XML

Webtechnologien

Themen

XML-Einführung

DTD

XML-Schema

XPath

XML & Stylesheet

XSLT

XML-Anwendungen

XML Beispiele

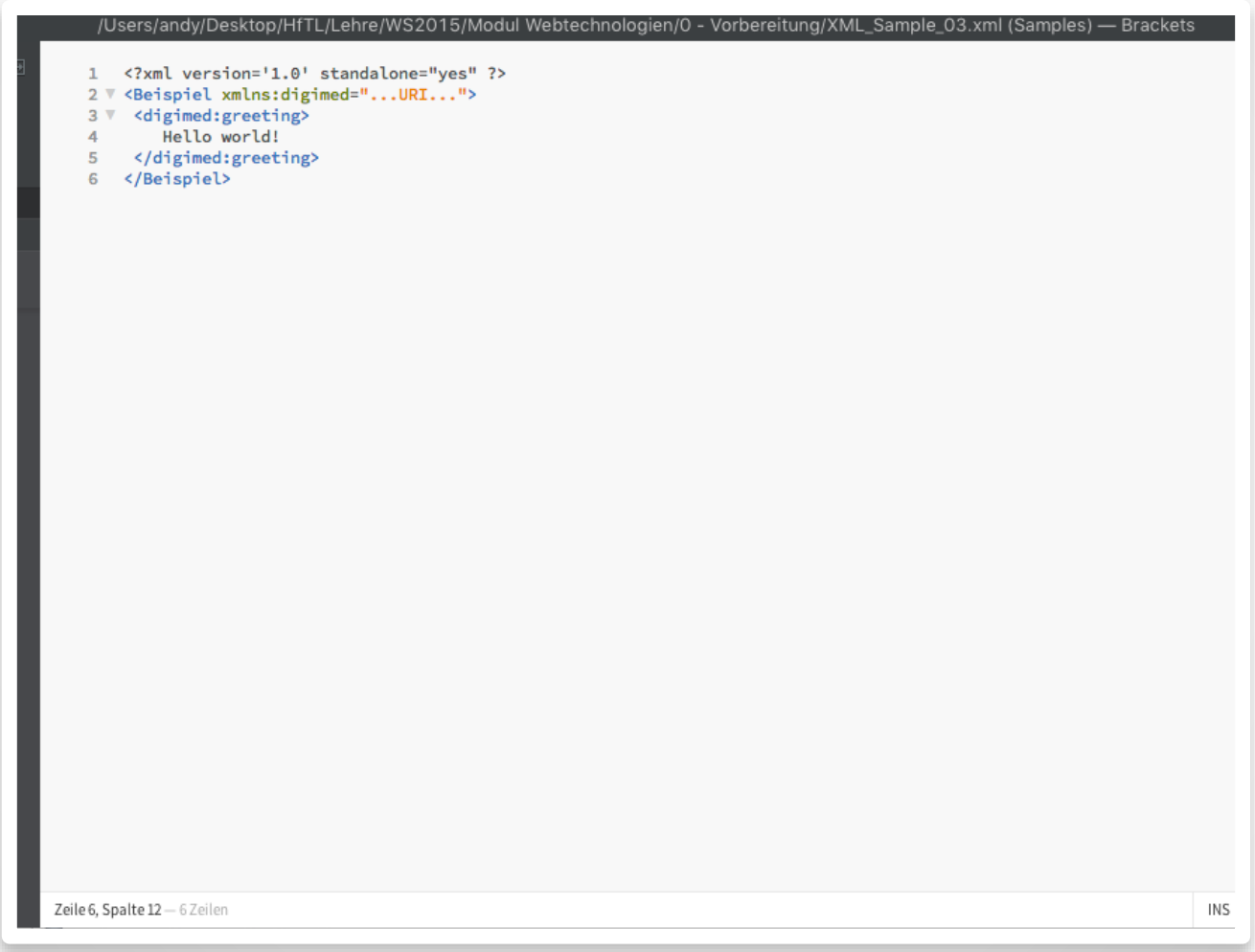
Quellen, Literatur

- Quellenangaben

  Der Inhalt richtet sich – soweit nicht anders gekennzeichnet – aus an:

  - [1] Einstieg in XML, Helmut Vonhoegen, Rheinwerk Verlag, Bonn, 2015 (http://d-nb.info/1070547034)
  - W3C
  - https://wiki.selfhtml.org
- Es handelt sich um eigene Abbildungen, wenn keine Quelle angegeben ist.
- Einige Inhalte sind an Wikipedia angelehnt. Als Quellangaben wurden die dort angegebenen Referenzen geprüft.

Unser 1. Beispiel



XML Sample 03

HTML vs. XML

HTML 4.01 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font). Framesets are not allowed.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

HTML 4.01 Transitional

This DTD contains all HTML elements and attributes, INCLUDING presentational and deprecated elements (like font). Framesets are not allowed.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

HTML 4.01 Frameset

This DTD is equal to HTML 4.01 Transitional, but allows the use of frameset content.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

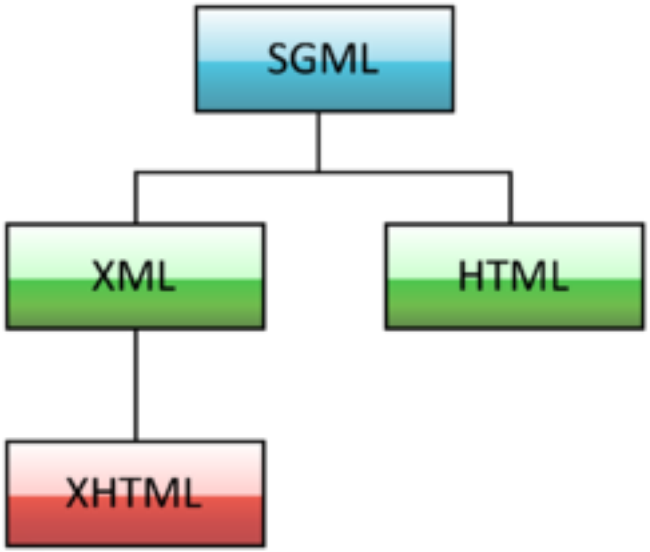
XHTML 1.0 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font). Framesets are not allowed. The markup must also be written as well-formed XML.

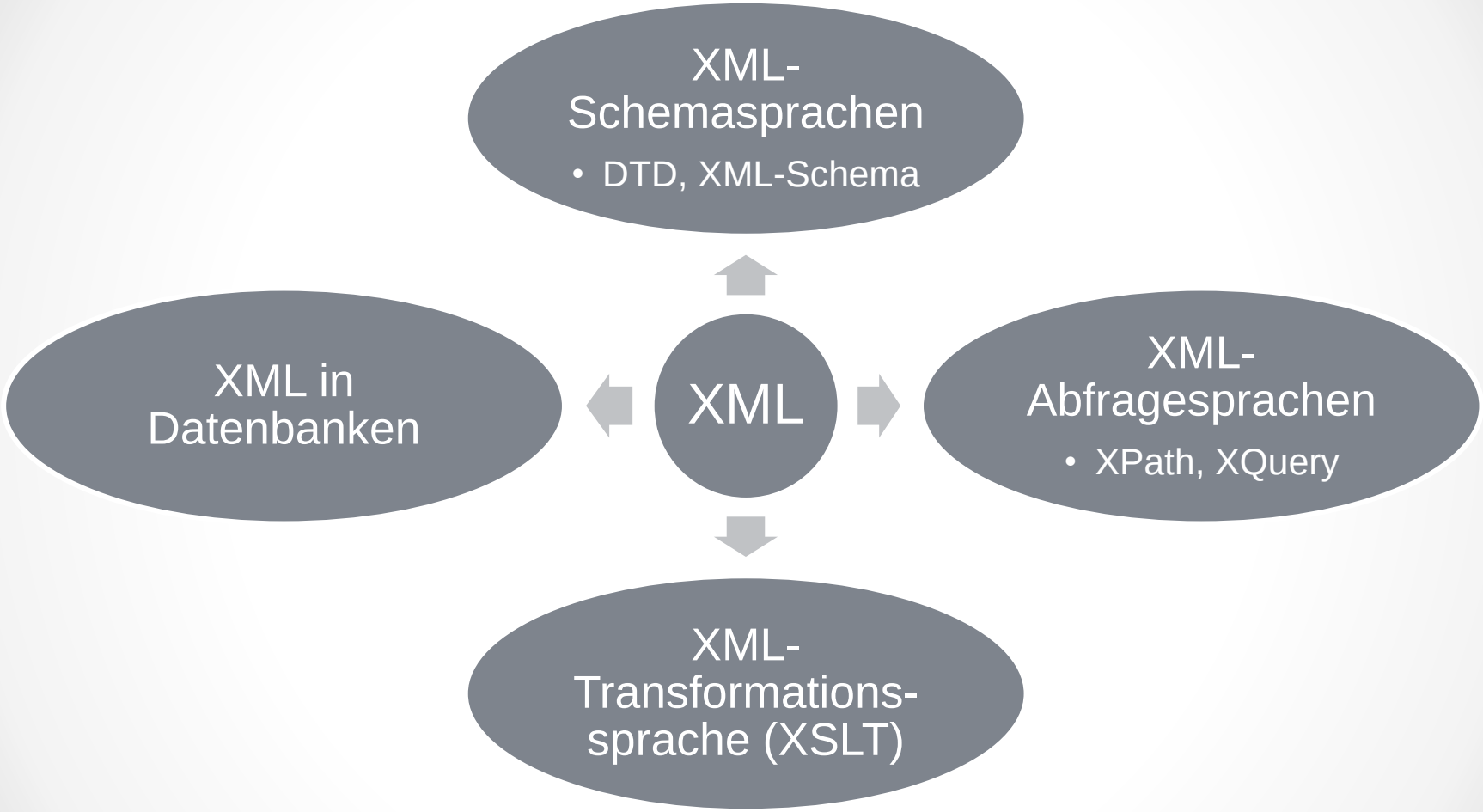
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Ist HTML = XML?

- Tatsächlich gab es mit XHTML Bestrebungen in diese Richtung und die Antwort war nicht klar.
- Heute hat sich der Standard HTML5 durchgesetzt.



Wir schauen uns jetzt XML genauer an. Was gehört alles dazu und was ist XML nun genau?

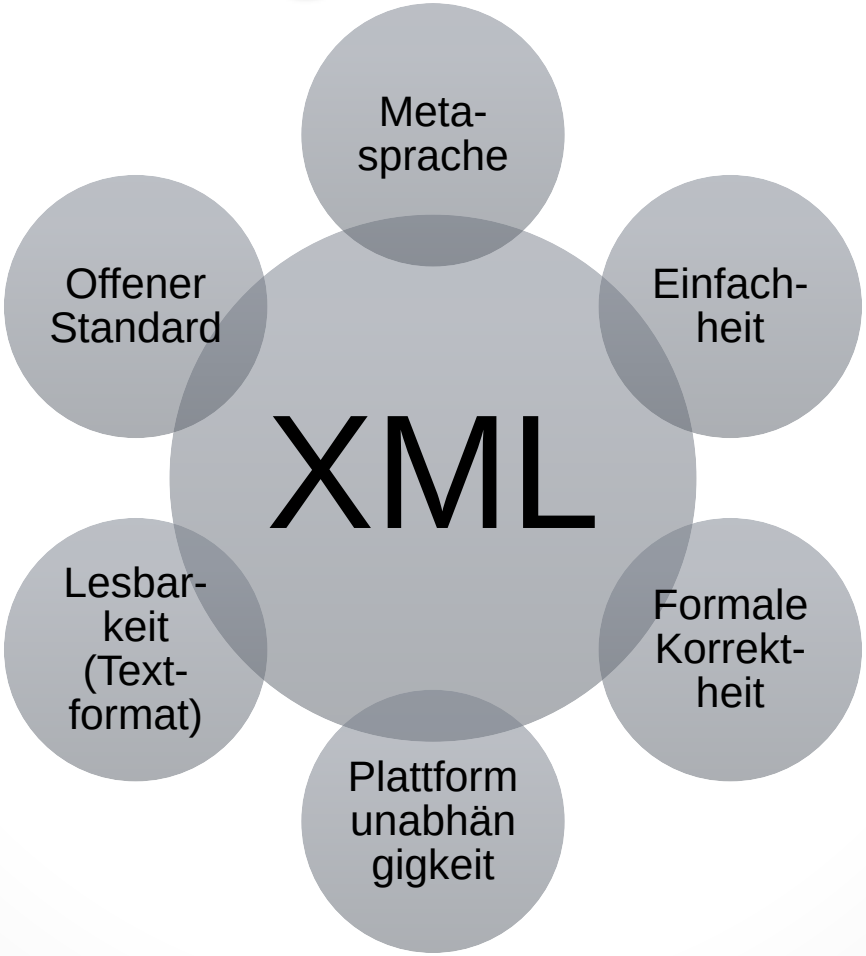

XML-
Schemasprachen
• DTD, XML-Schema
XML in
Datenbanken
XML
XML-
Abfragesprachen
• XPath, XQuery
XML-
Transformations-
sprache (XSLT)

XML = eXtensible Markup Language

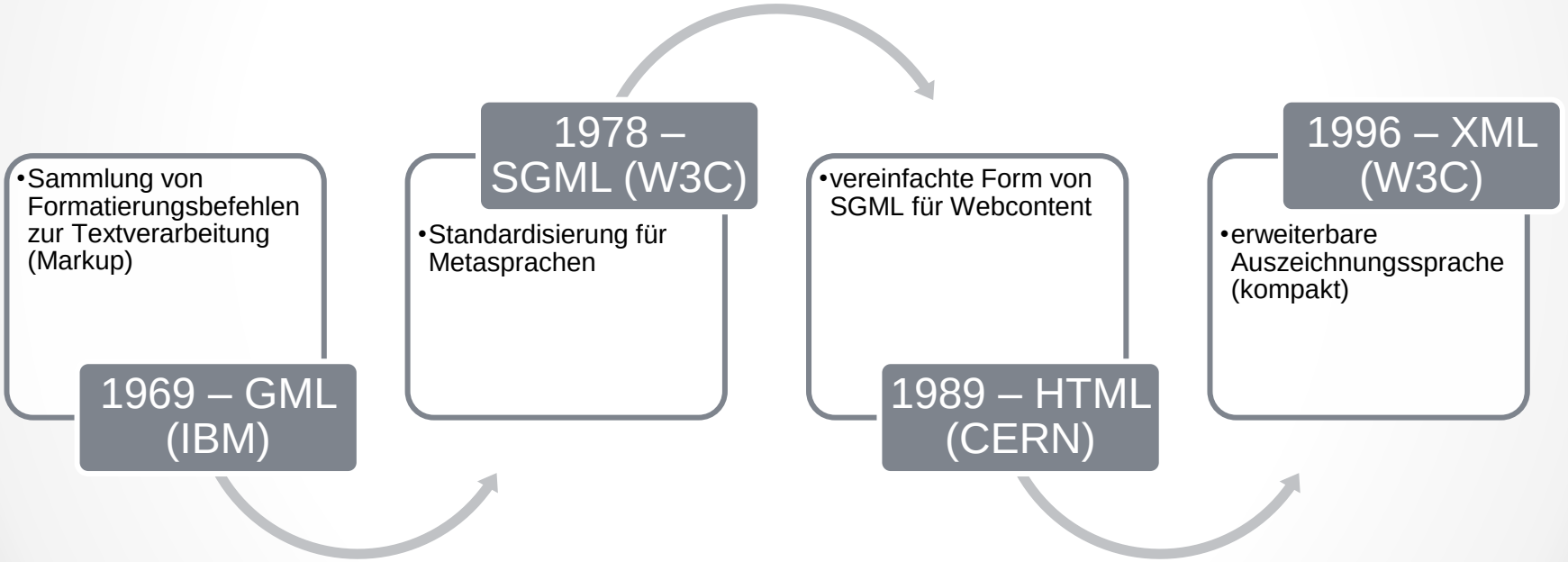
• • •

XML steht für...

XML Eigenschaften



XML Herkunft



XML allg. Definition (en)

The Extensible Markup Language (XML) is a subset of SGML […]. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

XML allg. Definition (dt)

XML (Abkürzung von engl.: extensible markup language) ist eine Metasprache für die Definition von anwendungsspezifischen Auszeichnungssprachen, die vom W3C normiert wird. Mittels XML können somit Auszeichnungssprachen für beliebige Anwendungsbereiche maßgeschneidert werden und anwendungsspezifische Datenstrukturen beschrieben werden.

Definition: Markup-Language

Eine Auszeichnungssprache (engl.: markup language) ist eine formale (nicht-natürliche) Sprache, die es ermöglicht, unterschiedliche Bestandteile eines Textes als solche zu kennzeichnen. Durch eine Auszeichnungssprache können beliebigen Textelementen auf deklarative Weise Eigenschaften zugewiesen werden, wodurch deren Bedeutung ausgedrückt werden kann.

Annotationen

- Synonym für Markup: *Annotation*
- Ein Text wird mit Annotationen versehen, um Bedeutung und Bestandteile auszuzeichnen.
- `<Tags>` in HTML sind Annotationen.
- Tags sind die am häufigsten verwendeten Annotationen (auch in anderen Sprachen).
- Annotationen werden auch als *Metadaten* bezeichnet.

