


9. Mediendokumente

- 9.1 Generische Auszeichnungssprachen: XML
- 9.2 XML und Style Sheets
- 9.3 XML für Multimedia: SMIL
- 9.4 XML Transformationen: XSLT 

Weiterführende Literatur:

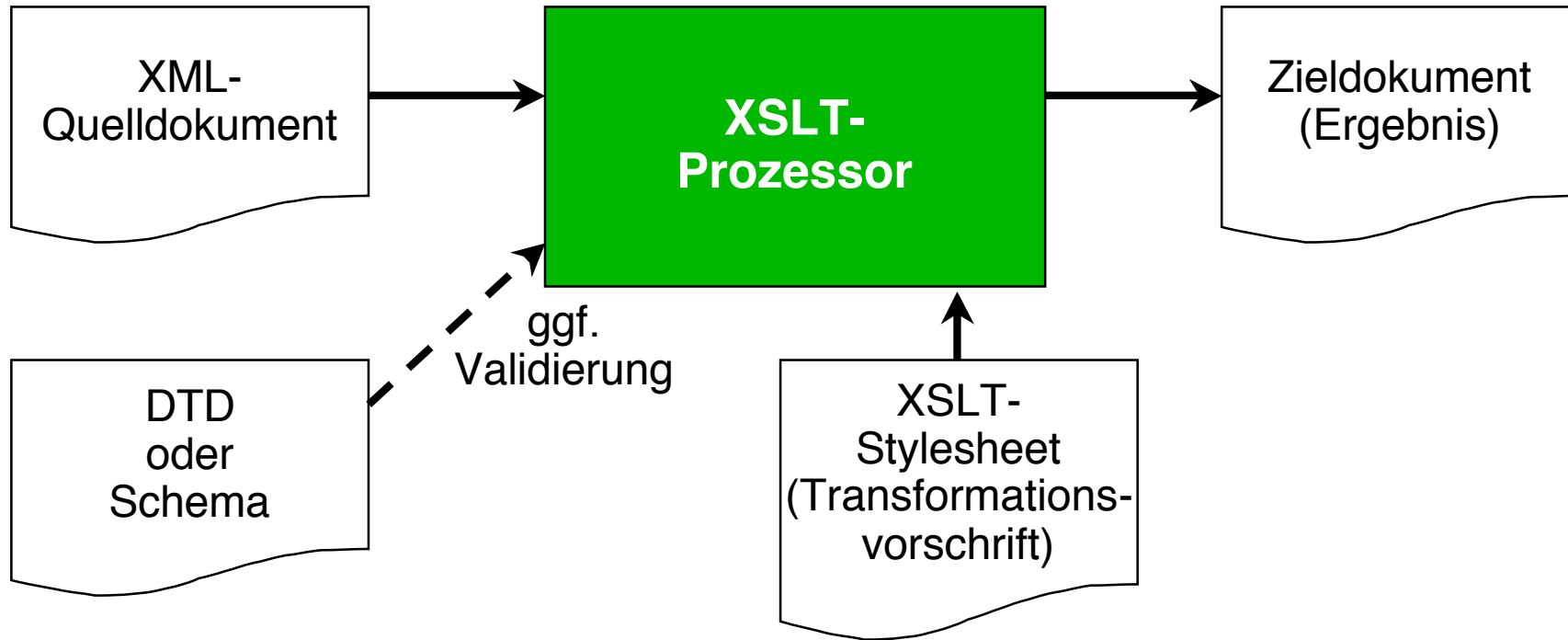
M. Knobloch, M. Kopp: Web-Design mit XML, dpunkt-Verlag 2001

Stylesheets, CSS und XSL

- Zweck von XML + Stylesheets:
 - Trennung des logischen Inhalts von der Präsentation
 - Flexibilität bezüglich der Darstellung auf verschiedenen Plattformen
 - Konsistenzsicherung bei mehrfach dargestellter Information
- Problematische Aspekte von klassischen "Cascading Style Sheets":
 - Verwendet spezielle Syntax ("properties") statt XML
 - *Struktur* der Präsentation muss Struktur des Inhalts folgen
 - » Schwierig: Auslassungen, Reihenfolgeänderungen, Mehrfachdarstellung
 - Keine gute Unterstützung für Druckmedien bzw. entsprechende Darstellung:
 - » Paginierung, Spalten, Kästen, Inhaltsverzeichnis, Index
- eXtensible Style Sheet Language XSL:
 - XSL Formatierungssprache (oft XSL Formatting Objects, XSL-FO genannt)
 - XSL Transformations
 - XPath Navigationssprache

