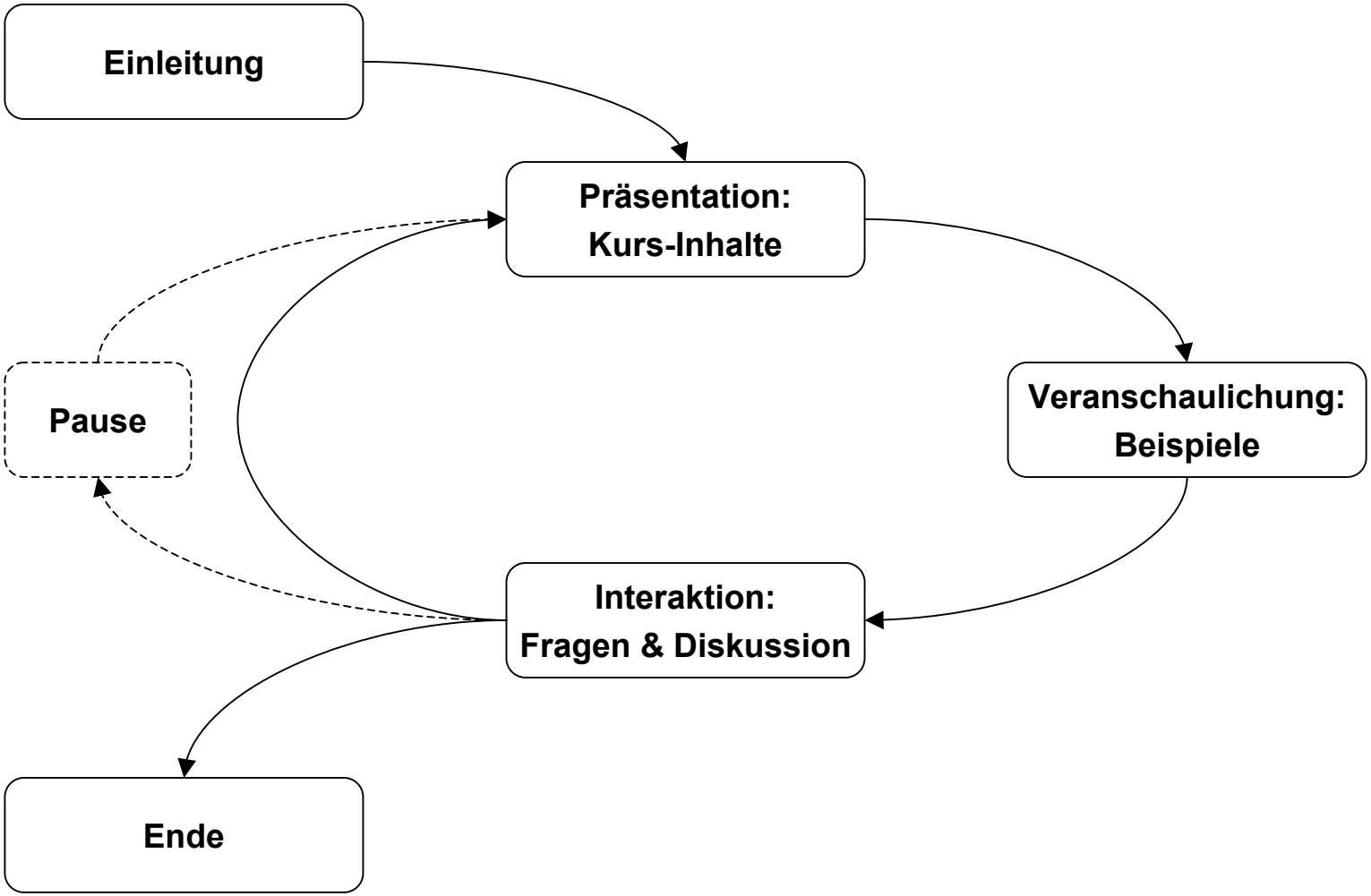
Einführung in XML und XSLT

Oder:

„Eine Guided-Tour von XML und DTD nach XSLT und XPath anhand von Beispielen.“

Ablauf des Kurses



Inhalt des Kurses

- **XML-Grundlagen**
  - Dokumente, Elemente, Attribute, Entitäten, Kommentare, CDATA, Wohlgeformtheit
  - Ausblick auf DOM und SAX
- **DTD-Grundlagen**
  - Elementdeklarationen, Attributdeklarationen
  - Modellierungsempfehlungen
- **XSLT**
  - XPath: Achsen, Filteranweisungen
  - XSL-Elemente: Templates, Kontrollstrukturen

Was dieser Kurs bietet

- **Grundlagen und Basiswissen für die selbständige Arbeit mit XML und XSLT:**
  - Erläuterungen zu den am häufigsten benötigten Teilaspekten von XML und XSLT
  - nur wenige kurze Hinweise auf Detailprobleme (...)
  - *„Stadtrundfahrt“* anstelle eines *„Stadtplans“*
- **Empfehlungen aus eigener Erfahrung** ich:
- **Beispiele, Beispiele, Beispiele**
- **...und was dieser Kurs nicht bieten kann:**
  - ein ausgearbeitetes Skript
  - ... und einen entspannten *„Spaziergang“*

„Was ist XML eigentlich?“

- **XML ist die Abkürzung für**
  - e**X**tensible **M**arkup **L**anguage
  - Übersetzt: erweiterbare Auszeichnungssprache
- **Meine Kurzbeschreibung für XML wäre:**
  - „Standard für textbasierte Markup-Sprachen“
- **XML selbst ist keine Markup-Sprache**
  - XML ist eine ***Meta-Sprache***:
    also eine Sprache, die andere Sprachen definiert

„Warum ist XML so toll?“

- **XML ist toll, weil XML ein Standard ist**
  - Der XML-Standard ist öffentlich zugänglich.
    - http://www.w3.org/TR/REC-xml
    - Die Nutzung kostet keine Lizenzgebühren.
  - Auf XML bauen andere Standards auf, diese vereinfachen die Arbeit mit XML z.B.:
    - DOM (Document Object Model)
    - SAX (Simple API for XML)
    - XSLT (kommt noch...)
  - in vielen Programmiersprachen gibt es Open-Source Implementierungen für den Zugriff auf XML-Daten
    - vermeidet Fehler – Eigenentwicklung nicht notwendig
    - lenkt den Schwerpunkt der Software-Entwicklung auf die Lösung der eigentlichen Aufgaben und Probleme

„Warum ist XML so toll?“

- **XML ist unabhängig von**
  - Hardware & Betriebssystemen (Plattformen)
  - Programmiersprachen
- **XML bietet Interoperabilität von Daten**
  - hierarchischen Daten können zwischen XML-fähigen Applikationen ausgetauscht werden
    - Vermeidung von evtl. verlustbehafteten Konvertierungen zwischen proprietären Datenformaten
  - XML bietet sich für den Informationsaustausch über das Internet an:
    - Web Services & SOAP, XML-RPC, XMLP, ...

