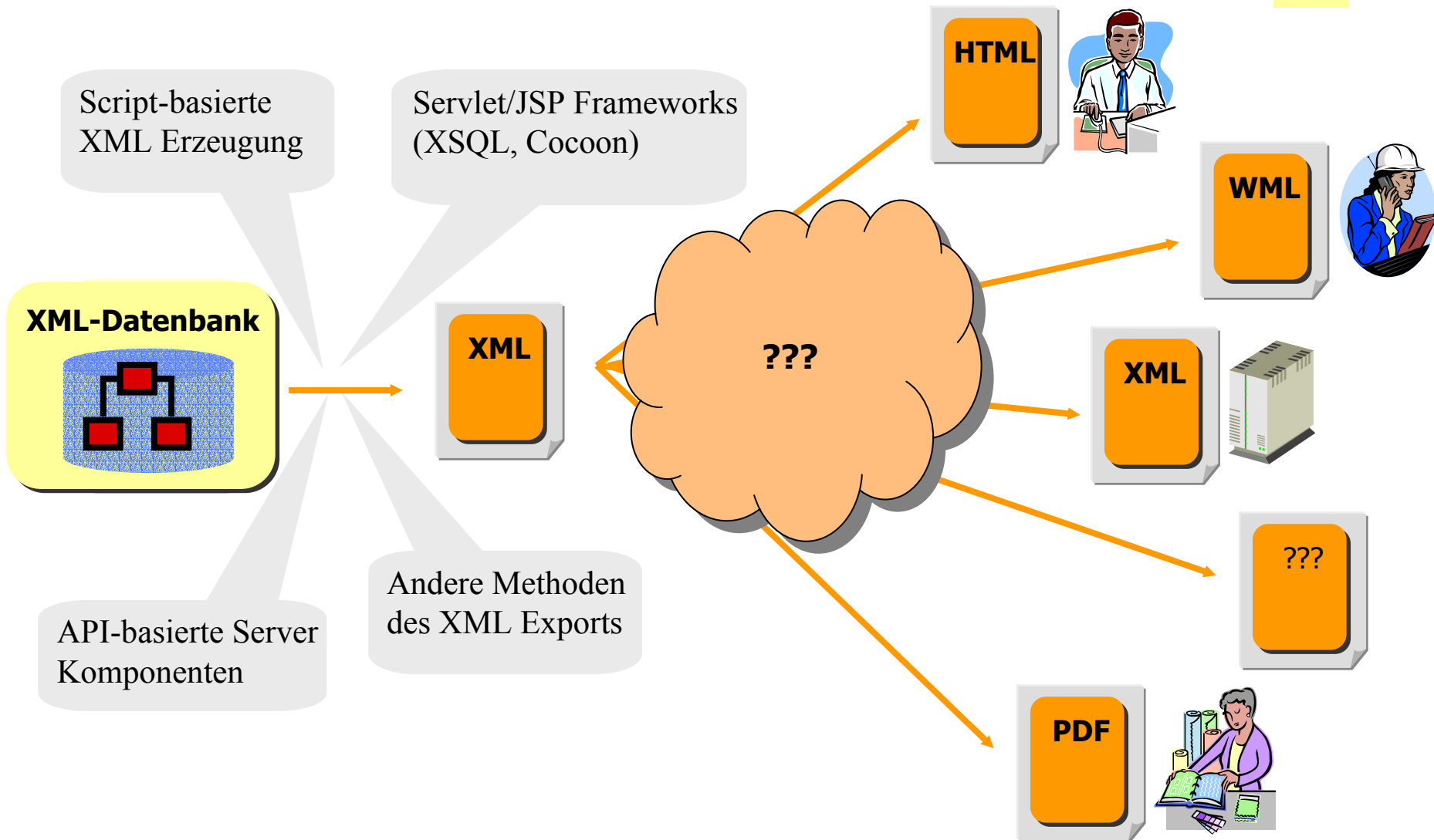


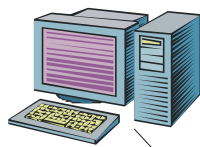
Verarbeitung von XML-Daten

XSLT & XPath

Heiko Paoli (FZI)

XML aus der DB: Und was nun?