im Folgenden behandelt

Transformation mit XML



- Mögliche Ergebnistypen:
 - XML-Baum
 - HTML-Baum
 - Text

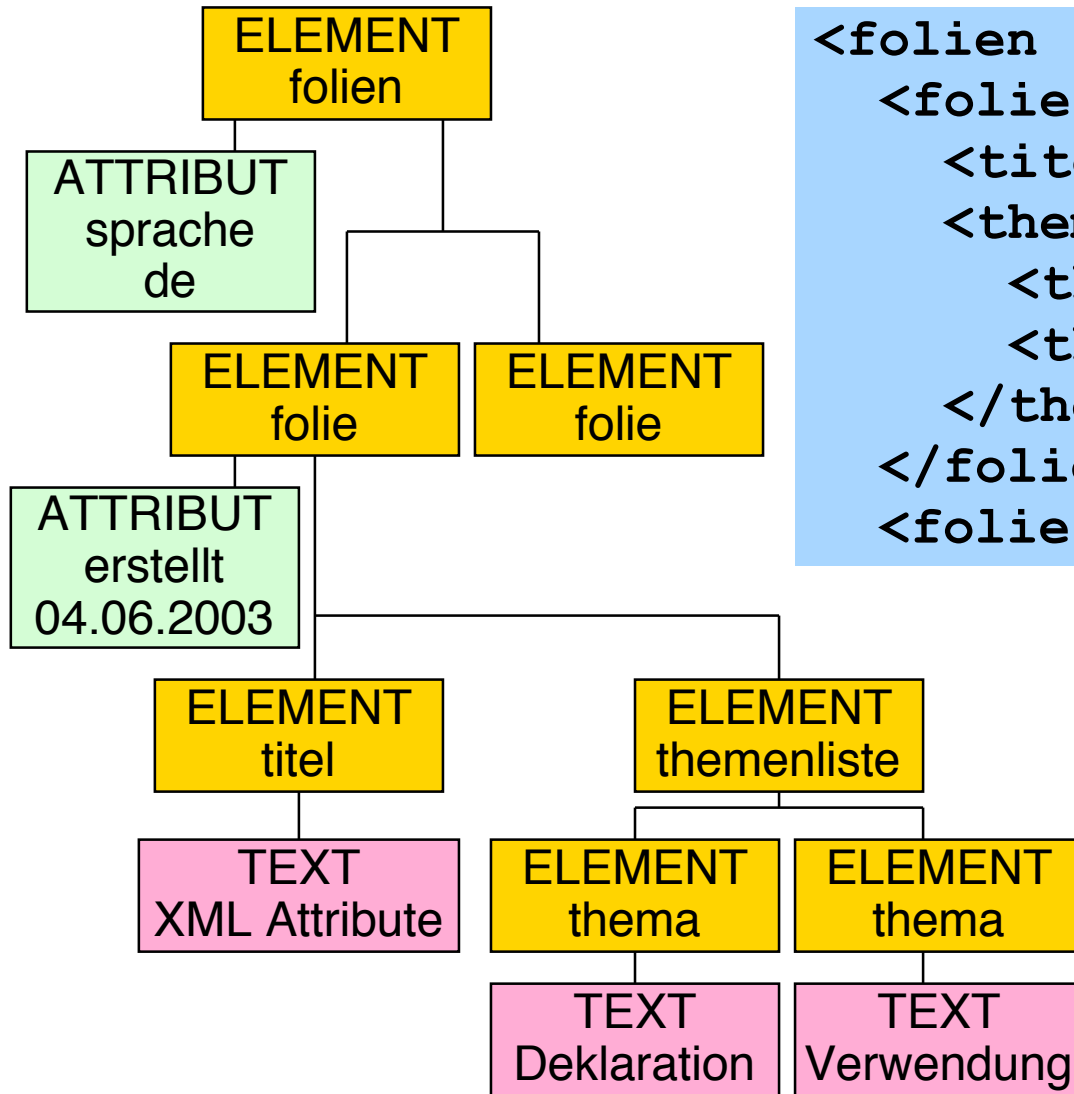
Praktische Bedeutung von XML-Transformation

- Vielfalt von Zugangs-Plattformen für netzgebundene Dienste
 - Handy, PDA, Laptop, PC, Kühlschrank, ...
- Darstellung muss auf umgebende Situation (*Kontext*) eingehen:
 - Restriktionen der Plattform (z.B. Display-Auflösung, Display-Größe)
 - Benutzerpräferenzen (z.B. ob Bilder bei Buch- oder Videoansicht)
- Darstellung der zu präsentierenden Information abstrakt und plattformunabhängig
 - Z.B. in XML
 - Diverse Transformationen von XML-formulierter Kerninformation in spezifische Darstellungskontexte
 - Beispiel: Flugplan o.ä. in HTML und WML (für WAP-Handys)

Wiederholung: XML-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="ftrans1.xsl" ?>
<!DOCTYPE folien SYSTEM "folien2.dtd">
<folien sprache="de">
  <folie erstellt="04.06.2003" ident="f1">
    <titel>Attribute in XML</titel>
    <themenliste>
      <thema>Deklaration in DTD</thema>
      <thema>Verwendung in XML-Dokument</thema>
    </themenliste>
  </folie>
  <folie erstellt="03.06.2003" ident="f2">
    <titel>Identifikatoren</titel>
    <themenliste>
      <thema>Eindeutigkeit</thema>
    </themenliste>
  </folie>
</folien>
```

XML-Dateien als Baumstruktur



```
<folien sprache="de">
  <folie erstellt="04.06.2003">
    <titel>XML Attribute</titel>
    <themenliste>
      <thema>Deklaration</thema>
      <thema>Verwendung</thema>
    </themenliste>
  </folie>
  <folie ...> ...
```

Knotenarten (Auswahl)

- Wurzel-Knoten (*root node*):
 - Ausgangspunkt des Dokuments
 - Kinder: Dokument-Element, Processing Instructions
- Element-Knoten (*element node*):
 - Entspricht *tag* im Dokument
 - Name = Tag-Name, evtl. Attribut-Knoten vorhanden
 - Kinder: Element-Knoten, Text-Knoten
- Attribut-Knoten (*attribute node*):
 - Spezielle Art der "Verwandtschaft" zum Element-Knoten
 - Name = Attribut-Name, Wert = Attribut-Wert, keine Kinder
- Text-Knoten (*text node*):
 - Zeichenkette aus dem Stylesheet-Dokument
 - kein Name, Wert = Zeichenkette, keine Kinder
- Kommentar-Knoten (*comment node*):
 - kein Name, Wert = Kommentartext, keine Kinder

ELEMENT
folien

ATTRIBUT
sprache
de

TEXT
XML Attribute

Einfaches Beispiel für Matching

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*">
    Element
  </xsl:template>
</xsl:stylesheet>
```

- Ergebnis:
 - Nur erstes Element wird gefunden
 - Keine Wiederholung des Matching (keine *Rekursion*)

Templates und Matching

`<xsl:template match="type">`

- Definiert eine *Schablone (template)*, die unter genau definierten Bedingungen auf einen oder mehrere Knoten paßt.

