



Forschungszentrum
Informatik



Universität
Karlsruhe (TH)



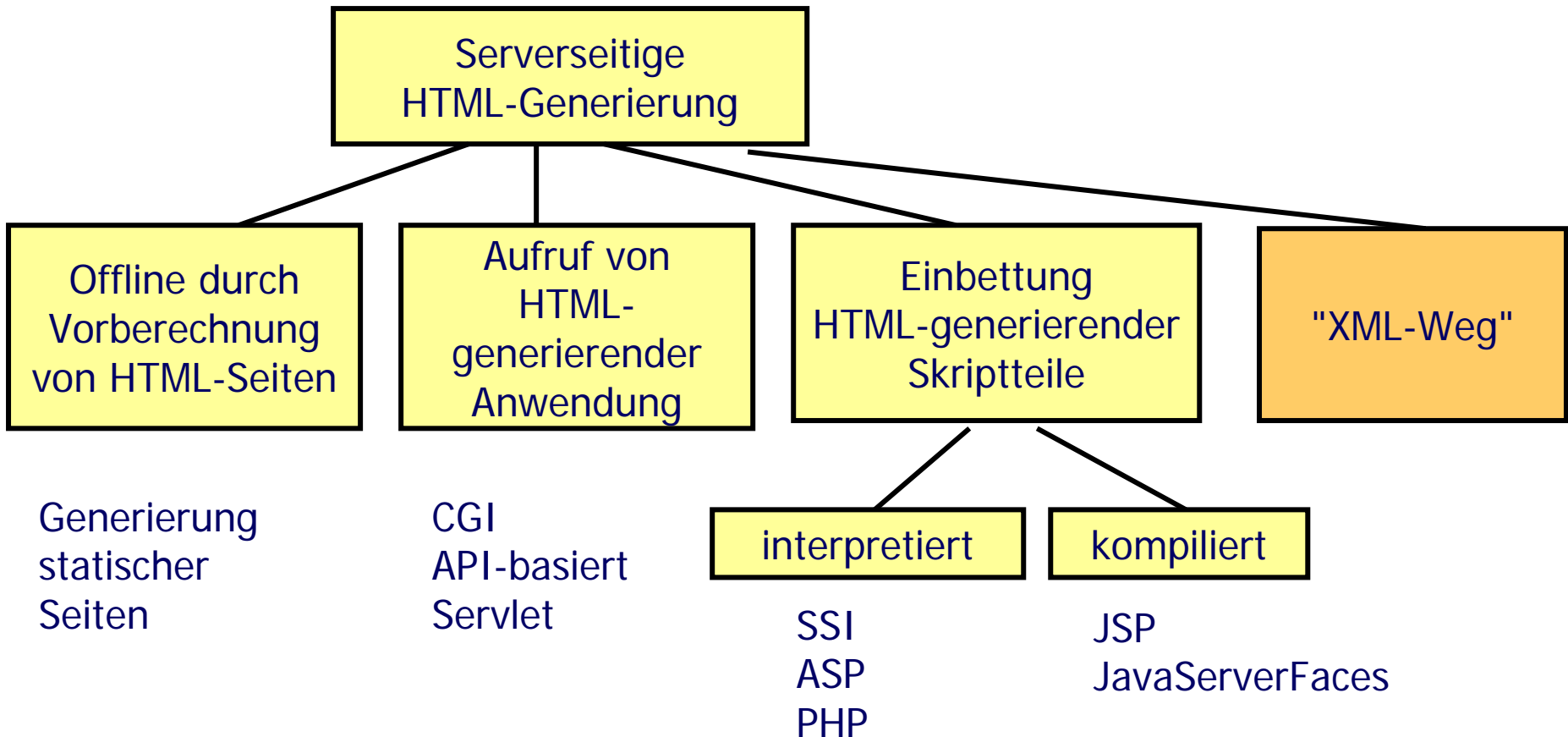
Information Process Engineering

Verarbeitung von Daten mit XML

Heiko Paoli

WS 2006/2007

Wiederholung: Server-seitige Ansätze



Wiederholung: XML-Weg

■ Zwischen den einzelnen Schichten: Brüchen in den Datenmodellen

- ▶ Datenbank: relational
- ▶ Anwendungslogik: objektorientiert
- ▶ Präsentation: semistrukturiert (XML)

■ XML tritt an, dies zu homogenisieren

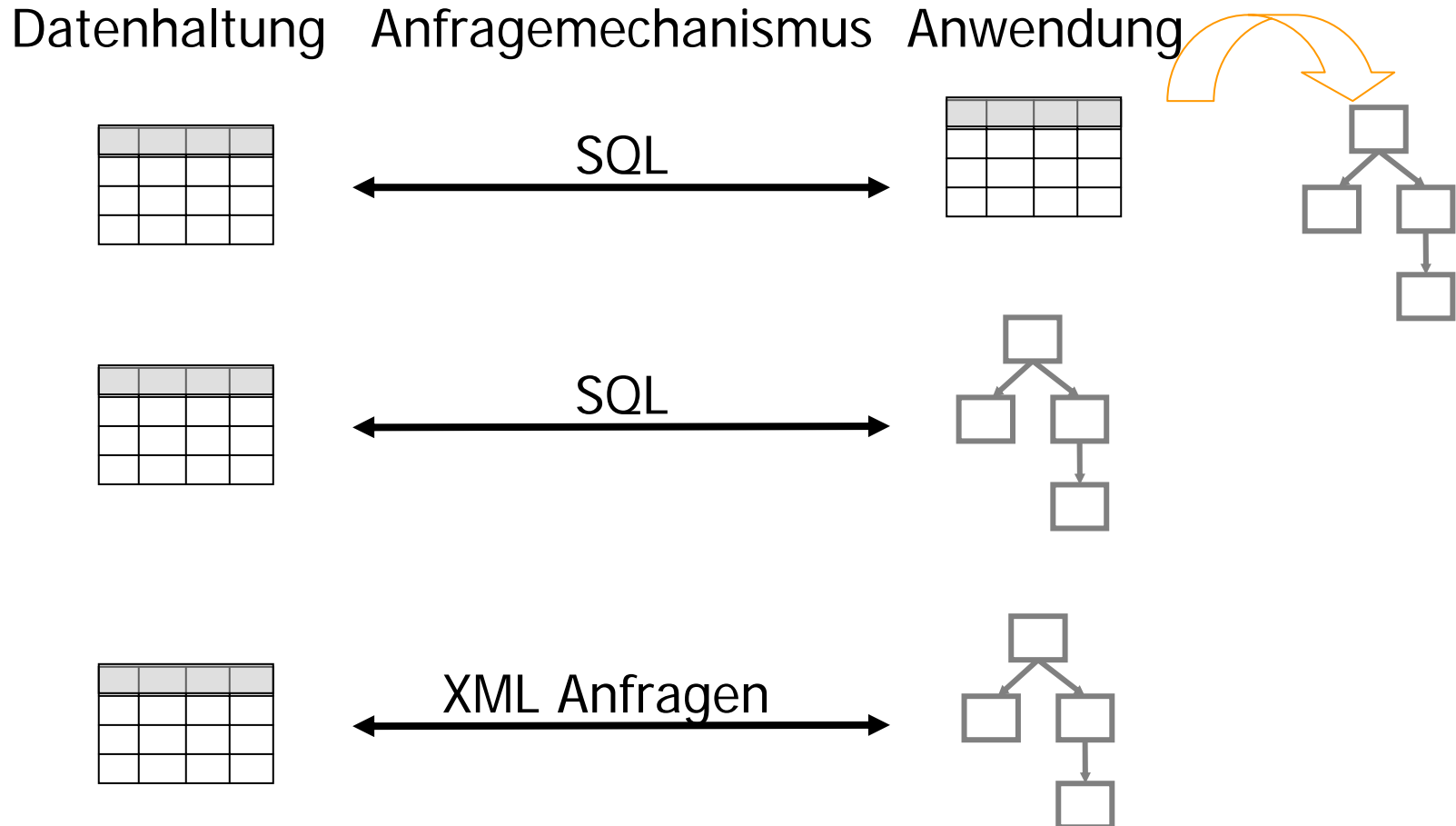
- ▶ XML als Datenmodell zur Speicherung
- ▶ Anwendungslogik als Transformation von XML Daten
- ▶ Präsentation als Transformation

- Datenbanken und XML
 - ▶ Grundgedanken
 - ▶ Abbildungsaspekte
 - ▶ FOR XML
 - ▶ XQuery
- XML-Anwendungsschnittstellen
 - ▶ DOM
 - ▶ SAX
- XML-Verarbeitung
 - ▶ XPATH
 - ▶ XSLT
 - ▶ XSL-FO

Datenbanken und XML

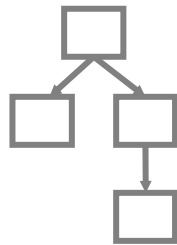


Grundgedanken (1)

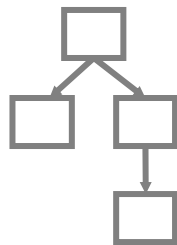
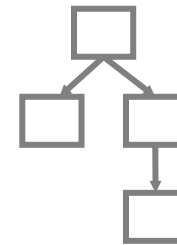


Grundgedanken (2)

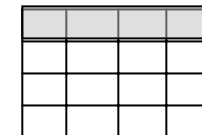
Datenhaltung Anfragemechanismus Anwendung