Applet



WAP



HTTP

Web-Technologien im Überblick

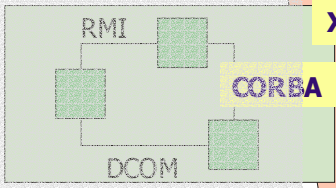
www.klick-and-bau.com

CGI JSP ASP Servlet

XSL-FO

Komponentenframeworks

Verarbeitung von XML-Daten



XSL-T

SQL

Verteilte Transaktionen

Datenaustausch und -zugriff mit XML

Architekturen und Systeme zur Informationsintegration

JDBC

CICS

XML-basierter Datenbankzugriff

XML

XML Schema

XML Schema XQuery

Mediator

Mediator

Semantische Integration

OEM

OEM

XML Schema XQuery

Wrapper

Wrapper

Wrapper

RDBMS

HOST

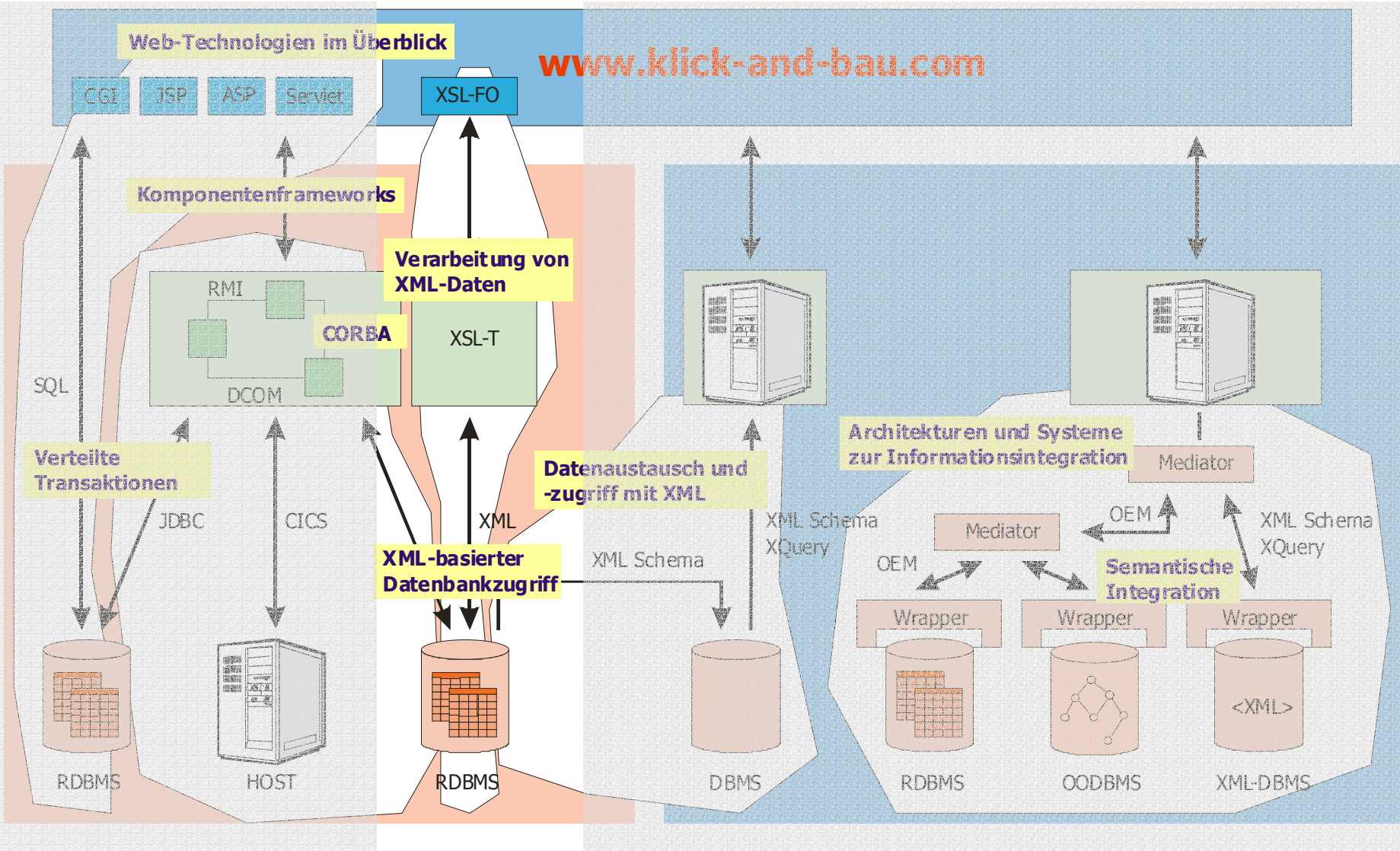
RDBMS

DBMS

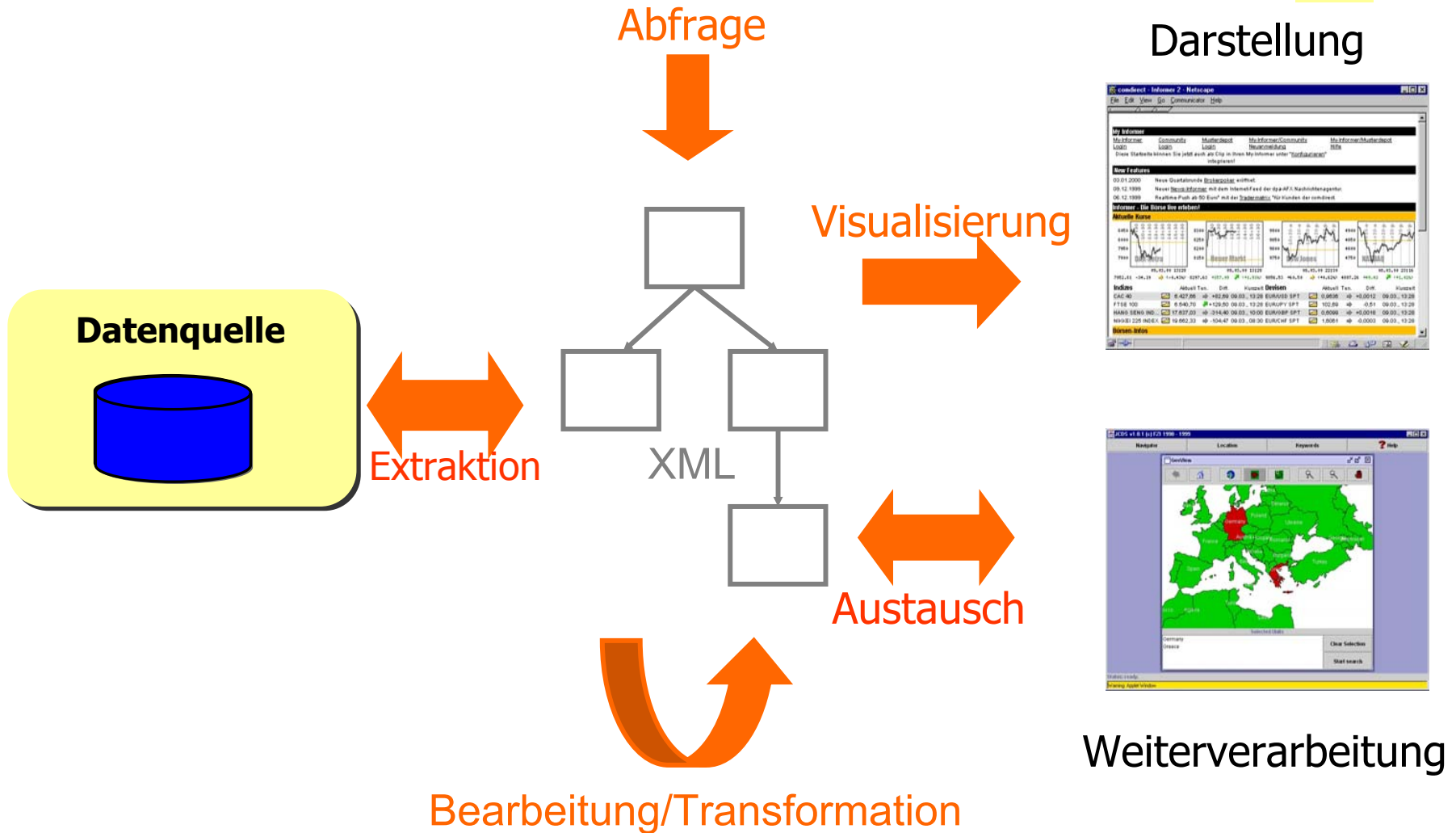
RDBMS

OODBMS

XML-DBMS



XML Verarbeitungsmodell

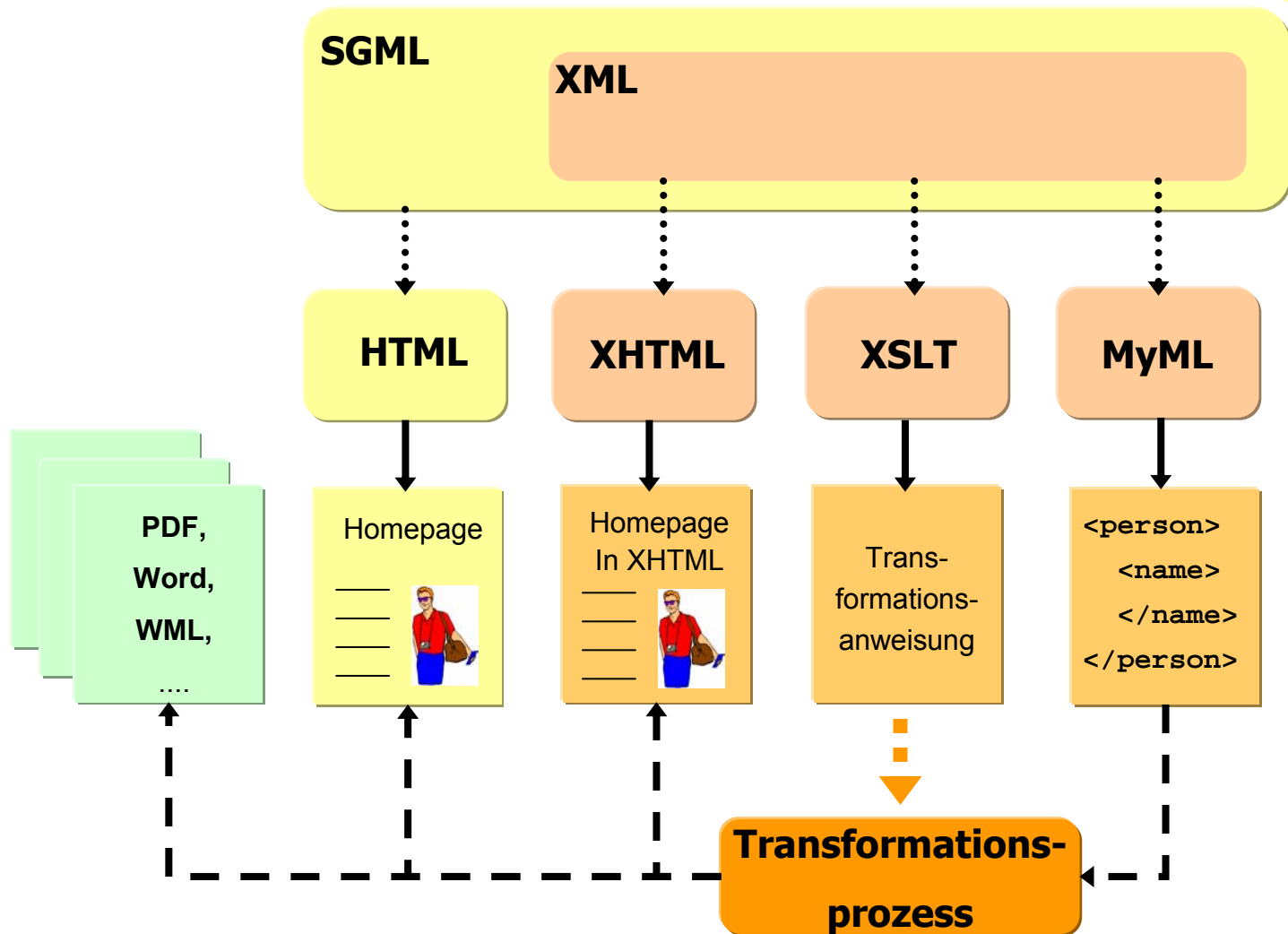


- Konzepte der Transformation von XML durch die „**eXtensible Stylesheet Language for Transformations** (XSLT)“
- Adressierung in XML mit Hilfe der „**XML Path language** (XPath)“
- XSLT als Sprache

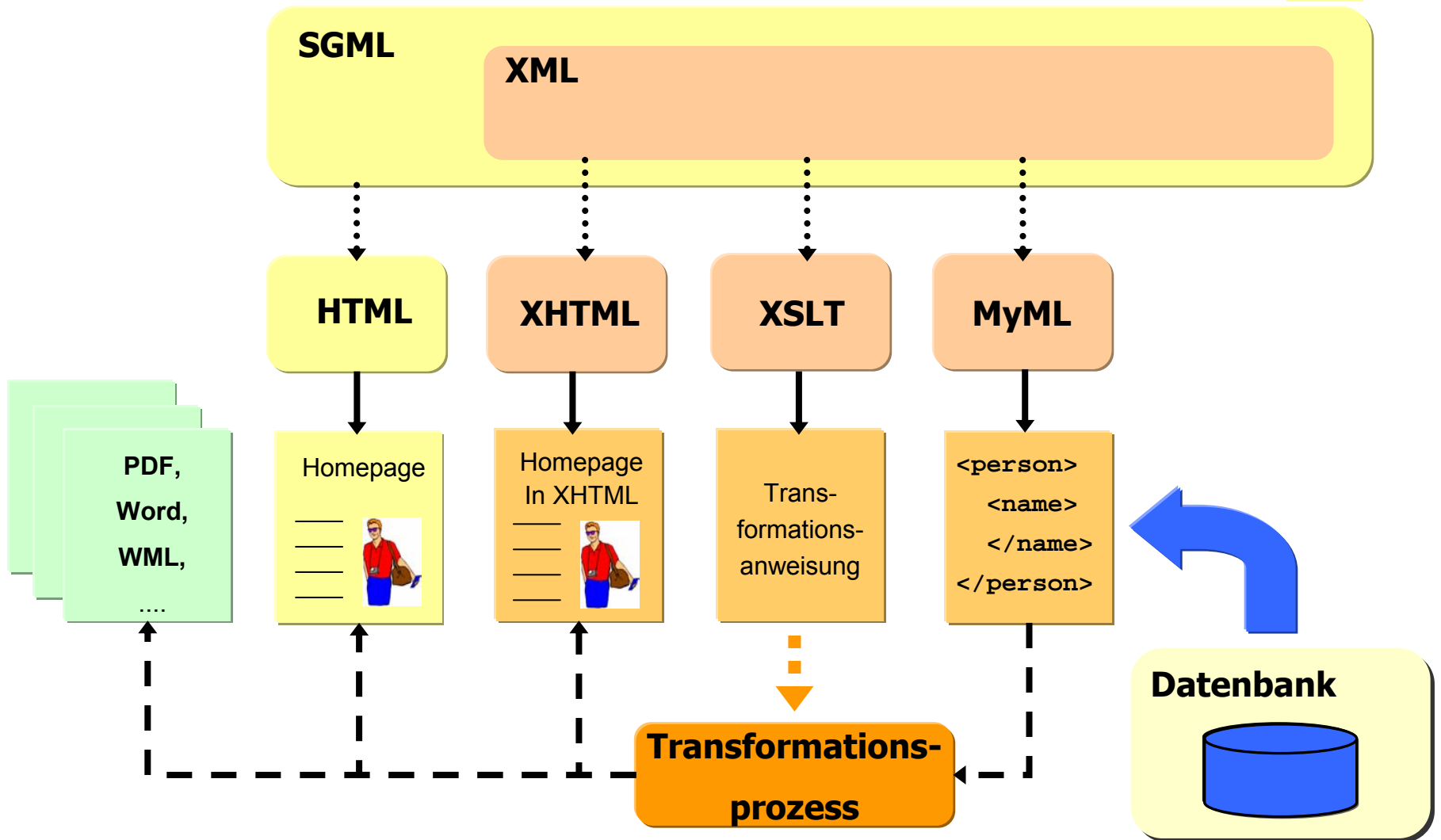
eXtensible Stylesheet Language for Transformations (XSLT)