„Warum ist XML so toll?“

- **XML-Dokumente sind Textdokumente**
  - einfach mit Text-Editor lesbar und erstellbar
- **XML basiert auf Unicode**
  - unterstützt Sonderzeichen und Idiogramme
  - auch für Daten nicht-westlicher Sprachen geeignet
- **Bestehende Systeme sind relativ einfach um XML-Datenausgaben erweiterbar**
  - erfordert lediglich die Ausgabe von Text

„Nicht ganz so toll an XML:“

- **Text als Datenformat ist nicht effizient:**
  - erfordert viel Speicherplatz
  - für Binärdaten nur schlecht geeignet
  - Einlesen von Daten erfordert Parse-Vorgang
- **Kodierung von Text ist eine Fehlerquelle**
  - Alle beteiligten Komponenten müssen die verwendeten Textkodierungen (text encoding) unterstützen und ggf. konvertieren können.

„Alles in allem...“

- **XML bietet für eine Vielzahl von Softwaresystemen eine geeignete Schnittstelle zur Verwaltung ihrer Daten.**
  - *„Wozu das Rad neu erfinden?“*

- **Anwendungsbeispiele:**
  - Im Büro: OpenOffice
  - Grafik: SVG, dia
  - Mathematik: MathML
  - E-Learning: LOM, Ilias, gimolus

XML-Fachbegriffe

- **Die wichtigsten XML-Fachbegriffe sind:**
  - Dokumente
  - Elemente
  - Attribute
  - Entitäten
  - Kommentare
  - CDATA-Abschnitte
  - Wohlgeformtheit (well-formedness)
  - Gültigkeit (validity)

  und werden im folgenden erläutert...

XML: Dokument

- **Ein XML-Dokument ist das äußerste logische Behältnis von XML-Daten.**
  - Im Allgemeinen eine Datei, Dokumente können aber auch nur im Speicher oder als Datenstrom vorliegen.
- **XML-Dokumente abstrahieren bereits von binärer Speicherform und Textkodierung:**
  - XML-Dokumente enthalten **nur** Unicode-Text
  - Zugriff auf die tatsächlich verwendete Repräsentation des Textes (auf Festplatte oder im Speicher) ist nicht möglich.

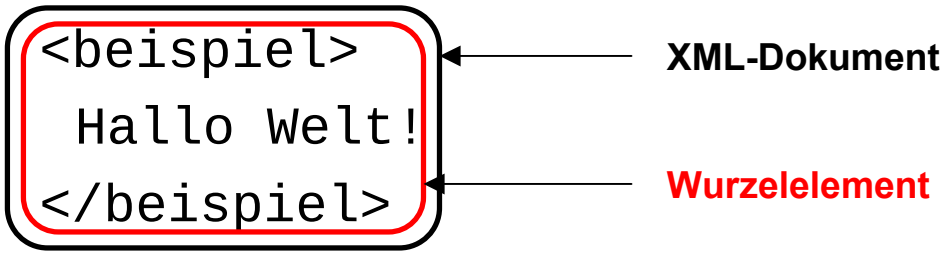
XML: Dokument

- **XML-Dokumente können enthalten:**
  - eine XML-Deklaration zu Beginn
  - Dokumenttypdeklarationen (Verweis auf DTDs)
  - Verarbeitungsanweisungen (processing instructions)
  - Kommentare
  - Whitespace (Leerzeichen, Tabulator, Zeilenumbruch)

- **Ein XML-Dokument enthält genau ein XML-Element, das Wurzelement (document root element).**
- **XML-Dokumente sind wohlgeformt (siehe später bei well-formedness – „Wohlgeformtheit“)**
- **... sonst ist es kein XML-Dokument.**

XML: Dokument

- **Beispiel DOK-1 – Hello World:**



  - ist bereits ein sehr einfaches XML-Dokument
  - enthält nur das Wurzelelement

XML: Dokument

- **Beispiel DOK-2 – mit XML-Deklaration:**

```
<?xml version="1.0" encoding="UTF-8"?>
<beispiel>
 Hallo Welt!
</beispiel>
```

  - XML-Deklaration kann von XML-Parsern genutzt werden, um die Textkodierung zu ermitteln.
    - Standardkodierung ohne XML-Deklaration ist UTF-8
  - XML-Version soll die Erweiterbarkeit des XML-Standards und die Kompatibilität von XML-Dokumenten sicherstellen.
    - zur Zeit ist nur Version „1.0“ verwendbar.

XML: Element

- **Elemente sind logische Einheiten innerhalb eines XML-Dokuments.**
- **Elemente bestehen aus:**
  - start-tag: <*Elementname*> (z.B. `<beispiel>`)
  - Elementinhalt (z.B. `Hallo Welt!`)
    - Text, andere Elemente oder beides
    - kann auch leer sein
  - end-tag </*Elementname*> (z.B. `</beispiel>`)
- **Elemente ohne Elementinhalt können auch abgekürzt geschrieben werden:**
  - anstelle von: `<beispiel></beispiel>`
    ein empty-tag: `<beispiel/>`

XML: Element

- **Elementnamen dürfen zusammengesetzt werden aus:**
  - Buchstaben „a“ ... „z“; „A“ ... „Z“
  - Ziffern „0“ ... „9“ (nicht am Anfang des Namens)
  - Idiogrammen „ä“, „ö“, „Ü“, „ß“, „チ“, „ぁ“, „Б“ etc.
  - Unterstrich „_“
  - Bindestrich „-“ (nicht am Anfang des Namens)
  - Punkt „.“ (nicht am Anfang des Namens)
- **Elementnamen dürfen nicht mit „xml“ (in allen groß/klein Varianten) beginnen.**

XML: Element

- **Ein Statement als Softwaretechniker:** 

  - Für die Vergabe von Elementnamen innerhalb eines Projekts sollte eine Namensrichtlinie vorgegeben werden.
  - Dadurch gewöhnt man sich im Laufe des Projekts schneller an die verwendeten Elementnamen.
  - Beispielsweise:
    - sprechende Elementnamen wählen
    - US-englische Worte benutzen
    - mehrere Worte durch Unterstrich trennen
  - Die Namensrichtlinie sollte sich an den (hoffentlich eingesetzten) Programmierrichtlinien für anderweitigen Source-Code im Projekt orientieren.

XML: Element

- **Beispiel EL-1 – leeres Element:**

```
<emptyness/>
```

Element

- **Beispiel EL-2 – Element mit Textinhalt:**

```
<text_inside>I am the content.</text_inside>
```

start-tag | Elementinhalt | end-tag

- **Beispiel EL-3 – Element mit Elementinhalt:**

```
<parent>
  <child>The child's content</child>
  <another_this_time_empty_child/>
</parent>
```

XML: Element

- **Beispiel EL-4 – gemischter Elementinhalt:**
  - „geht auch, ist in der Datenhaltung aber oftmals Zeichen nicht ausreichender Strukturierung“
  - Daher nun ein Beispiel mit XHTML:


```
<p>
 <img src=“green_unicode_characters.png“/>
 Neo: What is the Matrix?
 <br/>
 Phone: Follow the white rabbit.
 <br/>
</p>
```

XML: Attribut

- **Information zu einem Element kann in Attributen abgelegt werden.**
- **Attribute werden dem start-tag (bzw. empty-tag) hinzugefügt. Syntax:**
  - <*Elementname Attributname* **=** “*Attributwert*“>
  - Es können anstelle von (“) auch einfache Anführungszeichen (‘) als Begrenzerzeichen verwendet werden.
  - Whitespaces zwischen Bestandteilen sind erlaubt
  - Attributnamen sind wie Elementnamen zu bilden
  - Attributwerte dürfen kein Begrenzerzeichen, „**&**“ oder „**<**„ enthalten