XML Anfragen

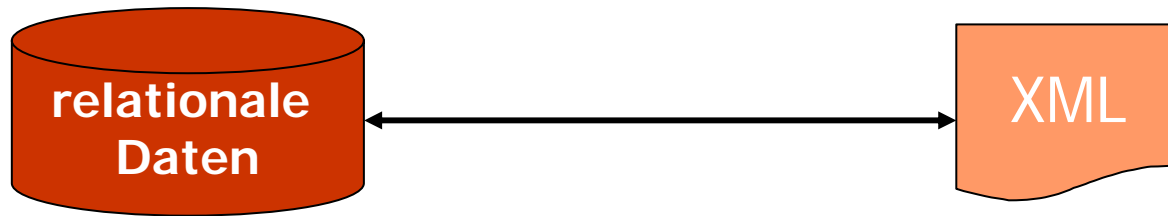


SQL



Grundgedanken (3)

- Unterschiedliche Anforderungen
 - ▶ Lesender und schreibender Zugriff
 - ▶ Wiederherstellbarkeit der Daten
- Stärke der Kopplung
 - ▶ Datenbank nur als Speicher vs. Verlagerung von Funktionalität in die Datenbank
 - ▶ Wie viel XML im Verarbeitungsmodell?
 - Vergleichbar mit OO-Datenbanken vs. Relationale Datenbanken
- Dokumente vs. Daten
 - ▶ Buchtexte werden anders verarbeitet als Adresslisten
 - ▶ Grad der Strukturierung



Abbildungsaspekte

Mögliche Probleme bei der Abbildung von XML auf relationale Datenbanken

Abbildungsaspekte (1)

- Schema muss nicht vorab bekannt sein
 - ▶ Optimierungsprobleme bei Speicherung
 - ▶ In realen Systemen wird diese Flexibilität oft fallengelassen (Schema muss also bekannt sein)
- Verschachtelung von Elementen
 - ▶ Verschachtelung beliebiger Tiefe
 - ▶ Zerlegung in Relationen ("shredding") und Verknüpfung über Fremdschlüssel
- Wiederholung von Elementen
 - ▶ Beliebige Wiederholung von Elementen möglich
 - ▶ Spalten in Relationen einmalig

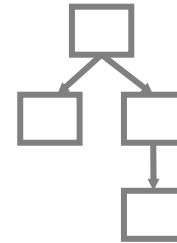
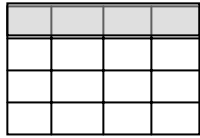
Abbildungsaspekte (2)

- Alternativen erlaubt
 - ▶ Elemente eines Elementtyps können unterschiedliche Kinderelemente enthalten
- Optionale Elemente
 - ▶ Kann auf NULL Werte abgebildet werden (ist aber semantisch nicht ganz das gleiche)
- Mixed Content
 - ▶ Abwechselnd Zeichen und Kinderelemente
 - ▶ Widerspruch zur ersten Normalform im relationalen Modell
- Attributwerte

Abbildungsaspekte (3)

- Elementreihenfolge
 - ▶ Elementare Eigenschaft von XML
 - ▶ Wird bei Validierung berücksichtigt
- Datentypen
 - ▶ In DTD nicht vorhanden (nur in XML Schema)
- Nullwerte
 - ▶ Abbildung von nicht vorhandenen Elementen auf NULL-Werte
 - ▶ Unterschied zwischen "" und null
- Binärdaten
- Processing Instructions

Datenhaltung Anfragemechanismus Anwendung

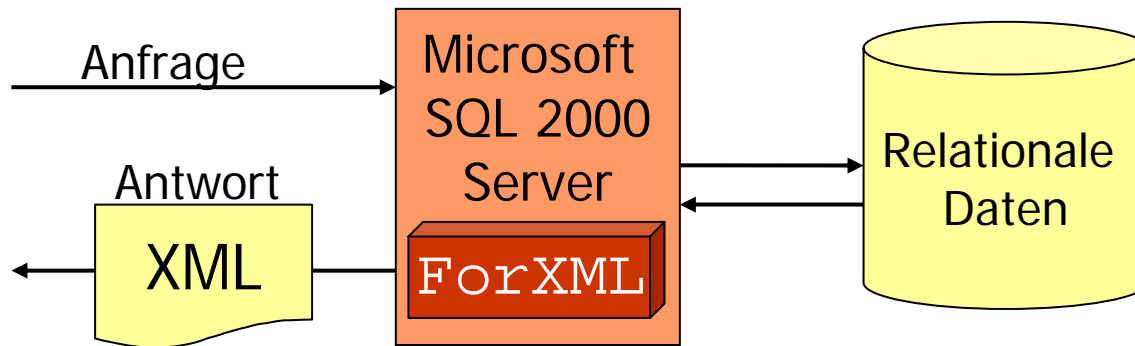


Beispiel: FOR XML



Relationale Daten nach XML

- Verwendung der SQL-Erweiterung For XML



- Transformierungsmodi von FOR XML:
 - RAW
 - AUTO
 - EXPLICIT

Beispielanfrage

■ SQL-Anfrage:

```
SELECT      Artikel.katalogelementId, Artikel.name
            Bestellung.beschreibung
FROM        Artikel
INNER JOIN  Bestellung
ON          Artikel.katalogelementId =
            Bestellung.katalogelementId
WHERE       Bestellung.katalogelementId = "47"
```

- ▶ Durch Anwendung von `FOR XML` und den Transformierungsmodi erfolgt Generierung der Ergebnisrelation zu XML-Dokument

Transformierungsmodi RAW & AUTO

■ RAW

- ▶ Ausgabebetupel als <row> Tag
- ▶ Spaltenname als Attributname
- ▶ Spaltenwert als Attributwert
 - `<row katalogelementId = "47" name="Sack Zement" beschreibung="Achtung! schnell bindend"/>`

■ AUTO

- ▶ Ergebnisrelation als verschachtelte XML
- ▶ Tabellename aus FROM-Anweisung als Element ausgegeben
- ▶ Gemäß in SELECT-Anweisung ausgewählte Ergebnisspalten werden Elementattributen zugeordnet

Elementverschachtelung

- ▶ Attributnamen gemäß Reihenfolge der Spaltendeklaration in SELECT-Anweisung:

```
<artikel katalogelementId = "47" name = "Sack Zement" />
```

Tabellenname

Spaltennamen

- ▶ Durch Einsatz von ELEMENTS-Direktive hinter FOR XML, Spalten als Kindelemente:

```
<artikel>
```

```
<katalogelementId>47</katalogelementId>
```

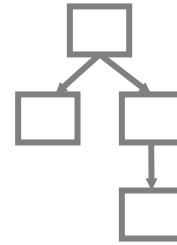
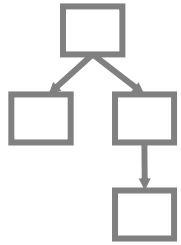
```
</artikel>
```

Spaltennamen

Datenhaltung

Anfragemechanismus

Anwendung



Beispiel: XQuery



XQuery - Überblick (1)

- Ende 1998
 - ▶ begann der Prozeß zur Entwicklung einer W3C-Empfehlung für eine XML-Anfragesprache
- Februar 2001
 - ▶ erster Entwurf auf der Basis von »Quilt«
- April 2001
 - ▶ erste XQuery-Prototypen
 - ▶ Syntaxebene: Microsoft
 - ▶ Algebraebene: Lucent/GMD-IPSI
- Aktuell
 - ▶ W3C Proposed Recommendation (21.11.2006)
 - ▶ Synchronisation mit anderen Standards (XPath, XSL-T)