Konzepte zur XML-Transformation

XSLT-Transformation

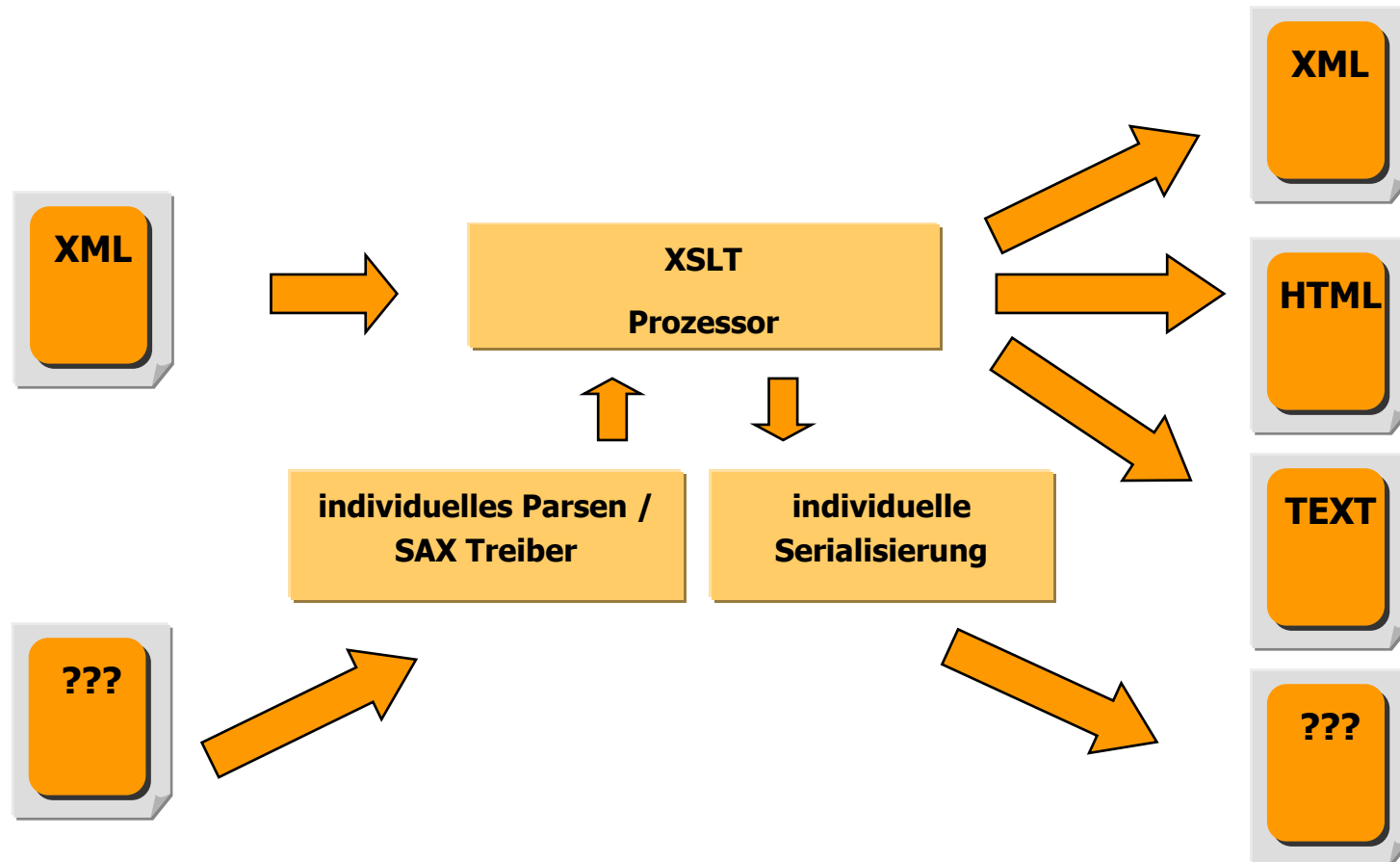


XSLT-Transformation

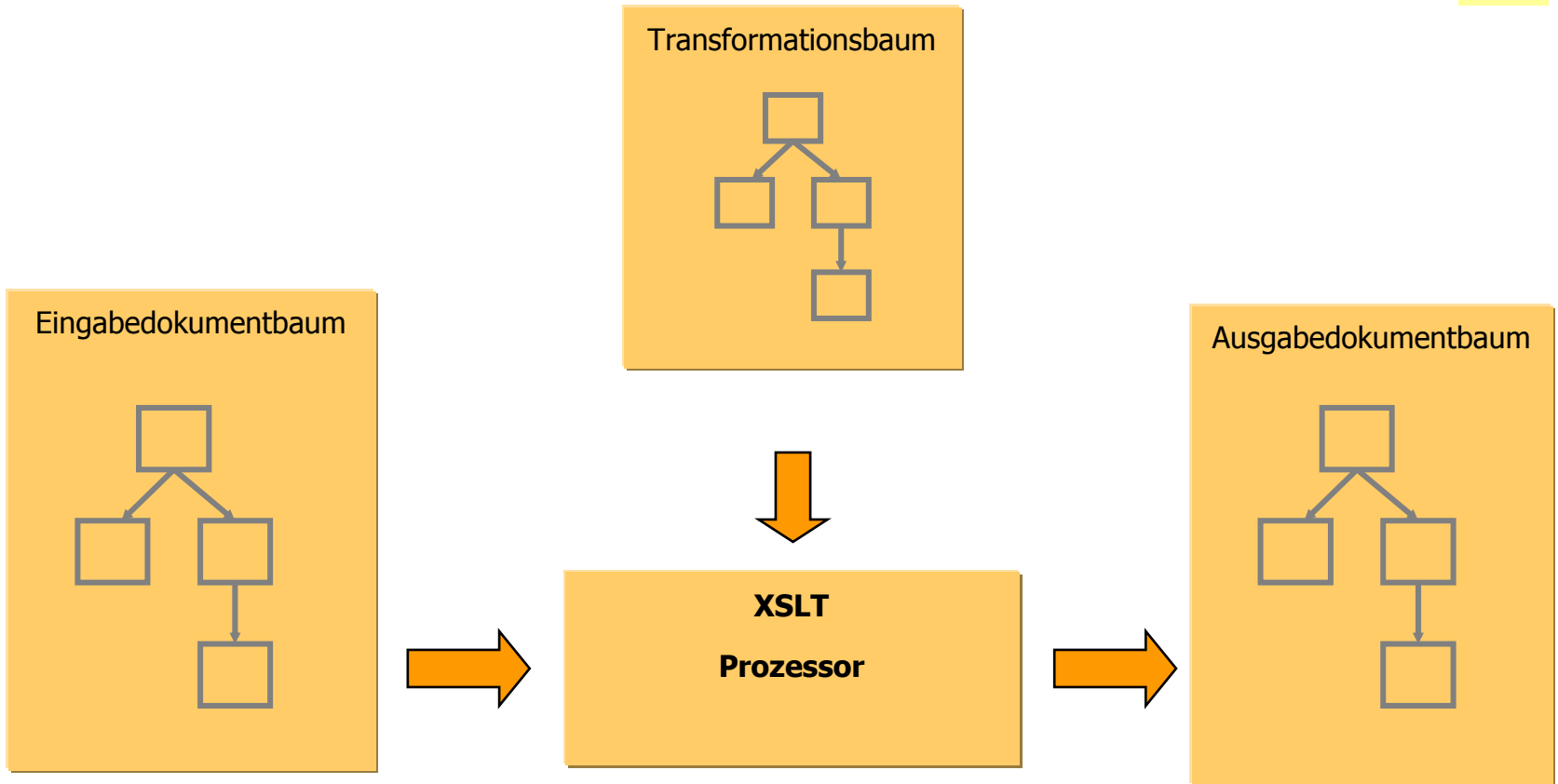


XSLT Konzepte: Ziel

Transformiere die Struktur der Dokumente



XSLT Konzepte: Bäume, Keine Dokumente



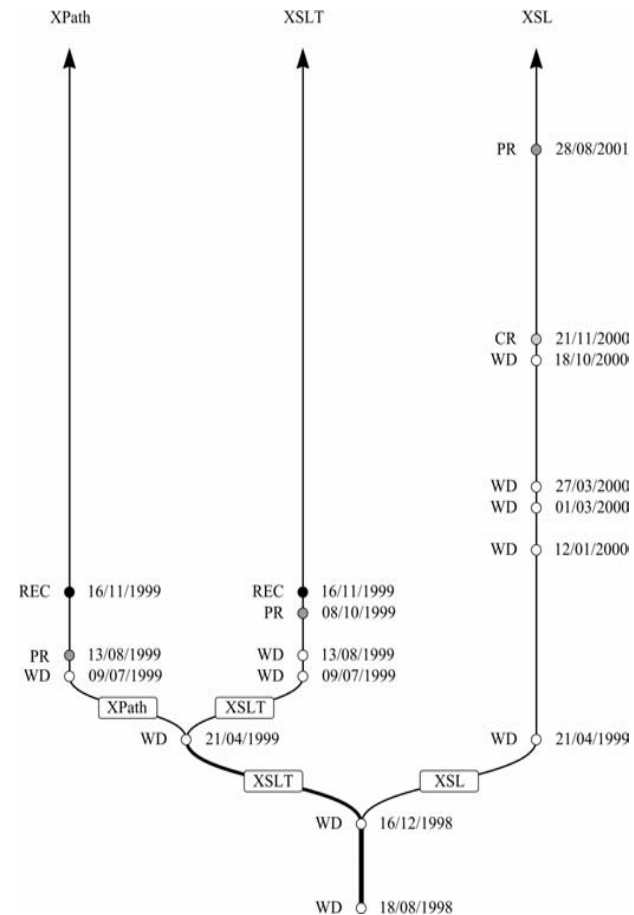
XSLT Konzepte: Allgemeine Idee



- 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
- Vielfältige Möglichkeiten um auf Dokumente zuzugreifen

XSLT: Historie

- 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 1.1 letzter Entwurf am 24.8.2001
- XSLT 2.0 letzter Entwurf am 15.11.2002



XSLT: Aktuell



- Gut entwickelte und gut unterstützte Sprache
- Wird sehr oft als Hilfstechnologie in umfangreichen Projekten eingesetzt
- Dutzende von industriellen Implementierungen inklusive diese von Microsoft (MSXML), Oracle (Oracle XDK), Sun und frei verfügbaren Initiativen: Xalan, Saxon
- Bibliotheken für Java, C/C++, COM, Perl, PHP, Python, PL/SQL
- Wird in XML-basierten „publishing frameworks“ (XSQL, Cocoon, Charlie) benutzt.

XML Path Language (XPath)

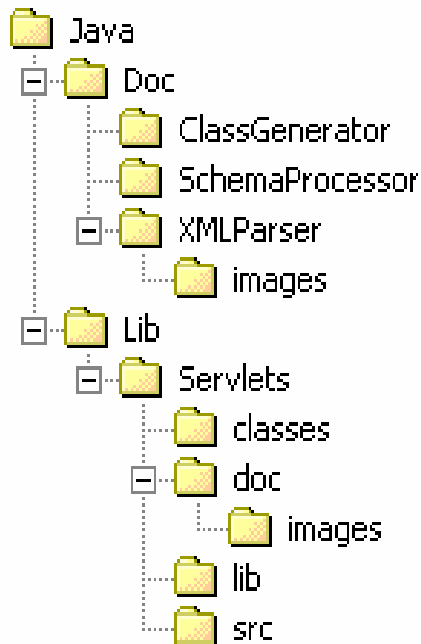
Adressierung in XML

Adressierung: XPath



- XPath ist eine Sprache, um Teile eines XML-Dokuments zu adressieren, und wird z.B. in XSLT, XPointer und XQuery verwendet.
- XPath kann Dokumentknoten unter Angabe verschiedener Kriterien selektieren und grundlegende Manipulationen an Zeichenketten, booleschen Werten und Knotenmengen durchführen.
- XPath unterstützt Mustererkennung auf Dokumentknoten (Testen gegen ein vorgegebenes Muster)
- XPath enthält eine einfache Funktionsbibliothek, welche durch benutzerdefinierte Funktionen erweitert werden kann.
- Kompakte, nicht an XML orientierte Syntax ähnlich zu Pfadangaben bei Betriebssystemen