XML: Attribut

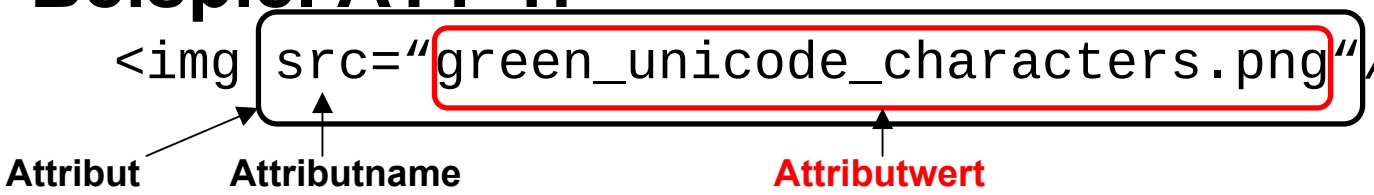
- **Eigenschaften von Attributen**
  - Attribute sind nicht geordnet (keine Reihenfolge)
  - ein Attribut darf in einem tag höchstens einmal vorkommen
- **Nutzung von Attributen**

  - Information in Attributen ist nicht zukunftssicher
    - keine weitere Strukturierung der Attributwerte möglich
    - Beschränkung der Kardinalität auf 0 oder 1
  - nur für Metainformation zu einem Element nutzen
  - die eigentliche Information in Elementinhalten unterbringen und hierarchisch strukturieren

XML: Attribut

- **Beispiel ATT-1:**

```
<img src="green_unicode_characters.png"/>
```



- **Beispiel ATT-2 – mit Whitespace-Zeichen:**
  - sorgt für bessere Übersichtlichkeit bei vielen Attributen bzw. langen Attributnamen/-werten

```
<tft_screen
 manufacturer = "Flat Screen's O-Rama Inc."
 horizontal_pixels = '1280'
 vertical_pixels   = '1024'
>
 <vga_connector type="D-Sub"/>
 <vga_connector type="DVI"/>
</tft_screen>
```

XML: Entität

- **Mit Entitätsreferenzen kann man sich auf woanders definierte Inhalte beziehen.**
  - überall in XML verwendbare Entitätsreferenzen:
    - < (anstelle von: **<**) und > (**>**)
    - " (**“**) und ' (**‘**)
    - & (**&**)
    - &#*Dezimalzahl*; oder &#x*Hexadezimalzahl*; beziehen sich auf Unicode-Zeichennummern
  - Man kann eigene Entitäten definieren und referenzieren...

    - gut für Konstanten, bzw. zum Einbinden von HTML-Entitäten
    - Flexibilisierung von DTDs möglich
    - auch externe Ressourcen (andere Dateien etc.) können als Entität definiert werden.

XML: Entität

- **Beispiel ENT-1 – das geht nicht:**

```
<formel>
  x<y   <-- Parser: „start-tag fehlerhaft!“
</formel>
```

  - Fehler während des Parse-Vorgangs, da ein start-tag eines Elements „y“ erwartet wird.

- **Beispiel ENT-2 – stattdessen:**

```
<formel>
  x&lt;y   <-- Entitätsreferenz
</formel>
```

XML: Kommentar

- **Kommentare dürfen (fast) überall zwischen anderen XML-Bestandteilen auftauchen.**
- **XML-Anwendungen dürfen Kommentare vollständig ignorieren**
  - Keine Anwendungsdaten in Kommentaren lagern!
- **Syntax:**
  - Kommentarbeginn: <!--
  - Kommentar: beliebige Zeichen außer „**--**“
  - Kommentarende: -->

XML: Kommentar

- **Beispiel KOM-1:**

```
<!-- monitor inventory list item -->
<tft_screen
 manufacturer = “Flat Screen‘s O-Rama Inc.“
 horizontal_pixels = ‘1280‘
 vertical_pixels   = ‘1024‘
>
 <!-- TODO: add DVI to D-Sub adapter -->
 <vga_connector type=“D-Sub“/>
 <vga_connector type=“DVI“/>
</tft_screen>
```

Kommentare

XML: CDATA-Abschnitt

- **CDATA-Abschnitte sind Bereiche, die beim Parse-Vorgang als Textbereich behandelt werden.**
  - Entitätsreferenzen werden nicht aufgelöst
  - „<“-Zeichen signalisieren darin kein start-tag bzw. empty-tag eines Elements
  - CDATA-Abschnitte können nicht geschachtelt werden.
- **Syntax:**
  - CDATA-Beginn: <![CDATA[
  - CDATA-Ende:]]>

XML: CDATA-Abschnitt

- **Beispiel CDA-1:**

```
<html_greeting>
 <![CDATA[<h1>
 Begr&uuml;&szlig;ung zum XML&XSLT-Kurs
 </h1>]]>
</html_greeting>
```

  - Das Element `html_greeting` enthält **nur** Text.
    - Der CDATA-Text enthält keinerlei XML-Struktur und damit keine Elementhierarchie. Ein Zugriff auf das `h1`-Element ist nicht möglich.
    - Der CDATA-Text verwendet keine Entitätsreferenzen, d.h. ü und ß werden nicht durch Entitäten ersetzt.
    - &XSLT wäre eine fehlerhafte Entitätsreferenz

XML: well-formedness

- **Wohlgeformtheit bedeutet:**
  - Ein Text entspricht syntaktisch den Regeln für ein XML-Dokument
  - **und** auch alle referenzierten Entitäten sind wohlgeformt.
- **Einige wichtige Wohlgeformtheitsregeln:**
  - Zu jedem start-tag muss ein end-tag existieren.
  - Elemente (start-/end-tags) überlappen sich nicht.
  - Es existiert genau ein Wurzelelement.
  - Ein Element darf nicht mehrere Attribute mit demselben Attributnamen besitzen.

XML: Ausblicke

- **DOM – Document Object Model**
  - ist ein Modell für den Zugriff auf XML-Dokumente aus einer Programmiersprache heraus
  - Jedem XML-Bestandteil (Element, Text, etc.) wird eine Klasse zugeordnet.
  - Objekte dieser Klassen können durch einen XML-Parser aus einer Textdatei erzeugt werden.
  - DOM-Objekte stellen Methoden zur Inspektion und Manipulation von Struktur und Inhalt zur Verfügung.
  - DOM bietet sich für Anwendungen an, die den zugleich auf unterschiedliche Bereiche eines XML-Dokuments zugreifen müssen.

XML: Ausblicke

- **SAX – Simple API for XML**
  - Ist eine Ereignis-basiertes (event-based) Modell, das es ermöglicht, den Parse-Vorgang an die eigenen Bedürfnisse anzupassen.
  - Unterstützung für selbst definierte Entitäten, processing instructions, etc. können damit realisiert werden.
  - SAX ermöglicht es im Gegensatz zu DOM auch nur Teile eines XML-Dokuments bearbeiten zu lassen.
  - SAX bietet sich für Anwendungen an, denen enge Speicher- oder Performancegrenzen gesetzt sind.