XQuery - Grundprinzip

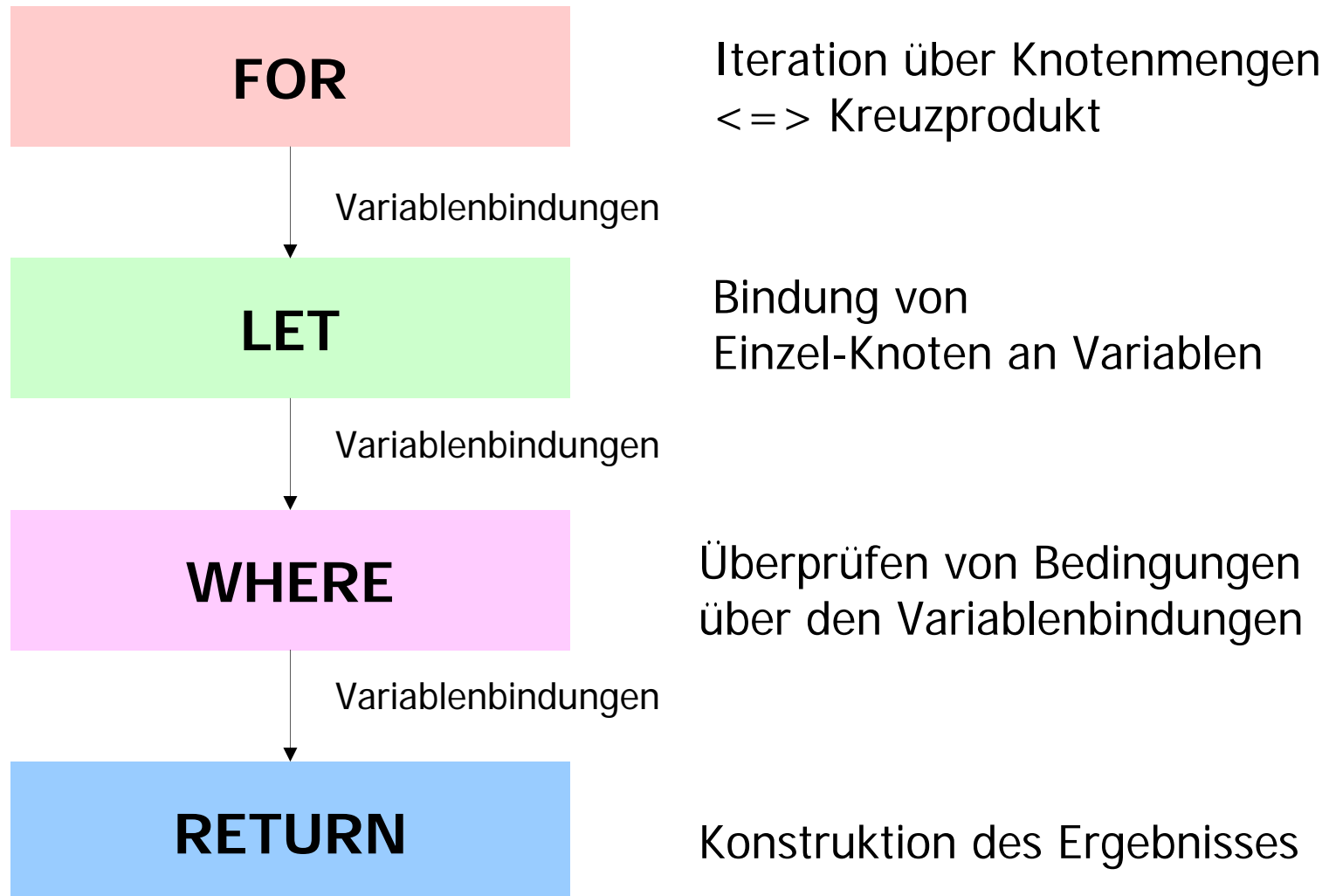
■ Selektionsaspekt

- ▶ Formulierung von Anfragebedingungen
- ▶ meist mittels (Baum-)Mustern
- ▶ Ergebnis sind Variablenbindungen

■ Transformationsaspekt

- ▶ Wie soll das Ergebnis aussehen?
- ▶ Erzeugung einer XML-Struktur aus diesen Variablenbindungen

XQuery: Struktur



Beispiel für Anfragebearbeitung

<kunden>

FOR \$k in //kunde

LET \$b = count(\$k//bestellung
[bestelldatum>2001-01-01])

WHERE \$b > 0

RETURN

<kunde>

\$k/name,

\$k/vorname,

<anzahl>\$b</anzahl>

</kunde>

</kunden>

\$k	\$b
Hans Mayer	0
Stefanie Bauer	1
Andreas Schmidt	5
Jochen Adam	0

\$k	\$b
Stefanie Bauer	1
Andreas Schmidt	5

```
<kunden>
  <kunde>
    <name>Bauer</name>
    <vorname>Stefanie</vorname>
    <anzahl>1</anzahl>
  </kunde>
  <kunde>
    <name>Schmidt</name> ...
```

XQuery – Beispiele Joins

<produkte>

```
FOR $b IN document("www.handwerkermarkt.de/angebote.xml")//artikel,  
      $a IN document("www.verbraucherportal.de/produkte.xml")//produkt  
WHERE $b/name = $a/bezeichnung  
RETURN
```

<produkt>

\$b/name,

<preis> \$a/preis/text() </preis> ,

<hersteller> \$a//herstellername/text() </hersteller> ,

<bewertung> \$b/rating/text() </bewertung>

</produkt>

SORTBY (\$b/name)

</produkte>

XQuery – Beispiel Quantoren

<kunden>

FOR \$k in //kunde

WHERE SOME \$p **IN** \$k//bestellung//produkt **SATISFIES**
\$p/kategorie = "Bohrmaschine"

RETURN

<kunde>

\$k/name,

\$k/vorname

</kunde>

</kunden>

- analog auch mit **EVERY ... IN ... SATISFIES** für Allquantor

XQuery – Beispiele Filter

<toc>

```
LET $b := document("bauvorschriften.xml")
```

```
RETURN filter($b,
```

```
    $b//abschnitt | $b//paragraph | $b//paragraph/überschrift)
```

</toc>

- Es bleiben nur die Elemente *abschnitt*, *paragraph* und *überschrift* übrig; der Baum wird »komprimiert«

XQuery: Weitere Möglichkeiten

- Funktionen können definiert werden
- XQuery-Anfragen sind beliebig schachtelbar
- Mengenoperatoren (UNION, INTERSECT, EXCEPT)
- IF ... THEN ... ELSE - Konstrukt

XQuery – Bewertung

- Mächtige und konzeptionell einfach gehaltene XML-Anfragesprache
 - ▶ gute Integration mit anderen XML-Standards, wie z.B. XPath
 - ▶ Schemaunterstützung (u.a. *instanceof*)
- aber
 - ▶ Kritisiert wird die Überlappung der Transformationsfunktionalität mit XSL-T
 - ▶ Noch keine Möglichkeit Werte mit XQuery zu ändern (XQuery Update Facility Requirements)

Anwendungsschnittstellen



Anwendungsschnittstellen - Ansätze

- Grundsätzlich zwei Ansätze:
 1. Das ganze Dokument wird eingelesen und in ein Objektmodell umgewandelt
 - ▶ DOM (Document Object Model)
 2. Das Dokument wird Schritt für Schritt durchwandert, und bei jedem Schnitt wird ein Ereignis (*event*) generiert, das eine Aktion zur Folge haben kann
 - ▶ SAX (Simple API for XML).
- DOM wesentlich mächtiger als SAX, aber SAX kann im Einzelfall sehr viel performanter sein.

Anwendungsschnittstellen - DOM

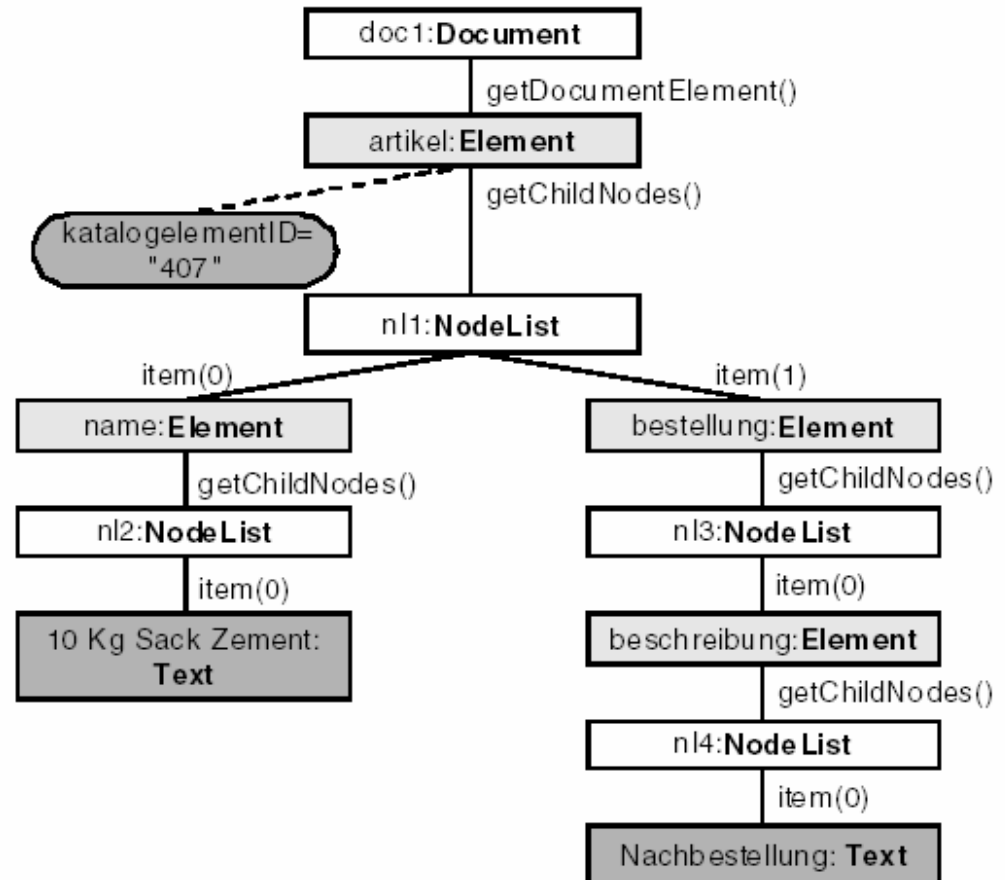
- DOM ist Empfehlung des W3C
- Ziel: Plattform- und Programmiersprachen-unabhängige Schnittstelle, um dynamisch auf XML-Dokumente zuzugreifen
 - ▶ Inhalt
 - ▶ Struktur
 - ▶ Darstellungseigenschaften
- Besteht aus verschiedenen Stufen, die aufeinander aufbauen
 - ▶ Derzeit höchste Stufe ist DOM Level 3 (seit April 2004)

DOM - Eigenschaften

- Plattform- und Programmiersprachenunabhängig
- Grundlegendes Objektmodell für HTML, CSS und XML-Dokumente
- Zusammensetzen und Zerlegen von Dokumenten
- Kein Ausschluss von externen Programmen oder eingebetteten Skripten, die auf Dokumentinhalte zugreifen
- Konsistente Namensgebung durch alle Stufen
- Visuelle Benutzerschnittstelle für Implementierung nicht erforderlich
- Spezielle Objektmodelle für HTML, CSS und XML orientieren sich an den zugrundeliegenden Konstrukten dieser Sprachen.
- Dokumente können eingelesen, bearbeitet und verändert ausgegeben werden, wobei ursprüngliche Struktur wieder herstellbar sein sollte
- Keine Probleme hinsichtlich Sicherheit, Gültigkeit von Dokumenten oder der Vertraulichkeit beim Anwender
- Keine Mechanismen ausschließen, um auf Dokumente zuzugreifen

DOM Dokumentstruktur - Beispiel

```
<artikel
  katalogelementID="407">
  <name>10 Kg Sack
  Zement</name>
  <bestellung>
    <beschreibung>
      Nachbestellung
    </beschreibung>
  </bestellung>
</artikel>
```



Anwendungsschnittstellen - SAX

- Die Simple API for XML (SAX) ist ein de-facto Standard für ein ereignisbasiertes Parsen von XML-Dokumenten
- SAX definiert eine Schnittstelle, die es erlaubt, die Informationen eines XML-Dokuments in linearer Reihenfolge zu durchlaufen **OHNE** eine Baumstruktur aufzubauen

Keine offizielle Spezifikation vom W3C vorhanden!