– Wichtigste Typen (Werte von *type*):

`/` Wurzelknoten

`*` Elementknoten

`xyz` Elementknoten des Tags *xyz*

`text()` Textknoten

`@*` Attributknoten

`@abc` Attributknoten mit Namen *abc*

`node()` Beliebiger Knoten außer Attributknoten und Wurzelknoten

– Weitere Einschränkungen, z.B.

» über Pfade (siehe XPath)

» über Bedingungen, z.B. für die relative Position

– Alternativen mit "|", z.B. "`* | @*`"

Rekursion

- Dokumentengetriebener Aufruf von Templates:
 - Rekursion explizit angestoßen mit
`<xsl:apply-templates select="pfad">`
 - *pfad*-Attribut kann fehlen, dann:
Standard-Rekursion über alle Nachfolge-Knoten *ohne* Attributknoten
 - *pfad*-Attribut zur Rekursion inklusive Attributknoten:
`<xsl:apply-templates select="*|@*">`
- Eingebaute Standard-Templates in XSLT
 - Stellen Text-Inhalte (hintereinander verkettet) lesbar dar
 - Sind überall (zusätzlich) wirksam, wo nicht durch eigene Templates "überschrieben"
- Direkter Aufruf von (benannten!) Templates:
`<xsl:call-template name="templateName">`

Einfaches Beispiel für Matching und Rekursion

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="*">
    Element
    <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```

- Ergebnis:
 - Alle Elemente erkannt
 - Standard-Templates geben zusätzlich Inhaltstext wieder

Beispiel: XSLT-Stylesheet (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Transformationsdemo</title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
```

Beispiel: XSLT-Stylesheet (2)

```
<xsl:template match="folie">
  <h2>
    <xsl:value-of select="titel"/>
  </h2>
  <ul>
    <xsl:apply-templates
      select="themenliste/thema"/>
  </ul>
  <p>
    <i>Foliename:
      <xsl:value-of select="@ident"/>,
      Erstellt am:
      <xsl:value-of select="@erstellt"/>
    </i>
  </p>
</xsl:template>
```

Beispiel: XSLT-Stylesheet (3)

```
<xsl:template match="thema">  
  <li>  
    <xsl:value-of select="."/>  
  </li>  
</xsl:template>  
</xsl:stylesheet>
```

Beispiel: Transformationsergebnis

- Für obige Beispielfolie:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=UTF8">
    <title>Transformationsdemo</title>
  </head>
  <body>
    <h2>Attribute in XML</h2>
    <ul>
      <li>Deklaration in DTD</li>
      <li>Verwendung in XML-Dokument</li>
    </ul>
    <p><i>Foliename: f1, Erstellt am: 04.06.2003</i></p>

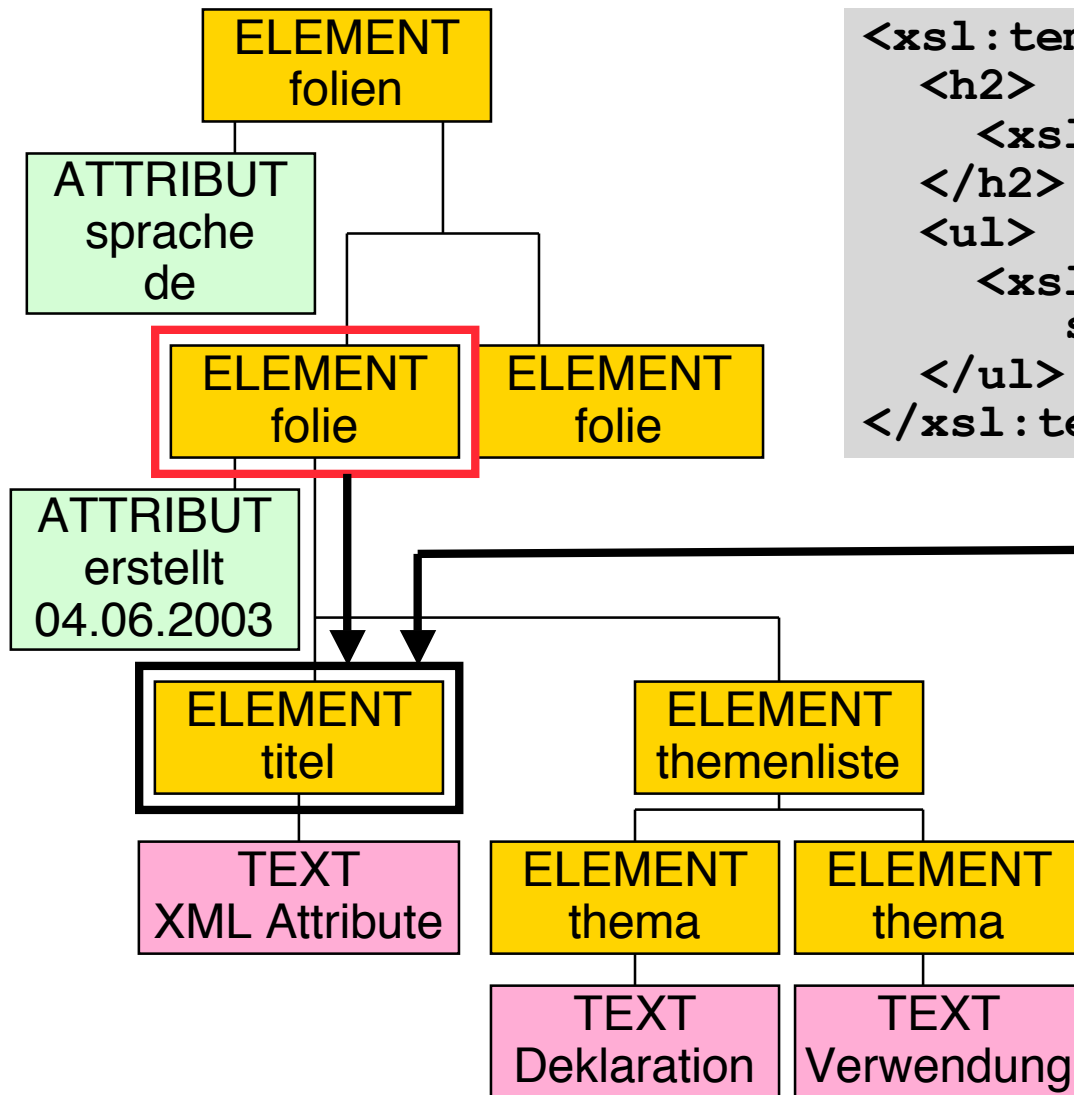
    <h2>Identifikatoren</h2>
    <ul>
      <li>Eindeutigkeit</li>
    </ul>
    <p><i>Foliename: f2, Erstellt am: 03.06.2003</i></p>
  </body>
</html>
```