XML: Ausblicke

- **Namensräume (namespaces)**
  - gruppieren Elemente einer Anwendung und machen sie unterscheidbar von gleichnamigen Elementen anderer Anwendungen
    - z.B: <person:titel xmlns:person="de:unistuttgart:myapp">
      Prof. Dr. rer. nat.
      </person:titel>

      und

      <buch:titel xmlns:buch="http://iso.org/isbn/author.dtd">
      The Lord of the Rings
      </buch:titel>
  - wird bei XSLT benötigt, um XSLT-Elemente verwenden zu können

XML: noch Fragen...?

- **Unklarheiten oder Fragen zu diesem Kapitel „XML-Fachbegriffe“?**
  - Jetzt melden!
- **10 Minuten Pause**
  - Danach geht es weiter mit dem Kapitel „DTD“.

„Aha, und wie kann ich XML einsetzen?“

- **Der Einsatz von XML für ein konkretes Projekt erfordert die Spezifikation einer projekteigenen XML-Sprache.**
  - diese Sprache bildet dann die Schnittstelle
    - zwischen verschiedenen Software-Komponenten (Client & Server, Browser & Web-Applikation etc.)
    - für zukünftige Erweiterungen
    - zu Fremd-Software Anbietern
    - für den Import von Daten aus bestehenden Systemen
- **Maßgeschneiderte XML-Sprachen können durch DTDs spezifiziert werden.**

„Was sind DTDs?“

- **DTD ist die Abkürzung für**
  - **D**ocument **T**ype **D**efinition
- **DTDs spezifizieren eine XML-Sprache**
- **Für Dokumente in dieser Sprache wird festgelegt:**
  - welche Elemente vorkommen dürfen
  - der Elementinhalt eines Elements
  - Attribute eines Elements
- **Nicht festgelegt wird:**
  - welches Element das Wurzelelement ist

„Was sind DTDs?“

- **Eine DTD wird i.A. in einer Textdatei abgelegt, die dann von XML-Dokumenten referenziert wird.**
  - Es ist auch möglich, DTD-Deklarationen in ein XML-Dokument einzubetten.
- **DTDs können modular aufgebaut werden**
  - Einsatz von Parameter-Entitäten
  - Referenzierung von Ressourcen (andere DTDs) innerhalb von integrierten Software-Anwendungen kann sich als sehr schwierig herausstellen!

DTD: Elementdeklaration

- **Syntax einer Elementdeklaration**
  - <!ELEMENT *Elementname Elementinhalt*>
- ***Elementinhalt* ist entweder:**
  - EMPTY „das Element muss leer sein“
  - ANY „Text oder deklarierte Elemente“
  - regulärer Ausdruck **in runden Klammern**:
    - *Elementname* „Ausdruck → ein Element mit diesem Namen“
    - #PCDATA „Ausdruck → zu parsender Text“
    - , „Sequenz von Ausdrücken“
    - | „entweder vorheriger oder nachfolgender Ausdruck“
    - ? „vorheriger Ausdruck ist optional“
    - + „vorheriger Ausdruck muss 1..n mal auftreten“
    - * „vorheriger Ausdruck muss 0..n mal auftreten“
    - (oder) „zur Zusammenfassung von Ausdrücken“

DTD: Elementdeklaration

- **Beispiel DTD-E1 – kein Elementinhalt:**

  ```
  <!ELEMENT br EMPTY>
  ```

  – br-Elemente sind nun deklariert und dürfen keine anderen Elemente oder Text enthalten.

- **Beispiel DTD-E2 – Text als Inhalt:**

  ```
  <!ELEMENT paragraph (#PCDATA)>
  ```

  – paragraph-Elemente dürfen Text enthalten.

  – Der Text im Elementinhalt kann auch leer sein.

  – Entitätsreferenzen im Text (außer in CDATA-Abschnitten) werden aufgelöst.

DTD: Elementdeklaration

- **Beispiel DTD-E3 – Sequenz:**

```
<!ELEMENT paragraph (title, text, notes)>
```

  - paragraph-Elemente müssen erst ein title-Element, dann ein text-Element und schließlich ein notes-Element enthalten.

- **Beispiel DTD-E4 – Alternative:**

```
<!ELEMENT paragraph (title, image?, text)>
```

  - wahlweise kann man zwischen title und text auch ein image-Element einfügen
  - Dieselbe Bedeutung hat:

```
<!ELEMENT paragraph
 (title, text | (image, text))
>
```

DTD: Elementdeklaration

- **Beispiel DTD-E5 – Wiederholungen:**

  ```
  <!ELEMENT paragraph (title, text+)>
  ```

  – paragraph-Elemente müssen erst ein title-Element und dann ein oder mehr text-Elemente enthalten.

  – Die Deklaration kann auch auf den *-Operator umformuliert werden. Entweder so:

  ```
  <!ELEMENT paragraph (title, text, text*)>
  ```

  – oder so:

  ```
  <!ELEMENT paragraph (title, text*, text)>
  ```

  – Generell sollte man Deklarationen zur besseren Verständlichkeit so kurz wie möglich formulieren.

DTD: Elementdeklaration

- **Beispiel DTD-E6 – Kombination:**

```
<!ELEMENT scientific_paper
 (title, paragraph+)
>
<!ELEMENT paragraph
 (title, text,
 ((image | table | text)*, text)? )
>
<!ELEMENT title   (#PCDATA)>
<!ELEMENT text    (#PCDATA)>
<!ELEMENT image   (caption?, counter?)>
<!ELEMENT caption (#PCDATA)>
<!ELEMENT counter (#PCDATA)>
<!ELEMENT table   (row+)>
<!ELEMENT row     (cell*)>
```

DTD: Attributdeklaration

- **Syntax einer Attributdeklaration**
  - <!ATTLIST *Elementname*
    (*Attributname Attributtyp Standardwert*)*>
  - *Attributtyp*:
    - CDATA „beliebiger Attributwerttext“
    - (*Attributwert*(| *Attributwert*)*) „selbst definierte Wertemenge“
    - *Token-Typen* – werden hier nicht behandelt (...)
  - Standardwert
    - “Attributwert“ „falls Attribut nicht angegeben, gilt dieser Wert“
    - #IMPLIED „Attribut ist optional“
    - #REQUIRED „Attribut muss angegeben werden“
    - #FIXED *Wert* „wenn Attribut angegeben ist, muss es den folgenden Wert haben – hier nicht behandelt“ (...)

DTD: Attributdeklaration

- **Beispiel DTD-A1 – ein Text-Attribut:**

```
<!ATTLIST image src CDATA #REQUIRED>
```

  - image-Elemente müssen ein src-Attribut erhalten
  - Der Attributwert des src-Attributs kann (im Rahmen von Attributwerten) beliebiger Text sein.

- **Beispiel DTD-A2 – mehrere Attribute:**

```
<!ATTLIST image
 src         CDATA #REQUIRED
 border      (thin | medium | thick) #IMPLIED
 transparent (true | false) “false“
>
```

  - border- und transparent-Attribute sind optional
  - Falls kein transparent-Attribut angegeben wird, **soll** dies Attributwert = “false“ bedeuten.

