# Softwareentwicklung 4

## Namespaces und Validität

Dominik Dolezal

Höhere Lehranstalt für Informationstechnologie

20. Februar 2017

# Inhalt

tgm
Die Schule der Technik

# Namespaces

*Namespaces* verhindern Namenskonflikte, wenn mehrere XML-Sprachen in einem Dokument gespeichert werden

```
<list>
  <h:table
    xmlns:h="http://www.w3.org/TR/html4">
    <h:tr>
      <h:td>Name</h:td>
      <h:td>Preis</h:td>
    </h:tr>
  </h:table>
  <f:table
    xmlns:f="http://www.w3schools.com/furniture">
    <f:name>Esstisch</f:name>
    <f:price currency="EUR">79</f:price>
  </f:table>
</list>
```

# Default Namespaces

- Namespace ohne Präfix
- Syntax: `xmlns="URI"`
- Gilt für: Aktuelles Element + alle Kindelemente ohne Präfix

```
<list
xmlns="http://www.w3.org/TR/html4"
xmlns:f="http://www.w3schools.com/furniture">
  <table>
    <tr>
      <td>Name</td>
      <td>Preis</td>
    </tr>
  </table>
  <f:table>
    <f:name>Esstisch</f:name>
    <f:price currency="EUR">79</f:price>
  </f:table>
</list>
```

# Validität

tgm
Die Schule der Technik

Korrektheit eines XML-Dokuments ist über zwei Eigenschaften definiert:

- **Wohlgeformtheit (well-formedness)**
  Einhaltung der allgemeinen XML-**Syntax**, z.B.
  - Genau 1 Wurzelelement
  - Keine Überlappungen der Tags
  - Korrekte Attribute
- **Gültigkeit / Validität (validity)**
  Dokument ist wohlgeformt und entspricht einer bestimmten **Grammatik**, definiert durch entweder
  - einer DTD oder
  - einem XML Schema

Für den Datenaustausch ist es sinnvoll, eine DTD bzw. ein Schema zu verwenden, um die Gültigkeit eines XML-Dokuments überprüfen zu können

# DTD Beispiel

tgm
Die Schule der Technik

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tracklist [
<!ELEMENT tracklist (track*, interpret*)>

<!ELEMENT track (name, length, review*)>
<!ATTLIST track id   ID  #REQUIRED
        style  (Rock|Funk|HipHop) "Rock">
<!ELEMENT name          (#PCDATA)>
<!ELEMENT length        (#PCDATA)>
<!ELEMENT review        (#PCDATA | link)*>
<!ELEMENT link       (#PCDATA)>
<!ATTLIST link  ref IDREF #IMPLIED>

<!ELEMENT interpret   (band | soloartist)>
<!ATTLIST interpret id  ID  #REQUIRED>
<!ELEMENT band       (name, person+)>
<!ELEMENT soloartist     (person)>

<!ELEMENT person       (firstname?, name, instrument*)>
<!ELEMENT firstname    (#PCDATA)>
<!ELEMENT instrument     (#PCDATA)>
<!ATTLIST person gender   (male|female) "female">
]>
<tracklist>...</tracklist>
```

# DTD

tgm
Die Schule der Technik

- DTDs werden oft als veraltet bezeichnet
- Sie sind aber nach wie vor weit verbreitet (z.B. HTML)
- DTDs verwenden leider eine andere Syntax als XML
- DTDs sind nicht so mächtig wie XML-Schemas
  - Keine Datentypen
  - Kein Namespace-Support
  - Anzahl an Kindelementen kaum beschränkbar
  - Nicht über DOM manipulierbar
  - Text innerhalb von Elementen kann nicht eingeschränkt werden

# XSD

tgm
Die Schule der Technik

- XML Schema Definition
- Basiert auf XML
- Unterstützung von Namespaces
- Unterstützung von Datentypen
- Unterstützung von Kardinalitäten
- u.v.m

# XSD Beispiel

tgm
Die Schule der Technik

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading"
          type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

# XSD Beispiel

tgm
Die Schule der Technik

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

<note
  xmlns="http://www.w3schools.com"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

# xs:schema

tgm
Die Schule der Technik

- Wurzelelement einer Schema-Datei
- Attribute:
  - XMLSchema Namespace
  - Namespace der XML-Datei (Instanz)
  - Default-Namespace (eher zu vermeiden)
  - Kann Namespaces der Instanz vorschreiben