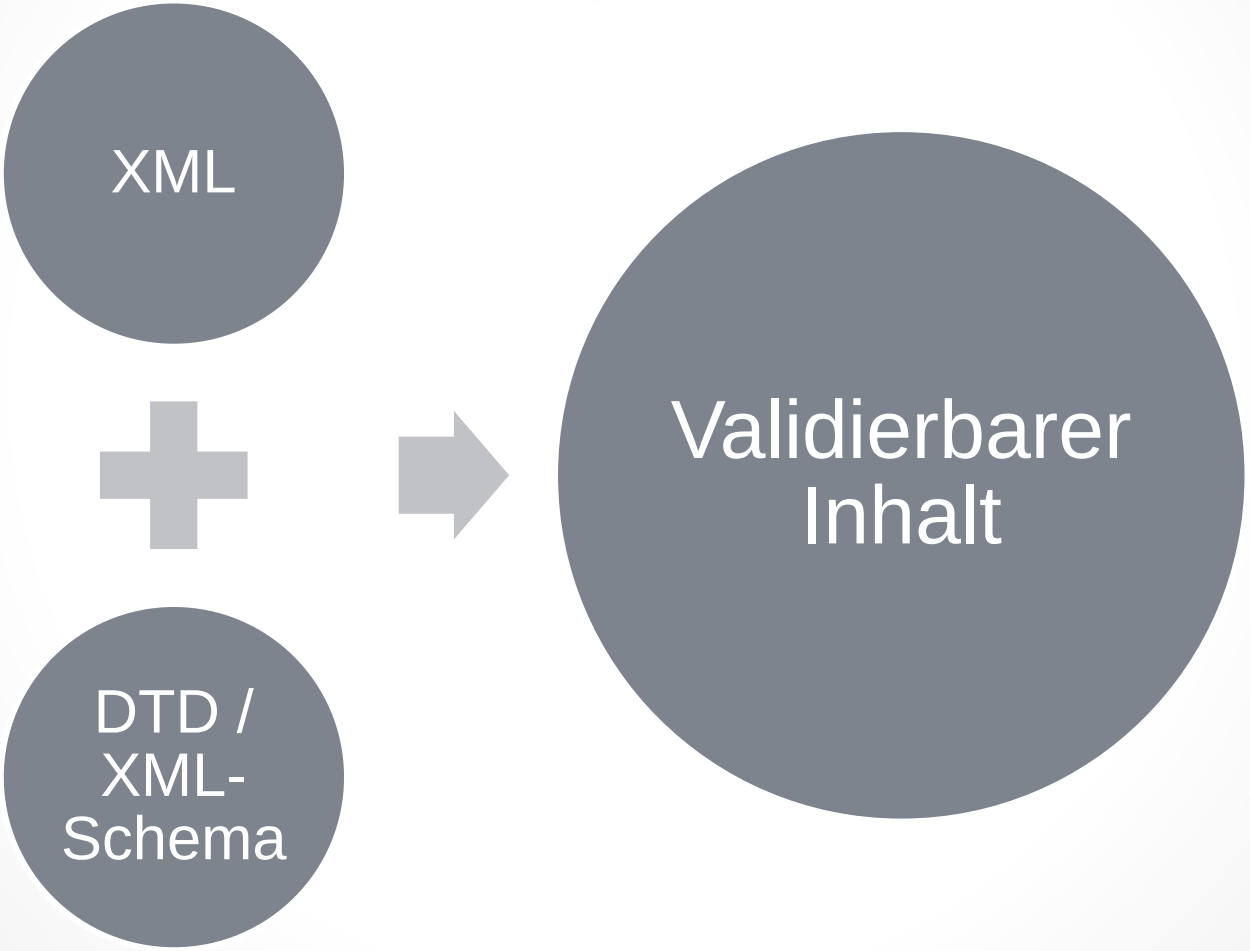
Annotationen für die Darstellung von Inhalt

Annotation

```
1  <!doctype html>
2  <html>
3   <head>
4   <meta charset="UTF-8">
5    <title>Titel: GallierRugby</title>
6   </head>
7   <body>
8    <h1>GallierRugby</h1>
9    <p>Unsere Beispielwebseite wird sich mit dem Thema <em>A
   Rugby-Team kombinieren und gegen die Römer antreten lassen
10   <h2>Das Team</h2>
11  </body>
12 </html>
13
```

Darstellung

Annotation für die Bedeutung von Inhalt



Exkurs: Syntax & Semantik

Syntax

(*griechisch: Zusammenstellung, Satzbau*)

- Struktur von Daten
- Korrektheit der Darstellung
- Wohlgeformtheit

Semantik

(*griechisch: zum Zeichen gehörend*)

- Bedeutung von Daten
- „Korrektheit“ des Informationsgehalts

„Die Macht, - erlernen Du musst.“

„Luke – ich bin Deine Mutter.“

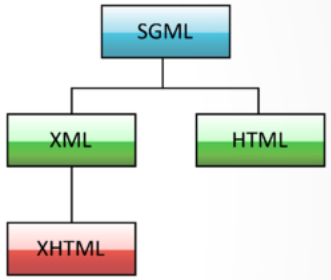
„Luke – ich bin Dein Vater.“

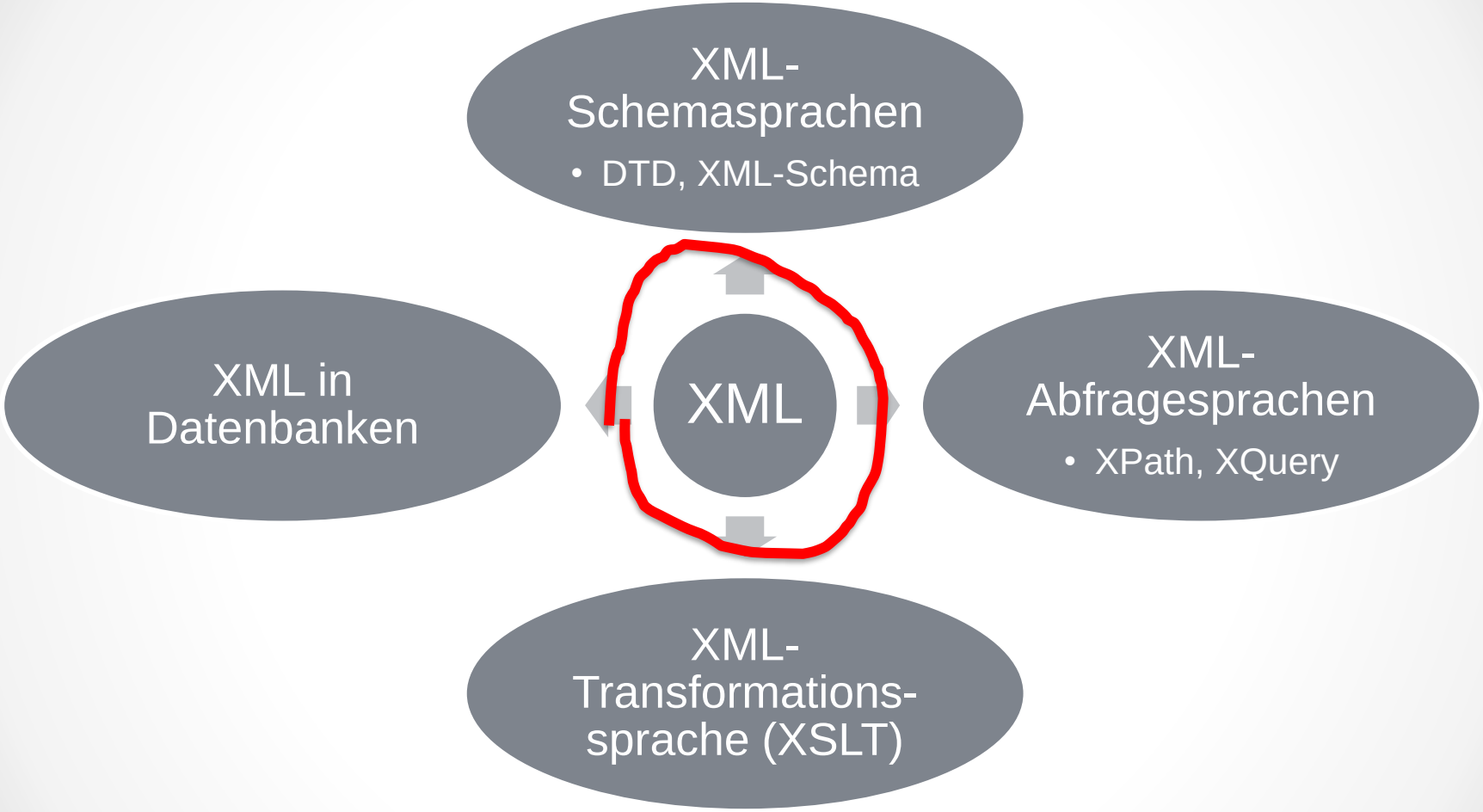
1. XML hilft, Daten im Web **a priori** syntaktisch korrekt zu formen
2. Das ist eine Voraussetzung, um die semantische Auswertung zu erleichtern.

Nochmal HTML...

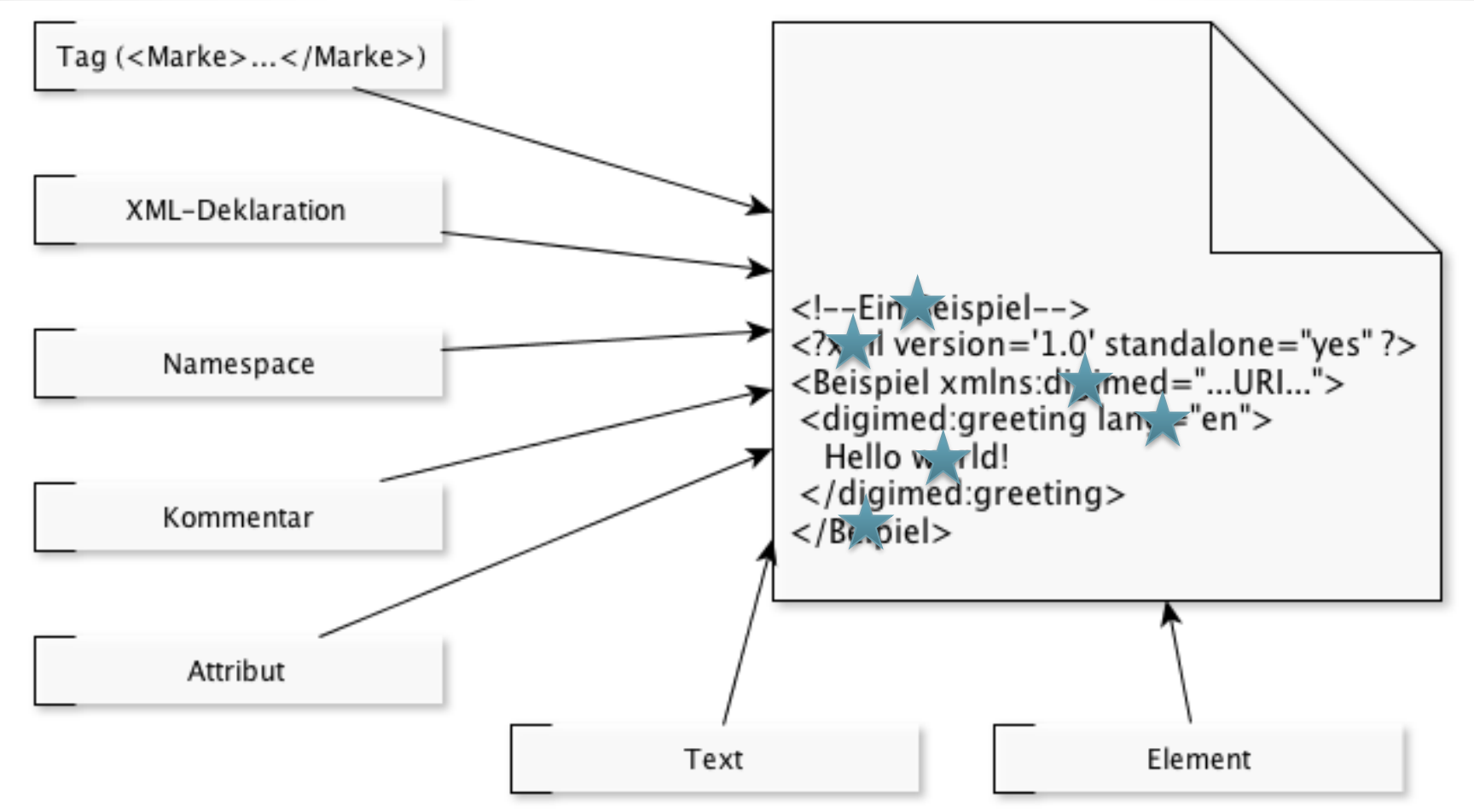
Wir vergleichen jetzt noch einmal XML und HTML:

- XML ist aus SGML abgeleitet (für das Web, weil SGML zu umfangreich,...)
- HTML ist eine Anwendung von SGML
- XHTML ist aus XML abgeleitet.
- Daher gab es auch reichlich Verwirrung mit XHTML 2.0 und der Entwicklung von HTML5



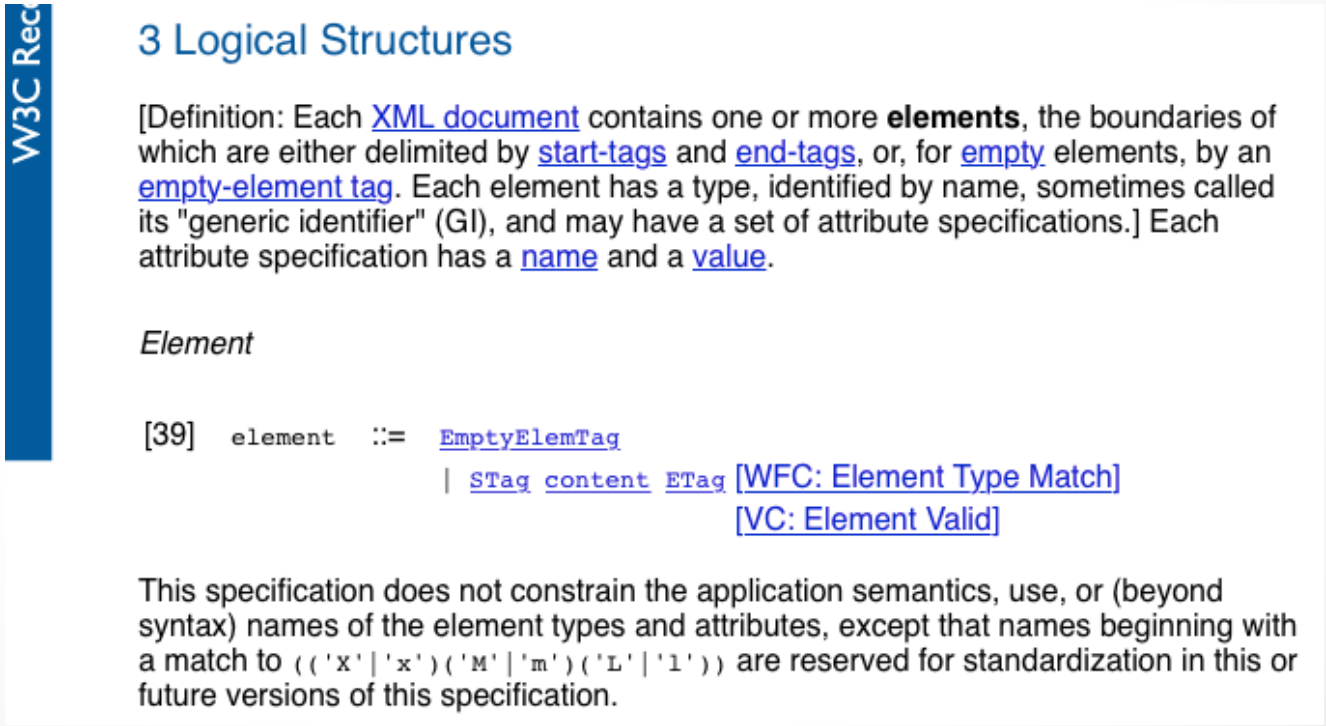

XML-
Schemasprachen
• DTD, XML-Schema
XML in
Datenbanken
XML
XML-
Abfragesprachen
• XPath, XQuery
XML-
Transformations-
sprache (XSLT)

XML unter die Lupe genommen



Spezifikation

http://www.w3.org/TR/2008/REC-xml-20081126



W3C Rec

3 Logical Structures

[Definition: Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications.] Each attribute specification has a name and a value.

Element

[39] `element` ::= `EmptyElemTag`
| `STag content ETag` [WFC: Element Type Match]
[VC: Element Valid]

This specification does not constrain the application semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

Syntax und Aufbau

- Prolog
- Elemente
- Attribute
- Namensräume
- Instruktionen und Anweisungen

Syntax-Regeln

- Elementnamen dürfen:
  - kein Leerzeichen enthalten
  - nicht mit einer Zahl beginnen
  - nur zugelassene Zeichen enthalten
  - nicht mit xml beginnen
- Attribute werden mit = zugewiesen und Werte in „“ eingeschlossen

Prolog `<?xml ...?>`

XML-Deklaration

übliche Startzeile eines XML-Dokuments

Deklaration

- ...muss die Version definieren
- ...kann den Zeichensatz definieren
- ...kann anzeigen, ob ein zugehöriges Schema existiert

```
<?xml version="1.0" encoding="ISO-8859-1”
standalone=“yes”?>
```

Elemente

- Zentrale Bausteine eine XML-Dokuments
  - Start-Tag
  - Inhalt
  - End-Tag

```
<greeting>moinsen</greeting>
```

- Inhalt besteht aus Text oder Elementen
- Diese Art der Schachtelung kennen wir aus HTML.
- `<emptyTag></emptyTag>` = `<emptyTag/>`

Beispiel eines leeren Elements aus HTML

```
<img src="asterix.gif" height="150" alt="Asterix"/>
```

HTML „passt nicht so genau auf und daher geht auch: >“

Attribute

- Zusatzinformationen für Elemente
  - Attributname
  - Gleichheitszeichen =
  - Literal
- Literale werden in einfachen oder doppelten Anführungszeichen angegeben

```
<greeting xml:lang=„en“>
 ...
</greeting>
```

Namespace

- In XML können eigene Namen für Marken (`<tags>`) definiert werden.
- Zur eindeutigen Identifizierung werden Namensräume eingeführt, die mit einer URI verbunden werden.
- Dadurch soll die Eindeutigkeit gewährleistet werden.