XPath: Übersicht



```
<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.

- ï `/Java/Lib/Servlets/src` wechselt in das `src` Verzeichnis oder selektiert das `src` Element
- ï `../../../../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`

XPath: 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
- Der Kontext kann von einem Verarbeitungsprozessor, innerhalb eines mehrstufigen Ausdruckes oder beim Auswerten des Prädikats geändert werden.

XPath: Datentypen

- Es gibt vier grundlegende Datentypen in XPath:
 - ▶ Boolean (Wahrheitswerte)
 - ▶ Number (IEEE 754, wie in Java)
 - ▶ String (Kette von Unicode-Zeichen)
 - ▶ Node-set (ungeordnete Menge von Dokumentknoten ohne Kinder)
- Externe Datentypen können auch benutzt werden. So ist in XSLT 1.0 ein Fragment des Ergebnisbaumes wie folgt definiert:
 - ▶ Korrespondiert zum Dokumentfragment des DOM (Document Object Model)
 - ▶ Verhält sich wie eine Knotenmenge mit dem einzigen Wurzelknoten des Baumes.
 - ▶ Kann nicht in eine Knotenmenge konvertiert werden
 - ▶ Wurde in XSLT 1.1 entfernt

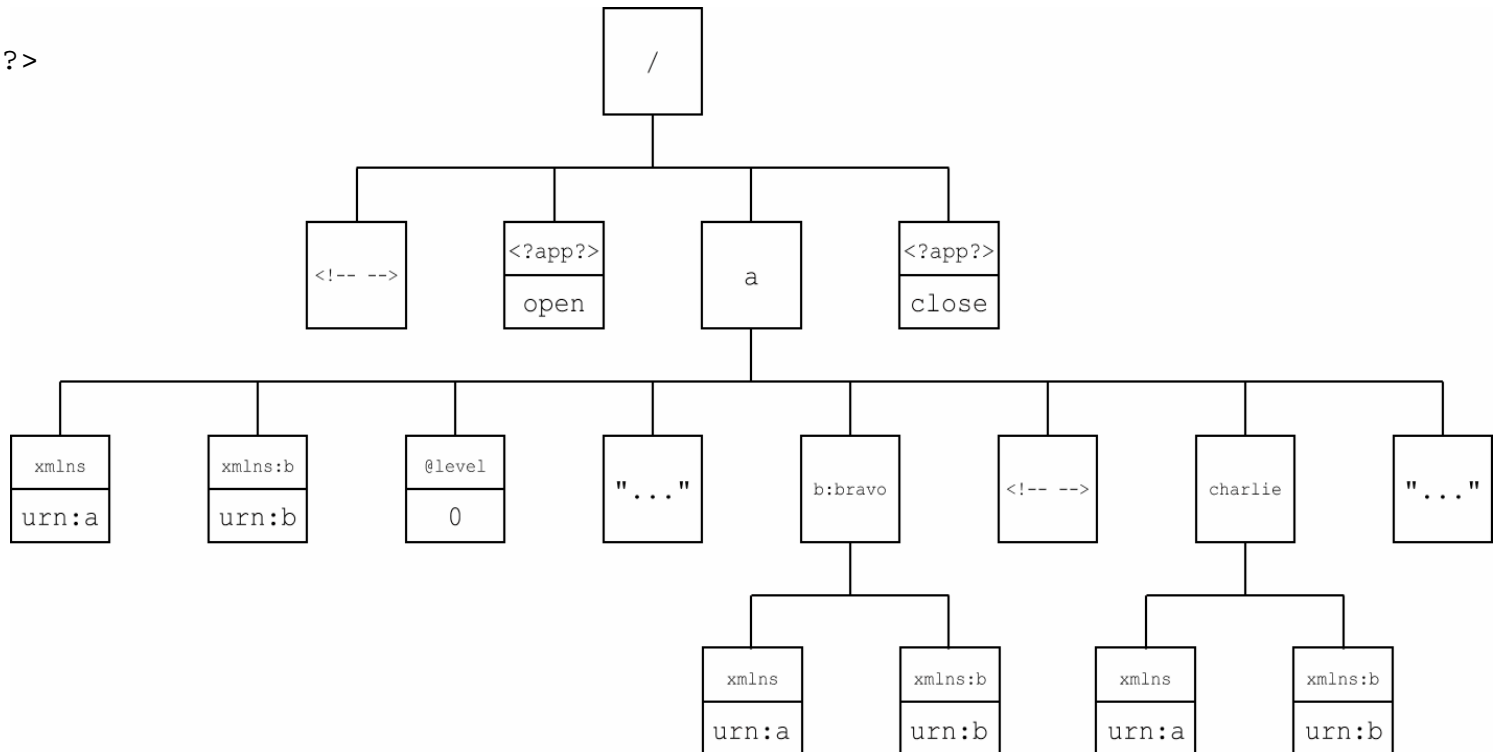
XPath: Dokumentenmodell



- Baumähnliches Datenmodell
- Weicht leicht vom DOM ab
- Sieben Typen von Knoten:
 - ▶ Wurzelknoten
 - ▶ Elementknoten
 - ▶ Attributknoten
 - ▶ Textknoten
 - ▶ Verarbeitungsanweisungsknoten
 - ▶ Kommentarknoten
 - ▶ Namensraumknoten
- Eltern-Kind-Beziehung zwischen Element und Attribut oder Element und Namensraum sind asymmetrisch.

XPath: Dokumentmodellbeispiel

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



XPath: Dokumentreihenfolge



- Obwohl Knotenmengen nicht geordnet sind, werden sie in vielen Fällen in der Dokumentreihenfolge oder umgekehrter Dokumentreihenfolge verarbeitet.
- Die Dokumentreihenfolge ist die Reihenfolge, welcher der syntaktischen Schreibweise entspricht.
 - ▶ Wurzelknoten zuerst
 - ▶ Elementknoten vor ihren Kindknoten, Attribut- und Namensraumknoten
 - ▶ Namensraumknoten vor Attributknoten
 - ▶ Attributknoten vor anderen Kindknoten

XPath: Dokumentreihenfolgenbeispiel

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

Dokumentenreihenfolge ist:

- Wurzelknoten;
- Kommentarknoten <!-- Start -->;
- Verarbeitungsanweisungen <?app open?>;
- Element a;
- Namensraum urn:a;
- Namensraum urn:b;
- Attribut level;
- Textknoten alpha;
- Element b:bravo;
- Namensraum urn:a;
- Namensraum urn:b;
- Kommentar <!-- To Do...-->
- Element charlie;
- Namensraum urn:a;
- Namensraum urn:b;
- Textknoten delta;
- PI <?app close?>.

XPath: 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:

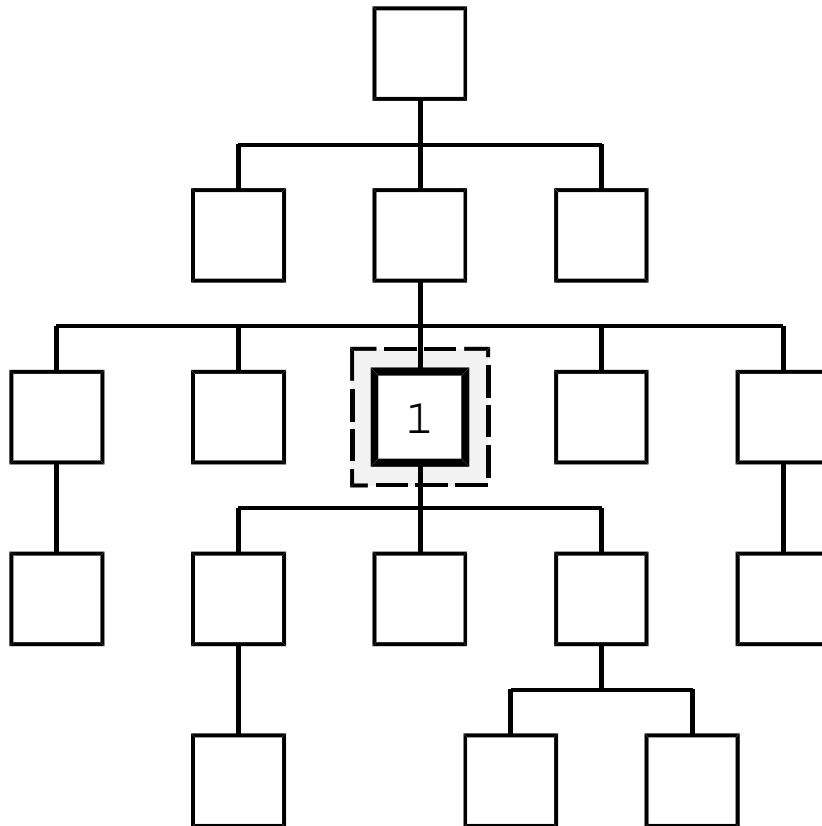
- Navigationachse `attribute` (oder `@` in abgekürzter Form)
- Knotentest `href`
- Prädikat `starts-with(., 'http://')`

XPath: 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:
 - ▶ *navigationsachse::knotenabfrage*
- Für die wichtigsten Navigationsachsen existiert eine Kurzschreibweise.

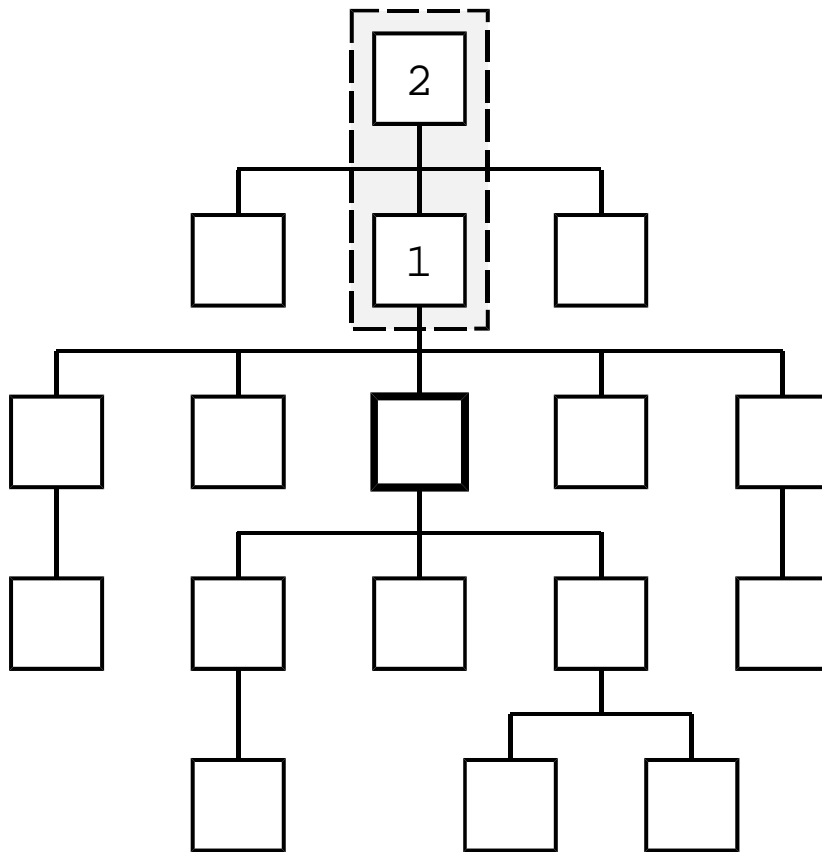
XPath: Navigationachsen (1)



Self-Achse:

- Enthält den Kontextknoten selbst.
- Abkürzende Schreibweise: `.`
- `self::node() = .`

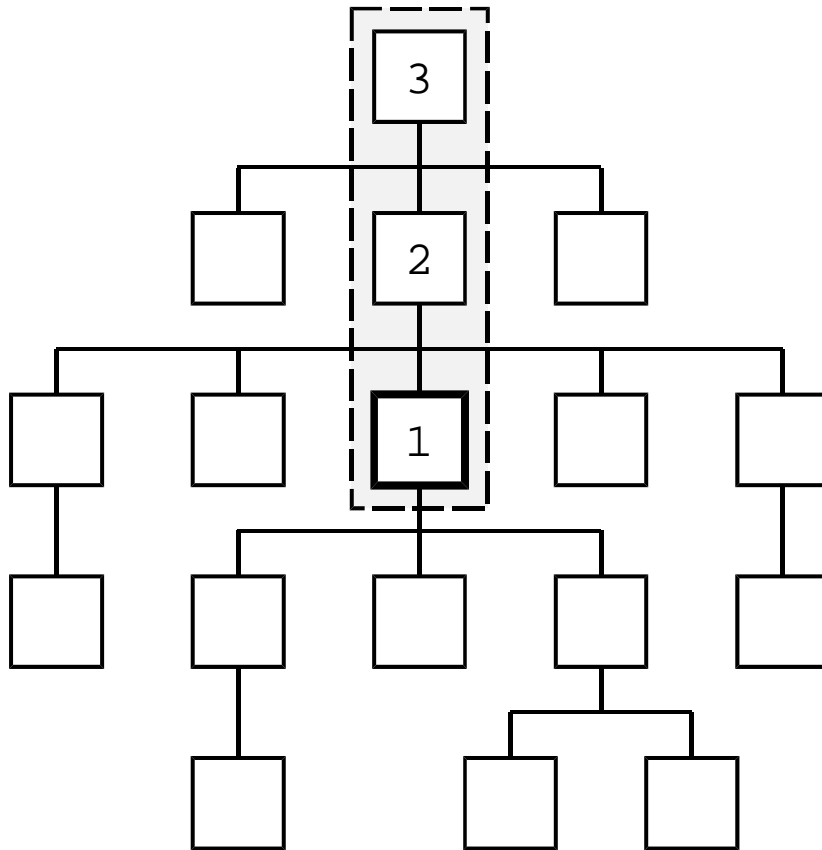
XPath: Navigationachsen (2)



Ancestor-Achse:

- Bis auf den Wurzelknoten haben alle Knoten Vorfahren.
- Bildet Achse vom Kontextknoten zum Wurzelknoten.