Auslesen von Information

`<xsl:value-of select="expression">`

- *expression* liefert Zeichenreihe, Knotenmenge oder Teilbaum
- *expression* erlaubt Navigation im Baum
 - mit XPath-Syntax (weitere Details sh. später)
 - » **xyz** Wert der Element-Unterknoten mit Name *xyz*
 - » **@xyz** Wert der Attribut-Unterknoten mit Name *xyz*
 - relativ zum aktuellen Knoten (*current node*) und einer aktuellen Knotenmenge (*current node set*) ausgewertet
- Wichtigste Funktionen in *expression*:
 - **current ()** oder **.** Wert des aktuellen Knotens
 - **name ()** Name des aktuellen Knotens

Beispiel zur Regelanwendung



```

<xsl:template match="folie">
  <h2>
    <xsl:value-of select="titel"/>
  </h2>
  <ul>
    <xsl:apply-templates
      select="themenliste/thema"/>
  </ul>
</xsl:template>
  
```

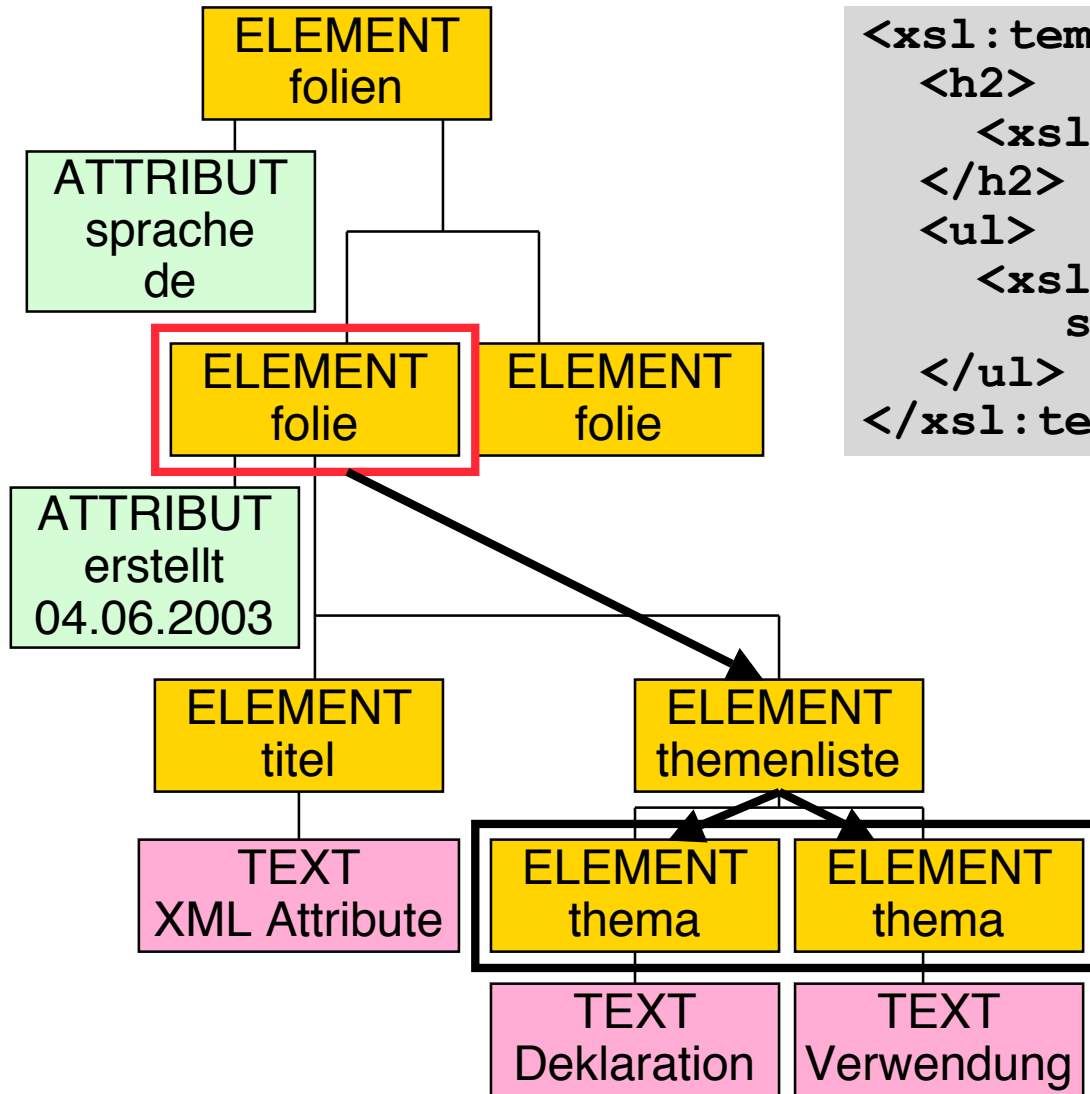
```

<h2>
  <xsl:value-of select=
    WertAktuellerKnoten/>
</h2>
<ul> ... </ul>
  