Was ist gemeint?

- Nehmen wir eine Marke mit dem Namen „Pfannkuchen“.
- Je nach Region hat Pfannkuchen unterschiedliche Bedeutungen.
- Wird die Marke mehrfach definiert, ist nicht mehr eindeutig klar, was sie bedeutet (ohne Kontext).
- Die Zuordnung zu einem Namensraum kann die Eindeutigkeit wieder herstellen.

Beispiel Namespace

```
<?xml version="1.0" standalone="yes" ?>
<beispiel
xmlns:localbakery="http://www.wiesnbaecker.de">
 <title>Namensraum</title>
 <localbakery:pfannkuchen>
      <anzahl>120</anzahl>
 </localbakery:pfannkuchen>
</beispiel>
```

Instruktionen und Anweisungen

<?xml
(siehe Prolog)

<!DOCTYPE

<? ?>

<![CDATA[...]]>

<!-- ... -->

<!DOCTYPE>

- Definition einer zugehörigen DTD (vgl. das Attribut standalone)
- SYSTEM und PUBLIC
- *DTDs können ähnlich zu CSS in HTML auch direkt in das XML-Dokument eingebettet werden.*

```
<!DOCTYPE root-element SYSTEM URI-of-DTD>
<!DOCTYPE root-element PUBLIC [name] URI-of-DTD>
```

<? ... ?>

- Verarbeitungsinstruktionen für Zielanwendungen
- prominentes Beispiel: Stylesheets

```
<?xml-stylesheet type="text/css"
href="filename.css"?>
```

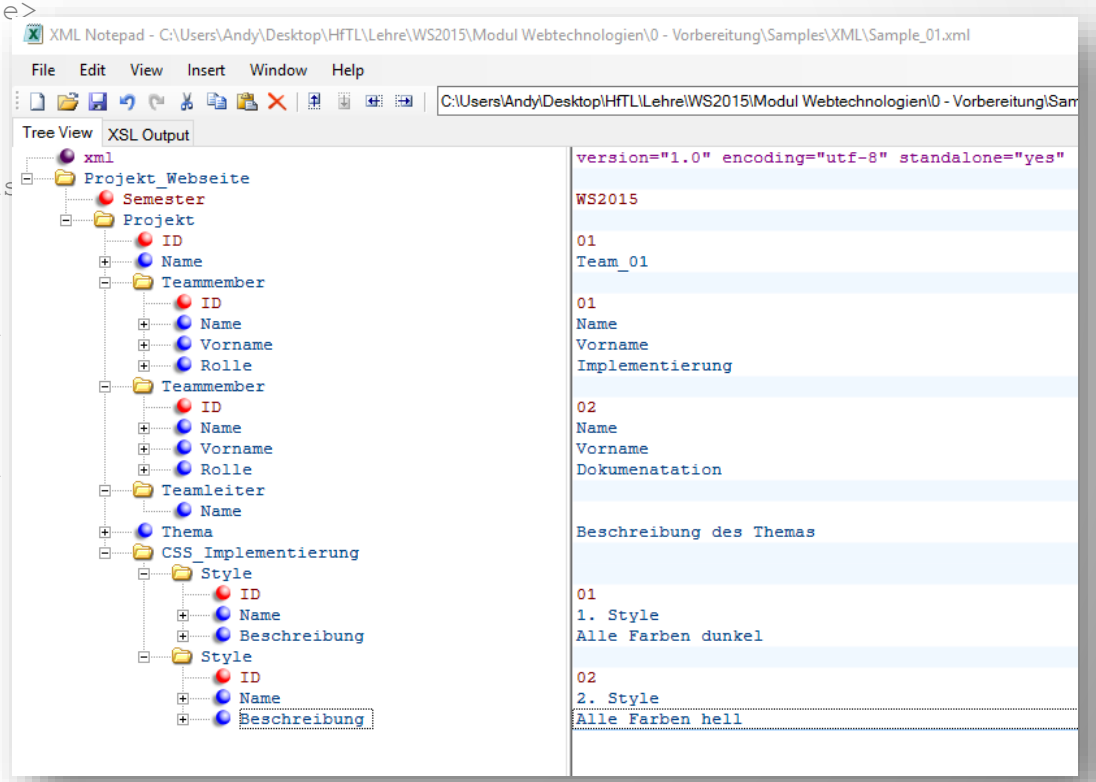
<![CDATA[...]]>

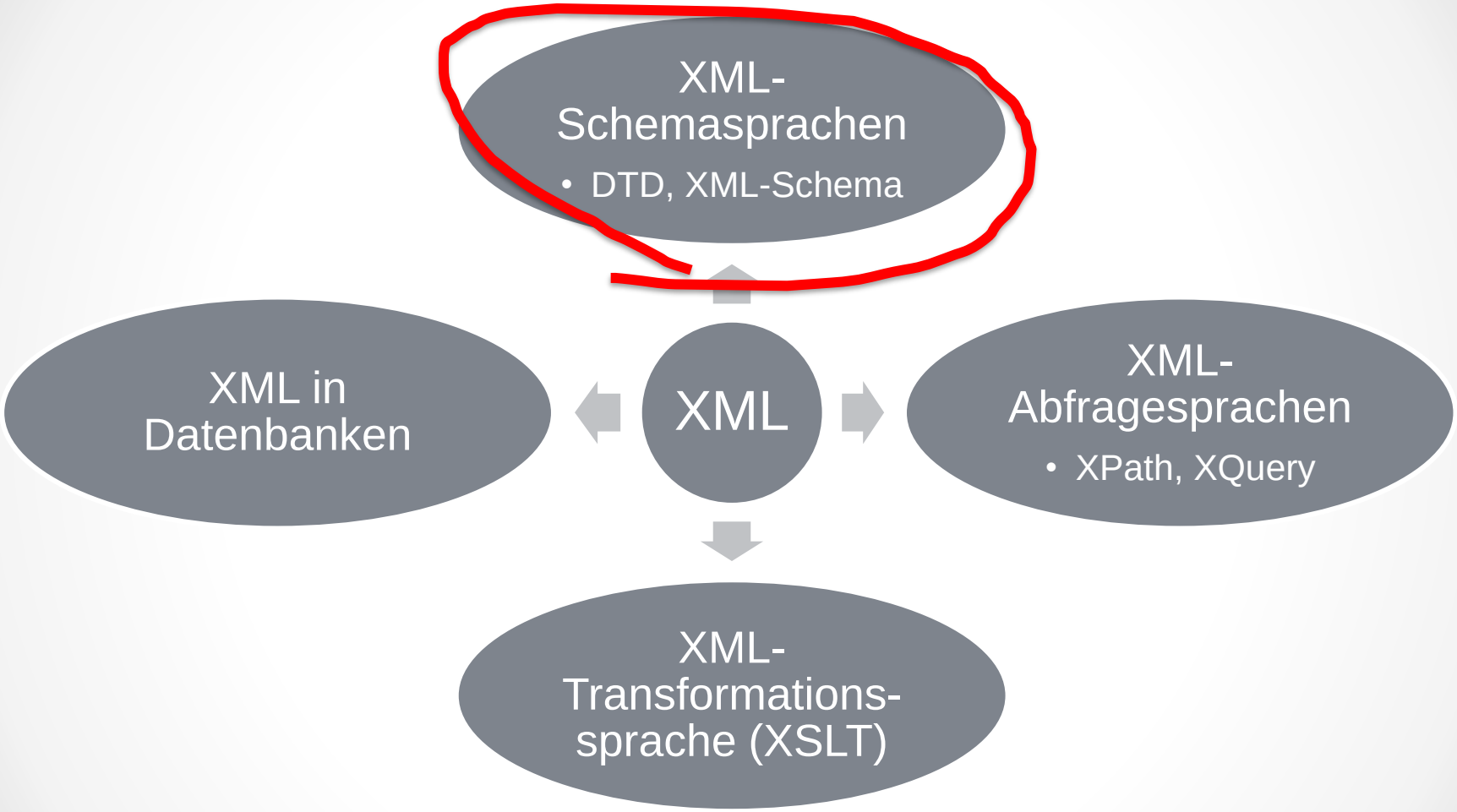
- Charakter-Data Sektion
- Zeichenfolgen, die der Parser NICHT interpretiert

```
<![CDATA[ Hier kommen alles Zeichen, die
der Parser
vollkommen <und> vollständig
ignoriert!
]]>
```

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Projekt_Webseite>
  <Projekt>
    <Name>Team_01</Name>
    <Teammember>
      <Name>Name</Name>
      <Vorname>Vorname</Vorname>
      <Rolle>Implementierung</Rolle>
    </Teammember>
    <Teammember>
      <Name>Name</Name>
      <Vorname>Vorname</Vorname>
      <Rolle>Dokumenatation</Rolle>
    </Teammember>
    <Teamleiter>
      <Name />
    </Teamleiter>
    <Thema>Beschreibung des Themas
    <CSS_Implementierung>
      <Style>
        <Name>1. Style</Name>
        <Beschreibung>Alle Farben
      </Style>
      <Style>
        <Name>2. Style</Name>
        <Beschreibung>Alle Farben
      </Style>
    </CSS_Implementierung>
  </Projekt>
</Projekt_Webseite>
```

Beispiel




XML-
Schemasprachen
• DTD, XML-Schema
XML in
Datenbanken
XML
XML-
Abfragesprachen
• XPath, XQuery
XML-
Transformations-
sprache (XSLT)

DTD + XML-Schema

- XML-Dokumente können durch Schema klassifiziert werden
- Beispiel: DTD und XML-Schema
- Optional!
- Das Schema definiert die Gültigkeit eines XML-Dokuments.

What is a DTD?

A DTD is a Document Type Definition.

A DTD defines the structure and the legal elements and attributes of an XML document.

Quelle: http://www.w3schools.com/xml/xml_dtd_intro.asp

Syntax vs. Gültigkeit

wohlgeformt != gültig

Nicht jedes wohlgeformte (syntaktisch korrekte) Dokument muss gültig sein.

gültig == wohlgeformt

Aber: jedes gültige Dokument ist wohlgeformt.

Diese Frage stellt sich nur, wenn ein Schema definiert ist.

DTD

Ein DTD strukturiert ein XML-Dokument. Es bildet die Grundlage der Validierung (Prüfung auf Gültigkeit)

Gibt es kein DTD dann enthält der Prolog das Attribut: `standalone=„yes“`

Prolog

- DTD wird ebenfalls mit einem Prolog eingeleitet.
- Dokumenttyp-Deklaration und Dokumenttyp-Definition
- Der Startpunkt ist der Wurzelknoten im XML-Dokument.

```
<?xml version='1.0' standalone="no"?>
<!DOCTYPE ABTEILUNGEN SYSTEM "AM.dtd">
```

```
<!DOCTYPE root-element SYSTEM URI-of-DTD>
<!DOCTYPE root-element PUBLIC [name] URI-of-DTD>
```

DTD Elementdeklarationen

Über DTD wird die Gültigkeit eines XML-Dokuments validiert.

daher muss für jedes auftretende Element eine Deklaration im DTD definiert sein.

Im folgenden schauen wir uns die Details der Deklaration an.

Merken: die Deklaration dient der späteren Validierung!

Was wird alles deklariert?

- Elemente
- Attribute
- Entitäten
- Notationen

Wir schauen uns Elemente und Attribute näher an.

DTD Syntax

Elemente

```
<!ELEMENT elementname rule>
```

Attribute

```
<!ATTLIST target_element attr_name attr_type default>
```

Elemente

Beispiel (Element)

Deklaration

```
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Rolle (#PCDATA)>
<!ELEMENT Teammember (Name ,Vorname
,Rolle )>
```

Instanz

```
<Teammember>
 <Name>Andreas</Name>
 <Vorname>Hartmann</Vorname>
 <Rolle>Projektleiter</Rolle>
</Teammember>
```

Typisierung

- ANY
  - Das Element kann beliebige Inhalte haben.
- PCDATA
  - Das Element enthält nur Zeichendaten.
- EMPTY
  - Das Element hat keinen Inhalt, kann aber Attribute besitzen.
- Gemischter Inhalt
  - Das Element kann Zeichendaten und Unterelemente enthalten.
- Elementinhalt
  - Das Element besitzt nur Unterelemente.

Operatoren

Operator	Bedeutung
+	muss mindestens 1x kann aber auch mehrfach vorkommen
?	kann 1x oder gar nicht vorkommen
*	kann beliebig oft vorkommen
,	Trennzeichen innerhalb einer Sequenz
|	Trennzeichen zwischen Alternativen (XOR)
()	Gruppierung

Beispiel

```
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Rolle (#PCDATA)>
<!ELEMENT Teammember (Name, Vorname, Rolle )>
<!ELEMENT Projekt (Name, Teammember+ )>
```

Ein Projekt hat mindestens ein Mitglied (kann auch mehr sein).

Attribute

Beispiel (Attribut)

Deklaration

```
<!ATTLIST Projekt
          id ID #REQUIRED
          pl IDREF #REQUIRED
>
<!ATTLIST Projekt_Webseite Semester ID
#REQUIRED>
<!ATTLIST Teammember id ID #REQUIRED>
```

Instanz

```
<Projekt_Webseite Semester="WS2015">
  <Projekt id="p01" pl="pm001">
    <Name>Team_01</Name>
    <Teammember id="pm001">
      <Name>Name</Name>
      <Vorname>Vorname</Vorname>
      <Rolle>Implementierung</Rolle>
    </Teammember>
…
```

10 Typen für Attribute

Ohne Struktur	Atomare Typen	Listen
• CDATA	• NMTOKEN • ID • IDREF • NOTATION • ENTITY • Aufzählung	• NMTOKENS • IDREFS • ENTITIES

Attribute können etwas differenzierter deklariert werden als Elemente.

Attributtypen

Typ	Beschreibung
Aufzählung	Liste von Token-Werten, von denen einer als Attributwert verwendet werden kann und muss
`CDATA`	einfache Zeichenkette (keine Marken!)
`ENTITY`	Name einer in der DTD deklarierten nicht geparsten Entität
`ENTITIES`	Durch Leerzeichen getrennte Liste von Entitäten
`ID`	Eindeutiger XML-Name (Schlüsselwert eines Elements)
`IDREF`	Verweis auf die ID eines anderen Elements im Dokument
`IDREFS`	Liste von Verweisen auf Ids durch Leerzeichen getrennt
`NMTOKEN`	Namenssymbol aus beliebigen Zeichen (entspr. XML-Namen)
`NMTOKENS`	Listen von Namenssymbolen durch Leerzeichen getrennt
`NOTATION`	Verweis auf eine Notation, z.B. Grafikdatei (no XML)

Vorgaben für die Attributwerte

Vorgabewert	Beschreibung
Attributwert	„default“ – Standardwert
`#IMPLIED`	keine Vorgabe und es ist auch kein Wert erforderlich
`#REQUIRED`	keine Vorgabe und es ist ein Wert erforderlich
`#FIXED Wert`	in jedem Fall wird Wert als Konstante verwendet

Beispiel

DTD

```
<!-- Elementtype Definition-->
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Vorname (#PCDATA)>
<!ELEMENT Rolle (#PCDATA)>
<!ELEMENT Teammember (Name ,Vorname ,Rolle )>
<!ELEMENT Teamleiter (Name )>
<!ELEMENT Thema (#PCDATA)>
<!ELEMENT Stylename (#PCDATA)>
<!ELEMENT Beschreibung (#PCDATA)>
<!ELEMENT Style (Stylename ,Beschreibung )>
<!ELEMENT CSS_Implementierung (Style+ )>
<!ELEMENT Projekt (Name,Teammember+
,Teamleiter ,Thema ,CSS_Implementierung+  )>
<!ELEMENT Projekt_Webseite (Projekt* )>
<!-- Attribute Definition-->
<!ATTLIST Projekt
          id ID #REQUIRED
          pl IDREF #REQUIRED
>
<!ATTLIST Projekt_Webseite Semester ID
#REQUIRED>
<!ATTLIST Teammember id ID #REQUIRED>
<!ATTLIST Style id ID #REQUIRED>
```

XML-Instanz

```
<?xml version="1.0" encoding="utf-8"
standalone="no"?>
<!DOCTYPE Projekt_Webseite SYSTEM "DigiMed.dtd">
<Projekt_Webseite Semester="WS2015">
  <Projekt id="p01" pl="pm001">
    <Name>Team_01</Name>
    <Teammember id="pm001">
      <Name>Name</Name>
      <Vorname>Vorname</Vorname>
      <Rolle>Implementierung</Rolle>
    </Teammember>
    <Teamleiter>
      <Name />
    </Teamleiter>
    <Thema>Beschreibung des Themas</Thema>
    <CSS_Implementierung>
      <Style id="style01">
        <Stylename>1. Style</Stylename>
        <Beschreibung>dunkel</Beschreibung>
      </Style>
    </CSS_Implementierung>
  </Projekt>
</Projekt_Webseite>
```

Beispiel DTD

DocBook

```
<!ELEMENT abstract (((title|titleabbrev)*,
(info)?), (anchor|para|formalpara|simpara)+)>

<!ATTLIST abstract
            xmlns       CDATA       #FIXED
            "http://docbook.org/ns/docbook"
            role        CDATA       #IMPLIED
            %db.common.attributes;
            %db.common.linking.attributes;