SAX - Eigenschaften



- SAX wurde von den Mitgliedern der XML-DEV Mailing List entwickelt
- Verfügt über eine breite Akzeptanz
- Wurde am 11. Mai 1998 in der Version 1.0 veröffentlicht
- Seit 28. Dezember 2000 gibt es die Version 2.0 (SAX2)
- Es existieren Implementierungen für die meisten gängigen Programmier- und Skriptsprachen, wie Java, Perl, Python und C++.

SAX - Ereignismodell (1)

- Bei SAX gibt es zwei wichtige Programmkomponenten:
 - ▶ Einen Treiber (XML Parser) der Ereignisse erzeugt
 - ▶ Einen Handler, der diese Ereignisse verarbeitet
- Beispiel:

```
<?xml version="1.0">  
<rowset>  
  <row>SAX Test</row>  
</rowset>  
=> Ereignisse:
```

1. StartDocument()
2. StartElement("rowset")
3. StartElement("row")
4. Characters("SAX Test")
5. EndElement("row")
6. EndElement("rowset")
7. EndDocument()

SAX - Ereignismodell (2)

- Es gibt in SAX grob folgende Ereignisklassen:
 - ▶ Ereignisse durch den Inhalt eines XML-Dokuments
 - ▶ Ereignisse durch Validierung
 - ▶ Ereignisse durch Fehler
- Reaktion auf Ereignisse durch *Handler* (Namen sind programmiersprachenabhängig)
- Wird zu einem Ereignis kein passender Handler gefunden, so bleibt dieses Ereignis unbehandelt.
- Zu den meisten Ereignissen werden vom Parser noch Parameter übergeben, zum Beispiel bei *StartElement* der Name des Elements oder bei *Error* der Fehlercode.

Vergleich DOM ↔ SAX

■ DOM

- ▶ Erste Wahl sollte auf DOM fallen
 - Mächtige Funktionen für das Einlesen von XML-Dokumenten
 - Wohldefinierte Schnittstellen auf das Objektmodell
- ▶ Komplexe Suchanfragen realisierbar
- ▶ Wahlfreier Zugriff auf XML-Dokumente
- ▶ DOM ist auch auf das Schreiben von XML-Dokumenten ausgelegt

■ SAX

- ▶ Lineares Parsen des XML-Dokuments möglich
 - Konstanter Speicherverbrauch auch bei großen XML-Dokumenten
- ▶ Dokument muss unter Umständen nicht komplett geparkt werden.
- ▶ SAX, wenn es sich um ein stark datenorientiertes XML-Dokument handelt, das eine flache Hierarchie hat.
- ▶ SAX kann ein Objektmodell erzeugen, mit dem man dann wie unter DOM arbeiten kann.

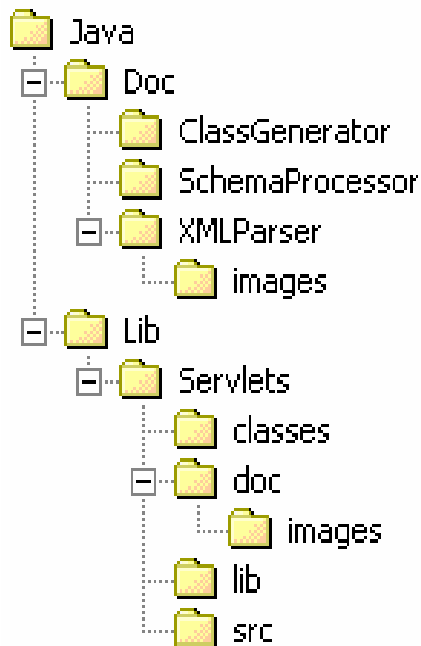
XML Path Language (XPath)

Adressierung in XML

Adressierung von XML

- Sprache, um Teile eines XML-Dokuments zu adressieren
 - ▶ Wird z.B. in XSLT, XPointer und XQuery verwendet.
- Kann
 - ▶ Dokumentknoten unter Angabe verschiedener Kriterien selektieren
 - ▶ Grundlegende Manipulationen an Zeichenketten, booleschen Werten und Knotenmengen durchführen.
- Unterstützt Mustererkennung auf Dokumentknoten (Testen gegen ein vorgegebenes Muster)
- Enthält eine einfache Funktionsbibliothek
 - ▶ Kann durch benutzerdefinierte Funktionen erweitert werden
- Kompakte, nicht an XML orientierte Syntax
 - ▶ ähnlich zu Pfadangaben bei Betriebssystemen
- XPath 1.0 freigegeben als Standard (TR) am 16.11.1999
- XPath 2.0 PR seit 21.11.2006

Konzept von XPath



```
<Java>
  <Doc>
    <ClassGenerator/>
    <SchemaProcessor/>
    <XMLParser>
      <images/>
    </XMLParser>
  </Doc>
  <Lib>
    <Servlets>
      <classes/>
      <doc>
        <images/>
      </doc>
      <lib/>
      <src/>
    </Servlets>
  </Lib>
</Java>
```

- XML-Dokumente können analog zu einer Verzeichnisstruktur in einem Dateisystem gesehen werden.
- XPath-Ausdrücke um Knoten in einem XML-Dokument zu selektieren entsprechen dem Wechseln zu einem anderen Verzeichnis in einem Dateisystem.

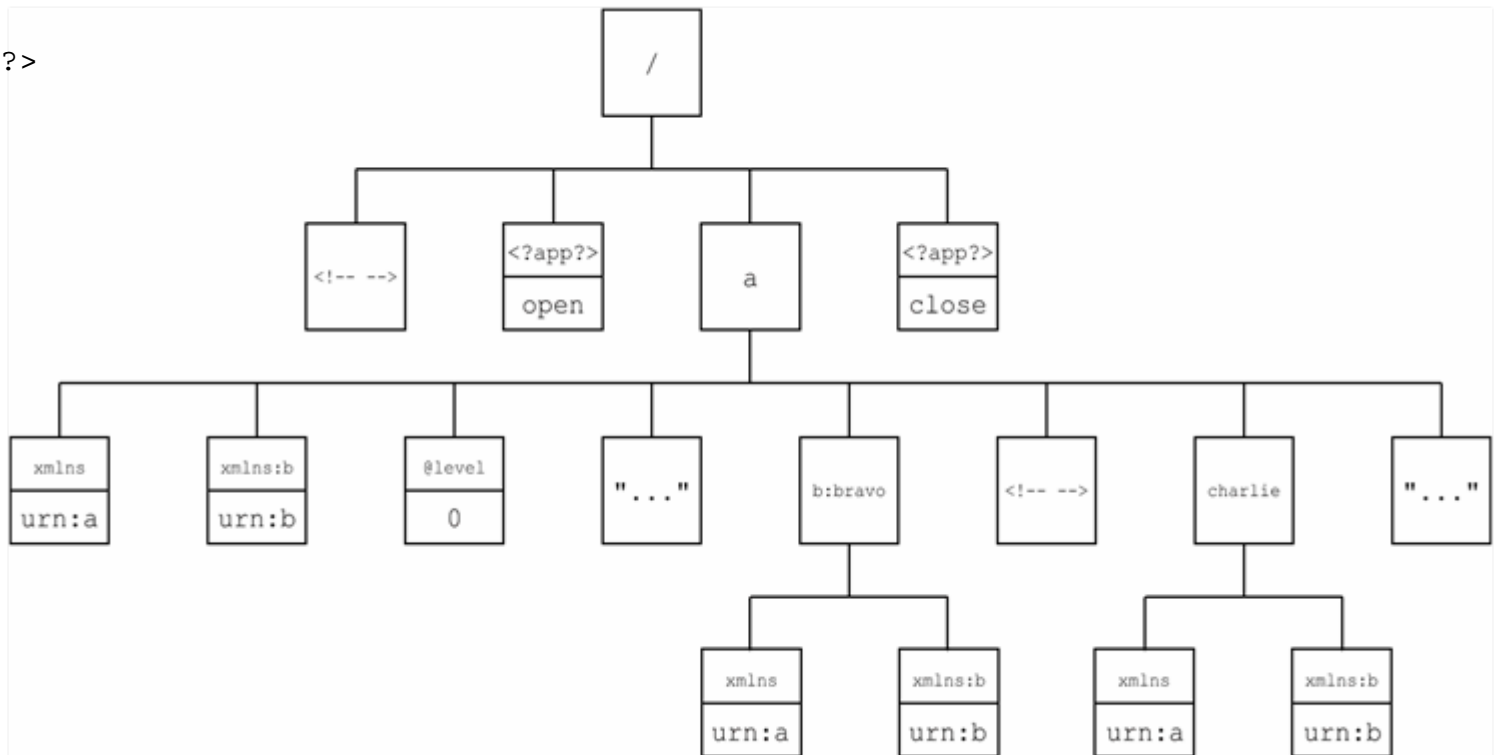
- `/Java/Lib/Servlets/src` wechselt in das `src` Verzeichnis oder selektiert das `src` Element
- `../../../../Doc/XMLParser/images` wechselt vom Verzeichnis `/Java/Lib/Servlets/doc/images` zum Verzeichnis `/Java/Doc/XMLParser/images` oder selektiert das Nachfolgerelement `images` des Elementes `Doc` vom Nachfolgerelement `images` des Elements `Lib`