DTD: Modellierung

- **Bei der Analyse der Datenstrukturen sehr sorgfältig vorgehen.**
  - Änderungen einer DTD im laufenden Betrieb können sehr zeitaufwändig (und Software-seitig besonders fehlerträchtig) sein.
- **Strukturen möglichst fein gliedern.**
  - erleichtert spätere Erweiterung (& XSLT-Handling)
  - Anstelle von:
    ```
    <!ELEMENT inventory (person+, item*)>
    ```
  - Vielleicht:
    ```
    <!ELEMENT inventory (customers, property)>
    <!ELEMENT customers (person+)>
    <!ELEMENT property  (item*)>
    ```

DTD: Modellierung

- **Standardwerte von Attributen mit Vorsicht genießen.**
  - Standardwerte werden durch den Parser **nicht** in ein Attribut mit entsprechendem Attributwert umgesetzt.
    - Standardwertbehandlung muss durchgängig in allen Software-Komponenten gehandhabt werden.
    - Änderung des Standardwerts in der DTD hat eine inhaltliche Veränderung aller gültigen XML-zur Folge!
- **Die Reihenfolge der Deklarationen ist nicht vorgeschrieben.**
  - Daher: Eine möglichst leicht lesbare Abfolge von Deklarationen anstreben.

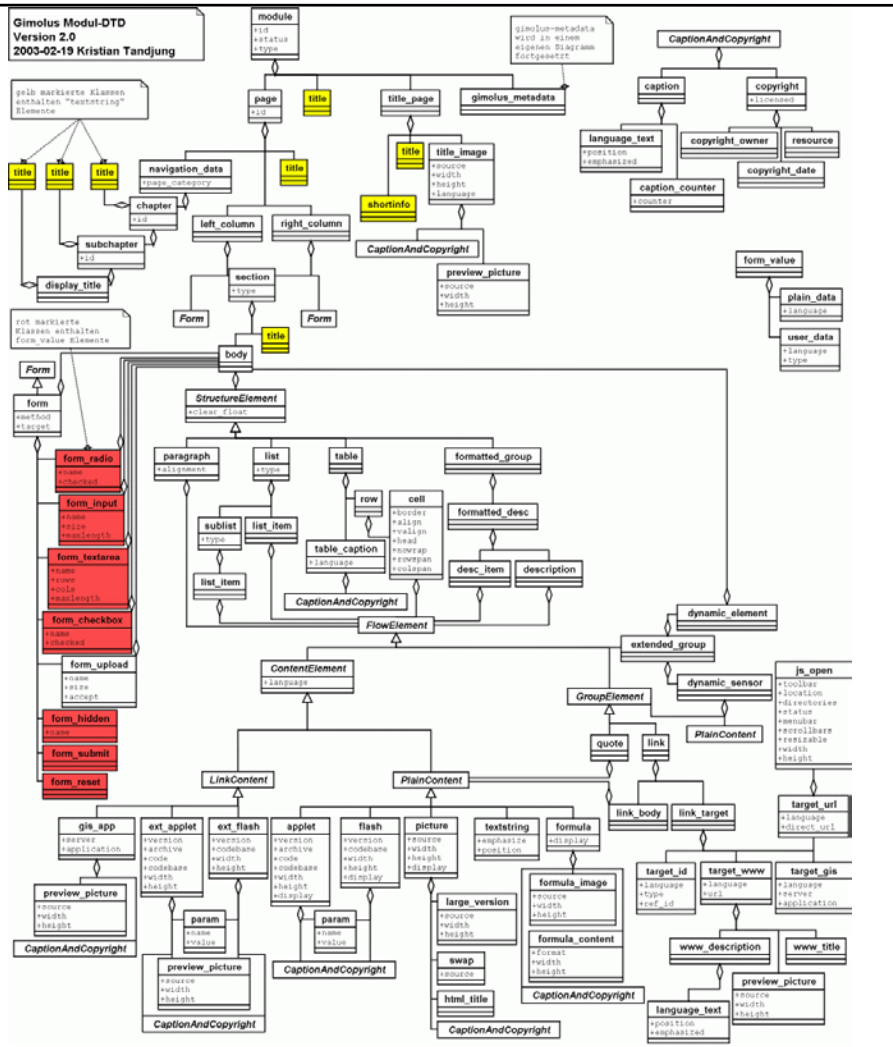
DTD: Modellierung

- **DTD sehr ausführlich Kommentieren.**
  - Da die DTD eine wesentliche Schnittstelle darstellt, wird sie von vielen Beteiligten verwendet.
  - Die DTD ist – anders als die begleitende Dokumentation – bei allen Beteiligten vorhanden.
  - Missverständnisse zur Bedeutung und Nutzung von Elementen und Attributen minimieren.
- **Bei der Erstellung graphische Hilfsmittel verwenden.**
  - Beziehungen der Deklarationen untereinander sind aus dem DTD-Text nur mühsam ermittelbar.
  - beispielsweise UML-Klassendiagramme nutzen

DTD: Modellierung

ich:

- **Ein Beispiel dafür, wie eine DTD im Laufe der Entwicklung „wachsen“ kann:**
  - ...und das ist noch nicht die **ganze** DTD.
  - Mit Hilfe des Diagramms noch handhabbar.



DTD: DOCTYPE

- **Ein XML-Dokument kann angeben, zu welcher DTD (und damit XML-Sprache) es gehört (bzw. gehören will).**
- **„Document Type Declaration“ hinzufügen**
  - Syntax:
    - <!DOCTYPE
    - *Wurzelelementname*
    - SYSTEM (es gibt auch PUBLIC)
    - *DTD-URI*
    - >
  - Die *DTD-URI* lautet bei einer DTD-Datei zum Beispiel: “file://article_exchange.dtd“

DTD: DOCTYPE

- **Beispiel DTD-D1 – DOCTYPE:**

```
<?xml version="1.0" standalone="no">
<!DOCTYPE scientific_paper SYSTEM
 "file://scientific_exchange.dtd">
<scientific_paper>
 <title>Einsatz von XML</title>
 <paragraph>
       <title>Geschichte von XML</title>
       <text>XML ist ein W3C-Standard</text>
       <image src="w3c.png"/>
       <text>SGML, HTML, Internet...</text>
       <!– und so weiter und so fort -->
 </paragraph>
</scientific_paper>
```

DTD: validity

- **Gültigkeit (validity) eines XML-Dokuments bezieht sich immer auf die referenzierte DTD in der „Document Type Declaration“.**

- **XML-Dokument ist gültig, wenn:**
  - Alle vorkommenden Elemente in der DTD deklariert wurden und die Elemente der Deklaration entsprechen.
  - Das Beispiel DTD-D1 ist gültig, wenn sich die Deklarationen aus Beispiel DTD-E6 in der referenzierten DTD-Datei befinden.

DTD: noch Fragen...?