>
```

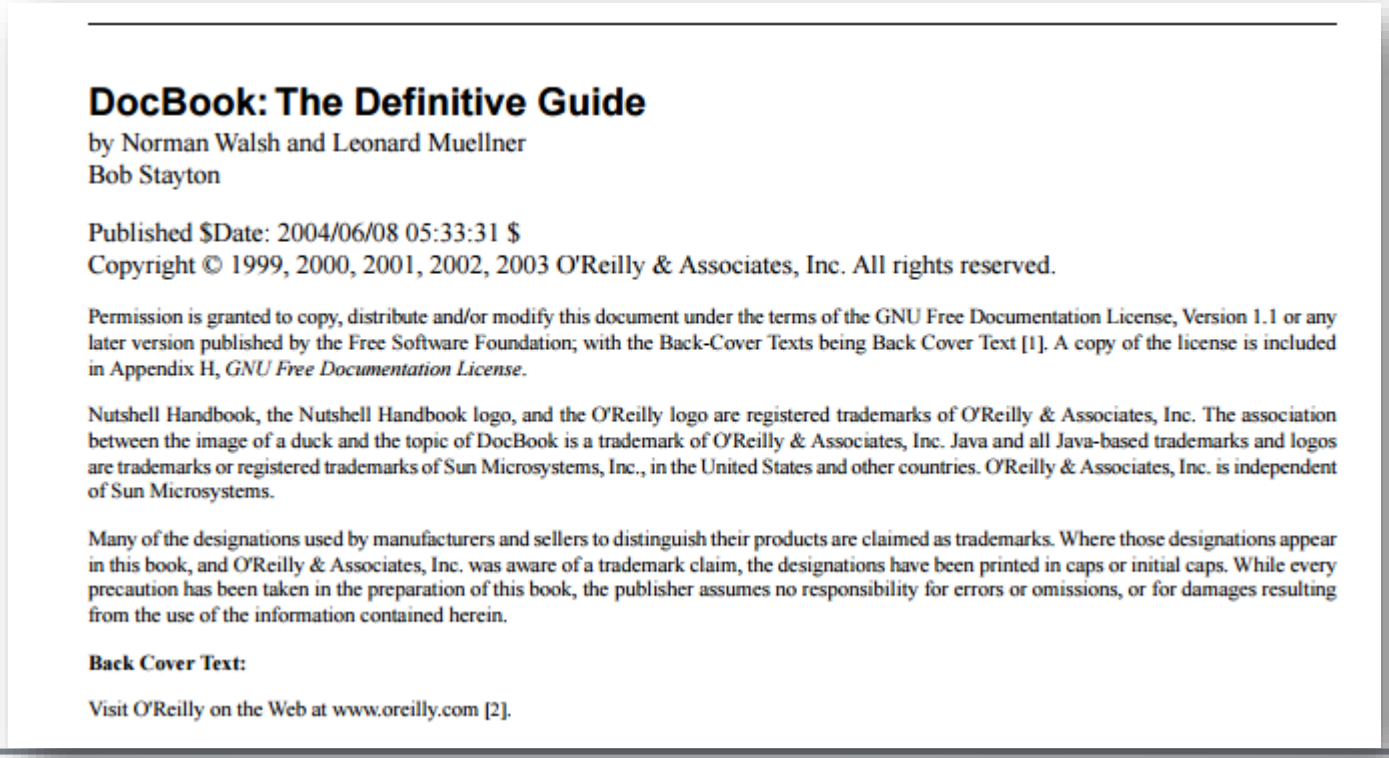
SVG

```
<!-- Basic Paint Attribute fill, fill-rule,
stroke, stroke-dasharray, stroke-dashoffset,
stroke-linecap, stroke-linejoin, stroke-
miterlimit, stroke-width, color, color-rendering
This module defines the Paint and Color attribute
sets. --> <!-- a 'fill' or 'stroke'
property/attribute value: <paint> -->

<!ENTITY % Paint.datatype "CDATA" > <!-- 'stroke-
dasharray' property/attribute value (e.g., 'none',
list of <number>s) --> <!ENTITY %
StrokeDashArrayValue.datatype "CDATA" >
```

Es handelt sich um öffentliche DTDs (auch XML-Anwendungen genannt).

DocBook

```
<?xml version='1.0'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN”
       "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd”>
```



DocBook: The Definitive Guide

by Norman Walsh and Leonard Muellner
Bob Stayton

Published $Date: 2004/06/08 05:33:31 $

Back Cover Text:

Visit O'Reilly on the Web at www.oreilly.com [2].

SVG

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

{ "id": "6a4a7304-8374-4c8e
"https://api.filepreviews.io/v2
0d636c487ecb/", "status": "s
"requested_size": "original",
"https://s3.amazonaws.com/
2913ee1dcdcde6685b9972
8974-0d636c487ecb_origina
"800" }, "size": { "height": "26
"resized": false, "requested_
"https://s3.amazonaws.com/demo.filepreviews.io/b47d2930a000fbb03e7843d
2913ee1dcdcde6685b9972460813ef7091533eaa9/6a4a7304-8374-4c8e-
8974-0d636c487ecb_original-1.png", "original_size": { "height": "262", "width":
"800" }, "size": { "height": "262", "width": "800" }, "page": 1 }], "original_file": {
"encoding": "us-ascii", "extension": "svg", "name": "Coca-Cola_logo",
"mimetype": "image/svg+xml", "total_pages": 1, "size": 8937, "metadata": { },
"type": "image" }, "user_data": null }

Quelle: http://www.w3.org/TR/SVG/struct.html

http://filepreviews.io/sandbox/

XML-Schema

XML-Schema dienen analog zu DTDs der Validierung von XML-Dokumenten.

Wieso XML-Schema?

- DTD hat einige Defizite bzgl. der Semantik
  - z.B. gibt es keine „zulässigen Wertebereiche“ –> Datumeingabe durch Nutzer
  - ein DTD prüft nicht, ob der Nutzer ein gültiges Datum dd.mm.yyyy eingibt
- DTD ist selbst kein XML-Dokument
- DTD ist vergleichsweise starr
- das XML-Schema soll diese Defizite lösen

Definition

Ein XML-Schema legt wie ein DTD das Vokabular und die Grammatik eines XML-Dokuments fest. Es ist dabei selbst ein XML-Dokument, welches auf Gültigkeit und Wohlgeformtheit geprüft werden kann.

Gültigkeit von Werten

```
<!ATTLIST lastchanged CDATA #REQUIRED>
```

- Im DTD ist das eine gültige Deklaration.
- Was, wenn der Nutzer kein Datum eingibt?
- XML-Schema:
  - `<xsd:attribute name=„lastchanged“ type=„xsd:date“ use=„required“>`

Wir erahnen, dass es in XML-Schema mehr Typen gibt...

Hinweise zu XML-Schema

- Merken: XML-Schema ist selbst ein XML-Dokument.
- Wurzel ist immer `<xsd:schema>`
- Namesraum: `xsd` (gehört zum XML-Schema)
  - `xmlns:xsd=„http://www.w3c.org/2001/XMLSchema“`
- Da XML-Schema selbst ein XML-Dokument ist, kann es durch ein DTD beschrieben werden. Unter W3C ist die DTD zu finden…

Quelle: W3C

XML Schema

15 October 2014

Table of contents

Introduction

This document describes the XML Schema namespace. It also contains a directory of links to these related resources, us

Related Resources for XML Schema

Schemas for XML Schema

DTD

XML Schema 1.1

A (non-normative) DTD XMLSchema.dtd for XML Schema. It incorporates an auxiliary DTD, datatypes.dtd.

XML Schema 1.0

A (non-normative) DTD XMLSchema.dtd for XML Schema. It incorporates an auxiliary DTD, datatypes.dtd.

XML Schema

XML Schema 1.1

An XML Schema schema document for XML Schema schema documents.

XML Schema 1.0

An XML Schema schema document for XML Schema schema documents. Last updated with release of XML Sch

DTD für XML-Schema (1)

```
<!-- The simpleType element and its constituent parts
     are defined in XML Schema: Part 2: Datatypes -->
<!ENTITY % xs-datatypes PUBLIC 'datatypes' 'datatypes.dtd' >

<!ENTITY % p 'xs:'> <!-- can be overriden in the internal subset of a
                         schema document to establish a different
                         namespace prefix -->
<!ENTITY % s ':xs'> <!-- if %p is defined (e.g. as foo:) then you must
                         also define %s as the suffix for the appropriate
                         namespace declaration (e.g. :foo) -->
<!ENTITY % nds 'xmlns%s;'>

<!-- Define all the element names, with optional prefix -->
<!ENTITY % schema "%p;schema">
<!ENTITY % complexType "%p;complexType">
<!ENTITY % complexContent "%p;complexContent">
<!ENTITY % simpleContent "%p;simpleContent">
<!ENTITY % extension "%p;extension">
<!ENTITY % element "%p;element">
<!ENTITY % unique "%p;unique">
<!ENTITY % key "%p;key">
<!ENTITY % keyref "%p;keyref">
<!ENTITY % selector "%p;selector">
<!ENTITY % field "%p;field">
```

DTD für XML-Schema (2)

```
<!--
        DTD for XML Schemas: Part 2: Datatypes
        $Id: datatypes.dtd,v 1.23 2001/03/16 17:36:30 ht Exp $
        Note this DTD is NOT normative, or even definitive. - - the
        prose copy in the datatypes REC is the definitive version
        (which shouldn't differ from this one except for this comment
        and entity expansions, but just in case)
  -->

<!--
        This DTD cannot be used on its own, it is intended
        only for incorporation in XMLSchema.dtd, q.v.
  -->