Auswertungskontext

- Ausdrücke werden in einem bestimmten Kontext ausgewertet
- Kontexte bestehen aus:
 - ▶ Kontextknoten
 - ▶ Kontextposition und -größe
 - ▶ Einer Menge von gebundenen Variablen (Name-Objekt-Paare)
 - ▶ Einer Menge von definierten Funktionen (Name-Funktions-Paare), einschließlich benutzerdefinierter Funktionen
 - ▶ Einer Menge von Namensraumdeklarationen
- Kontext kann von einem Verarbeitungsprozessor, innerhalb eines mehrstufigen Ausdruckes oder beim Auswerten des Prädikats geändert werden

Dokumentmodellbeispiel

```
<!-- Start -->
<?app open?>
<a level="0" xmlns:b="urn:b" xmlns="urn:a">
  alpha
  <b:bravo/><!-- To do... --><charlie/>
  delta
</a>
<?app close?>
```



Lokalisierungspfad

- Der Lokalisierungspfad (location path) ist das wichtigste XPath-Konstrukt
- Lokalisierungspfade werden benutzt, um Knotenmengen zu selektieren
- Der Lokalisierungspfad ist eine Folge von Lokalisierungsschritten
- Lokalisierungspfade können absolut oder relativ sein

Der Ausdruck `/html/body/a/@href` ist ein absoluter Lokalisierungspfad mit vier Lokalisierungsschritten:

- `html` – selektiert alle `html` Elemente beginnend vom Wurzelknoten (da der Lokalisierungspfad absolut ist);
- `body` – selektiert `body` Elemente;
- `a` – selektiert `a` Elemente;
- `@href` – selektiert `href` Attribute.

Der Ausdruck liefert `href` Attribute von allen obersten Ankern (`a` Elementen) des Dokumentenkörpers zurück.

XPath: Lokalisierungsschritten

- Der Lokalisierungspfad ist eine Folge von Lokalisierungsschritten.
- Lokalisierungsschritte bestehen aus drei Teilen:
 - ▶ Navigationsachse (Wohin bewegen wir uns in dieser Stufe?)
 - ▶ Knotentests (Nach was suchen wir in dieser Stufe?)
 - ▶ Null oder mehrere Prädikate (Was sind die Eigenschaften von dem, was wir suchen?)

Der Lok.-Schritt: `@href[starts-with(., 'http://')]`

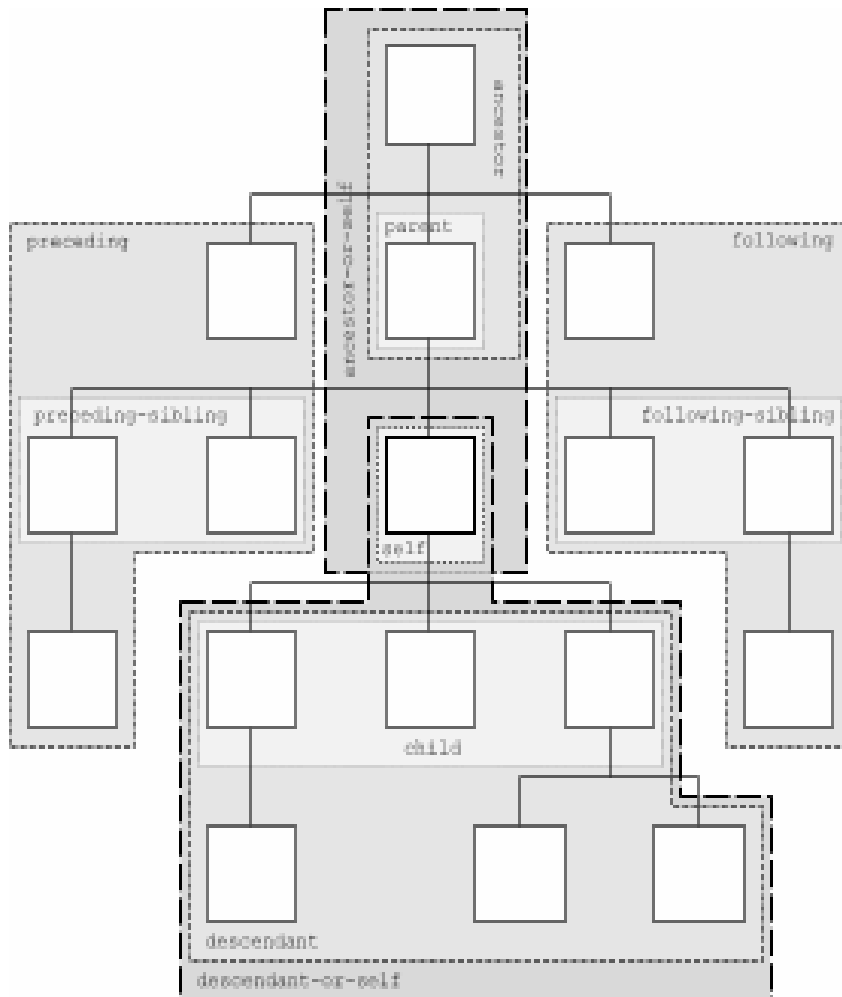
besteht aus:

- Navigationachse `attribute` (oder `@` in abgekürzter Form)
- Knotentest `href`
- Prädikat `starts-with(., 'http://')`

Navigationsachsen

- Im Gegensatz zu Pfaden in einem Dateisystem gibt es in XPath zusätzlich noch Navigationsachsen.
- Alle Navigationsachsen beziehen sich immer auf den aktuellen Kontextknoten.
- Es gibt insgesamt 13 verschiedene Navigationsachsen.
- Schreibweise:
 - ▶ *navigationssachse::knotenabfrage*
- Für die wichtigsten Navigationsachsen existiert eine Kurzschreibweise.

Übersicht: Navigationachsen



Dreizehn Navigationsachsen:

- self (self::node() = .)
- ancestor
- ancestor-or-self
- parent (parent::node() = ..)
- child (kann als Abkürzung weggelassen werden)
- descendant
- descendant-or-self (descendant-or-self::node() = //)
- following
- following-sibling
- preceding
- preceding-sibling
- attribute (abgekürzt mit @)
- namespace

Knotentest und Prädikate

Knotentests können wie folgt sein:

■ Namenstest (`href`)

■ Knotentypstest

- ▶ `node()` – jeder Knotentyp;
- ▶ `text()` – Textknoten;
- ▶ `comment()` – Kommentarknoten;
- ▶ `processing-instruction()` – PI-Knoten;
- ▶ `processing-instruction(name)` – PI mit gegebenen Namen;
- ▶ * - jeder Knoten des Hauptachsentyps
 - `attribute` für Attributachse,
 - `namespace` für Namensraumachse
 - `element` für alle andere Achsen