```

```

<h2>
  XML Attribute
</h2>
<ul> ... </ul>
  
```

Beispiel zur Regelanwendung (2)



```

<xsl:template match="folie">
  <h2>
    <xsl:value-of select="titel"/>
  </h2>
  <ul>
    <xsl:apply-templates
      select="themenliste, thema"/>
  </ul>
</xsl:template>
  
```

```

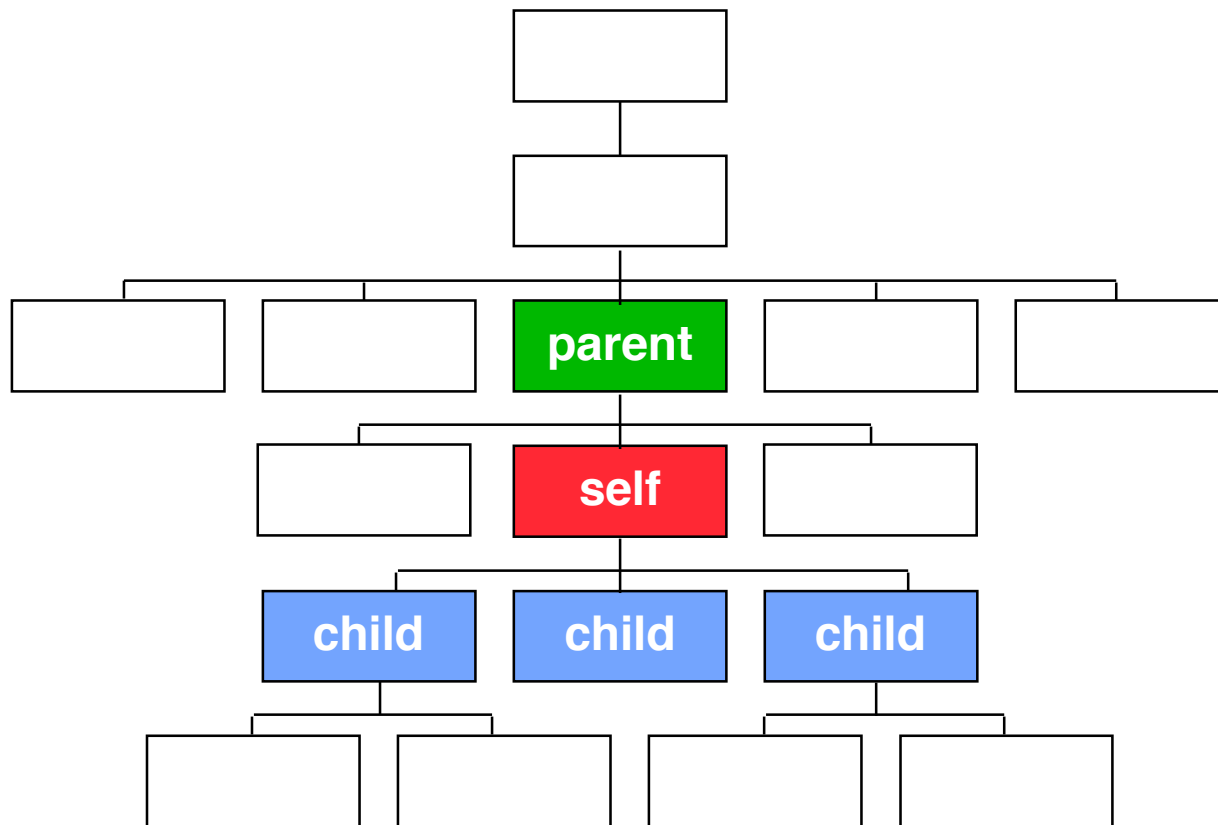
<h2>
  XML Attribute
</h2>
<ul>
  <xsl:apply-templates
    select = />
</ul>
  
```

XPath: Grundkonzepte

- Navigation in XML-Baumstruktur:
 - nicht nur in XSL benötigt, sondern auch in anderen Standards (z.B. XPointer)
 - eigener W3C-Standard "XPath"
 - XSLT nicht ohne XPath verwendbar
- Grundidee: Pfadausdrücke zur Selektion von Werten in Bäumen
 - Mengen von Knoten als Ergebnis
 - Auswertung relativ zu einer bestimmten Position im Baum ("self")
- Einfache Beispiele für XPath-Ausdrücke:
 - .
 - `themenliste/thema`
- Komplexe Beispiele
 - Verwendung 13 verschiedener Baum-Dimensionen ("Achsen")

