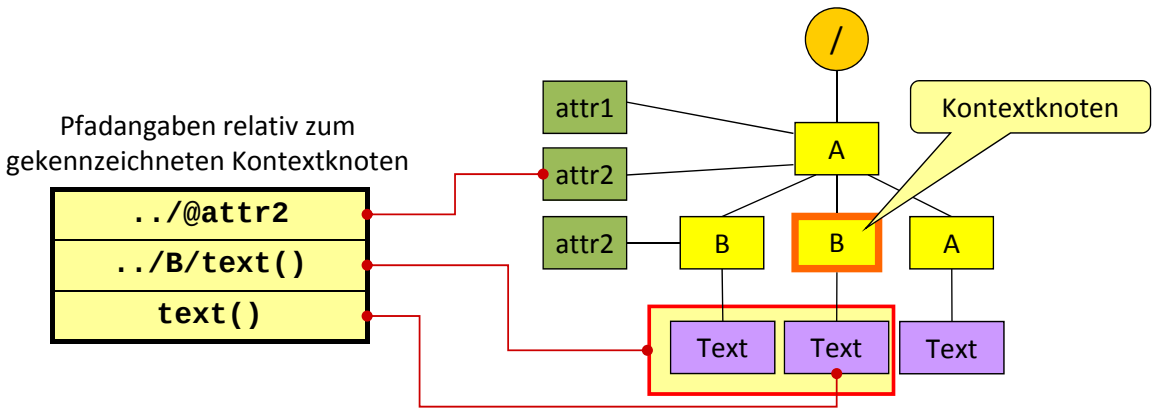
- die Attributknoten eines Elements werden durch einen vorangestellten Klammeraffen (@) adressiert
  - mit **@*** erhält man alle Attributknoten der spezifizierten Elemente
  - wie alle anderen Knoten können auch Attributknoten mit Hilfe des rekursiven Abstiegs (//) in Unterbäumen gesucht werden

```
<?xml version="1.0"?>
<A attr1="val1" attr2="val2">
  <B attr2="val3">
    Text
  </B>
  <B>Text</B>
  <A>Text</A>
</A>
```



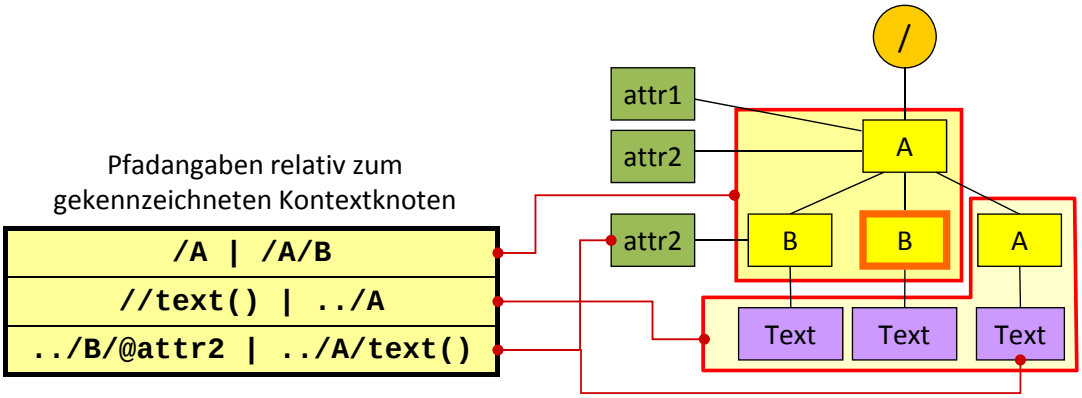
XPath: relative Pfadangaben

- eine XPath-Pfadangabe muss nicht zwingend bei der Wurzel beginnen
  - eine Pfadangabe kann relativ zu einem beliebigen Knoten notiert werden
  - der Knoten, auf den sich die Pfadangabe bezieht, wird *Kontextknoten* genannt
- alle bisher vorgestellten Pfadangaben bewirken einen Abstieg zu den Nachfahren
- mit .. erhält man den Elternknoten des aktuellen Knotens