- Enthält **NICHT** den Kontextknoten.

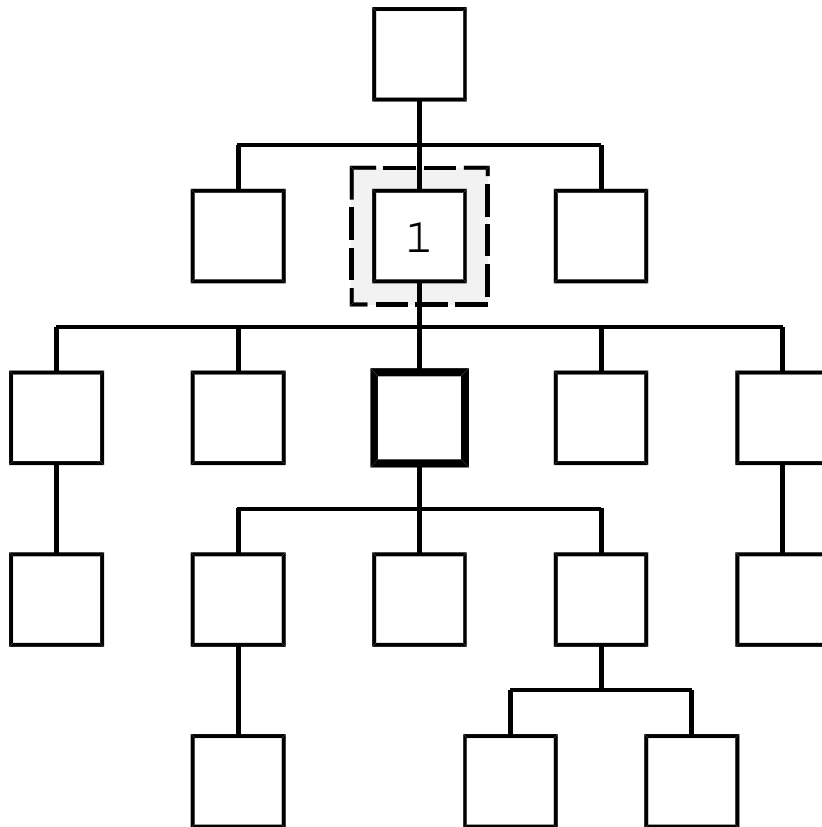
XPath: Navigationachsen (3)



Ancestor-or-self-Achse:

- Wie Ancestor-Achse aber mit Kontextknoten.
- Bildet Achse vom Kontextknoten zum Wurzelknoten.
- Enthält den Kontextknoten.

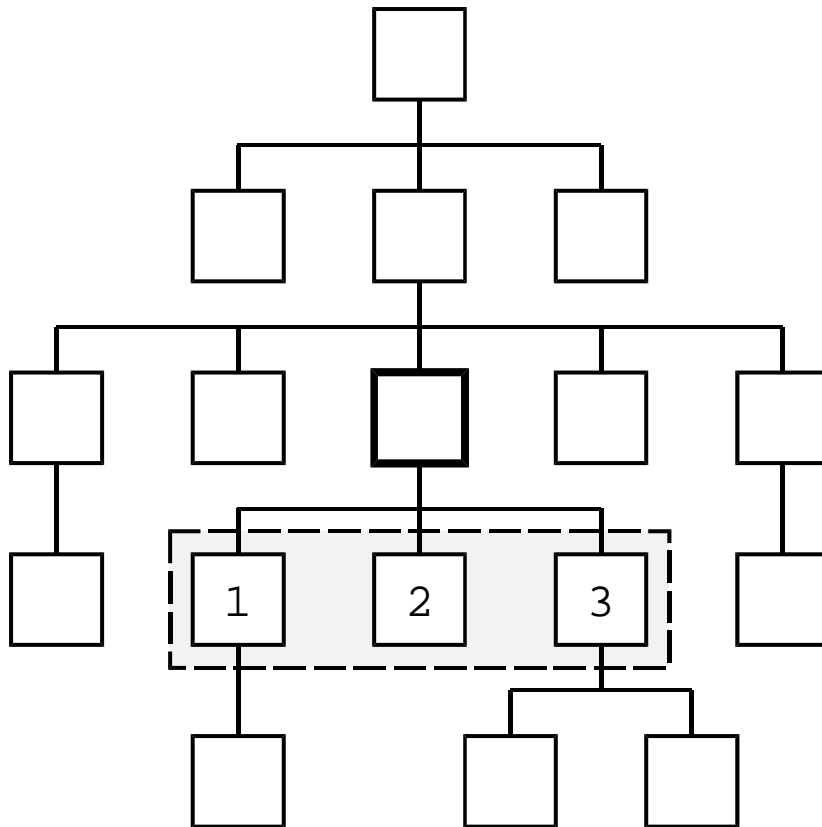
XPath: Navigationachsen (4)



Parent-Achse:

- Direkter Elternknoten des Kontextknoten.
- Abkürzende Schreibweise: ..
- `parent::node() = ..`

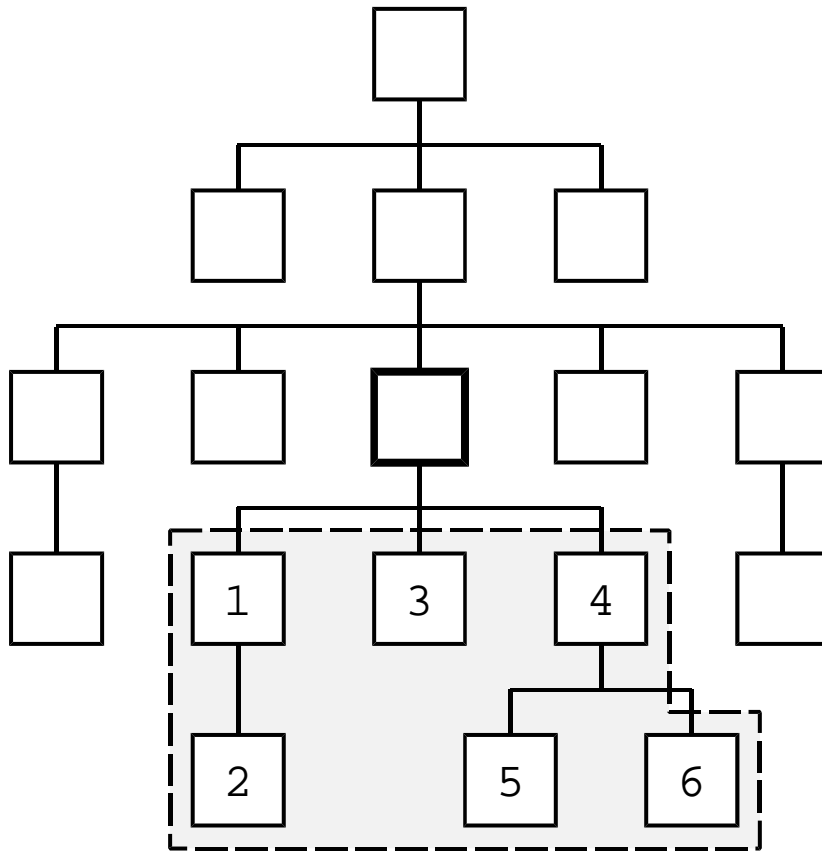
XPath: Navigationachsen (5)



Child-Achse:

- Direkte Kindknoten des Kontextknoten.
- Kann weggelassen werden.
- `artikel/id = child::artikel/child::id`

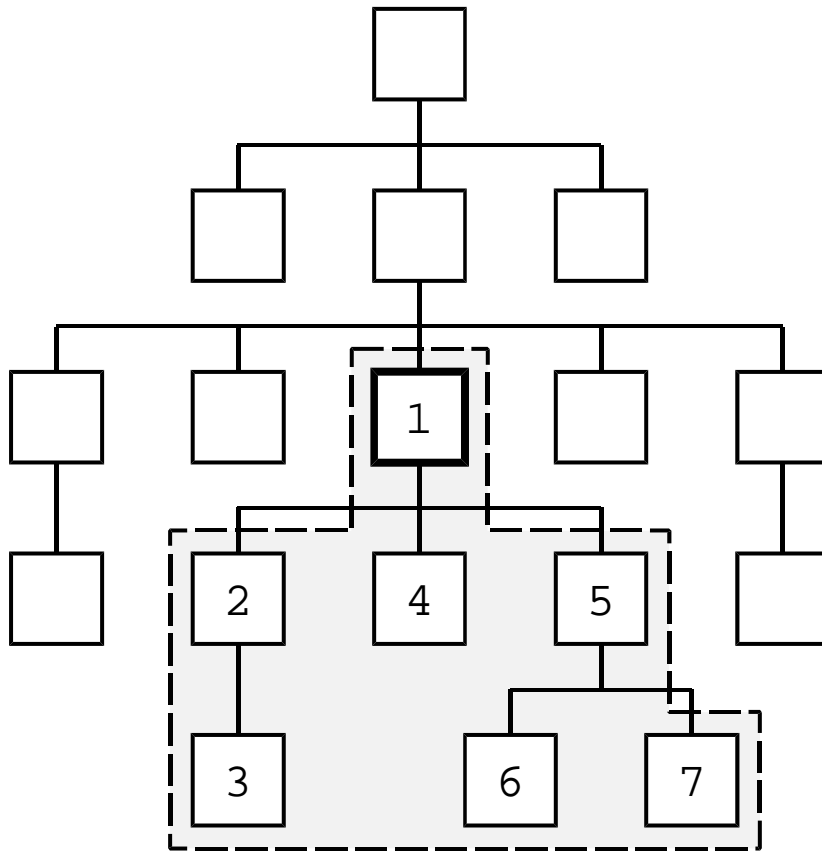
XPath: Navigationachsen (6)



Descendant-Achse:

- Alle Kind- und Kindeskindknoten bis hin zu den Blättern.
- Enthält **NICHT** den Kontextknoten.

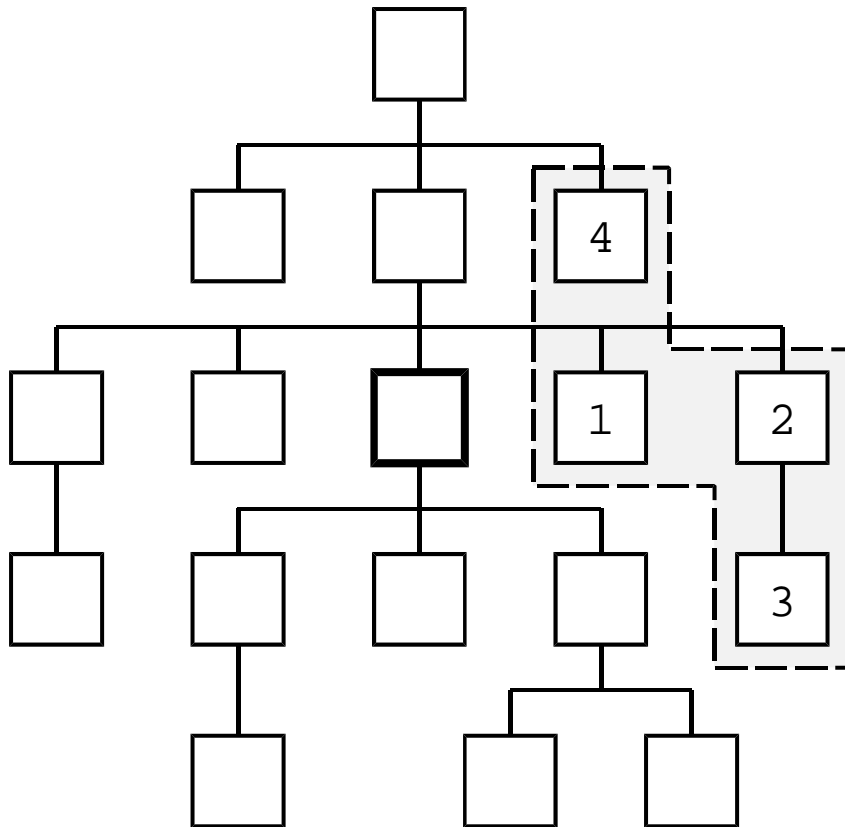
XPath: Navigationachsen (7)



Descendant-or-self-Achse:

- Wie Descendant-Achse aber mit Kontextknoten.
- Enthält den Kontextknoten.
- Abkürzende Schreibweise: //
- `descendant-or-self::node() = //`

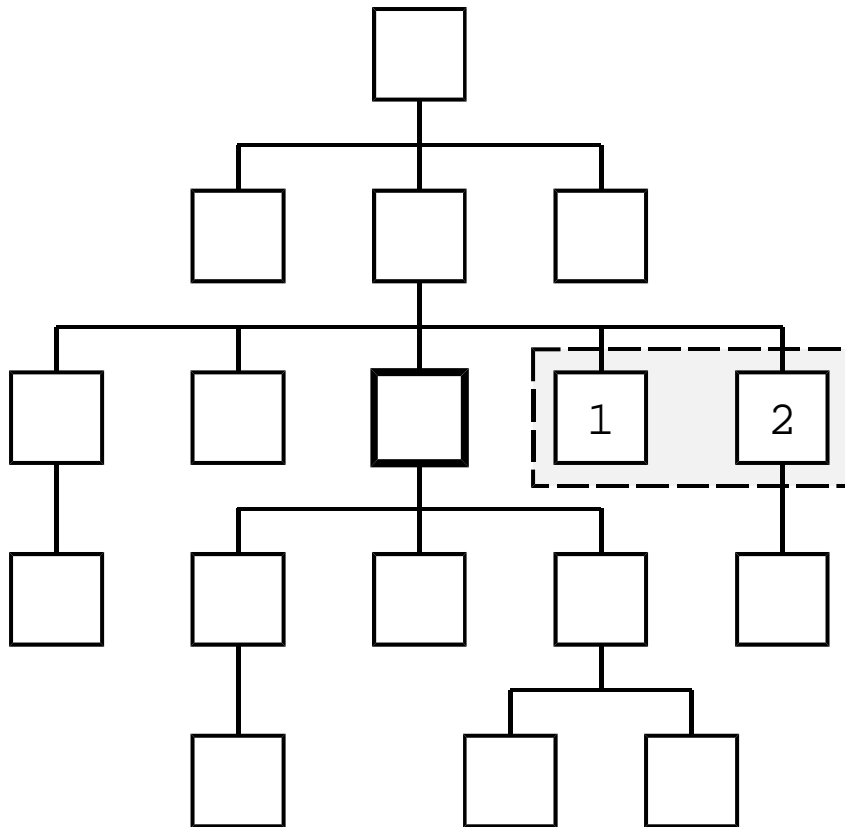
XPath: Navigationachsen (8)



Following-Achse:

- Alle in der Dokumentenordnung nachfolgenden Knoten, ohne Kind- und Kindeskindknoten.
- Enthält **NICHT** den Kontextknoten.

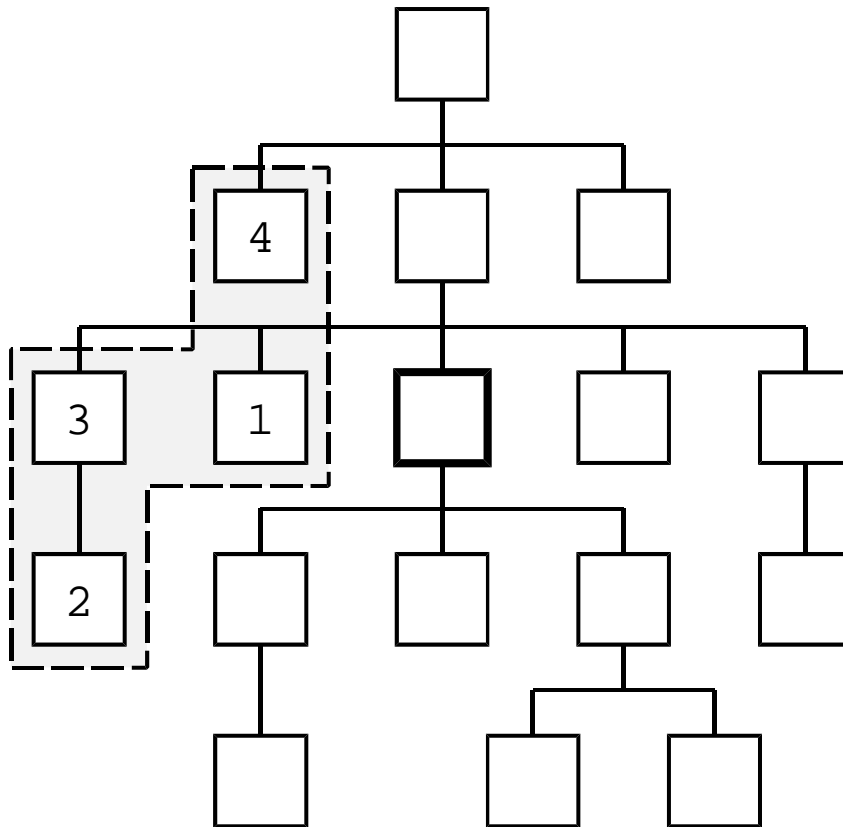
XPath: Navigationachsen (9)



Following-sibling-Achse:

- Alle in der Dokumentenordnung nachfolgenden Knoten auf gleicher Ebene des Kontextknotens.
- Enthält **NICHT** den Kontextknoten.
- Leer, falls Kontextknoten ein Attribut- oder Namespace-Knoten ist.

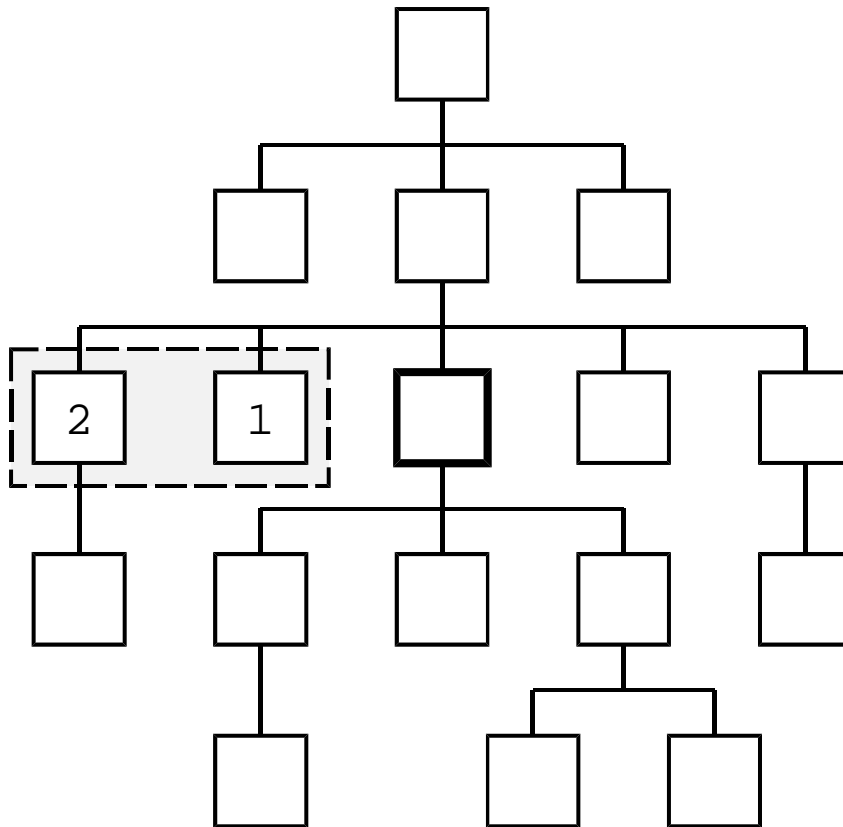
XPath: Navigationachsen (10)



Preceding-Achse:

- Alle in der Dokumentenordnung vorhergehenden Knoten, ohne Vorfahren.
- Enthält **NICHT** den Kontextknoten.

XPath: Navigationachsen (11)



Preceding-sibling-Achse:

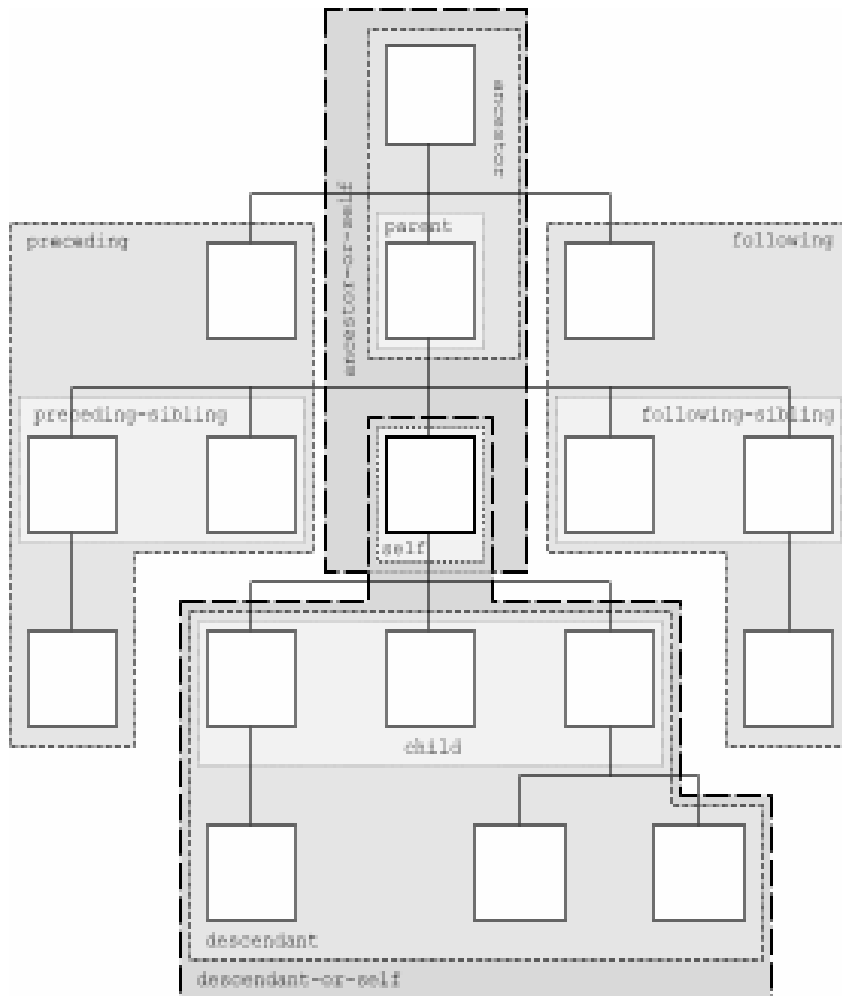
- Alle in der Dokumentenordnung vorhergehenden Knoten auf gleicher Ebene des Kontextknotens.
- Enthält **NICHT** den Kontextknoten.
- Leer, falls Kontextknoten ein Attribut- oder Namespace-Knoten ist.

XPath: Navigationsachsen(12)

- Zusätzlich gibt es noch die beiden Navigationsachsen für Attribut- und Namespace-Knoten (*attribute* und *namespace*).
- Abkürzende Schreibweise für Attribute-Achse: @
- `artikel[attribute::id=„xyz“]` = `artikel[@id=„xyz“]`

XPath: Navigationachsen

Zusammenfassung



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`

XPath: 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 lediglich XPath-Ausdrücke in Rechteckklammern.

eXtensible Stylesheet Language for Transformations (XSLT)



XSLT als Sprache

XSLT: Sprache



- eXtensible Stylesheet Language for Transformations
- Transformationen sind in XSLT eine Menge of Regelschablonen, welche definieren, wie bestimmte Knoten verarbeitet werden
- Regelschablonen können sich auch rekursiv aufrufen, dadurch sehr flexible Möglichkeit ein ganze Dokument zu verarbeiten
- XSLT Elemente konstruieren das Ausgabedokument
- XPath wird für Knotenselektion und Berechnung verwendet
- Seiteneffekte-frei, sehr nah zu funktionalen Programmiersprachen

XSLT: „Hello, World!“

Eingabedokument:

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

Ausgabedokument:

```
<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>
```

XSLT: Schablonen-basierte Sprache

Die Transformation ist eine Menge von Regelschablonen, welche „Templates“ genannt werden. Ein „Template“ wird durch ein `xsl:template` Element definiert:

- ▶ Attribut `match` definiert Muster für die Knoten, welche für die Verarbeitung durch diese Template ausgewählt werden
- ▶ Rumpf eines Templates ist eine Anzahl von Anweisungen, welche ausgeführt werden, wenn das Template aufgerufen wird
- ▶ Attribut `name` erlaubt, dass das Template durch Angabe eines Namen aufgerufen werden kann
- ▶ Ein Template kann andere Templates aufrufen durch `xsl:apply-templates` oder `xsl:call-template` Elemente

Beispiel 1

Eingabe

```
<source>
<title>XSL</title>
<author>John Smith</author>
</source>
```

Ausgabe

```
<h1>XSL</h1>
<h2>John Smith</h2>
```

Stylesheet

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

  <xsl:template match="/">
    <h1>
      <xsl:value-of select="//title"/>
    </h1>
    <h2>
      <xsl:value-of select="//author"/>
    </h2>
  </xsl:template>

</xsl:stylesheet>
```

Stylesheet Struktur

- Wurzelement des Stylesheets ist `xsl:stylesheet`
- Attribut `version` gibt die verwendete XSLT Version an
- XSLT Namensraum muss immer wie folgt lauten:
<http://www.w3.org/1999/XSL/Transform>
- Ein `xsl` Namensraum-Präfix wird oft verwendet

Beispiel

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

Anpassung der Ausgabe

Das oberste Element `xsl:output` wird benutzt, um die Ausgabe anzupassen

- Attribut `method` setzt die Ausgabemethode (`xml`, `html`, `text`)
- Attribut `encoding` setzt Ausgabeformat
- Attribut `indent` bestimmt, ob Einrückung benutzt werden soll

Eingerückte HTML Ausgabe

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

  <xsl:output method="html" indent="yes"/>
  ...
</xsl:stylesheet>
```

XSLT: Transformationskontext



Der Transformationskontext besteht aus:

- Aktuellem Knoten
- Aktueller Knotenliste

Dieser Kontext wird geändert, wenn `xsl:apply-template` oder `xsl:for-each` Elemente ausgeführt werden. Ausdrücke in `select` Attributen bestimmen neue, aktuelle Knotenliste.

Jeder Knoten der aktuellen Knotenliste wird zuerst zum aktuellen Knoten und dann verarbeitet.

Transformationskontexte beeinflussen die Auswertung eines Ausdruckes.

Beispiel 2

Eingabe

```
<source>
<b>Hello, world.</b>
<red>I am </red>
<i>fine.</i>
</source>
```

Ausgabe

```
<p>
  <b>Hello, world.</b>
</p>
<p style="color:red">I am
</p>
<p>
  <i>fine.</i>
</p>
```

Stylesheet

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

<xsl:template match="source">
  <xsl:apply-templates select="bold|red|italic"/>
</xsl:template>

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

<xsl:template match="red">
  <p style="color:red"><xsl:value-of select="."/></p>
</xsl:template>

<xsl:template match="italic">
  <p><i><xsl:value-of select="."/></i></p>
</xsl:template>

</xsl:stylesheet>
```

XSLT: Aufruf von Templates



- `xsl:apply-templates` – wählt die aktuelle Knotenliste aus, dann wird jeder Knoten der Reihe nach zum Aktuellen und die entsprechende Schablonenregel wird ausgeführt.
- `xsl:call-template` – ruft Template mit dem angegebenen Namen auf. Ändert den Kontext nicht.