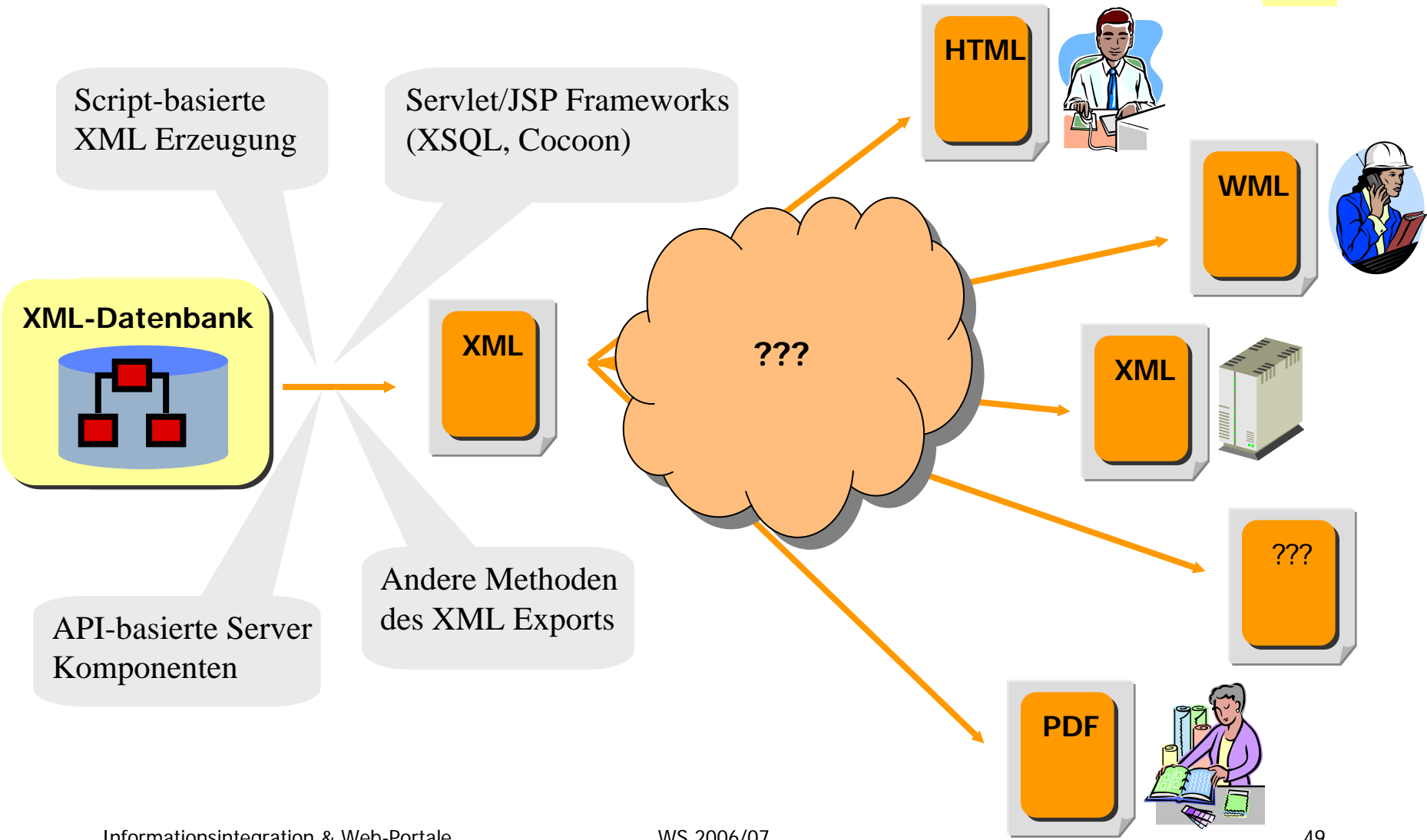
■ Prädikate sind weitere XPath-Ausdrücke in Rechteckklammern

eXtensible Stylesheet Language for Transformations (XSLT)

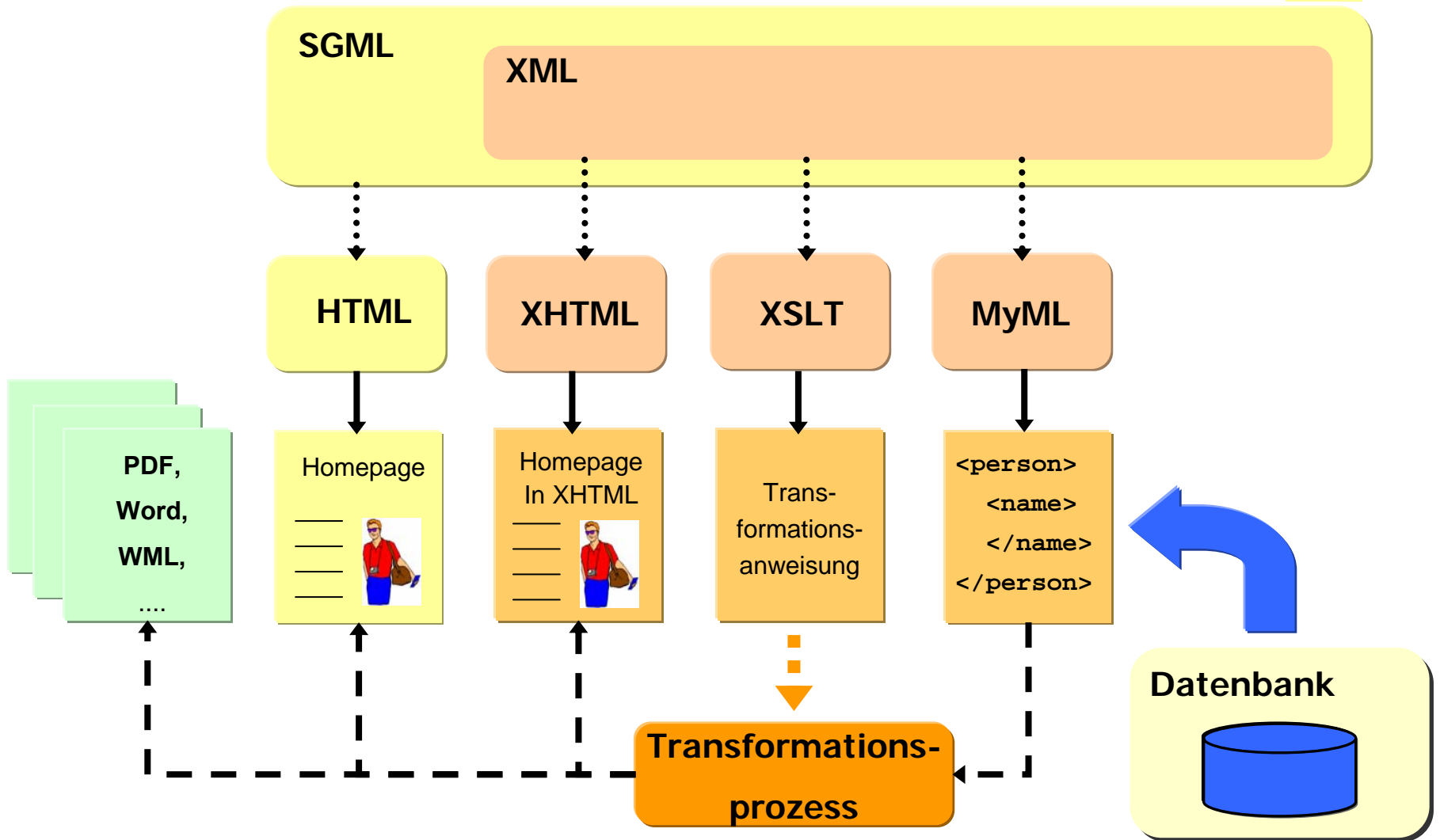


XSLT als Sprache

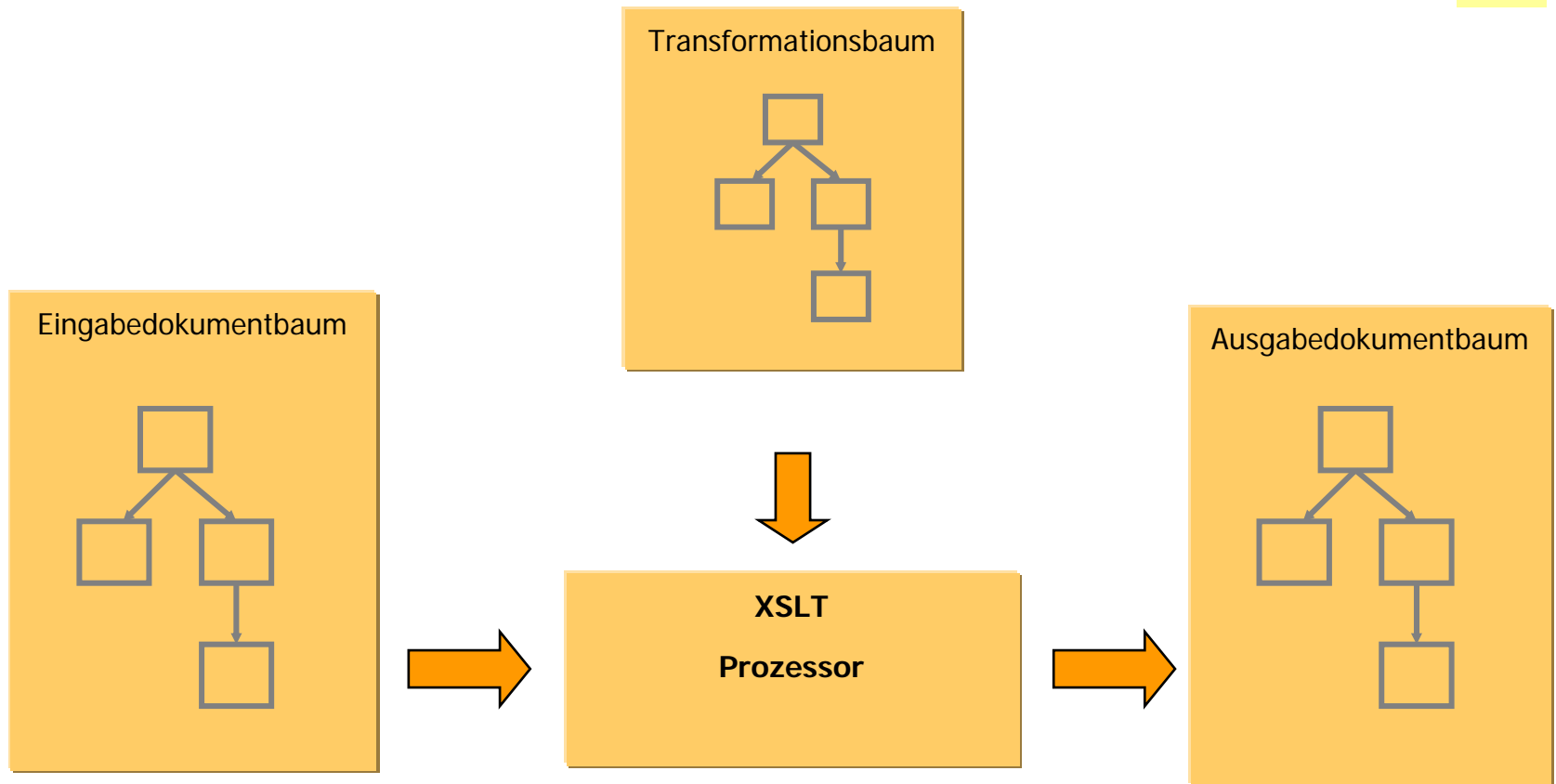
XML aus der DB: Und was nun?