XPath: Kombination von Knotensequenzen

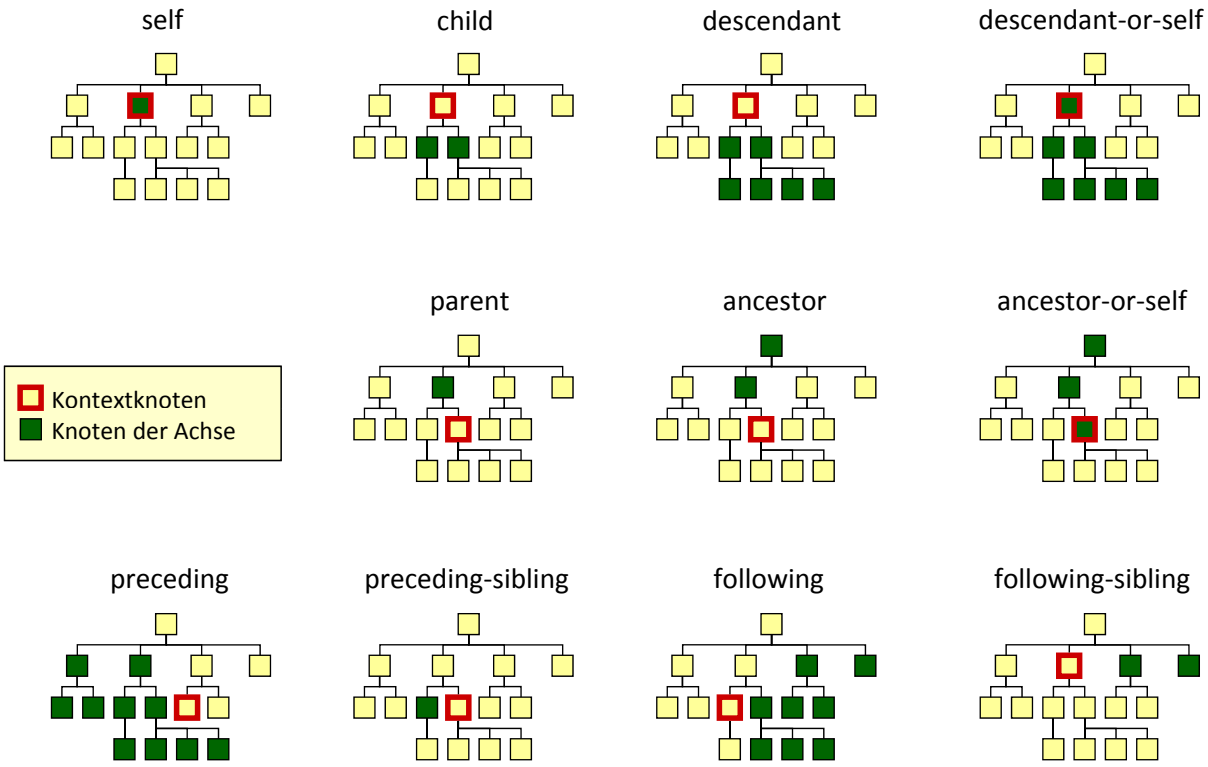
- mit Hilfe des Kompositionsoperators „|“ lassen sich zwei Knotensequenzen vereinigen
  - auf beiden Seiten des Operators steht eine Pfadangabe
  - relative Pfadangaben beziehen sich jeweils auf den aktuellen Kontextknoten
  - der Kompositionsoperator bindet schwächer als alle anderen Pfadoperatoren
  - die Sortierung entspricht der Dokumentreihenfolge



XPath: Achsen

- Knotentests beziehen sich normalerweise auf die Kinder des aktuellen Kontextknotens
  - **/person/titel** prüft, ob der Dokumentknoten einen *person*-Kindknoten und dieser wiederum einen *titel*-Kindknoten enthält
- um von einem Kontextknoten aus in andere Baumbereiche navigieren zu können, stellt XPath 13 *Achsen* bereit
  - Achsen beschreiben Bereiche des XML-Baums relativ zum Kontextknoten, z.B. Geschwisterknoten, Parallelknoten, Vorgängerknoten usw.
- Achsen werden durch einen Bezeichner gefolgt von zwei Doppelpunkten vor einem Knotentest notiert
  - **parent::A** liefert den Elternknoten, wenn es ein Element A ist
  - **ancestor::A** liefert alle Vorgängerknoten, die vom Elementtyp A sind