<!-- Define all the element names, with optional prefix -->
<!ENTITY % simpleType "%p;simpleType">
<!ENTITY % restriction "%p;restriction">
<!ENTITY % list "%p;list">
<!ENTITY % union "%p;union">
```

Anders in XML-Schema

- Im Vergleich zur DTD gibt es in XML-Schema deutlich mehr Bestandteile, insbesondere Typen. Eine kurze Übersicht schauen wir uns dazu an.

Komponenten

Primäre:

Typen (einfach, komplex), Elemente, Attribute

Sekundäre:

Attributgruppen, Eindeutigkeitsbeschränkungen, Schlüsselreferenzen, Modellgruppen, Anmerkungen

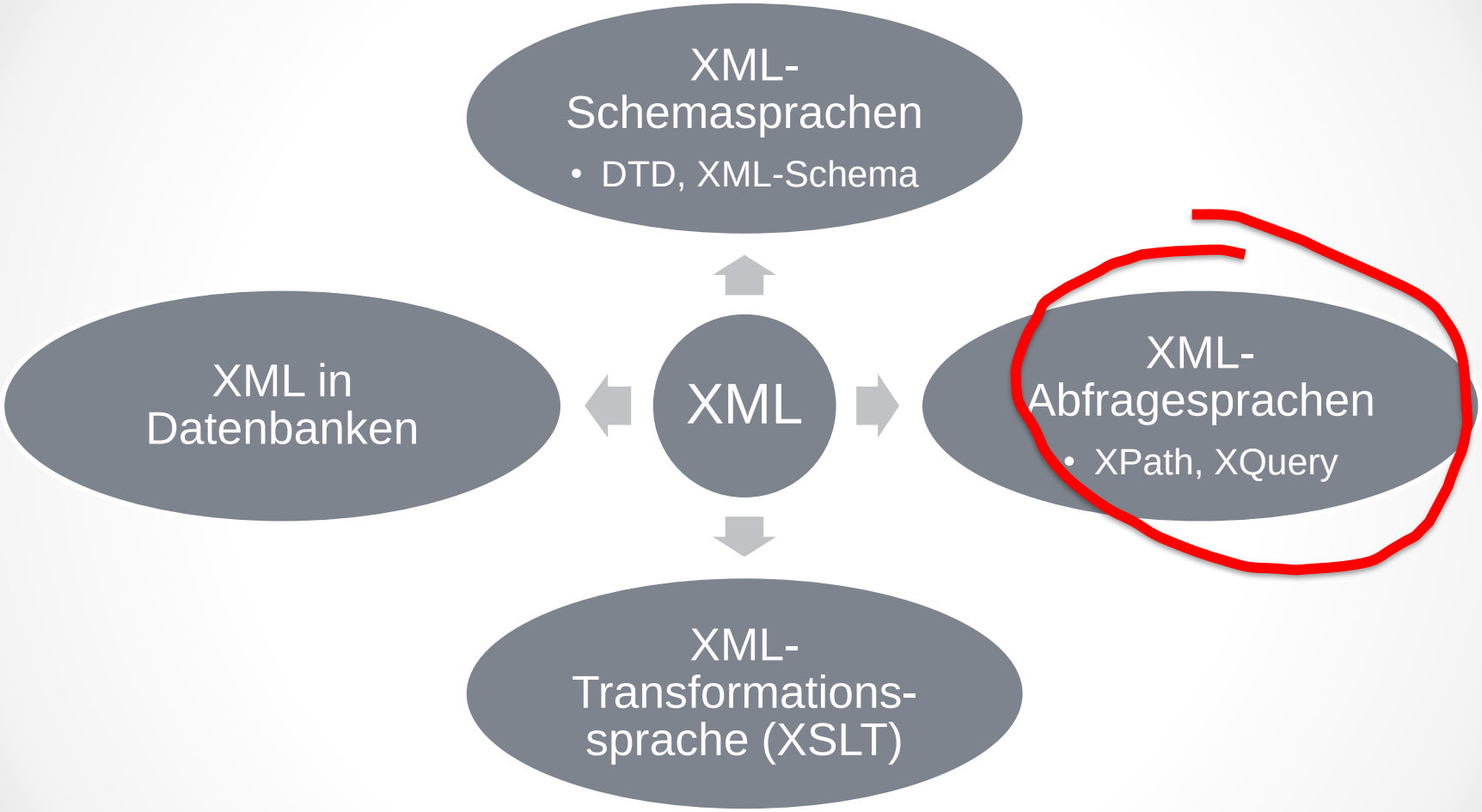
Tertäre (Hilfskomponenten als Teil von anderen Komponenten):

Anmerkungen, Modellgruppen, Partikel, Joker, Festlegungen über die Verwendung von Attributen

Im Detail können wir nicht weiter auf die Komponenten eingehen. Siehe Literatur!

Beispiel Kennwort

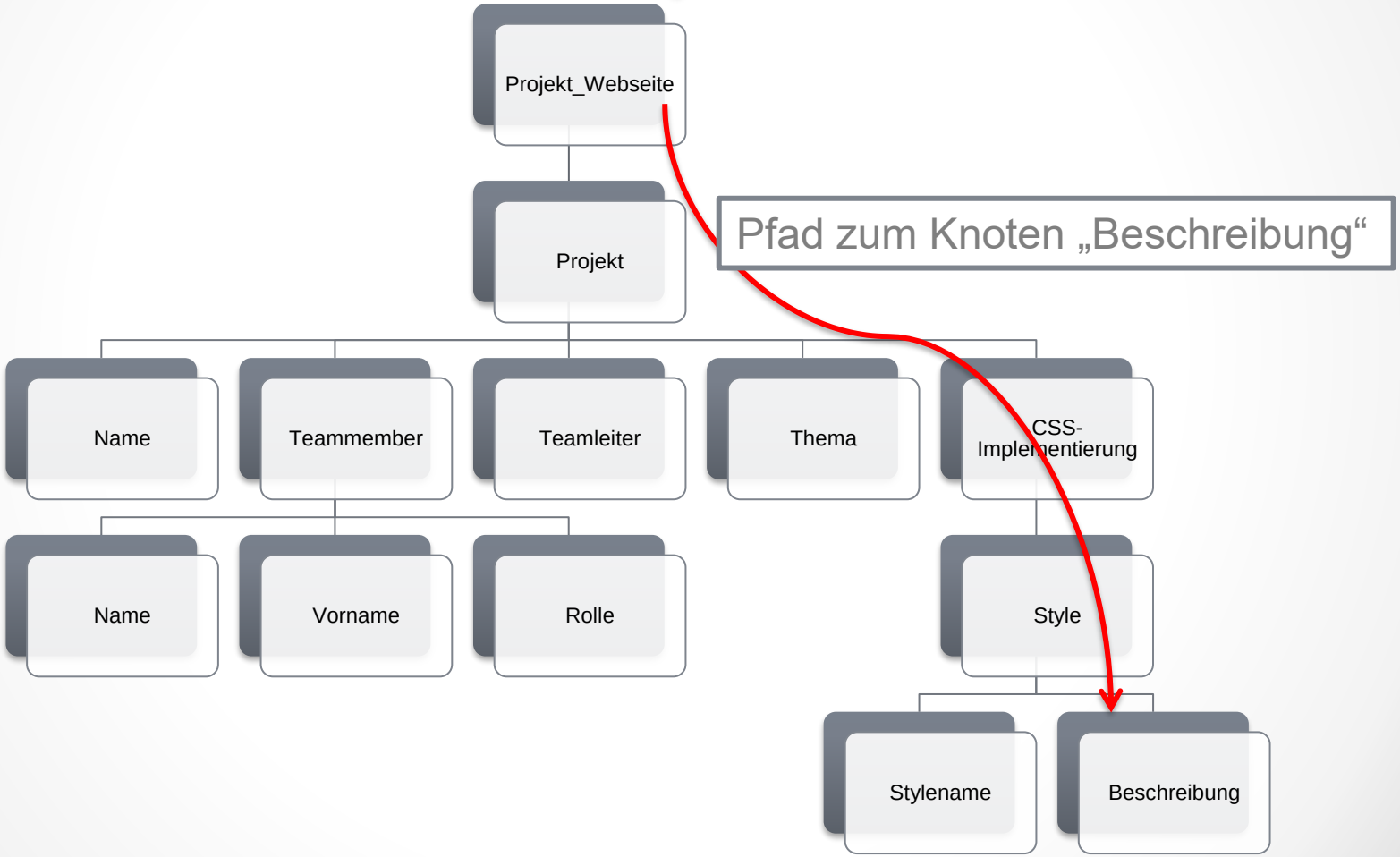
```
<xsd:simpleType name=„password“>
 <xsd:restriction base=„xsd:string“>
  <xsd:minLength value=„8“>
  <xsd:maxLength value=„15“>
 </xsd:restriction>
</xsd:simpleType>
```


XML-
Schemasprachen
• DTD, XML-Schema
XML in
Datenbanken
XML
XML-
Abfragesprachen
• XPath, XQuery
XML-
Transformations-
sprache (XSLT)

XPath

- XML-Dokumente haben eine Baumstruktur
- Knoten enthalten in der Regel Daten
- Wie können Daten schnell aufgefunden werden?
- spezielle Adressierungssprache *XML Path Language – XPath*

Beispiel


Projekt_Webseite
Projekt
Pfad zum Knoten „Beschreibung“
Name
Teammember
Teamleiter
Thema
CSS-Implementierung
Name
Vorname
Rolle
Style
Stylename
Beschreibung

Merkmale

XPath ist selbst keine XML-Anwendung

XPath ist eine eigene Syntax zur Bildung von Zeichenketten, über die sich Elemente eines XML-Dokuments adressieren lassen.

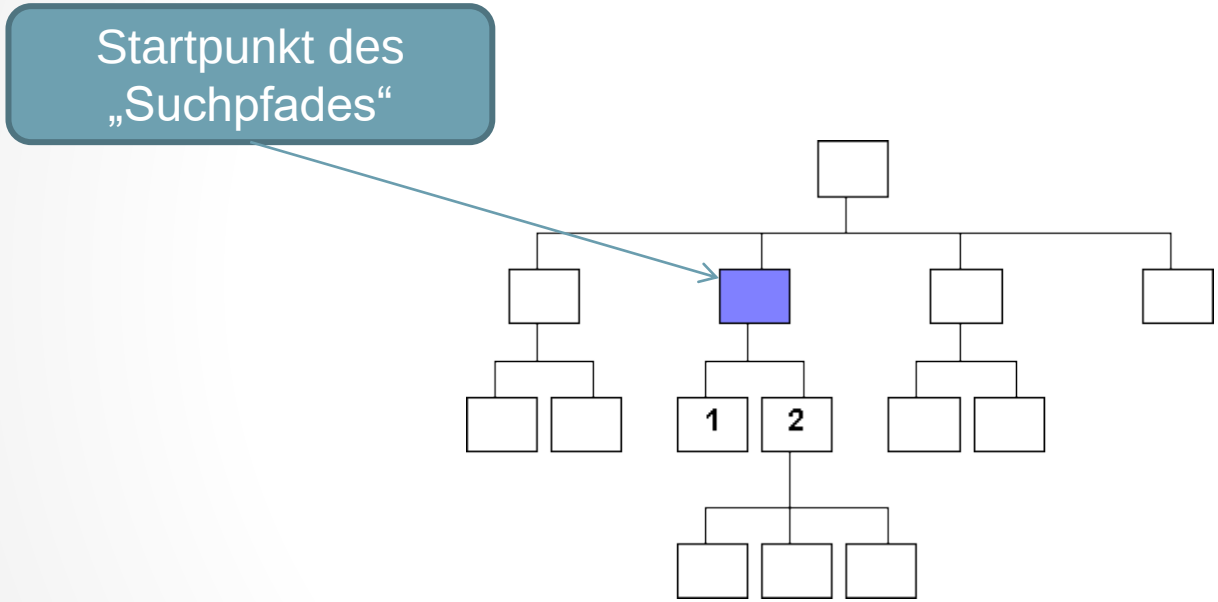
http://www.w3.org/TR/xpath

XPath Anwendung

- Basis ist ein Baummodell (ähnlich DOM)
- Abfrage durch so genannte *XPath-Ausdrücke*
- ein *XPath-Prozessor* übersetzt ein XML-Dokument in einem Baumstruktur und wendet einen Ausdruck an

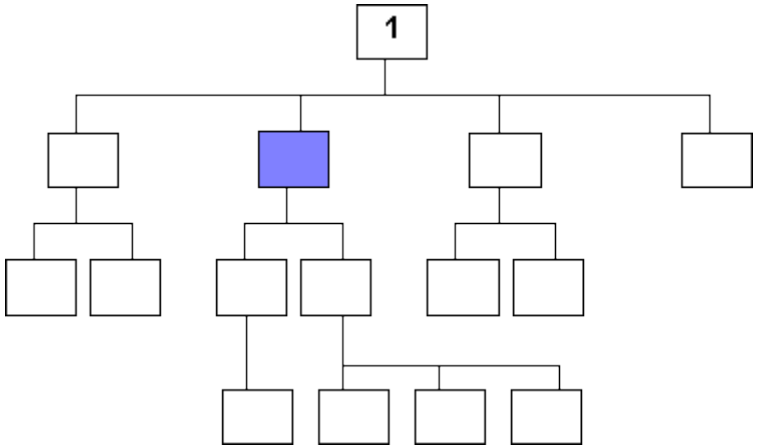
Schauen wir uns noch einmal den Baum und die Beziehungen der Knoten untereinander an!

Child



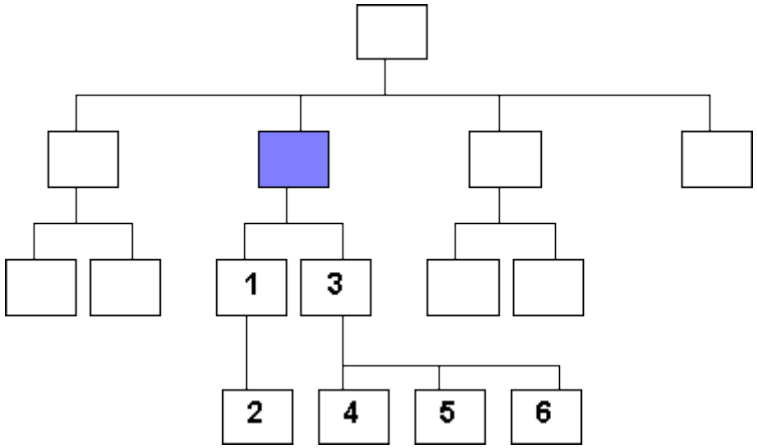
https://wiki.selfhtml.org/wiki/Datei:Xml_child.gif; MScharwies

Parent



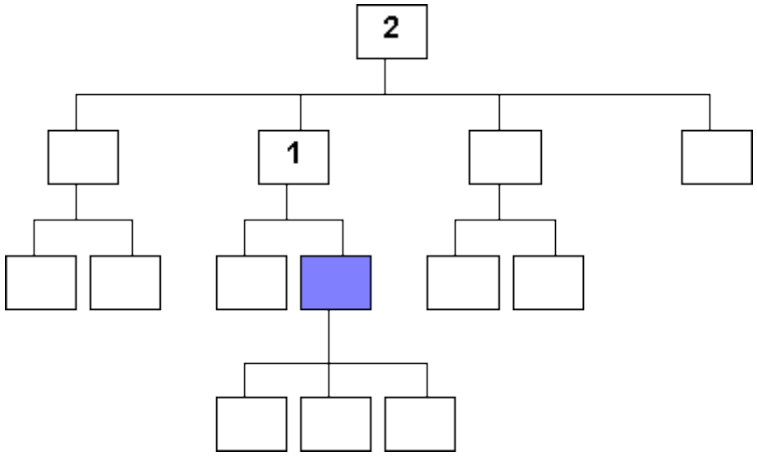
https://wiki.selfhtml.org/wiki/Datei:Xml_parent.gif; MScharwies

Descendant



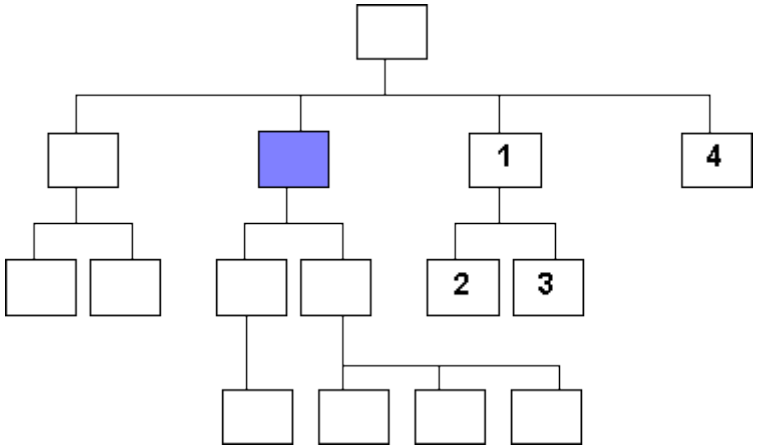
https://wiki.selfhtml.org/wiki/Datei:Xml_descendant.gif; MScharwies

Ancestor



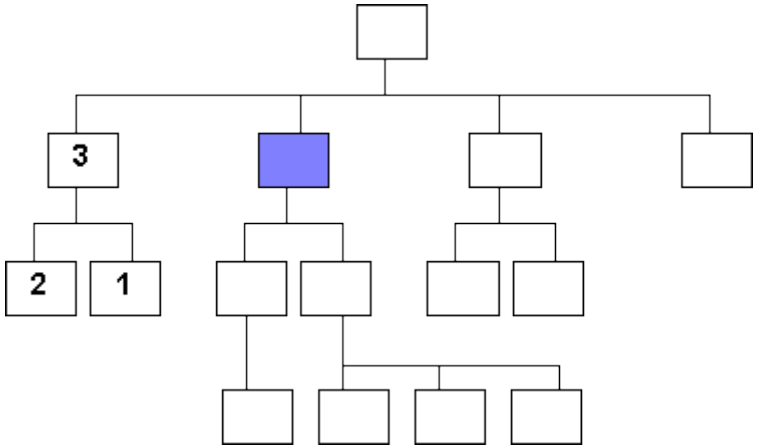
https://wiki.selfhtml.org/wiki/Datei:Xml_ancestor.gif; MScharwies

Nachfolger (following)



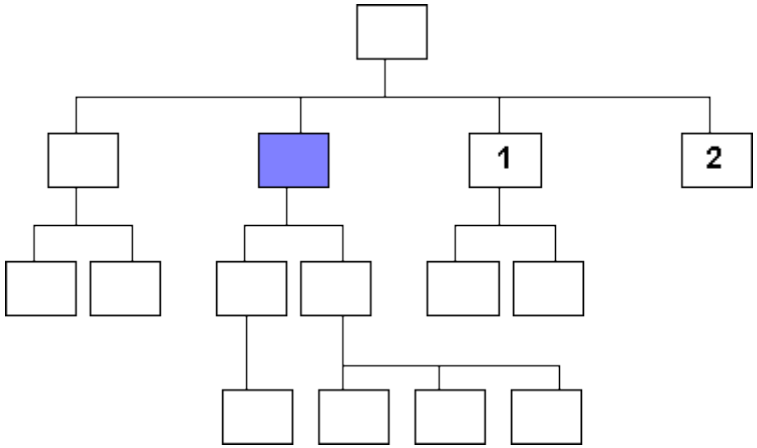
https://wiki.selfhtml.org/wiki/Datei:Xml_following.gif; MScharwies

‚Vorhergend‘ (preceding)



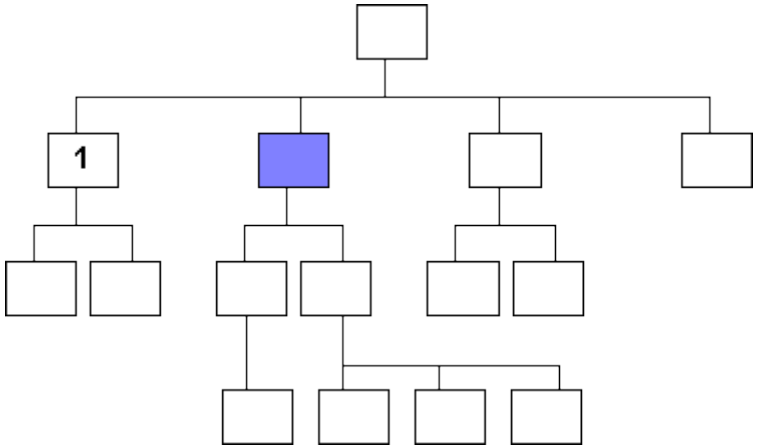
https://wiki.selfhtml.org/wiki/Datei:Xml_preceding.gif; MScharwies

following sibling



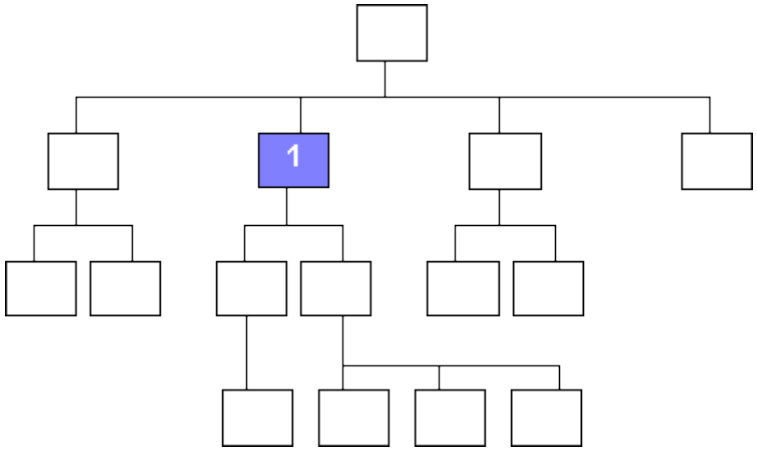
https://wiki.selfhtml.org/wiki/Datei:Xml_following_sibling.gif; MScharwies

preceding sibling



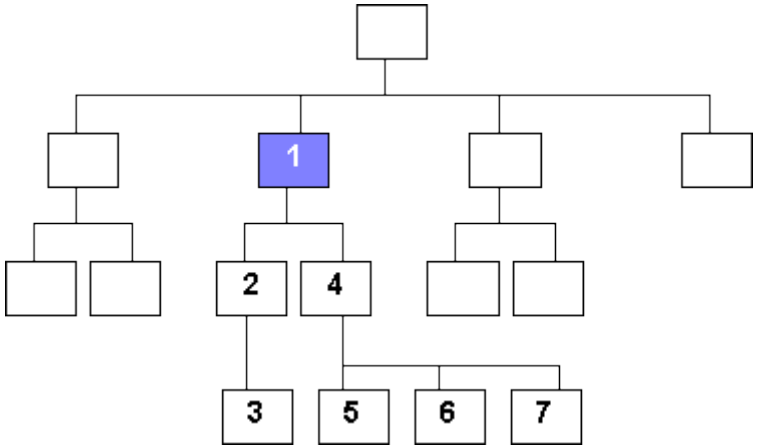
https://wiki.selfhtml.org/wiki/Datei:Xml_preceding_sibling.gif; MScharwies

self



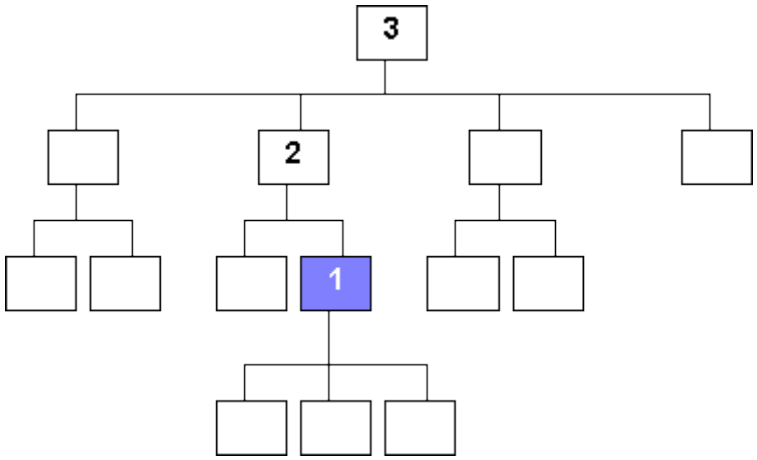
https://wiki.selfhtml.org/wiki/Datei:Xml_self.gif; MScharwies

descendant or self



https://wiki.selfhtml.org/wiki/Datei:Xml_descendant_or_self.gif; MScharwies

ancestor or self



https://wiki.selfhtml.org/wiki/Datei:Xml_ancestor_or_self.gif; MScharwies

Beispiel für unser Bsp.

```
/Projekt_Webseite/Projekt/CSS-
Implementierung/Style/Beschreibung/@attribut
```

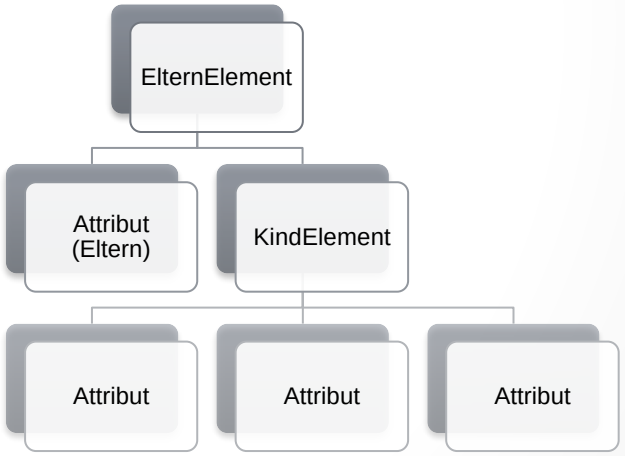
XPath-Ausdruck zum *Attribut* des Elements „Beschreibung“.

XPath kennt nur 4 Datentypen

- node-set (Menge von Knoten, auch leer)
- string (Zeichenfolge, auch leer)
- number (Fließkomma 64bit double)
- boolean (true, false)

Unterschied zu DOM

- Baum wird speziell für Suchoperationen erstellt
- Knoten haben keine Eigenschaften oder Werte
- so sind z.B. Attribute eines Elements wiederum Kindknoten des Element-Knotens



Somit gibt es spezielle Knotentypen im Suchbaum

Wurzelknoten

- root node
- Wurzel des XML-Dokuments
  - Ist selbst kein Element, sondern ein abstraktes Objekt welches auf die Ursprungswurzeln im Baum zeigt

Elementknoten

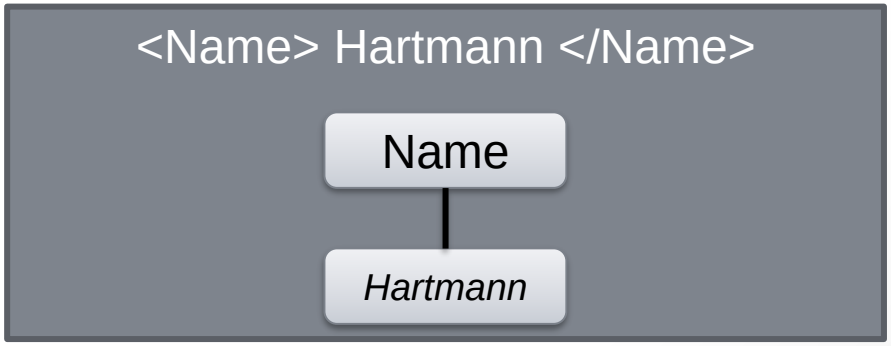
- Entspricht den Elementen im XML-Dokument
- ...also `<element>some content</element>`
- Unterelemente sind ebenfalls Elementknoten

Attributknoten

- Im Unterschied zum DOM werden Attribute in Form von Knoten angesprochen
- Jedem Attribut ist ein Attributknoten zugeordnet, bestehend aus:
  - Name
  - Wert

Textknoten

- Beinhaltet ein Element keine weiteren Elemente, sondern nur noch Zeichen, handelt es sich um einen zugeordneten Textknoten
- Vgl. Anzeige des Baumes im XML-Editor!
  - Hier ist jedem Elementknoten noch ein Textknoten zugeordnet, welcher die Zeichenkette des Elements enthält.

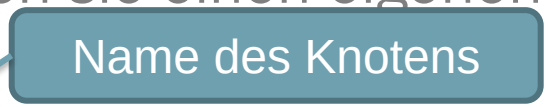


Namensraumknoten

- Elemente und Attribute können Namensraumangaben enthalten
- …werden analog zu Attributknoten modelliert und dem Elementknoten zugeordnet

Verarbeitungsanweisungs knoten

- Verarbeitungsanweisungen sind keine Elemente im Dokumentbaum
- Daher stellen sie einen eigenen Knotentyp dar




```
<?xml-stylesheet type="text/xsl" href="darstellung.xsl"?>
```

Die XML-Deklaration gehört allerdings **nicht** dazu!