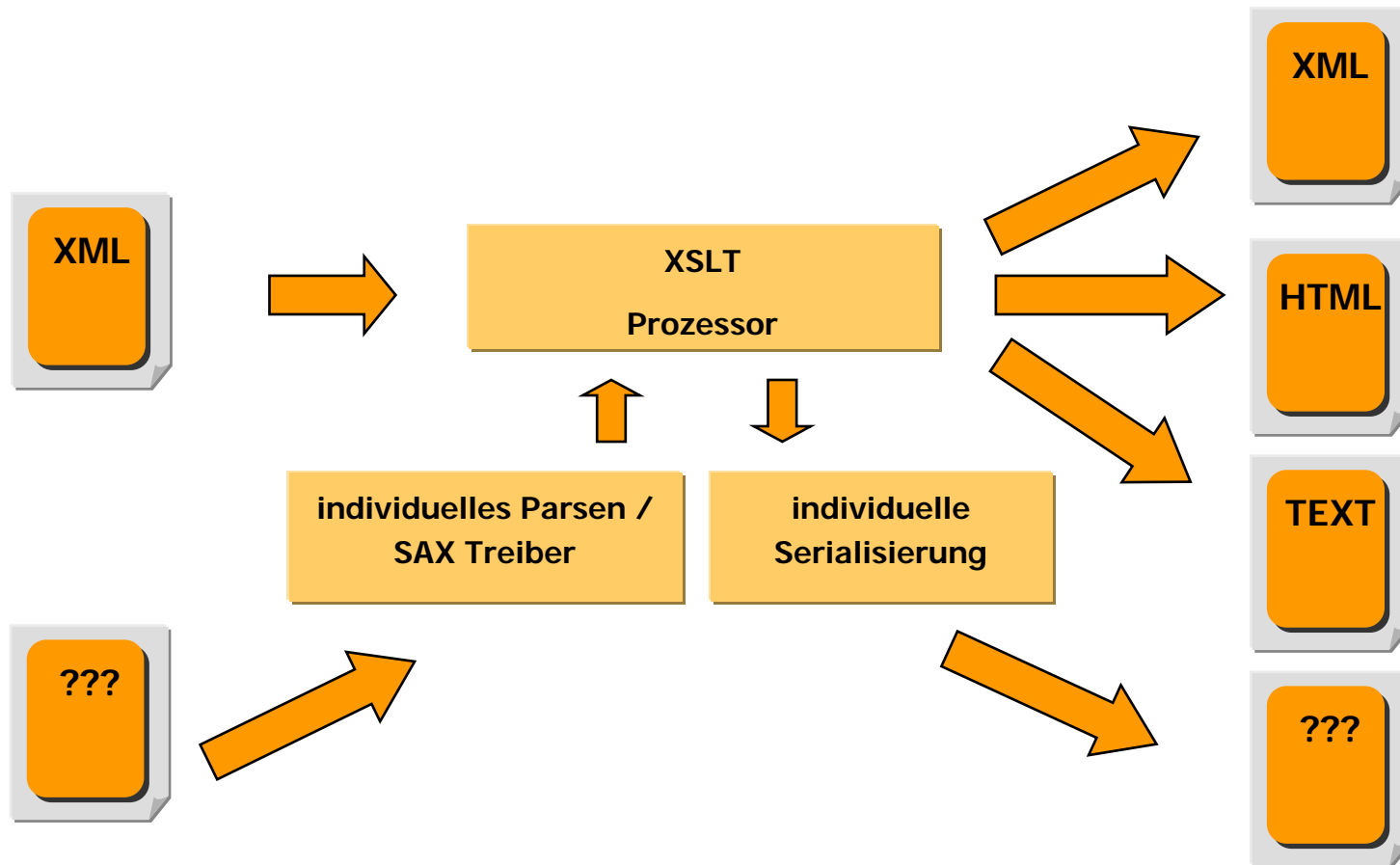
XSLT-Transformation



Konzept: Bäume, Keine Dokumente!

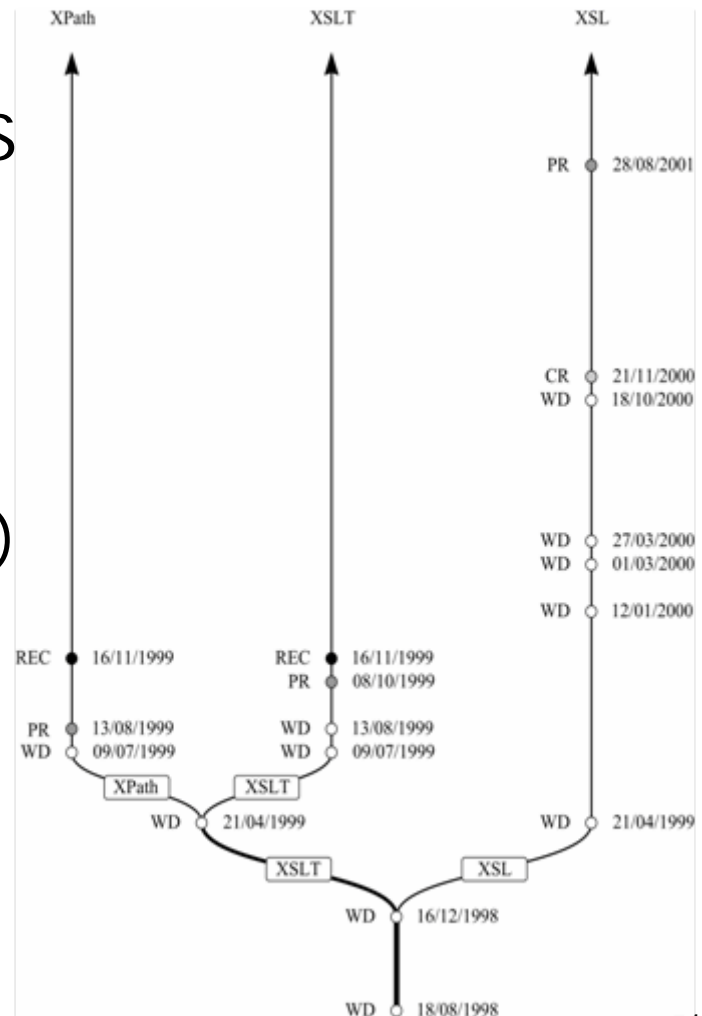


Transformiere die Struktur der Dokumente



- Deklarativ, keine prozedurale Sprache
- Definiert in XML-Syntax
- Nicht nur “stilgebende” wie CSS, sondern auch restrukturierende Möglichkeiten
- Unabhängig von physikalischen Modellen
 - ▶ Arbeitet nur auf der logischen Darstellung
 - ▶ Syntax-unabhängig

- Stammt von der XSL Initiative
- Nachfahre von DSSSL and CSS
- Drei verbundene aber unterschiedliche Aufgaben: Adressierung (XPath), Transformation (XSLT) and Darstellung (XSL oder XSL-FO)
- XSLT 1.0 freigegeben als Standard (TR) am 16.11.1999
- XSLT 2.0 PR seit 21.11.2006



- Gut entwickelte und gut unterstützte Sprache
- Oft Hilfstechnologie in umfangreicheren Projekten
- Viele industriellen Implementierungen
 - ▶ Microsoft (MSXML)
 - ▶ Oracle (Oracle XDK)
 - ▶ Sun
 - ▶ Frei verfügbaren Initiativen: Xalan, Saxon, ...
- Bibliotheken für Java, C/C++, COM, Perl, PHP, Python, PL/SQL
- Wird in XML-basierten „publishing frameworks“ (Cocoon, Charlie) benutzt.

Spracheigenschaften

- Transformationen in XSLT
 - ▶ Menge von Regelschablonen
 - ▶ Definieren, wie bestimmte Knoten verarbeitet werden
 - ▶ Können auch rekursiv aufgerufen werden
- XSLT-Elemente konstruieren das Ausgabedokument
- XPath wird für Knotenselektion und Berechnung verwendet
- Seiteneffektfrei
 - ▶ Sehr nah zu funktionalen Programmiersprachen

Beispiel: „Hello, World!“

Eingabe

```
<msg>Hello, world!</msg>
```

Ausgabe

```
<message>Hello, world!</message>
```

Transformation:

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

  <xsl:template match="msg">
    <message>
      <xsl:value-of select="."/>
    </message>
  </xsl:template>

</xsl:stylesheet>
```

Schablonenbasierte Sprache

- Transformation ist eine Menge von Regelschablonen (Templates)
- Eine Regel wird durch `xsl:template` definiert
 - ▶ Attribut `match` definiert ein Muster in XPath
 - Muster bestimmt, welche Knoten von der Regel bearbeitet werden
 - Anweisungen im Rumpf einer Regel werden ausgeführt werden, falls Muster
 - ▶ Attribut `name` erlaubt expliziten Aufruf der Regel durch Angabe eines Namens
- Ein Regel kann andere Regeln aufrufen
 - ▶ `xsl:apply-templates`
 - ▶ `xsl:call-template`

Transformationskontext

- Transformationskontext beeinflusst Auswertung eines Ausdrucks
- Transformationskontext besteht aus
 - ▶ Aktuellem Knoten
 - ▶ Aktueller Knotenliste
- Transformationskontext wird z.B. geändert durch
 - ▶ `xsl:apply-template`
 - ▶ `xsl:for-each`
 - ▶ Dabei bestimmt jeweils das Attribut `select` die neue, aktuelle Knotenliste
- Jeder Knoten der aktuellen Knotenliste wird zuerst zum aktuellen Knoten und dann verarbeitet
- Kontext wird z.B. nicht geändert durch `xsl:call-template`

Noch einmal „Hello World!“

Stylesheet

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

  <xsl:template match="/">
    <html>
      <head>
        <title>Message</title>
      </head>
      <body>
        <xsl:apply-templates select="msg"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="msg">
    <b>
      <xsl:value-of select="."/>
    </b>
  </xsl:template>

</xsl:stylesheet>
```