XPath: Achsen



XPath: Kurzschreibweisen

- für häufig verwendete Achsen gibt es Kurzbezeichner

Kurzform	ausführliche Form
`.`	`self::node()`
`..`	`parent::node()`
`//`	`/descendant-or-self::node()/`
`name`	`child::`*`name`*
`@name`	`attribute::`*`name`*

- Beispiel:

  `A//B/@attrib` ist eine Kurzform von
  `child::A/descendant-or-self::node()/child::B/attribute::attrib`

XPath: Prädikate

- *Prädikate* grenzen Resultate von Knotentests durch Boolsche Ausdrücke weiter ein
  - alle Knoten, auf die der Ausdruck nicht zutrifft, werden aus der Ergebnissequenz entfernt
- Prädikate werden in eckigen Klammern hinter einem Knotentest angegeben
  - Beispiel: **`A[@attrib = 'value']`**
    liefert alle *A*-Knoten, die einen Attributknoten *attrib* besitzen, der wiederum den Wert *value* enthält
  - Beispiel: **`//A[B]`**
    liefert alle *A*-Knoten, die einen *B*-Kindknoten besitzen

a* = *b	*a* gleich *b*
a* != *b	*a* ungleich *b*
a* < *b	*a* kleiner *b*
a* > *b	*a* größer *b*
a* <= *b	*a kleiner oder gleich b*
a* >= *b	*a* größer oder gleich *b*

a* + *b	Addition
a* – *b	Subtraktion
a* * *b	Multiplikation
a* div *b	Division
a* mod *b	Modulo
a* or *b	logisches ODER
a* and *b	logisches UND

XPath: Prädikate

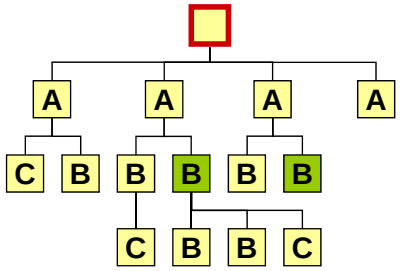
- XPath definiert eine Reihe von Funktionen, die u.a. zur Formulierung von Prädikaten verwendet werden können
- eine kleine Auswahl:

Funktion	Returntyp	Beschreibung
`concat(`*`s1`*`,...,`*`sn`*`)`	string	verkettet die Strings *s1* bis *sn* zu einem neuen String
`contains(`*`s1`*`, `*`s2`*`)`	bool	prüft, ob String *s1* den String *s2* enthält
`count(`*`K`*`)`	number	Anzahl der Knoten in Sequenz *K*
`name(`*`K`*`)`	string	Name des ersten Knotens in Sequenz *K*
`not(`*`A`*`)`	bool	boolsche Negation
`position()`	number	Position des Kontextknotens in der Ergebnissequenz
`string-length(`*`s`*`)`	number	Länge von String *s*
`substring(`*`s`*`, `*`n`*`, `*`l`*`)`	string	Teilstring von *s*, beginnend beim *n*-ten Zeichen und Länge *l*

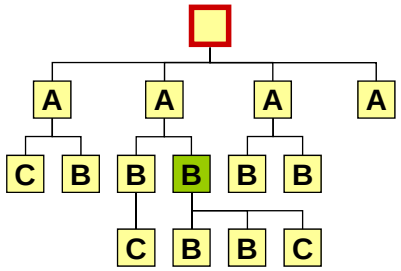
- Beispiele:
  - **`A[count(B) >= 3]`**
  - **`A[contains(., 'Hallo')]`**
  - **`A[position() = 2]`** oder kurz **`A[2]`**
  - **`concat('Nr. ', @pos)`**

XPath: Prädikate

- Achtung: auch Prädikate mit Positionsangaben können mehrere Knoten bezeichnen
- Beispiel: **`/A/B[2]`**



- Wie muss die Pfadangabe aussehen, um nur den ersten B-Knoten auszuwählen?
- **`/A[1]/B[2]`**



XPath vs. XQuery

- mit XPath kann man:
  - in XML-Bäumen navigieren
  - einzelne Knoten und Knotensequenzen aus XML-Bäumen auswählen
- mit XPath kann man nicht:
  - Daten ändern
  - Daten sortieren
  - Daten neu gruppieren
- XQuery ist eine deklarative Programmiersprache mit XPath als Untermenge
  - ermöglicht komplexe Abfragen von Daten aus XML-Dokumenten
  - bietet Konstrukte zur Erzeugung neuer XML-Bestandteile
  - Ändern von XML-Daten nur mit *XQuery Update Facility* möglich
- Auswahl kostenloser XQuery-Tools
  - *BaseX* XML-Datenbank mit GUI, Uni Konstanz (Java)
  - *Saxon* von Michael Kay (Java, .NET)
  - *Zorba* (C++)