- **Unklarheiten oder Fragen zu diesem Kapitel „DTDs“?**
  - ...oder im Zusammenhang mit dem vorherigen Kapitel „XML-Grundlagen“?
  - Jetzt melden!
- **Weiter geht es mit dem Kapitel „XSLT“...**

„Warum XSLT?“

- **XML ist ein verbreiteter Standard, um Daten zu speichern oder auszutauschen.**
- **Datenkonvertierung (wenn auch „nur“) zwischen verschiedenen XML-Sprachen ist weiterhin notwendig.**
- **Auf XML-Daten kann eine Vielzahl unterschiedlicher Sichten erwünscht sein:**
  - (X)HTML, WAP: zur Darstellung im Browser
  - SQL: zur Manipulation von Datenbankinhalten
  - Text: zusammenfassender Eintrag in Logfile

„Warum XSLT?“

- **Warum soll für jede neue Textkonvertierung von XML-Daten ein neues Konvertierungsprogramm implementiert werden?**
  - Sinnvoller ist es, stattdessen neue **Regeln** für die Konvertierung zu definieren.
  - Diese Regeln sind selbst auch Daten und sollten mit XML verwaltet werden (→ Stylesheet).

- **Genau das bietet XSLT.**

„Was ist XSLT?“

- **XSLT ist die Abkürzung für**
  - e**X**tensible **S**tylesheet **L**anguage: **T**ransformations
    - Übersetzt in etwa:
      erweiterbare Gestaltungsvorlagensprache: Transformationen
  - XSLT ist eine XML-Sprache
    - Elemente und Attribute besitzen eine genau definierte Bedeutung für die Konvertierung
- **XSLT-Prozessor-Implementierungen**
  - Müssen in der Lage sein:
    - ein beliebiges XML-Eingabedokument
    - anhand in XSLT spezifizierter Regeln (Stylesheet)
    - in Text oder in ein XML-Ausgabedokument zu transformieren.

XSLT: XML-Strukturmodell

- **XSLT betrachtet XML-Strukturen (genauso wie DOM) als Baum**
  - das XML-Dokument ist Wurzel (root) des Baums
  - Elemente und Attribute sind Knoten
  - zusammenhängender Text bildet einen Knoten
  - Knoten im Elementinhalt sind Kinder des Elementknotens
  - Attributknoten sind keine Kinder des Elementknotens
  - Es gibt noch weitere Knotentypen. Diese werden hier nicht behandelt...

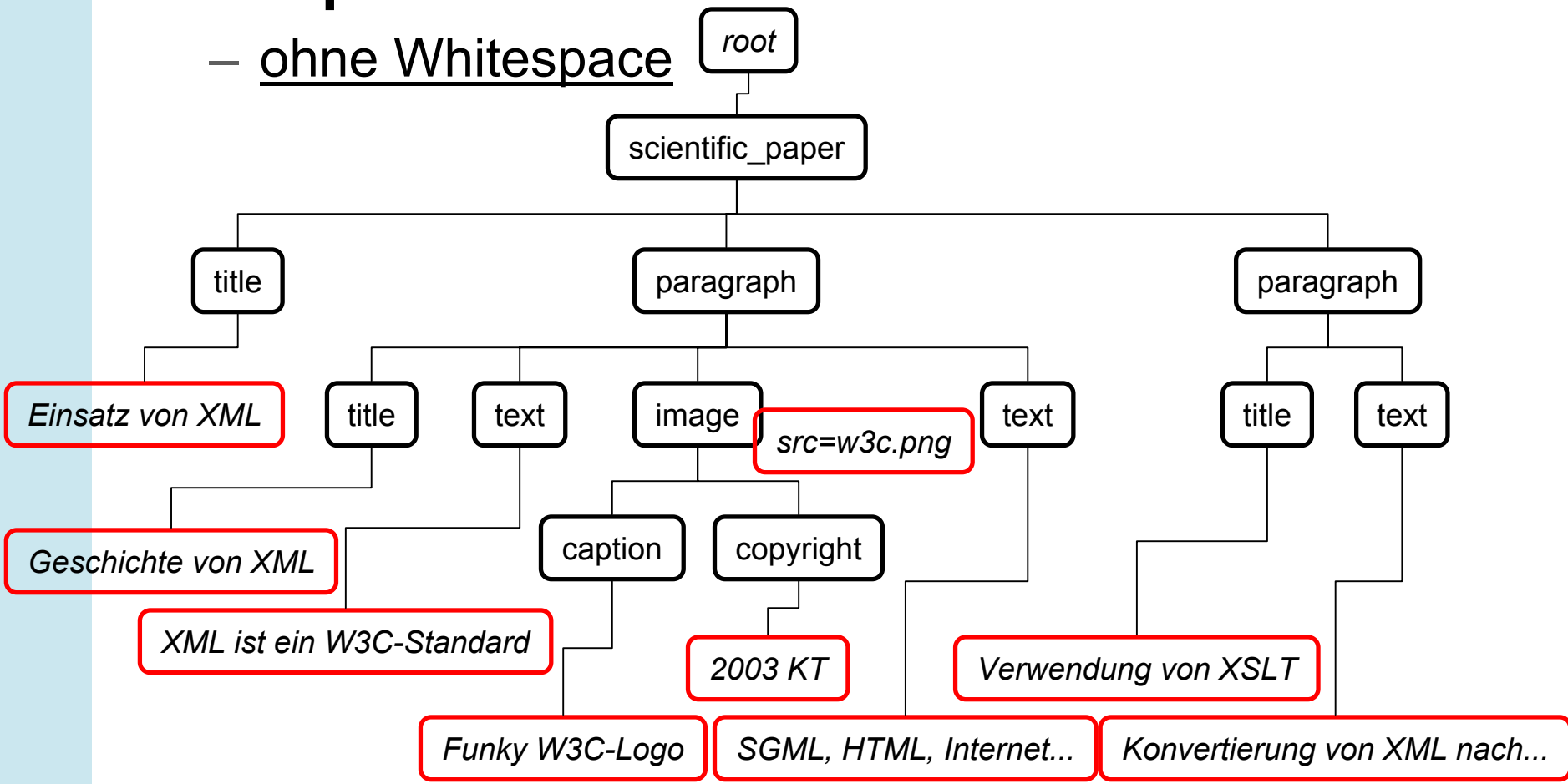
XSLT: Beispieldokument

- **Das folgende XML-Dokument wird für alle weiteren XSLT-Beispiele verwendet:**

```
<scientific_paper>
 <title>Einsatz von XML</title>
 <paragraph>
        <title>Geschichte von XML</title>
        <text>XML ist ein W3C-Standard</text>
        <image src="w3c.png">
               <caption>funky W3C-Logo</caption>
               <copyright>2003 KT</copyright>
        </image>
        <text>SGML, HTML, Internet...</text>
 </paragraph>
 <paragraph>
        <title>Verwendung von XSLT</title>
        <text>Konvertierung von XML nach...</text>
 </paragraph>
</scientific_paper>
```