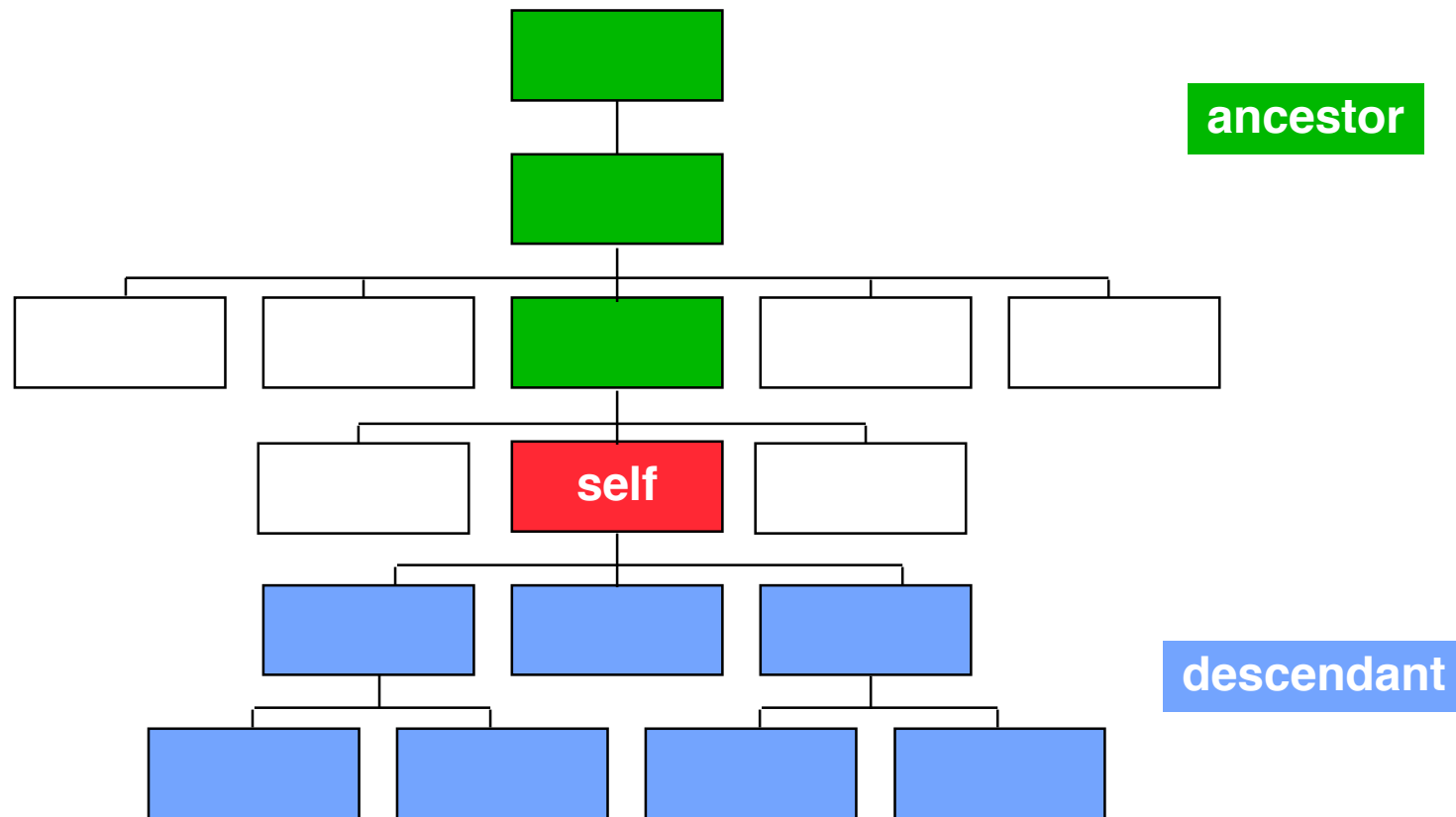
XPath: Achsen (1)

- self-Achse: nur aktueller Kontextknoten
- child-Achse: Kind-Knoten des aktuellen Knotens
- parent: Eltern-Knoten des aktuellen Knotens