XQuery: FLWOR-Ausdrücke

- zentrales XQuery-Konstrukt sind die sog. FLWOR-Ausrücke
  - wird wie engl. *flower* ausgesprochen
  - FLWOR steht für **F**or, **L**et, **W**here, **O**rder by, **R**eturn
  - Reihenfolge, in der die 5 Anweisungen innerhalb eines Ausdrucks angeordnet werden
- **let**: Definition von Variablen
  - Variablen werden durch ein vorangestelltes $-Zeichen markiert
  - Beispiel: **let $personen := //person[vorname = 'Maria']**
  - Variablen können sowohl atomare Werte (Zahlen, Strings, Booleans usw.) als auch Knoten und Sequenzen aufnehmen
  - einmal definierte Variablen können nicht geändert werden (es gibt keinen Zuweisungsoperator in XQuery)
- **return**: Resultate zurückgeben
  - legt fest, welche Werte eines FLWOR-Ausdrucks als Ergebnis zurückgeliefert werden sollen

```
let $personen := //personen[vorname='Maria']
return $personen/nachname
```


XQuery: FLWOR-Ausdrücke

- **for**: Iteration über Sequenzen
  - ermöglicht iterativen Zugriff auf die Elemente einer Sequenz

```
for $person in //personen[vorname='Maria']
return $person/nachname
```

  - ermöglicht das Durchnummerieren der Sequenzelemente

```
for $person at $i in //personen[vorname='Maria']
return concat($i,' ',$person/nachname,'&#10;')
```

- **where**: Iteration beschränken
  - es werden nur Sequenzelemente berücksichtigt, die die angegebene Bedingung erfüllen

```
for $person at $i in //personen[vorname='Maria']
where $i > 3
return concat($i,' ',$person/nachname,'&#10;')
```

XQuery: FLWOR-Ausdrücke

- **order by**: Sortieren der Ergebnissequenz

```
for $person at $i in //personen[vorname='Maria']
let $nachname := $person/nachname
where $i >= 3
order by $nachname
return concat($i,' ',$nachname,'&#10;')
```

- Variable mit Ergebnis eines FLWOR-Ausdrucks definieren

```
(: Nachnamen-Elemente sortiert in Var. ablegen :)
let $nachnamen :=
  for $person in //personen
  let $n := $person/nachname
  order by $n
  return $n

(: Nachnamen nummeriert ausgeben; doppelte entfernen :)
for $n at $i in distinct-values($nachnamen)
return concat($i,' ',$n,'&#10;')
```


XQuery: Element-Konstruktoren

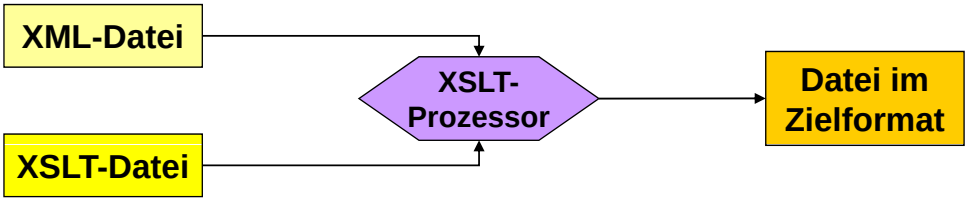
- XQuery-Skripte können XML-Fragmente enthalten
  - Inhalte von XML-Elementen und -Attributen werden nur ausgewertet , wenn sie mit {...} eingeklammert werden

```
<html>
  <head>
    <title>Nachnamen</title>
  </head>
  <body>
    <ul>{
      for $person in //personen
      let $nachname := $person/nachname
      order by $nachname
      return <li>{data($nachname)}</li>
    }
    </ul>
  </body>
</html>
```

liefert den Textinhalt vom Element $nachname („entfernt die Tags“)