XSLT: 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>
```

Ausgabe

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

XSLT: Erzeugung des Inhalts

- Benutze literale Ergebniselemente – schreibe einfach `<elem att=î...î>...</elem>` damit dieses Element in der Ausgabe enthalten ist.
- Benutze `xsl:value-of` um einen XPath-Ausdruck auszuwerten und das Ergebnis als Textknoten in die Ausgabe zu übernehmen.
- Benutze `xsl:copy-of` um einen XPath-Ausdruck auszuwerten und das Ergebnis, so wie es ist, in die Ausgabe einzufügen.
- Benutze `{XPath expression}` in Attributen um XPath-Ausdrücke zu berechnen und `{...}` mit dem Ergebnis zu ersetzen.
- Benutze `xsl:element`, `xsl:attribute`, `xsl:text`, `xsl:comment`, `xsl:processing-instruction` um die entsprechenden Knoten in der Ausgabe zu erzeugen.

Beispiel 3

Eingabe

```
<a>"a" consist of <b>"b"</b>,
<c>"c"</c> and <d>"d"</d>.</a>
```

Ausgabe

```
<value-of-a>"a" consist of "b", "c"
and "d".</value-of-a>
<copy-of-a>
  <a>"a" consist of <b>"b"</b>,
<c>"c"</c> and <d>"d"</d>.</a>
</copy-of-a>
```

Stylesheet

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

  <xsl:output indent="yes"/>

  <xsl:template match="/">
    <xsl:element name="value-of-{name(*)}">
      <xsl:value-of select="a"/>
    </xsl:element>

    <xsl:element name="copy-of-{name(*)}">
      <xsl:copy-of select="a"/>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

XSLT: Ablaufsteuerung

- Benutze `xsl:apply-templates`, `xsl:call-template` und `xsl:apply-imports` um Templates, benannte Templates und importierte Template aufzurufen.
- Benutze `xsl:if` um „Wenn-dann“-Steuerung zu realisieren.
- Benutze `xsl:choose`, `xsl:when` und `xsl:otherwise` Kombinationen um „Wenn-dann-anderenfalls“-Steuerung oder mehrfache Auswahlmöglichkeiten zu realisieren.
- Benutze `xsl:for-each` um die angegebenen Anweisungen für jeden Knoten einer ausgewählten Knotenmengen zu wiederholen.

Beispiel 4

Eingabe

```
<list active="Bravo">
  <item>Alpha</item>
  <item>Bravo</item>
  <item>Charlie</item>
</list>
```

Ausgabe

```
<option>Alpha</option>
<option selected>Bravo</option>
<option>Charlie</option>
```

Stylesheet

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

  <xsl:output method="html" indent="yes"/>

  <xsl:template match="list">
    <xsl:apply-templates select="item"/>
  </xsl:template>

  <xsl:template match="item">
    <option>
      <xsl:if test=". = ../@active">
        <xsl:attribute
name="selected">selected</xsl:attribute>
      </xsl:if>
      <xsl:value-of select="."/>
    </option>
  </xsl:template>

</xsl:stylesheet>
```

Identische Transformation, Beispiel 5

Template für identische Transformation

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
</xsl:template>
```

Stylesheet

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

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="*[not(node())]">
    <xsl:copy>empty</xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Eingabe

```
<a>
  One little indian
<b/>
  <c>Two little indians</c>
<d></d>
  <e>Three little indians</e>
</a>
```

Ausgabe

```
<a>
  One little indian
  <b>empty</b>
  <c>Two little indians</c>
  <d>empty</d>
  <e>Three little indians</e>
</a>
```

XSLT: Zusätzliche Möglichkeiten

- Behandle „Whitespaces“ mit `xsl:preserve-space` oder `xsl:strip-space`
- Realisiere alternative Sortierreihenfolge durch `xsl:sort`
- Realisiere Nummerierungen durch `xsl:number`
- Realisiere effizienteren Zugriff auf Knoten mittels Schlüssel durch `xsl:key`
- Sende Nachrichten an den Benutzer durch `xsl:message`
- Realisiere Aliase für Namensräume durch `xsl:namespace-alias`
- Bestimme das Ausgabeformat von Zahlen durch `xsl:decimal-format`

Beispiel 6 [1]

Eingabe

```
<doc>
  <chapter title="First
chapter">
    <section title="First
section">
      <para>paragraph 1</para>
      <para>paragraph 2</para>
      <para>paragraph 3</para>
    </section>
    ...
  <section title="Sixth section">
    <para>paragraph 17</para>
    <para>paragraph 18</para>
  </section>
</chapter>
</doc>
```

Stylesheet

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

  <xsl:output method="text"/>

  <xsl:template match="doc">
    <xsl:text>Resulting document&#xA;</xsl:text>
    <xsl:text>=====&#xA;</xsl:text>
    <xsl:apply-templates select="chapter"/>
  </xsl:template>

  <xsl:template match="chapter">
    <xsl:number format="1. "/>
    <xsl:value-of select="@title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:apply-templates select="section"/>
  </xsl:template>

  ...
```

Beispiel 6 [2]

Ausgabe

```
Resulting document
=====
1. First chapter
  1.1 First section
    a) paragraph 1
    b) paragraph 2
    c) paragraph 3
  1.2 Second section
    d) paragraph 4
    e) paragraph 5
2. Second chapter
  2.1 Third section
    f) paragraph 6
    g) paragraph 7
    h) paragraph 8
    i) paragraph 9
  ...
3. Third chapter
  3.1 Sixth section
    q) paragraph 17
    r) paragraph 18
```

Stylesheet

```
...
  <xsl:template match="section">
    <xsl:number format=" 1.1 "
      level="multiple"
      count="chapter|section"/>
    <xsl:value-of select="@title"/>
    <xsl:text>&#xA;</xsl:text>
    <xsl:apply-templates select="para"/>
  </xsl:template>

  <xsl:template match="para">
    <xsl:number
      format="    a) "
      level="any"
      count="para"/>
    <xsl:value-of select="."/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

XSLT: Variablen und Parameter

- Werden definiert durch `xsl:variable` und `xsl:param` Elemente:
 - ▶ Attribut `name` definiert Variablen-/Parametername.
 - ▶ Attribut `select` definiert Variablen-/Parameterwert. Darf für Variablen nicht geändert werden. Darf für Parameter vom aufrufenden Element gesetzt werden (`xsl:apply-templates`, `xsl:call-template`) durch `xsl:with-param` Element.
- Werte der Variablen und Parameter sind zur Laufzeit nicht änderbar (seiteneffektfrei, funktionaler Stil).
- Benutzt: in komplexen, mehrstufigen Berechnung als temporärer Speicher um den Kontext in kontextsensitiven Ausdrücken zu speichern, um Baumfragmente zu speichern.
- Parameter erlauben parametrisierte, rekursive Aufrufe von Templates.

Beispiel 7

Eingabe

```
<args>
  <x>2</x>
  <y>18</y>
</args>
```

Ausgabe

```
2 plus 18 equals 20
```

Stylesheet

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

  <xsl:output method="text"/>

  <xsl:variable name="x" select="/args/x"/>
  <xsl:variable name="y" select="/args/y"/>

  <xsl:template match="/">
    <xsl:variable name="xPLUSy" select="$x + $y"/>
    <xsl:value-of select="$x"/> plus <xsl:value-of
select="$y"/> equals <xsl:value-of select="$xPLUSy"/>
  </xsl:template>

</xsl:stylesheet>
```

Beispiel 8

Eingabe

```
<args>
  <x>1</x>
  <f>2</f>
  <d>18</d>
  <e>5</e>
  <d>-9.4</d>
  <z>3</z>
</args>
```

Ausgabe

```
1
4
324
25
88.360000000000001
9
```

Stylesheet

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

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:for-each select="args/*">
      <xsl:call-template name="sqr">
        <xsl:with-param name="x" select="."/>
      </xsl:call-template>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:template>

  <xsl:template name="sqr">
    <xsl:param name="x"/>
    <xsl:value-of select="$x * $x"/>
  </xsl:template>

</xsl:stylesheet>
```

Fortgeschrittene XSLT Techniken



Probleme mit offensichtlichen, aber sehr schlechten XSLT Lösungen:

- Gruppierung
- Mengenoperationen
- Schleifen

Gruppierung

Eingabe

```
<items>
  <item source="a" name="A"/>
  <item source="b" name="B"/>
  <item source="a" name="C"/>
  <item source="c" name="D"/>
  <item source="b" name="E"/>
  <item source="b" name="F"/>
  <item source="c" name="G"/>
  <item source="a" name="H"/>
</items>
```

Ausgabe

```
<sources>
  <source name="a">
    <item source="a" name="A"/>
    <item source="a" name="C"/>
    <item source="a" name="H"/>
  </source>
  <source name="b">
    <item source="b" name="B"/>
    <item source="b" name="E"/>
    <item source="b" name="F"/>
  </source>
  <source name="c">
    <item source="c" name="D"/>
    <item source="c" name="G"/>
  </source>
</sources>
```

Stylesheet

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

  <xsl:output indent="yes"/>

  <xsl:key name="src" match="item" use="@source"/>

  <xsl:template match="items">
    <sources>
      <xsl:apply-templates
        select="item[generate-id(.)=generate-id(key('src',@source)))]"/>
    </sources>
  </xsl:template>

  <xsl:template match="item">
    <source name="{@source}">
      <xsl:copy-of select="key('src',@source)"/>
    </source>
  </xsl:template>

</xsl:stylesheet>
```

Operationen auf Knotenmengen

Der Ausdruck:

$$\text{count}(\$node | \$nodeset) = \text{count}(\$nodeset)$$

erlaubt den Test, ob $\$node$ zu $\$nodeset$ gehört.

- Vereinigung: $\$A | \B
- Schnitt: $\$A[\text{count}(. | \$B) = \text{count}(\$B)]$
- Differenz: $\$A[\text{count}(. | \$B) \neq \text{count}(\$B)]$
- Symmetrische Differenz:
 $\$A[\text{count}(. | \$B) \neq \text{count}(\$B)] |$
 $\$B[\text{count}(. | \$A) \neq \text{count}(\$A)]$

- “Allgemeine” Schleifen (wie „for“- oder „while“- Schleifen in den meisten Programmiersprachen) gibt es in XSLT nicht!
- `xsl:apply-templates` oder `xsl:for-each` verarbeiten Knotenmengenelemente, dadurch können keine Schleifen implementiert werden.
- Schleifen müssen durch Rekursion realisiert werden!

Beispiel 10 [1]

Eingabe

```
<pre>One little rabbit  
Two little rabbits  
Three little rabbits</pre>
```

Ausgabe

```
<pre>One little rabbit<BR/>  
Two little rabbits<BR/>  
Three little rabbits</pre>
```

Stylesheet

```
<xsl:stylesheet  
  version="1.0"  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  
  <xsl:output indent="yes"/>  
  
  <xsl:template match="pre">  
    <xsl:copy>  
      <xsl:apply-templates mode="replace"/>  
    </xsl:copy>  
  </xsl:template>  
  
  ...
```

Beispiel 10 [2]