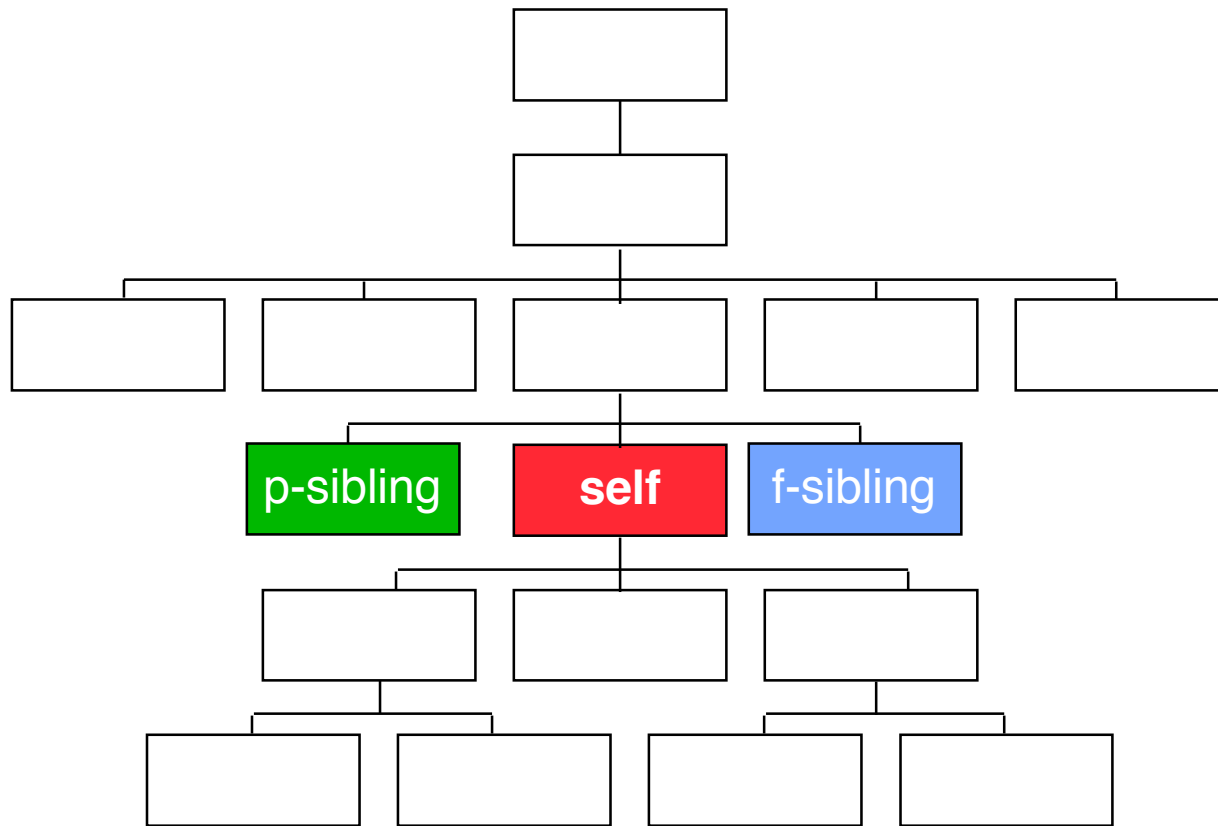
XPath: Achsen (2)

- descendant-Achse: Nachkommen ohne aktuellen Kontextknoten
- descendant-or-self-A.: Nachkommen incl. dem aktuellen Kontextknoten
- ancestor-Achse: Vorfahren ohne aktuellen Kontextknoten
- ancestor-or-self-A.: Vorfahren incl. dem aktuellen Kontextknoten



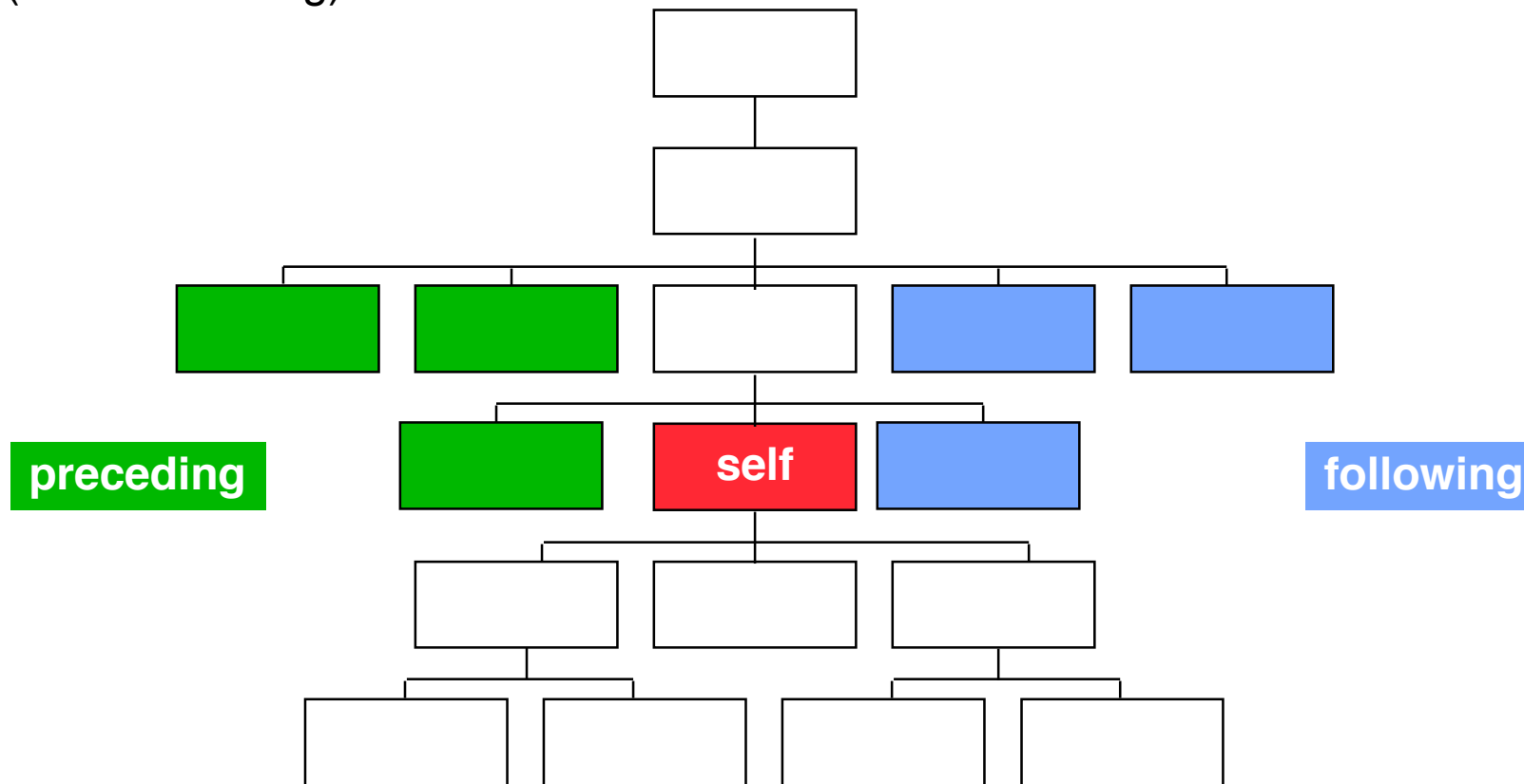
XPath: Achsen (3)