```
<?xml version="1.0"?>
```

Kommentarknoten

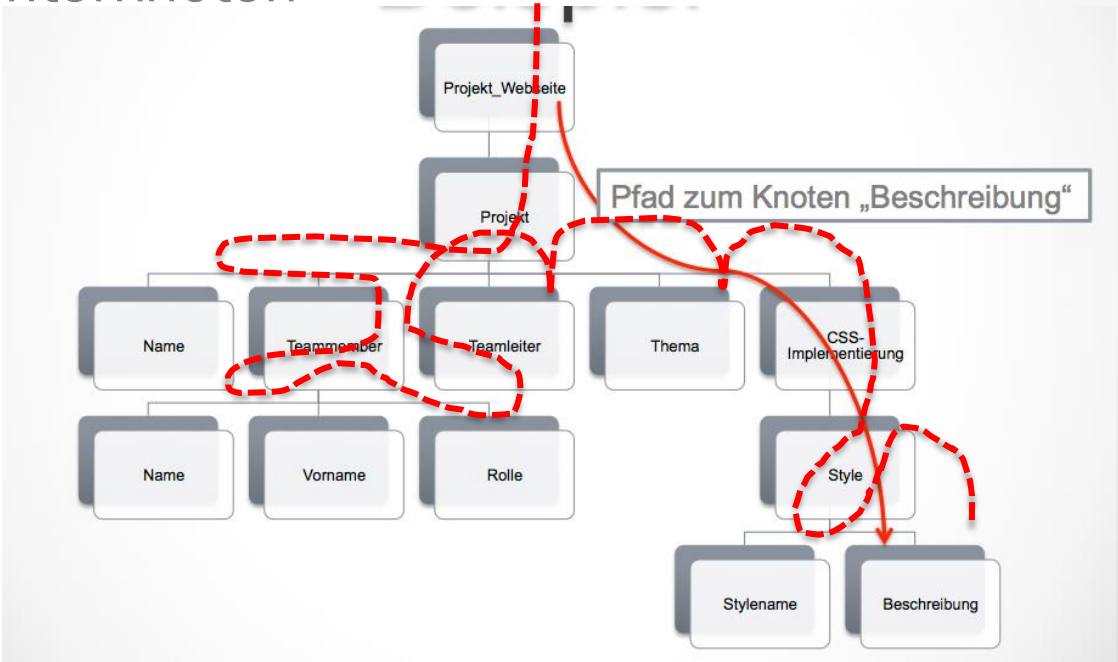
- `<!--Kommentar-->`

Lokalisierungspfad

- vgl. dazu auch unseren Exkurs Graphentheorie (Suchbaum)
- Der Lokalisierungspfad liefert den Zugriff auf Elemente.
- Das Ergebnis ist eine Knotenmenge.
  - (Merken: im Suchbaum sind auch Attribute = Knoten)

Pfadangabe

- relative Pfade
  - `../../Thema (aus Sicht von „Style“)`
- absolute Pfade
  - `/Projekt-Webseite/Projekt/CSS-Implementierung/Style/Beschreibung`
- Abkürzung: //gesuchterknoten
  - `//Beschreibung`



Adressierung von Attributen

- z.B.:
  - `STG/@name`

<xml …
<STG name=„DKMI“>

Beispiel	Notation	Erläuterung
self::liste	Ausführlich	adressiert das aktuelle Element, jedoch nur dann, wenn es vom Typ liste ist.
child::listeneintrag[position()=3]	ausführlich	adressiert das 3. Kindelement vom Typ listeneintrag mit Hilfe der XPath-Funktion position().
listeneintrag[3]	verkürzt	adressiert das 3. Kindelement vom Typ listeneintrag.
child::listeneintrag[position()=last()]	ausführlich	adressiert das letzte Kindelement vom Typ listeneintrag mit Hilfe der XPath-Funktion last().
listeneintrag[last()]	verkürzt	adressiert das letzte Kindelement vom Typ listeneintrag mit Hilfe der XPath-Funktion last().
listeneintrag[last()-1]	verkürzt	adressiert das vorletzte Kindelement vom Typ listeneintrag mit Hilfe der XPath-Funktion last().
/buch/kapitel[2]/abschnitt[4]/absatz[17]	verkürzt	adressiert in einem Buch absolut den 17. Absatz des 4. Abschnitts im 2. Kapitel (entsprechende XML-Elementnamen vorausgesetzt).
child::augen[attribute::farbe='blau']	ausführlich	adressiert das Kindelement augen, jedoch nur dann, wenn es ein Attribut farbe mit dem Wert blau hat.
augen[@farbe='blau']	verkürzt	
child::augen[attribute::farbe and attribute::sehstaerke]	ausführlich	adressiert das Kindelement augen, jedoch nur dann, wenn es sowohl ein Attribut farbe als auch ein Attribut sehstaerke hat (egal mit welchen zugewiesenen Werten).
augen[@farbe and @sehstaerke]	verkürzt	
/child::*[self::inhaltsverzeichnis or self::stichwortverzeichnis]	ausführlich	adressiert in einem Buch absolut alle Kindelemente unterhalb der Wurzel, jedoch nur solche, die als Elementnamen inhaltsverzeichnis oder stichwortverzeichnis haben.
.	verkürzt	adressiert den aktuellen Knoten.
..	verkürzt	adressiert den Elternknoten des aktuellen Knotens.
kapitel//absatz	verkürzt	adressiert alle Elemente vom Typ absatz, die aus Sicht eines aktuellen Knotens unterhalb von Elementen vom Typ kapitel liegen. Die Syntax // wird auch als recursive descent operator bezeichnet. //Xbezeichnet das gleiche wie /descendant-or-self::node()/X.
buch//kapitel[position() < 5]	verkürzt	adressiert die ersten vier Kindelemente des Typs kapitel innerhalb eines Elements vom Typ buch.

Quelle: https://wiki.selfhtml.org/wiki/XML/XSL/XPath/Einführung

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation - jedoch nur, wenn vor dem Operator keines der folgenden Zeichen bzw. keine der folgenden Zeichenfolgen steht: @, ::, (, [,), ,
=	gleich (Vergleich zweier Werte)
!=	ungleich (Vergleich zweier Werte)
< bzw. <	kleiner als (Vergleich zweier Werte). Maskierung innerhalb von XSLT erforderlich.
> bzw. >	größer als (Vergleich zweier Werte). Maskierung innerhalb von XSLT zu empfehlen.
<= bzw. <=	kleiner als oder gleich (Vergleich zweier Werte). Maskierung innerhalb von XSLT erforderlich.
>= bzw. >=	größer als oder gleich (Vergleich zweier Werte). Maskierung innerhalb von XSLT zu empfehlen.
and	logische Und-Verknüpfung (beide Ausdrücke)
or	logische Oder-Verknüpfung (einer von beiden Ausdrücken)
div	Fließkommateilung
mod	Rest der Fließkommateilung

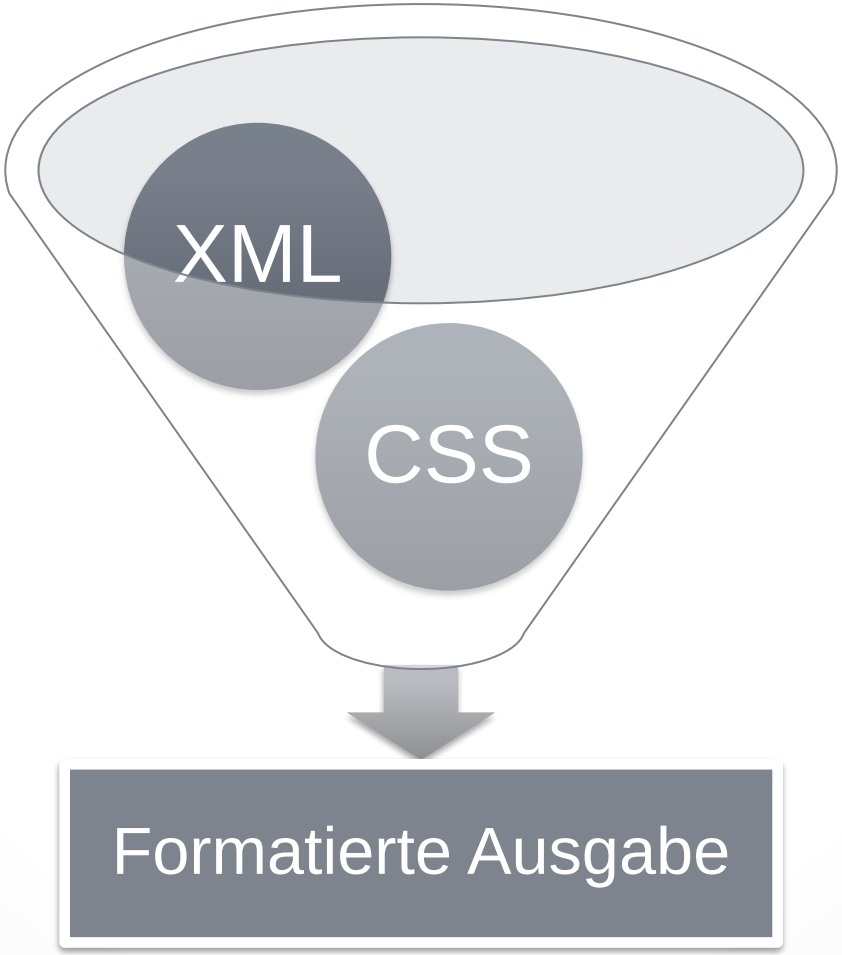
Mehr zum Thema…

- XPath ist eine mächtige Sprache mit komplexen Ausdrücken zur Navigation in XML-Dokumenten.
- Details siehe Literatur.

XML & CSS

- Bisher haben wir uns die Strukturierung von Daten sowie die Suche nach Daten angesehen.
- Ein anderer Punkt ist die geeignete Repräsentation.
- Ähnlich HTML – ohne CSS sahen unsere Webseiten noch recht karg aus!
- Erfreulicherweise funktioniert das in XML analog.

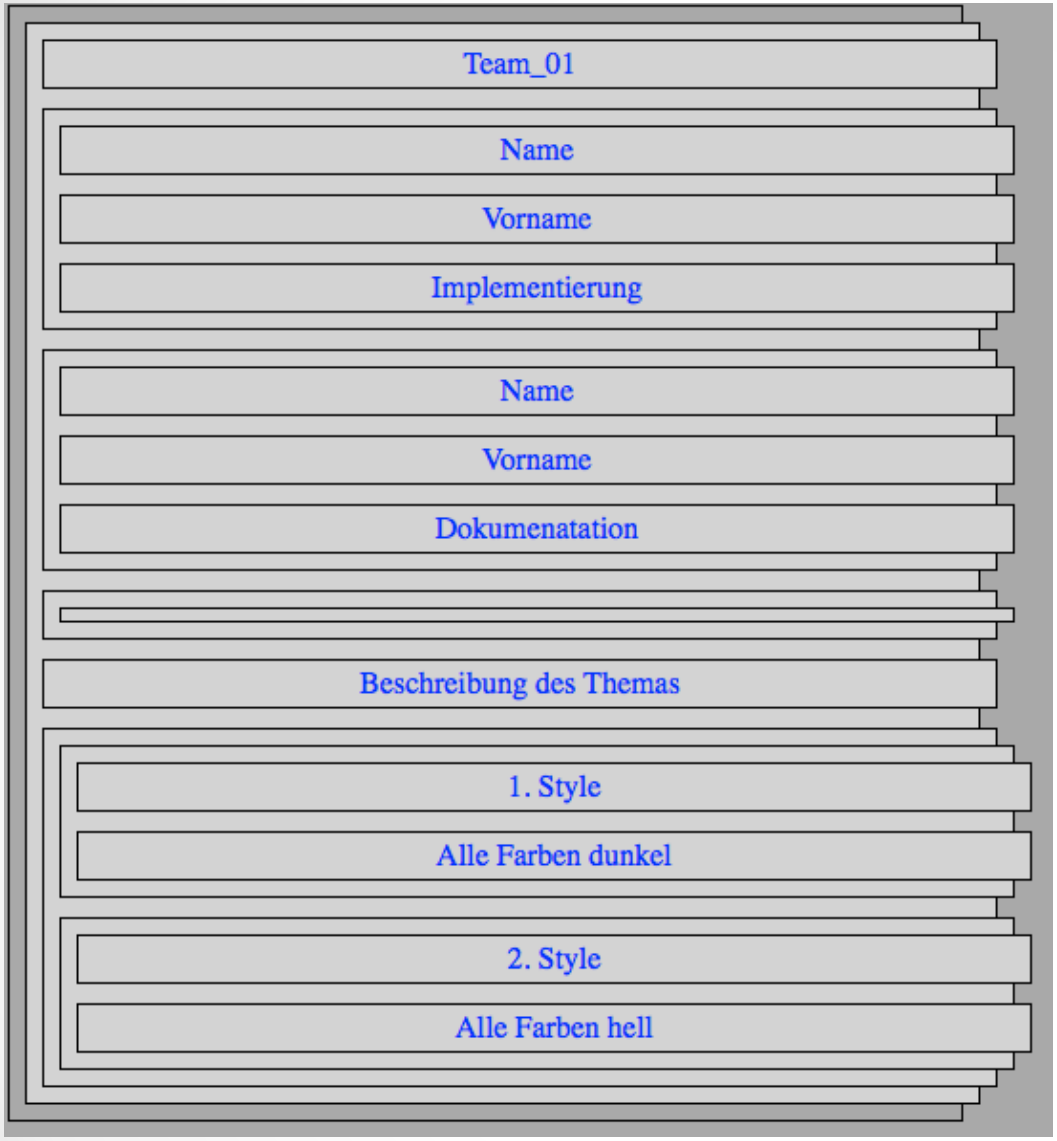
Das Prinzip



Anwendung

Die Anwendung von Stylesheets kennen wir bereits aus HTML. In XML-Dokumenten passiert das auf analoge Weise:

- Stylesheet einbinden
- Stylesheet bezieht sich auf Elemente im XML-Dokument
- Aussehen (d.h. Darstellung) steuern



Beispiel für unser Bsp.

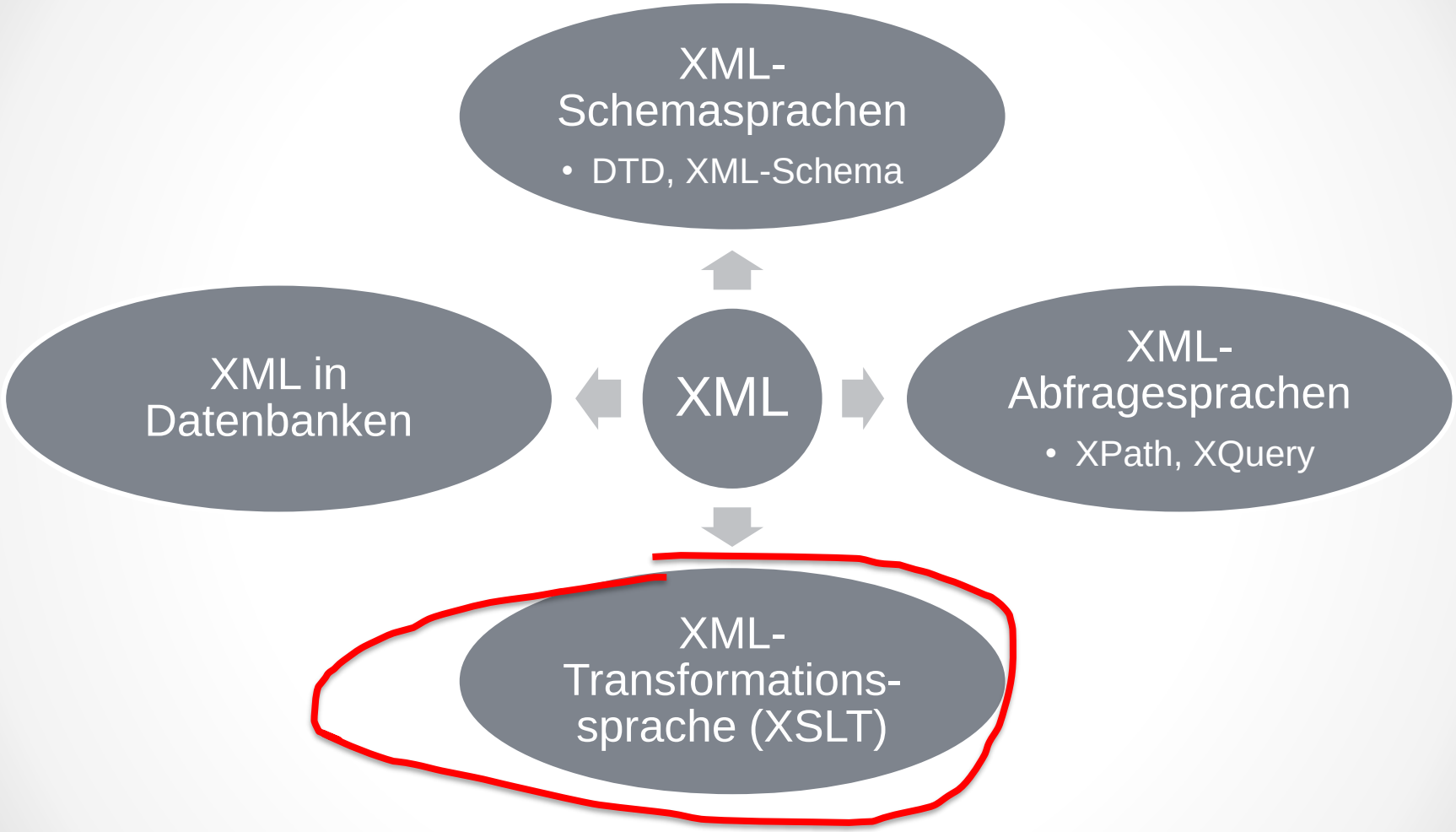
```
* {
    color: blue;
    background: lightgrey;
    margin: 5px;
    padding: 3px;
    border: 1px solid black;
    text-align: center;
    display: flex;
    flex-direction: column;
    width: 500px;
}