Stylesheet

```
...
<xsl:template name="replace" match="text()" mode="replace">
  <xsl:param name="str" select="."/>
  <xsl:param name="search-for" select="'&#xA;'" />
  <xsl:param name="replace-with">
    <xsl:element name="BR" />
    <xsl:text>&#xA;</xsl:text>
  </xsl:param>
  <xsl:choose>
    <xsl:when test="contains($str, $search-for)">
      <xsl:value-of select="substring-before($str, $search-for)" />
      <xsl:copy-of select="$replace-with" />
      <xsl:call-template name="replace">
        <xsl:with-param name="str"
          select="substring-after($str, $search-for)" />
        <xsl:with-param name="search-for" select="$search-for" />
        <xsl:with-param name="replace-with" select="$replace-with" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$str" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

Erweiterungen

- 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)

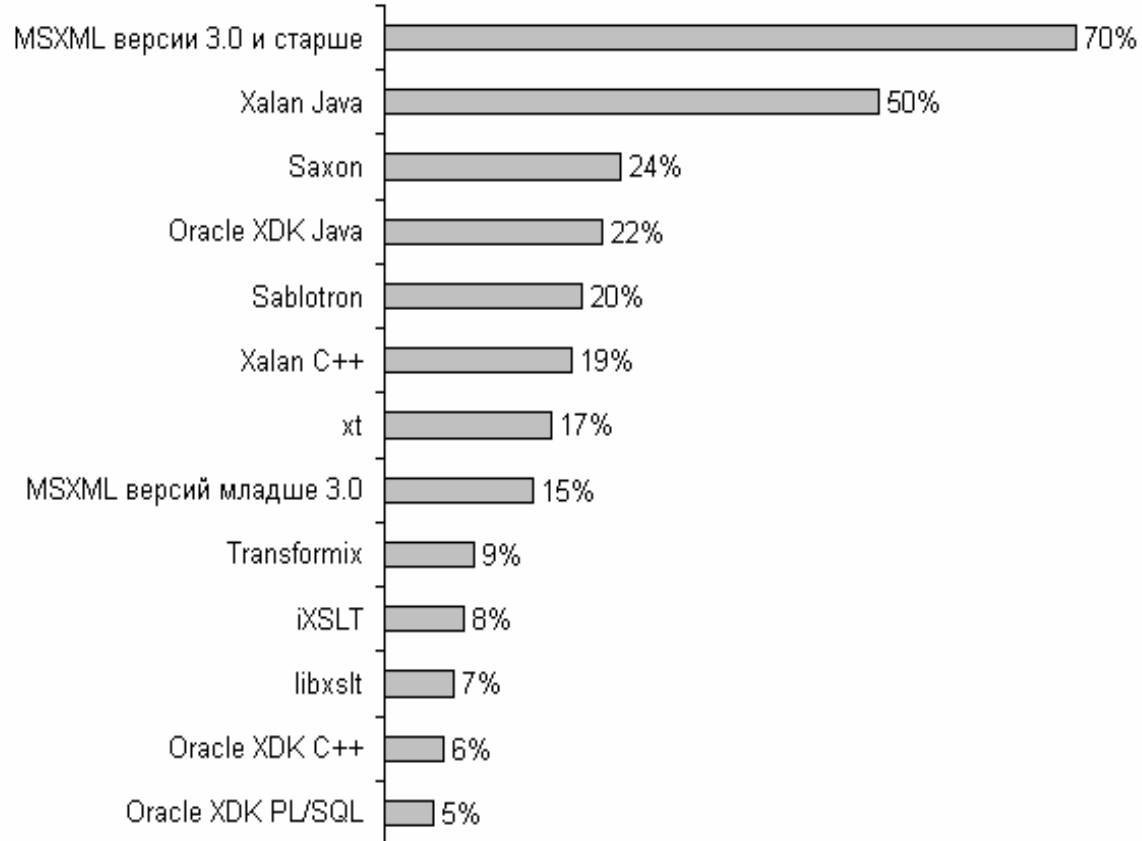
```
<xsl:value-of select="math:round(0.6)"  
  xmlns:math="java:java.lang.Math" />
```
- Java ist die häufigste Sprache für XSLT-Erweiterungen
- Erweiterungen beschränken die Portabilität

- 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?

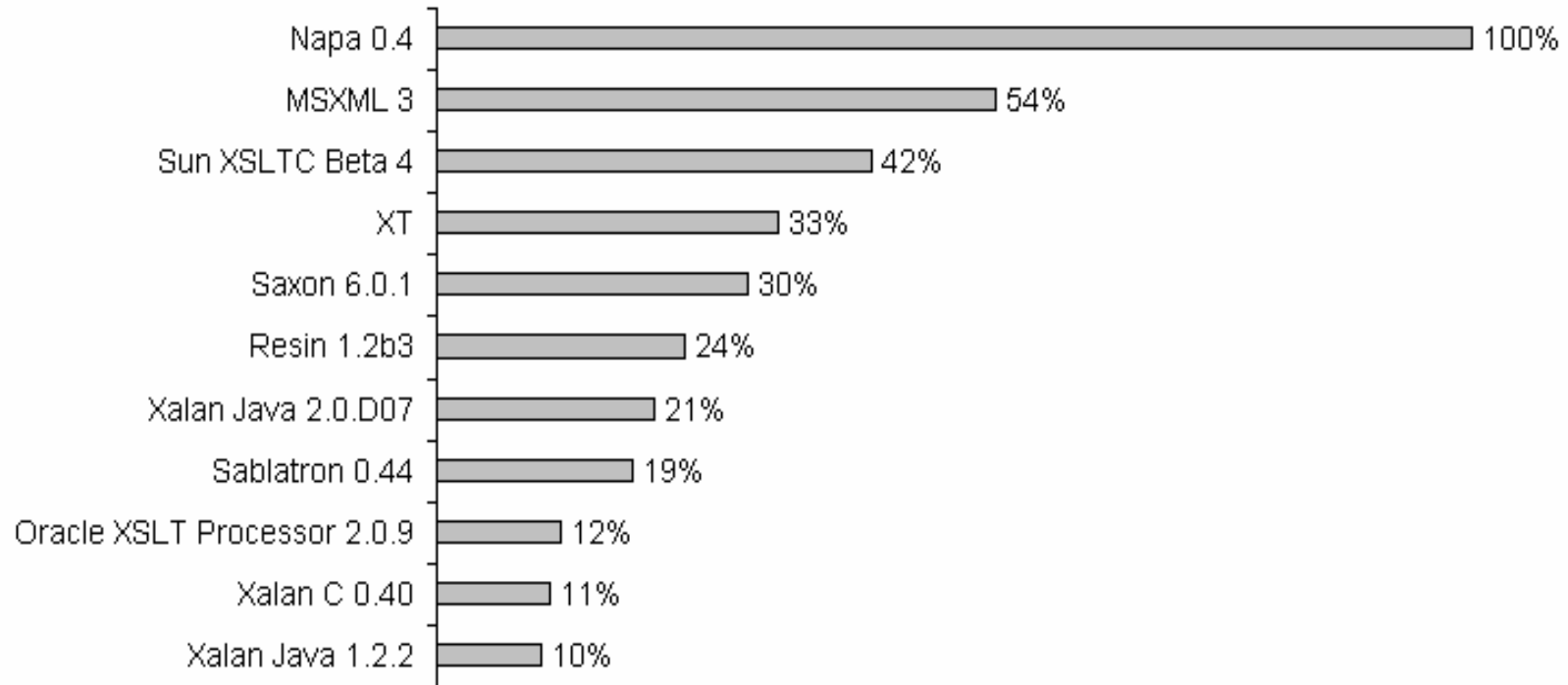
Stärken und Schwächen

- 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

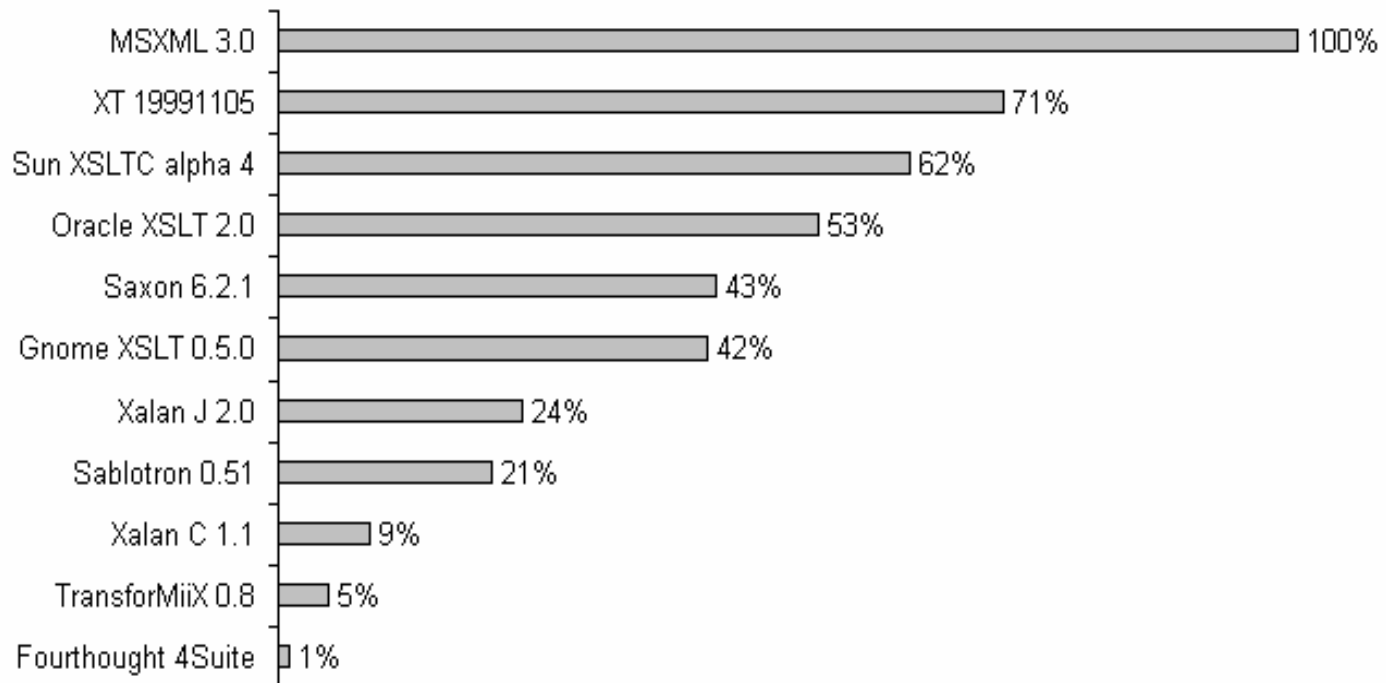
XSLT: Процессорbeliebtheit



XSLT: Prozessoreffizienz [1]



XSLT: Prozessoreffizienz [2]



XSLT: Werkzeuge nach Sprache



■ Java

- ▶ Xalan
- ▶ Saxon
- ▶ Oracle XDK

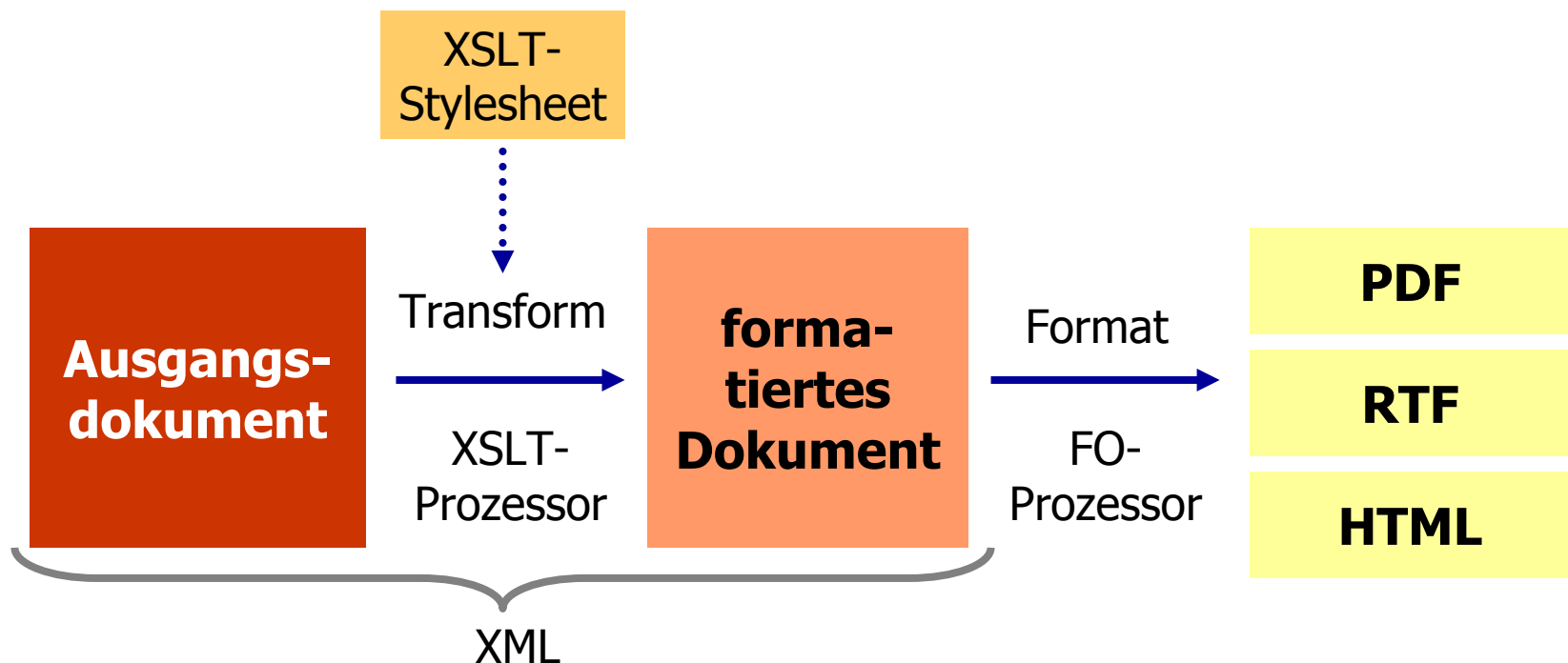
■ COM

- ▶ MSXML

■ Serverseitige Skriptsprachen (Perl, PHP, Python)

- ▶ Sablotron
- ▶ libxslt (von „The Gnome Project“)

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



Empfohlene Literatur:

- Michael Kay. XSLT Programmer's Reference – WROX 2001

- Wassili Kazakos, Andreas Schmidt, Peter Tomczyk: Datenbanken und XML, Springer, April 2002

- W3C
 - ▶ Allgemein: <http://www.w3c.org>
 - ▶ XSL-Allgemein: <http://www.w3c.org/Style/XSL>
 - ▶ XPath-Spezifikation: <http://www.w3c.org/TR/xpath>
 - ▶ XSLT-Spezifikation: <http://www.w3c.org/TR/xslt>