Eingabe

```
<msg>Hello, world!</msg>
```

Ausgabe

```
<html>
  <head>
    <title>Message</title>
  </head>
  <body>
    <b>Hello, world!</b>
  </body>
</html>
```

Erzeugung des Inhalts

- Literale Ergebniselemente werden direkt in die Ausgabe übernommen
- Mit `xsl:value-of` wird ein
 - ▶ XPath-Ausdruck ausgewertet und
 - ▶ Ergebnis als Textknoten in die Ausgabe übernommen
- Mit `xsl:copy-of` wird ein
 - ▶ XPath-Ausdruck ausgewertet und
 - ▶ Ergebnis unverändert in die Ausgabe übernommen
- Mit `{XPath expression}` in Attributen
 - ▶ Werden XPath-Ausdrücke berechnet und
 - ▶ `{...}` durch das Ergebnis ersetzt
- Mit `xsl:element`, `xsl:attribute`, `xsl:text`, `xsl:comment`, `xsl:processing-instruction` können dynamisch entsprechende Knoten für die Ausgabe erzeugt werden

■ Regelaufruf durch

- ▶ `xsl:apply-templates`
- ▶ `xsl:call-template`
- ▶ `xsl:apply-imports`

■ Fallunterscheidung durch

- ▶ `xsl:if` für einzelne Fälle
- ▶ `xsl:choose`, `xsl:when` und `xsl:otherwise` für mehrfache Fälle

■ Schleifen durch `xsl:for-each`

- ▶ Allgemeine Schleifen nicht möglich

Zusätzliche Möglichkeiten

- Behandlung von „Whitespaces“ durch
 - ▶ `xsl:preserve-space`
 - ▶ `xsl:strip-space`
- Sortieren durch `xsl:sort`
- Nummerierungen durch `xsl:number`
- Effizienteren Zugriff auf Knoten mittels Schlüssel durch `xsl:key`
- Nachrichten an den Benutzer senden durch `xsl:message`
- Aliase für Namensräume durch `xsl:namespace-alias`
- Ausgabeformat von Zahlen durch `xsl:decimal-format`

Variablen und Parameter

- Variablen und Parameter werden definiert durch
 - ▶ `xsl:variable` und
 - ▶ `xsl:param`
- Attribut `name` definiert Variablen-/Parametername
- Attribut `select` definiert Variablen-/Parameterwert
 - ▶ Darf für Variablen nicht mehr geändert werden
 - ▶ Darf für Parameter vom aufrufenden Element gesetzt werden
 - ▶ Werte der Variablen und Parameter sind zur Laufzeit nicht änderbar (seiteneffektfrei, funktionaler Stil).
- Nötig in komplexen, mehrstufigen Berechnungen als temporärer Speicher
 - ▶ Z.B. um den Kontext zwischenspeichern
 - ▶ Z.B. um Baumfragmente zu speichern
- Parameter erlauben parametrisierte, rekursive Aufrufe von Regeln

- XSLT erlaubt benutzerdefinierte Funktionen und Elemente durch „**external**“
- Erweiterungselemente und -funktionen dürfen keine leeren und keine von XSLT verwendete Namensräume haben.
- Normalerweise bezeichnet der Namensraum Art und Ort der Erweiterungsfunktion (z.B. Klassenname)

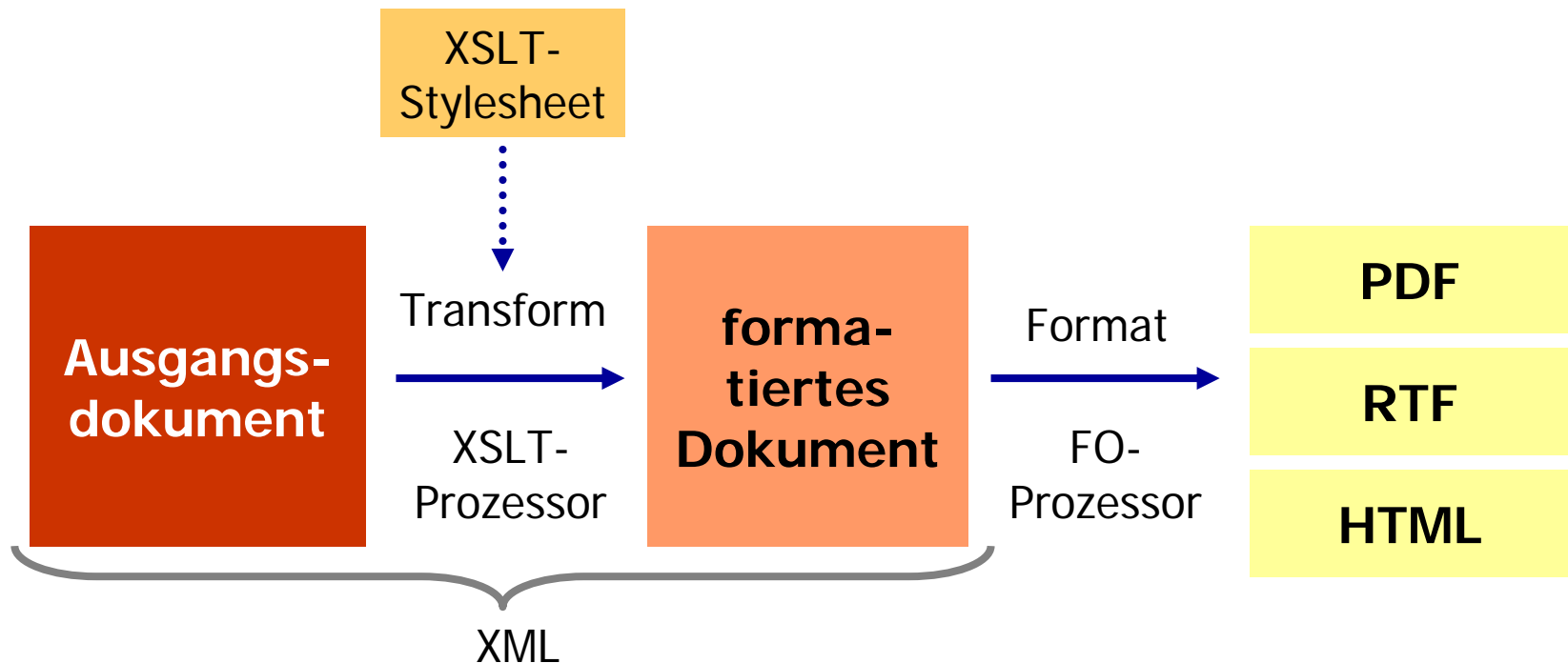
```
<xsl:value-of select="math:round(0.6)"  
  xmlns:math="java:java.lang.Math" />
```
- Java ist die häufigste Sprache für XSLT-Erweiterungen
- Erweiterungen beschränken die Portabilität

- Gibt es Alternativen zu XSLT?
 - ▶ Verarbeitung von XML durch eine Programmiersprache
 - ▶ Java Servlets, Java Server Pages (JSTL), Active Server Pages
 - ▶ PHP, Perl
 - ▶ ...
- Wann sollte dann XSLT verwendet werden und wann nicht?

Stärken und Schwächen

- XSLT ist eine Technologie, die gut für geeignet ist für Strukturtransformationen und einfache Datenkonvertierungen:
 - ▶ XML (Format A) -> XML (Format B)
 - (textbasiertes Format) -> XML
 - ▶ XML -> (textbasiertes Format)
 - ▶ Präsentation von Daten (Publishing)
- XSLT ist weniger gut geeignet für komplexe Werttransformationen und Transaktionen:
 - ▶ Komplexe Stringoperationen (Perl)
 - ▶ Berechnung von Ergebnissen unter Verwendung externer Dienste
 - ▶ Elemente bezeichnen Objekte mit Eigenleben, welche die Transformation beeinflussen können

- Stellt ein standardisiertes Vokabular für den medienunabhängigen Dokumentensatz bereit
 - ▶ Pagination, Textfluß, Positionierung, Schriften, ...



- Wassilios Kazakos, Andreas Schmidt, Peter Tomczyk:
Datenbanken und XML, Springer, April 2002
(www.datenbanken-und-xml.de)
- Michael Kay. XSLT Programmer's Reference – WROX 2004
- W3C
 - ▶ Allgemein: <http://www.w3c.org>
 - ▶ XQuery: <http://www.w3.org/XML/Query>
 - ▶ DOM: <http://www.w3.org/DOM/>
 - ▶ XSL-Allgemein: <http://www.w3c.org/Style/XSL>
 - ▶ XPath-Spezifikation: <http://www.w3c.org/TR/xpath>
 - ▶ XSL-T-Spezifikation: <http://www.w3c.org/TR/xslt>
- SAX: <http://www.saxproject.org/>