Projekt_Webseite {
    background: darkgrey;
}
```

Allerdings ist uns CSS nicht flexibel genug. Das geht bestimmt noch besser…

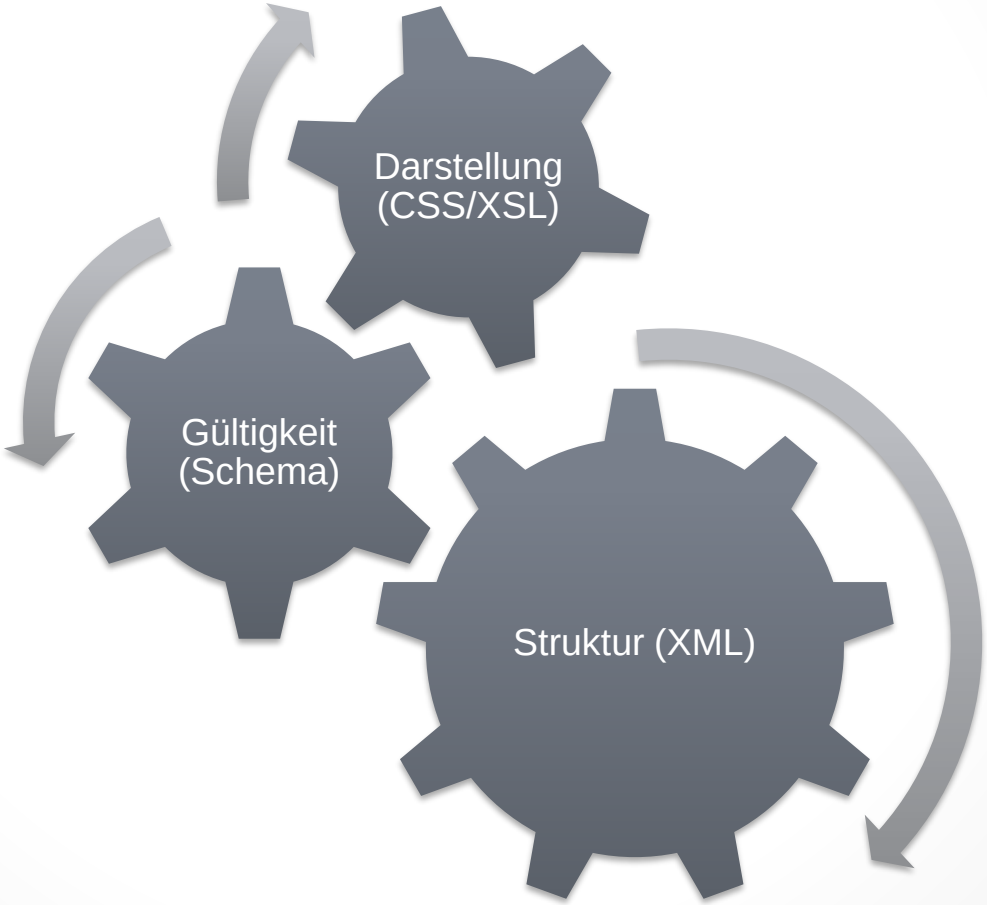
XSL

- XSL – Extensible Stylesheet Language
- XSL ist deutlich mächtiger als CSS
- gezielte Formatierung der Ausgabe
- z.B. FOP (Apache Software Foundation)
  - Formatting Objects Processor
  - gibt es als Java-Implementierung

...dafür fehlt uns leider die Zeit...


XML-
Schemasprachen
• DTD, XML-Schema
XML in
Datenbanken
XML
XML-
Abfragesprachen
• XPath, XQuery
XML-
Transformations-
sprache (XSLT)

...bisher...



Für einen wirklichen Mehrwert als Webtechnologie fehlt jetzt noch die Unterstützung für dynamische Inhalte.

XSL - Transformation

- Es geht darum XML-Daten dynamisch zu transformieren, bevor sie dargestellt werden.
  - Sortierung von Daten aus einer Datenbank
  - Umwandlung bestimmter Formate, z.B. nach HTML
  - Übersetzung zwischen verschiedenen Vokabularen
- XSL = Extensible Stylesheet Language

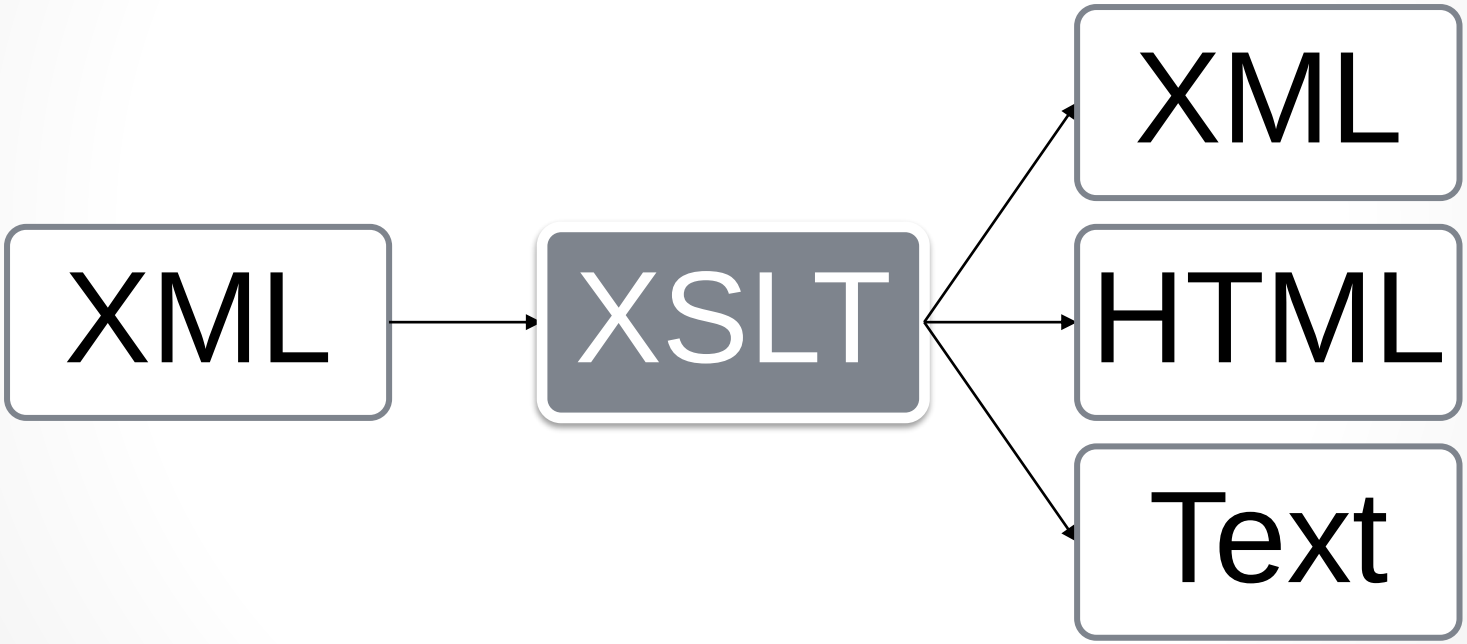
Workflow


Quelldokument
Transformation (XSLT)
Ergebnisdokument + Formatierte Ausgabe

XSLT vs. CSS

- XSL Transformationen ähneln Stylesheets – mit dem Unterschied, dass zusätzlich zur formatierten Ausgabe noch eine Umwandlung stattfinden kann.
- Darüber hinaus ist XSLT dynamischer als CSS.

Ausgabeformate



Grundlagen

- XSLT ist eine in XML definierte Sprache.
- DTDs und XML-Schema spielen bei der Transformation mit XSLT keine Rolle.
- Das Quelldokument muss wohlgeformt sein.
  - HTML geht also nicht, XHTML dagegen schon.
- Die Transformation wird durch einen so genannten XSLT-Prozessor durchgeführt.
  - Xalan (Apache Software Foundation)
  - Saxon (Open Source)
  - interne Prozessoren von XML-Editoren (Altova XMLSpy)
  - tlw. im Browser implementiert – Transformation auf dem Client

Nur ein kleines Bsp.

- Wir wenden als nächstes eine XSL-Transformation auf unser XML-Dokument an.
- Dabei soll eine Ausgabe in HTML erzeugt werden, die die Projekte aus der Liste darstellt.

Schritt 1: XSLT entwerfen

Prolog

```
<?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"/>
```

Template-Regeln entwerfen

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

```
<xsl:value-of select="@id"/>
```

```
<xsl:value-of select="Name"/>
```

Templates einbinden

```
<xsl:apply-templates select="/Projekt_Webseite/Projekt" />
```

Hinweis

- Template-Regeln definieren ein Suchmuster
  - `<xsl:template match="Projekt">`
- XSLT-Anweisungen steuern den XSLT-Prozessor
  - `<xsl:apply-templates select="/Projekt_Webseite/Projekt" />`
- Literale Ergebniselemente werden ausgegeben
  - `<html>...</html>`

Umfangreiche Sprache

- Wir können nicht vollständig auf den Sprachumfang von XSLT eingehen. Siehe [1] bei Interesse.

Schritt 2: XSL einbinden

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="Sample_02.xslt"?>
```

Das Ergebnis

- Wird das Beispieldokument jetzt im Browser geöffnet:
  - findet die Transformation wie im XSLT-File definiert statt
  - Ausgabeformat ist HTML
  - Literale werden ausgegeben (<html>, …) und bilden die HTML-Elemente
  - XSLT-Anweisungen binden die Daten aus dem XML-Dokument ein
  - XSLT-Templates bestimmten die Daten anhand eines Suchmusters

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="Sample_02.xslt"?>

<Projekt_Webseite Semester="WS2015">
  <Projekt id="p01" pl="pm001">
    <Name>Team_01</Name>
    <Teammember id="pm001">
      <Name>Mister X</Name>
      <Vorname>Joe</Vorname>
      <Rolle>Implementierung</Rolle>
    </Teammember>
    <Teammember id="pm002">
      <Name>Mister Y</Name>
      <Vorname>John</Vorname>
      <Rolle>Dokumenatation</Rolle>
    </Teammember>
    <Teamleiter>
      <Name />
    </Teamleiter>
    <Thema>X and Y</Thema>
    <CSS_Implementierung>
      <Style id="style01">
        <Stylename>1. Style</Stylename>
        <Beschreibung>Alle Farben dunkel</Beschreibung>
      </Style>
      <Style id="style02">
        <Stylename>2. Style</Stylename>
        <Beschreibung>Alle Farben hell</Beschreibung>
      </Style>
    </CSS_Implementierung>
  </Projekt>
</Projekt_Webseite>
```

Unser XML-Dokument

```
<?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>Projektliste</title>
            </head>
            <body>
                <h1>Liste der eingetragenen Projekte</h1>
                <xsl:apply-templates
select="/Projekt_Webseite/Projekt" />
            </body>
        </html>
    </xsl:template>
    <xsl:template match="Projekt">
        <div>
            <h2>
                <xsl:value-of select="@id"/>:<xsl:value-of
select="Name"/>
            </h2>
            <p>
                Thema: <xsl:value-of select="Thema"/>
            </p>
            <h3>Mitglieder</h3>
            <xsl:apply-templates
select="/Projekt_Webseite/Projekt/Teammember" />
        </div>
    </xsl:template>
            <xsl:template match="Teammember">
                    <p>
                        ID= <xsl:value-of select="@id"/> //
                        Name: <xsl:value-of select="Name"/>
                    </p>
            </xsl:template>
</xsl:stylesheet>
```

XSL-Transformation

Liste der eingetragenen Projekte

p01:Team_01

Thema: X and Y

Mitglieder

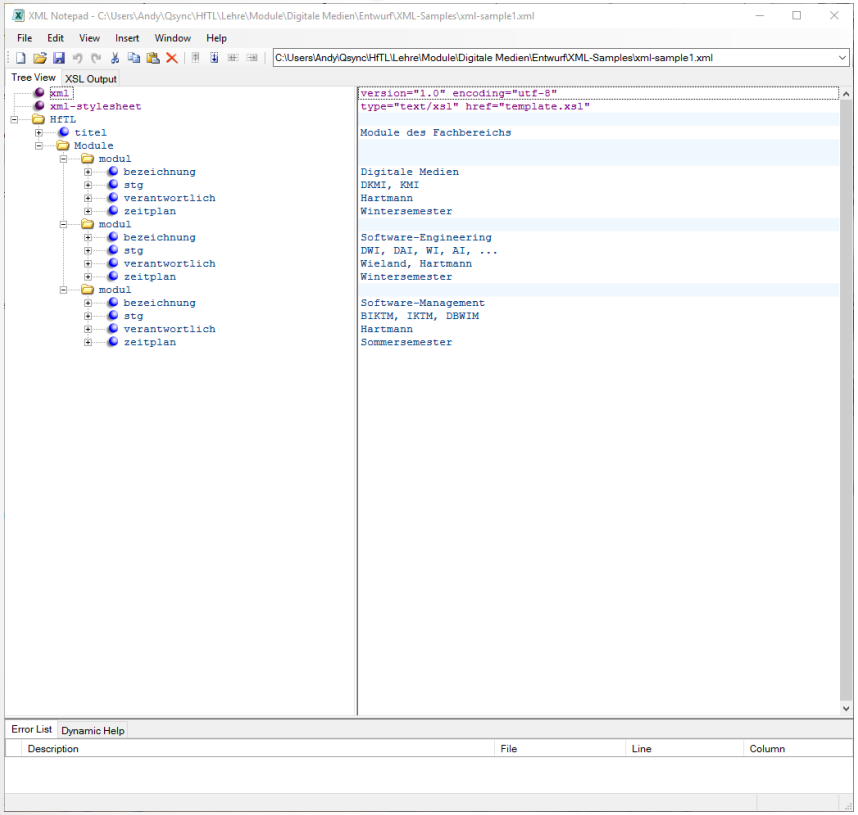
ID= pm001 // Name: Mister X

ID= pm002 // Name: Mister Y

Ausgabe in Safari

Wir sehen uns jetzt ein paar weitere Beispiele an…

Templates (XML-Dokument)



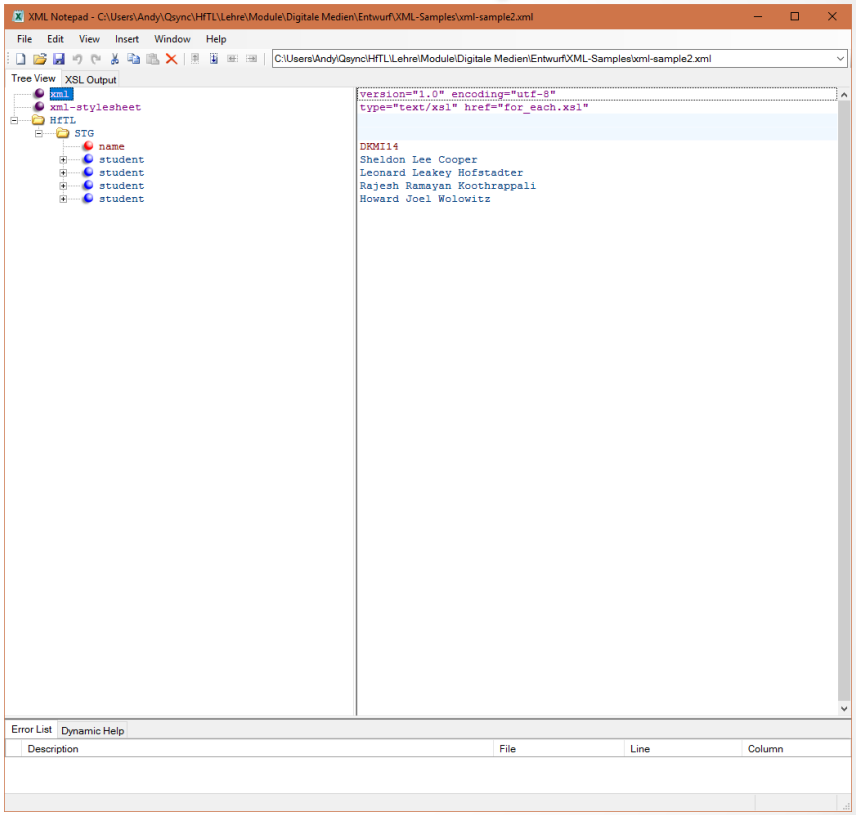

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="template.xsl" ?>
    <HfTL>
        <titel>Module des Fachbereichs</titel>
        <Module>
            <modul>
                <bezeichnung>Digitale Medien</bezeichnung>
                <stg>DKMI, KMI</stg>
                <verantwortlich>Hartmann</verantwortlich>
                <zeitplan>Wintersemester</zeitplan>
            </modul>
            <modul>
                <bezeichnung>Software-Engineering</bezeichnung>
                <stg>DWI, DAI, WI, AI, ...</stg>
                <verantwortlich>Wieland,
Hartmann</verantwortlich>
                <zeitplan>Wintersemester</zeitplan>
            </modul>
            <modul>
                <bezeichnung>Software-Management</bezeichnung>
                <stg>BIKTM, IKTM, DBWIM</stg>
                <verantwortlich>Hartmann</verantwortlich>
                <zeitplan>Sommersemester</zeitplan>
            </modul>
        </Module>
    </HfTL>
```