XSLT: Baumstruktur

- **Beispiel XSLT-1 – Baumstruktur:**
  - ohne Whitespace



XSLT & Xpath

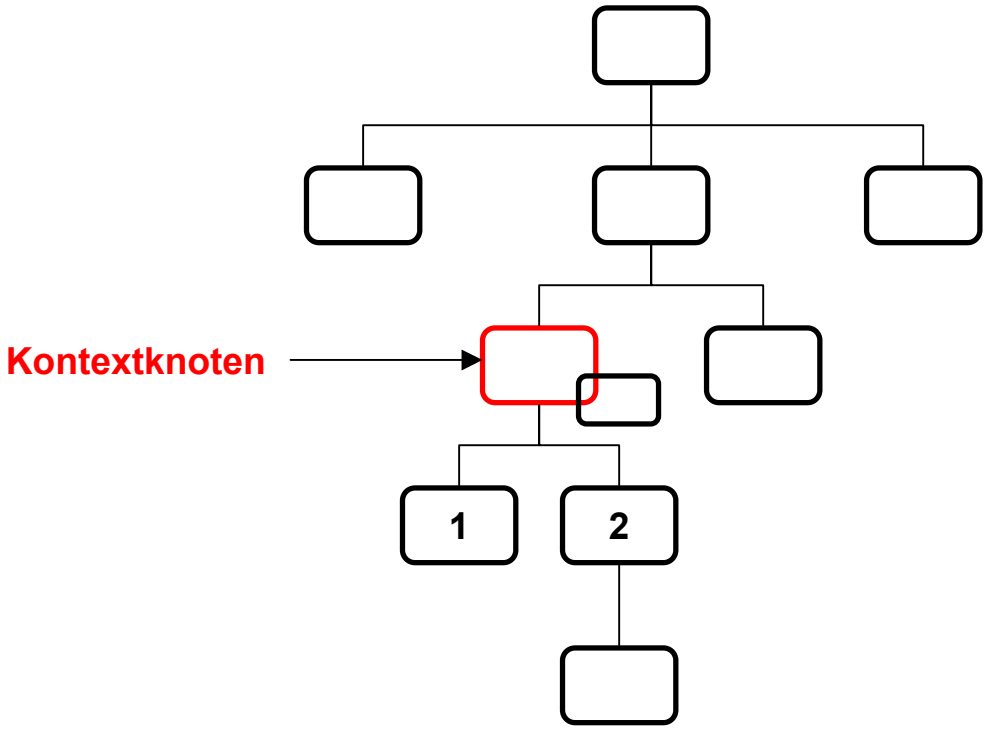
- **XPath ist keine XML-Sprache**
- **XPath ist ein eigenständiger Standard**
- **XPath ist eine Ausdruckssprache, mit der**
  - String-Manipulationen,
  - arithmetische Berechnungen oder
  - Bool‘sche Ausdrücke ausgewertet werden können.
- **Aber vor allem ist XPath zur Selektion von Knoten geeignet.**
  - XSLT greift auf XPath zurück, um die nächste zu bearbeitende Knotenmenge auszuwählen.

XPath: Achsen

- **Relativ zu einem Knoten (Kontextknoten) kann der XML-Baum auf verschiedene Weisen traversiert werden.**
  - XPath stellt sogenannte Achsen (axis) dafür zur Verfügung.
  - Achsen werden mit *Achsenname*:: angegeben
  - Einige wichtige XPath-Achsen werden im folgenden erklärt:
    - child
    - parent
    - attribute
    - self

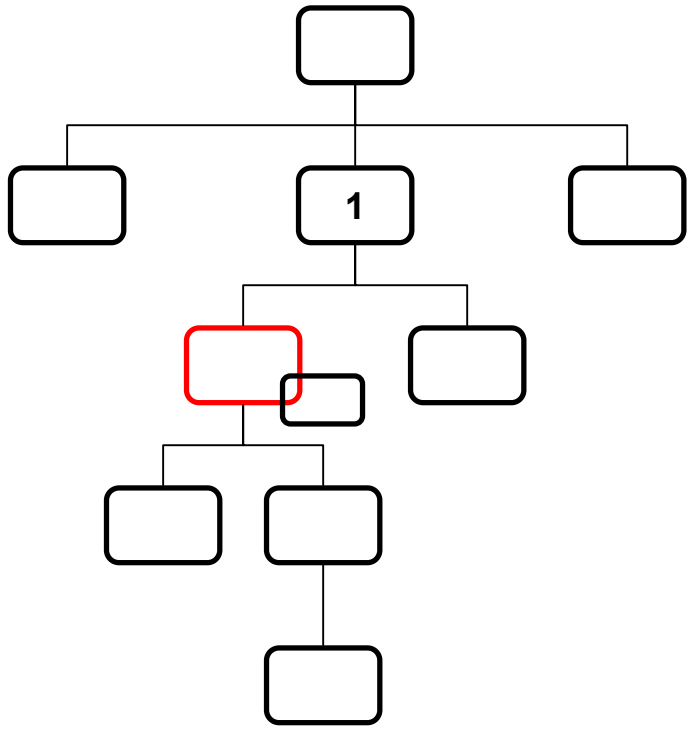
XPath: child

- **Beispiel XPath-1 – child axis:**
  - Abkürzung: „“ (nichts)