XSLT: Extensible Stylesheet Language Transformations

- XSLT ist eine deklarative Programmiersprache im XML-Format, mit der Transformationen von XML-Dateien in andere Formate beschrieben werden können
  - XSLT-Dateien sind portabel, d.h. die Transformationen können unabhängig von Plattform und verwendeter Programmierumgebung eingesetzt werden
  - verschiedene Zielformate durch Austausch der Stylesheets
  - XML-Daten können unverändert genutzt werden
- XSLT-Stylesheets werden von einem XSLT-Prozessor ausgeführt
  - die bekanntesten XSLT-Prozessoren sind
    - *Saxon* von Michael Kay (http://saxon.sourceforge.net)
    - *Xalan* von der Apache Group (http://xml.apache.org/xalan-j)
    - *libxslt* des Gnome Projekts (http://xmlsoft.org/XSLT)



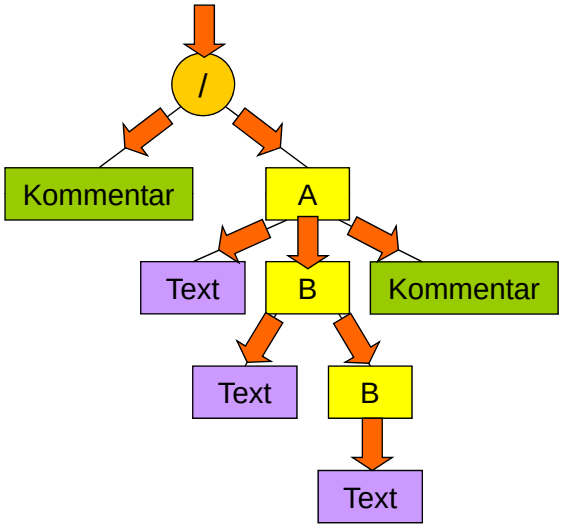
XSLT-Beispiel

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
        <ul>
          <xsl:apply-templates/>
        </ul>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="abschnitt">
    <li>
      <xsl:value-of select="@titel"/>
      <ul>
        <xsl:apply-templates/>
      </ul>
    </li>
  </xsl:template>

  <xsl:template match="text()"/>
</xsl:stylesheet>
```

```
<buch>
  <abschnitt titel="Erstes Kapitel">
    <abschnitt titel="Unterkapitel">
    ...
    </abschnitt>
  </abschnitt>
  <abschnitt titel="Zweites Kapitel">
    <abschnitt titel="Unterkapitel">
    ...
    </abschnitt>
    <abschnitt titel="Unterkapitel">
    ...
    </abschnitt>
  </abschnitt>
</buch>
```

XSLT: Push-Processing

- das rekursive Weiterreichen des Template-Matchings von der Wurzel bis zu den Blättern wird *Push-Processing* genannt
  - ein allein auf Push-Processing basierendes Stylesheet enthält in aller Regel zahlreiche einfach aufgebaute Templates




```
<xsl:template match="/">
  <html>
    <body>
       <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="A">
  Ein A-Knoten:
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="B">
  Ein B-Knoten:
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="text()">
  Ein Textknoten:
  <xsl:value-of select="."/>
</xsl:template>
```


XSLT: Implizite Templates

- findet der XSLT-Prozessor für den aktuellen Knoten im Stylesheet kein passendes Template, wird ein Standard-Template verwendet
- Diese Standard-Templates sehen wie folgt aus:

Elemente	`<xsl:template match="*">`
	`  <xsl:apply-templates/>`
	`</xsl:template>`

Attribute	`<xsl:template match="@*">`
	`  <xsl:value-of select="."/>`
	`</xsl:template>`

Text	`<xsl:template match="text()">`
	`  <xsl:value-of select="."/>`
	`</xsl:template>`

Kommentare	`<xsl:template match="comment()"/>`

XSLT: weitere Features

- XSLT stellt eine Reihe weiterer Konstrukte zum Erstellen umfangreicher Stylesheets zur Verfügung
  - Variablendefinitionen (wie bei XQuery nicht nachträglich änderbar)
  - Fallunterscheidungen (**if**, **choose**/**when**/**otherwise**)
  - Iterationen über Sequenzen (**for-each**, ähnlich wie bei XQuery)
  - Anweisungen zum Sortieren und Gruppieren (**sort**, **key**, **for-each-group**)
  - Definition eigener Funktionen (**exsl:function**, **xsl:function**)
  - ...
- Mehr dazu in der Veranstaltung XML und XSLT im kommenden Wintersemester