Templates
(XSLT + Ausgabe)

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

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

    <xsl:template match="titel">
        <h1><xsl:value-of select="." /></h1>
    </xsl:template>

    <xsl:template match="Module/modul">
        <p style="font-family:Arial,Helvetica,sans-serif; font-size:16px">
            <xsl:apply-templates />
        </p>
    </xsl:template>

    <xsl:template match="bezeichnung">
        <b style="color:blue"><xsl:value-of select="." />: </b><br
    </xsl:template>

    <xsl:template match="stg">
        <em><xsl:value-of select="." /></em>
    </xsl:template>

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

</xsl:stylesheet>
```

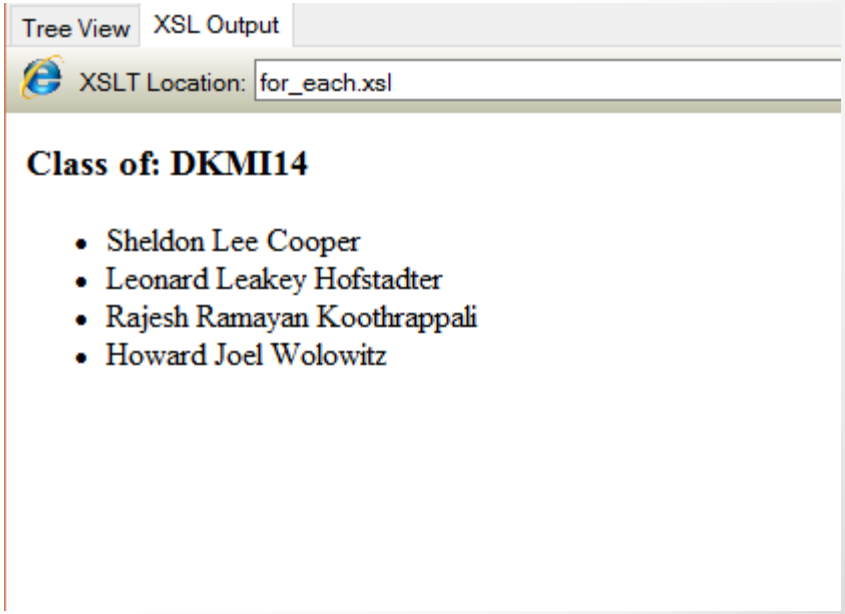
Schleife (XML-Dokument)

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl"
href="for_each.xsl" ?>
 <HfTL>
  <STG name="DKMI14">
   <student>Sheldon Lee Cooper</student>
   <student>Leonard Leakey
Hofstadter</student>
   <student>Rajesh Ramayan
Koothrappali</student>
   <student>Howard Joel Wolowitz</student>
  </STG>
 </HfTL>
```

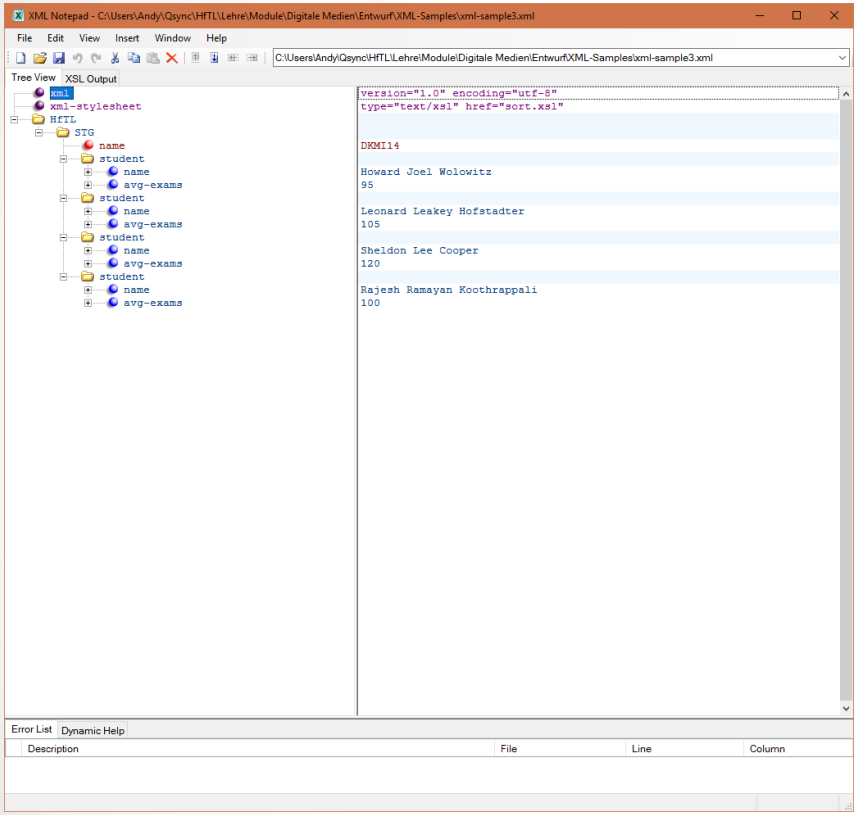


Schleife
(XSLT + Ausgabe)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <head>
 </head>
 <body>
  <xsl:apply-templates />
 </body>
 </html>
</xsl:template>
<xsl:template match="STG">
  <h3>Class of: <xsl:value-of select="@name" /></h3>
  <ul>
  <xsl:for-each select="student">
    <li><xsl:value-of select="." /></li>
  </xsl:for-each>
  </ul>
</xsl:template>

</xsl:stylesheet>
```



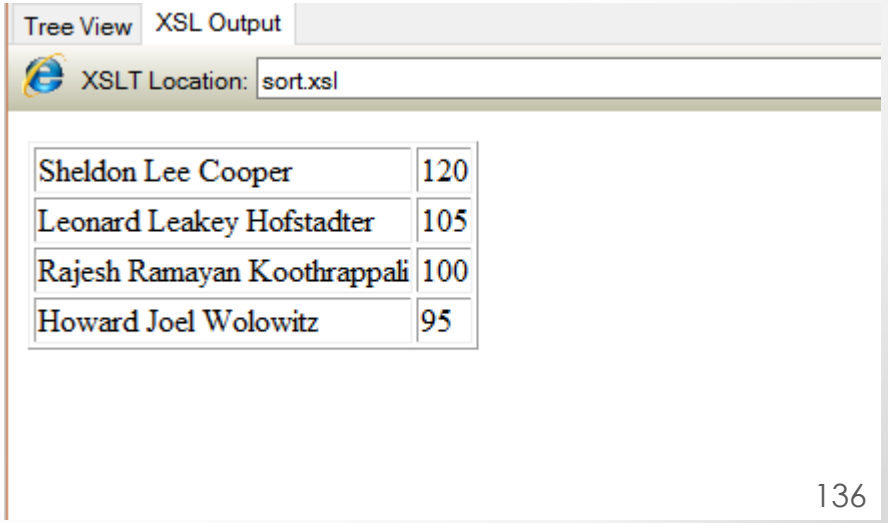
Sort (XML-Dokument)




```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl"
href="sort.xsl" ?>
    <HfTL>
        <STG name="DKMI14">
            <student>
                <name>Howard Joel
Wolowitz</name>
                <avg-exams>95</avg-exams>
            </student>
            <student>
                <name>Leonard Leakey
Hofstadter</name>
                <avg-exams>105</avg-exams>
            </student>
            <student>
                <name>Sheldon Lee
Cooper</name>
                <avg-exams>120</avg-exams>
            </student>
            <student>
                <name>Rajesh Ramayan
Koothrappali</name>
                <avg-exams>100</avg-exams>
            </student>
        </STG>
    </HfTL>
```

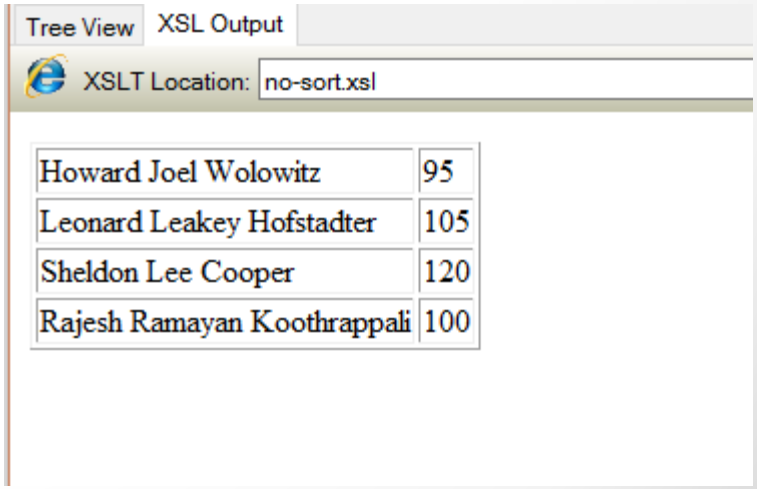
Sortierung
(XSLT + Ausgabe)

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <html>
  <head>
  </head>
  <body>
   <table border="1">
    <xsl:for-each select="HfTL/STG/student">
     <xsl:sort select="avg-exams" order="descending"
data-type="number" />
      <tr>
       <td>
        <xsl:value-of select="name" />
       </td>
       <td>
        <xsl:value-of select="avg-exams" />
       </td>
      </tr>
    </xsl:for-each>
   </table>
  </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```



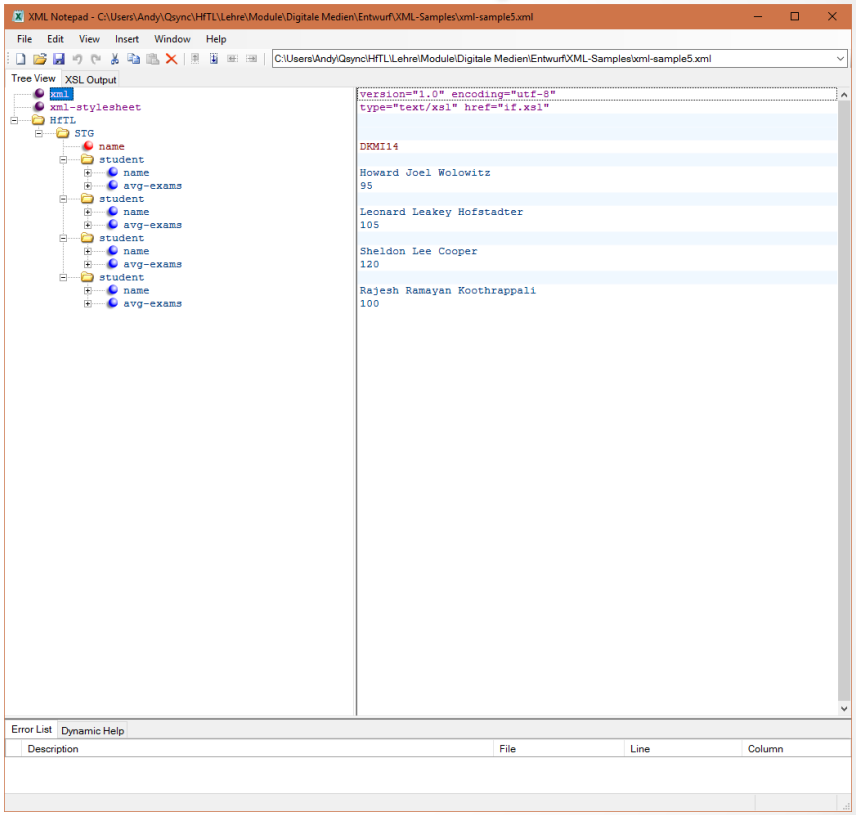
```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
        <head>
        </head>
        <body>
            <table border="1">
                <xsl:for-each select="HfTL/STG/student">
                    <tr>
                        <td>
                            <xsl:value-of select="name" />
                        </td>
                        <td>
                            <xsl:value-of select="avg-exams" />
                        </td>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Wie eben, ohne Sortierung



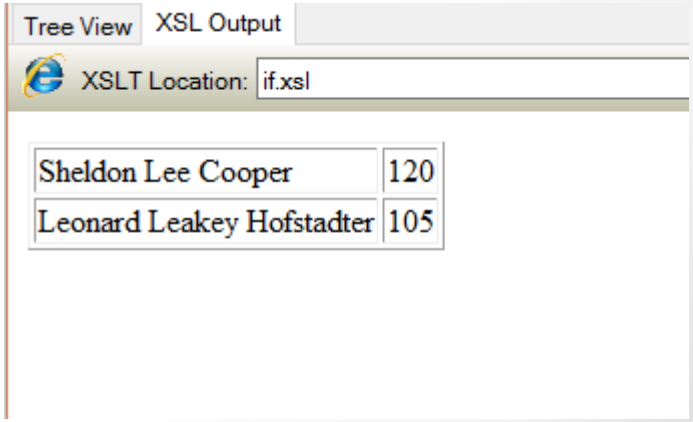
Bedingung (XML-Dokument)

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="if.xsl" ?>
    <HfTL>
        <STG name="DKMI14">
            <student>
                <name>Howard Joel Wolowitz</name>
                <avg-exams>95</avg-exams>
            </student>
            <student>
                <name>Leonard Leakey
Hofstadter</name>
                <avg-exams>105</avg-exams>
            </student>
            <student>
                <name>Sheldon Lee Cooper</name>
                <avg-exams>120</avg-exams>
            </student>
            <student>
                <name>Rajesh Ramayan
Koothrappali</name>
                <avg-exams>100</avg-exams>
            </student>
        </STG>
    </HfTL>
```



Bedingung
(XSLT + Ausgabe)

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
        <head>
        </head>
        <body>
            <table border="1">
                <xsl:for-each select="HfTL/STG/student">
                    <xsl:sort select="avg-exams"
order="descending" data-type="number" />
                    <xsl:if test="avg-exams &gt;100">
                    <tr>
                        <td>
                            <xsl:value-of select="name" />
                        </td>
                        <td>
                            <xsl:value-of select="avg-exams" />
                        </td>
                    </tr>
                    </xsl:if>
                </xsl:for-each>
            </table>
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```



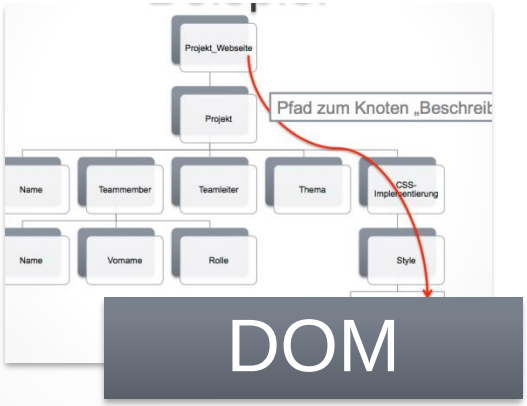
Entwickeln Sie ein Verständnis für die Transformation von XML-Dokumenten. Nur so lassen sich wirklich dynamische Inhalte im Web modellieren!

DOM und SAX

Verarbeitungsschnittstellen für XML-Dokumente

Verarbeitung

- XML-Anwendung
- Benötigt einen Parser
  - Nicht vergessen: ein Textdokument kommt zunächst als serieller Datenstrom an (File, HTTP,...)
- Parser können auch validieren
- 2 Ansätze:



DOM

SAX

DOM

- Der Parser baut eine hierarchische Baumstruktur als Speichermodell auf
- Typisch: Out-Trees
- Vgl. HTML

SAX

- Der Parser liest den Datenstrom sequenziell und liefert Events an die XML-Anwendung.
- Es ist nicht notwendig, das ganze XML-Dokument im Speicher abzubilden.
- Callback-Methoden und Event-Handler reagieren auf die Events.
- Beispiel:
  - ContentHandler
    - startDocument( ); endDocument( ); startElement( );

Was Sie sich merken sollten...

Zusammenfassung

- XML ist eine Metasprache
- XML ist nicht gleich HTML
- mittels XML lassen sich Daten strukturiert bereitstellen
- XML ist wohlgeformt
- XML-Schema und DTD ermöglichen die Validierung der Daten
- XPath ermöglicht die Addressierung von XML-Elementen
- mittels CSS und XSL kann die Darstellung gesteuert werden
- XSLT dient der Transformation und formatierten Ausgabe

XML in der Praxis

- Prominente Beispiele
  - OpenStreetMap
  - OSIS (Annotationen in der Literatur)
  - SVG
  - OpenDocument
  - ...

Ende

• • •

Danke!