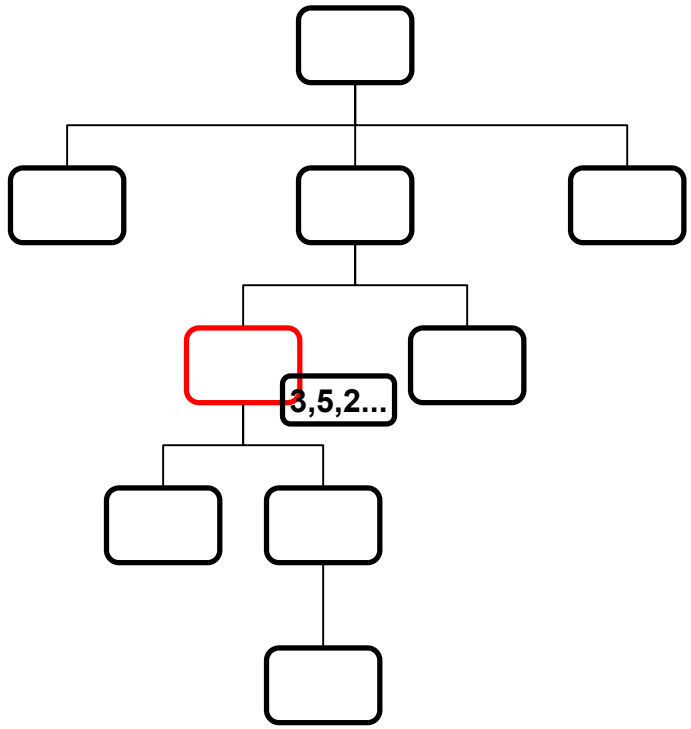
XPath: parent

- **Beispiel XPath-2 – parent axis:**
  - Abkürzung: „..“



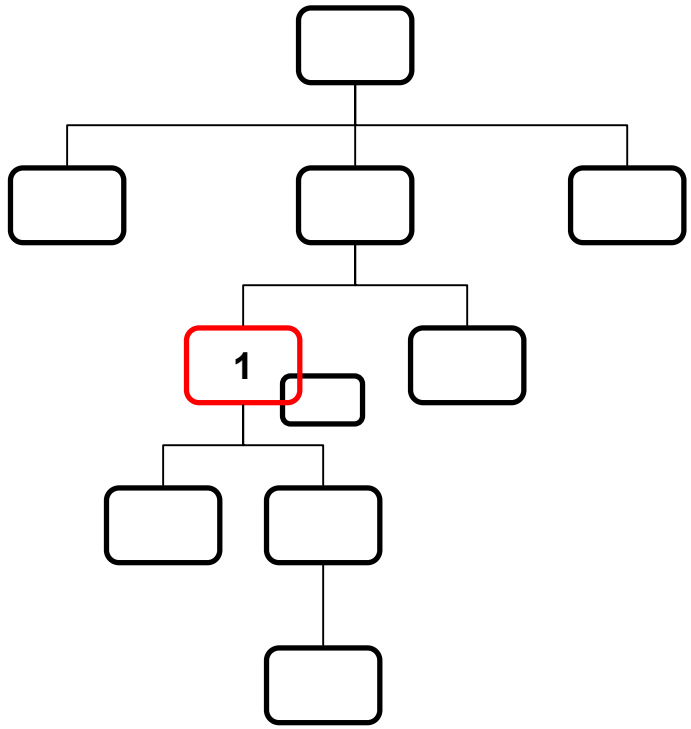
XPath: attribute

- **Beispiel XPath-3 – attribute axis:**
  - Abkürzung: „@“



XPath: self

- **Beispiel XPath-4 – self axis:**
  - Abkürzung: „.“



XPath: location path

- **Um eine Knotenmenge auszuwählen bedarf es eines „location path“**
  - Dieser besteht aus:
    - *XPath-Achse*,
    - *Knotenmenge* und wahlweise
    - *Filter-Ausdrücken*
  - Auswahl der *Knotenmenge* erfolgt u.a. durch (...)
    - *Elementname* – Elementknoten mit gleichem Namen
    - „*“ – alle Elementknoten
  - *Filter-Ausdrücke* werden innerhalb eckiger Klammern angefügt (...)
    - [*Zahl*] – wähle nur den *Zahl*-ten Knoten aus

XPath: location path

- **Beispiele XPath-5 – location path:**
  - „.“
    - wählt den Kontextknoten aus
  - „@source“
    - wählt den Attributknoten des Kontextknotens aus, das den Attributnamen „source“
  - „paragraph[2]/text“
    - wählt alle Kind-Elementknoten mit Namen „text“ des zweiten Kind-Elementknotens mit Namen „paragraph“ aus
  - „/scientific_paper/*“
    - wählt alle Kind-Elementknoten des Wurzelelements aus

XSLT: Stylesheet

- **Jedes Stylesheet muss dem XSLT-Namespace zugeordnet sein.**
  - In der Praxis: Im start-tag des Wurzelelements wird ein entsprechendes Präfix (meist „xsl“) angegeben:
    - `<xsl:stylesheet version="1.0"`
      `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
      `>`
- **Ein Stylesheet enthält zur Steuerung der Transformation mehrere „templates“.**
  - „templates“ enthalten Anweisungen, die Ausgaben erzeugen können oder die Verarbeitung an andere „templates“ weitergeben.

XSLT: Stylesheet

- **Eingebautes Verhalten in einem XSLT-Prozessor:**
  - tritt in kraft, wenn kein template in einem Stylesheet zur Bearbeitung des Kontextknotens herangezogen werden kann
  - Elementknoten:
    - Bearbeitung mit allen Kindknoten fortsetzen
  - Textknoten:
    - Ausgabe des Textes

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-2 – Minimalismus:**
  - Keinerlei Templates im Stylesheet
  - Ausgabe:

```
Einsatz von XML

    Geschichte von XML
    XML ist ein W3C-Standard

          funky W3C-Logo
          2003 KT

    SGML, HTML, Internet...


    Verwendung von XSLT
    Konvertierung von XML nach...
```

  - Whitespaces in Elementen sind auch Textknoten

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-3 – Push-Processing:**
  -
```
<xsl:template match="/">
 <html><head/><body>
 <xsl:apply-templates select="scientific_paper"/>
 </body></html>
</xsl:template>

<xsl:template match="scientific_paper">
 <h1><xsl:apply-templates select="title"/></h1>
 <xsl:apply-templates select="paragraph"/>
</xsl:template>

<xsl:template match="paragraph">
 <xsl:apply-templates select="text"/>
</xsl:template>
```

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-3 – Push-Processing:**
  - Ausgabe:

```
<html>
<head/>
<body>
<h1>Einsatz von XML</h1>XML ist ein W3C-StandardSGML, HTML,
  Internet... Konvertierung von XML nach...</body>
</html>
```

  - Prozessor darf die Bedeutung-nicht-verändernde Whitsepaces einfügen
  - html und head Elemente können durch den Prozessor konfiguriert werden

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-4 – Pull-Processing:**

```
<xsl:template match="/">
  <html><head/><body>
  <xsl:apply-templates/>
  </body></html>
</xsl:template>
<xsl:template match="scientific_paper">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="paragraph">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="title">
  <h1><xsl:value-of select="."/></h1>
</xsl:template>
<xsl:template match="text">
  <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="image"/>
```

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-4 – Pull-Processing:**
  - Ausgabe:

```
<html><head/><body>

<h1>Einsatz von XML</h1>

<h1>Geschichte von XML</h1>
        XML ist ein W3C-Standard

        SGML, HTML, Internet...


        <h1>Verwendung von XSLT</h1>
        Konvertierung von XML nach...

</body></html>
```

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-5 – Attribute:**

```
<xsl:template match="image">
 <img>
       <xsl:attribute name="src">
             <xsl:value-of select="@source"/>
       </xsl:attribute>
       <xsl:attribute name="alt">
             <xsl:value-of select="caption"/>
       </xsl:attribute>
 </img>
</xsl:template>
```

  - (Teil-)Ausgabe:

```
<img alt="funky W3C-Logo" src="w3c.png"/>
```

XSLT-Beispiel: XML⇨HTML

- **Beispiel XSLT-7 – if:**

```
<xsl:template match="image">
 <img>
       <xsl:attribute name="src">
             <xsl:value-of select="@source"/>
       </xsl:attribute>
       <xsl:if test="not(@border)">
             <xsl:attribute name="class">
             borderless</xsl:attribute>
       </xsl:if>
 </img>
</xsl:template>
```

  - (Teil-)Ausgabe:

```
<img class="borderless" src="w3c.png"/>
```

XSLT: Ausblicke

- xsl:choose / xsl:when
- xsl:for-each
- xsl:param / xsl:with-param
- xsl:import / xsl:include
- xsl:sort
- xsl:element
- xsl:key

- Allerlei XPath-Funktionen...

Information zu XML / XSLT

- Standards bei W3C
  - XML: http://www.w3.org/TR/REC-xml
  - XSLT: http://www.w3.org/TR/xslt
- Bücher
  - Elliotte Rusty Harold, W. Scott Means,
    XML in a Nutshell,
    O'Reilly
  - Michael Kay,
    XSLT Programmer's Reference 2nd Edition,
    Wrox Press
- Apache Open-Source Software
  - XML-Parser: Apache Xerces
  - XSLT-Prozessor: Apache Xalan
  - http://xml.apache.org

„Letzte Chance: Fragen?“

- **Fragen zu übergreifenden Problemen von XML, XSLT, XPath?**
- **Interesse an genaueren Informationen zu Detailfragen?**
- **Zum Abschluss bitte noch die Rückmeldungsbögen ausfüllen.**
- **Dankeschön... Mahlzeit.**