- preceding-sibling-Achse: vorangehende Geschwisterknoten
- following-sibling-Achse: nachfolgende Geschwisterknoten



XPath: Achsen (4)

- preceding-Achse: voranstehende Knoten im (Text-)Dokument ohne Vorfahren
- following-Achse: nachstehende Knoten im (Text-)Dokument ohne Nachkommen
- (Präfix-Ordnung)



XPath: Achsen (5)

- Bei allen vorhergehenden Achsen sind grundsätzlich die Attribut- und Namensraum-Knoten ausgeschlossen.
- **Attribut-Achse:**
 - alle Attributknoten des aktuellen Knotens
 - nur existent, wenn aktueller Knoten Elementknoten ist
- **Namensraum-Achse:**
 - alle Namensraumknoten des aktuellen Knotens
 - nur existent, wenn aktueller Knoten Elementknoten ist

XPath: Pfad-Syntax (Ausführliche Fassung)

`<relative location path> ::= <step> | <relative location path> / <step>`

`<step> ::= <axis name> : : <node test> <predicate>*`

`<axis name>`

- Eine der Achsen-Bezeichnungen (sh. oben)

`<node test>`

- Name des Knotens (z.B. `title`) oder
- `*` (alle Knoten der betreffenden Achse) oder
- Typtest, z.B. `text()`, `comment()`, `node()`

`<predicate> ::= [<expression>]`

- Ein Boolescher Ausdruck unter Verwendung weiterer Pfadausdrücke und vordefinierter Funktionen, z.B. `position()` und arithmetischer Operationen

Absolute Pfade:

- starten an der Wurzel: Vorangestelltes `/`

XPath: Beispiele für ausführliche Pfad-Syntax

- `descendant-or-self::titel`
- `descendant::* / attribute::*`
- `descendant::* / attribute::erstellt`
- `descendant::folie / child::themenliste / child::thema / child::text() [string-length() > 20]`
- `descendant::themenliste / child::thema [position() = 2]`
- `descendant::themenliste / preceding-sibling::*`

XPath: Verkürzte Syntax

- Zur Verkürzung der Pfadausdrücke gelten folgende Abkürzungen:
 - `child::` kann weggelassen werden
 - `attribute::` kann als `@` geschrieben werden
 - `/descendant-or-self::node()` kann als `//` geschrieben werden
 - `self::node()` kann als `.` geschrieben werden
 - `parent::node()` kann als `..` geschrieben werden
 - `[position()=n]` kann als `[n]` geschrieben werden
- Beispiele:
 - `//titel`
 - `//@*`
 - `//@erstellt`
 - `//folie/themenliste/thema/text() [string-length() > 20]`
 - `//themenliste/thema[2]`

"Schleifen" in XSLT

```
<xsl:for-each select="XPathExpression">  
  Rumpf  
</xsl:for-each>
```

- Sequenz von Knoten wird berechnet
 - kann auch sortiert werden
- Rumpf wird einmal für jeden Knoten ausgeführt
- Günstig für Transformationen, die nur in einem Kontext benutzt werden
 - Spart Template-Definitionen
 - Macht Kontrollfluss transparenter

"Variablen" in XSLT

```
<xsl:variable name="VarName" select="XPathExpression">  
  Rumpf  
</xsl:variable>
```

- Rumpf wird einmal ausgewertet
- Ergebnis wird unter *VarName* für weitere Verwendung gespeichert
 - Eigentlich Konstantendeklaration
 - Entspricht "let" in funktionaler Programmierung
- Gültigkeit (innerhalb von Templates): Element, in dem Variable deklariert wird und dessen Nachfolger
- Zugriff: Innerhalb von Ausdrücken mittels *\$VarName*

Automatische Nummerierung

`<xsl:number/>`

- Bestimmt aktuelle Position des aktuellen Knotens innerhalb der aktuellen Knotensequenz
- Genauere Steuerung durch Attribute
 - z.B. Formatierung
 - z.B. hierarchische Numerierung

Beispiel: Kompakte Transformation (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="folien">
    <html>
      <head>
        <title>Transformationsdemo</title>
      </head>
      <body>
        <xsl:for-each select="folie">
          <xsl:variable name="fnr">
            <xsl:number/>
          </xsl:variable>
          <h2> Folie <xsl:value-of select="$fnr"/>:
            <xsl:value-of select="titel"/>
          </h2> ...
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Beispiel: Kompakte Transformation (2)

...

```
<ul>
  <xsl:for-each select="themenliste/thema">
    <li> Thema <xsl:value-of select="$fnr"/>
      .<xsl:number/>: <xsl:value-of select="."/>
    </li>
  </xsl:for-each>
</ul>
<p>
  <i>Foliename: <xsl:value-of select="@ident"/>,
  Erstellt am: <xsl:value-of select="@erstellt"/>
</i>
</p>
</xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```


Transformationsergebnis

Folie 1: Attribute in XML

- Thema 1.1: Deklaration in DTD
- Thema 1.2: Verwendung in XML-Dokument

Foliename: f1, Erstellt am: 04.06.2003

Folie 2: Identifikatoren

- Thema 2.1: Eindeutigkeit

Foliename: f2, Erstellt am: 03.06.2003