```
<?xml version="1.0"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
...
</xs:schema>
```

# simpleType

tgm
Die Schule der Technik

- Wir unterscheiden zwischen complexType und simpleType
- simpleType sind einfache Elemente
- Sie enthalten keine Attribute oder Kindelemente
- Somit kann nur Text zwischen den Tags stehen
- Beispiel aus Schema:

```
<xs:element name="to" type="xs:string"/>
```

- XML-Instanz:

```
<to>Tove</to>
```

# simpleType: type

- Es gibt eingebaute primitive **Datentypen**

```
<xs:element name="lastname"
    type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
<xs:element name="start" type="xs:time"/>
<xs:element name="account"
    type="xs:decimal"/>
<xs:element name="discount"
    type="xs:boolean"/>
```

- XML-Instanz:

```
<lastname>Renoir</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
<start>10:03:30</start>
<account>-234.50</account>
<discount>true</discount>
```

# simpleType: default & Wertebereiche

tgm Die Schule der Technik

- Es gibt die Möglichkeit, **Standardwerte** für leere Elemente zu definieren

```
<xs:element name="color" type="xs:string"
    default="red"/>
```

- Es können **Wertebereiche** festgelegt werden

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# simpleType: Eigene Datentypen

tgm
Die Schule der Technik

- Alternativ: **Eigenen Datentyp** erstellen und mehrfach verwenden

```
<xs:element name="age" type="ageType" />

<xs:simpleType name = "ageType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="120"/>
  </xs:restriction>
</xs:simpleType>
```

# simpleType: enumeration

tgm
Die Schule der Technik

- **Aufzählungen** (Wertemengen) können definiert werden

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# simpleType: pattern & length

- Reguläre Ausdrücke (**Regular Expressions**) möglich

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Länge** des Inhalts kann eingeschränkt werden (auch min-max)

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

# complexType: attribute

tgm
Die Schule der Technik

- Komplexe Typen erlauben das Definieren von Attributen und Kindelementen sowie gemischten Inhalten (Text + Kindelemente)
- Beispiel eines Attributes im Schema:

```
<xs:attribute name="lang" type="xs:string"/>
```

- XML-Instanz:

```
<lastname lang="EN">Smith</lastname>
```

- Attribute können optional (`use="optional"`) oder verpflichtend sein (`use="required"`)
- Attribute können Standardwerte haben (`default="123"`) oder fixiert sein (`fixed="10"`)

# complexType: sequence

tgm
Die Schule der Technik

- Kindelemente in bestimmter **Reihenfolge**

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname"
          type="xs:string"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</employee>
```

# complexType: all

- Jedes Element muss **einmal** vorkommen, Reihenfolge egal

```
<xs:element name="employee">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname"
          type="xs:string"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <lastname>Smith</lastname>
  <firstname>John</firstname>
</employee>
```

# complexType: choice

- **Eines** der Elemente muss vorkommen

```
<xs:element name="employee">
  <xs:complexType>
    <xs:choice>
      <xs:element name="firstname"
          type="xs:string"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <firstname>John</firstname>
</employee>
```

# complexType: maxOccurs & minOccurs

tgm
Die Schule der Technik

- **Mehrmaliges** Vorkommen

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname"
          type="xs:string" maxOccurs="3"
          minOccurs="0"/>
      <xs:element name="lastname"
          type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<employee>
  <firstname>John</firstname>
  <firstname>James</firstname>
  <lastname>Smith</lastname>
</employee>
```

# complexType: Beispiele

tgm
Die Schule der Technik

- **Attribut** (ohne Kindelemente)

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid"
        type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<product prodid="1345" />
```

# complexType: Beispiele

tgm Die Schule der Technik

- Element mit **Textinhalt und Attribut** (ohne Kindelemente)

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country"
            type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<shoesize country="france">35</shoesize>
```

# complexType: Beispiele

tgm
Die Schule der Technik

- Element mit **gemischten** Inhalt

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name"
          type="xs:string"/>
      <xs:element name="orderid"
          type="xs:positiveInteger"/>
      <xs:element name="shipdate"
          type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- XML-Instanz:

```
<letter>Dear Mr.<name>John Smith</name>.
    Your order <orderid>1032</orderid> will
    be shipped on <shipdate> 2001-07-13
    </shipdate>.</letter>
```

# XPath

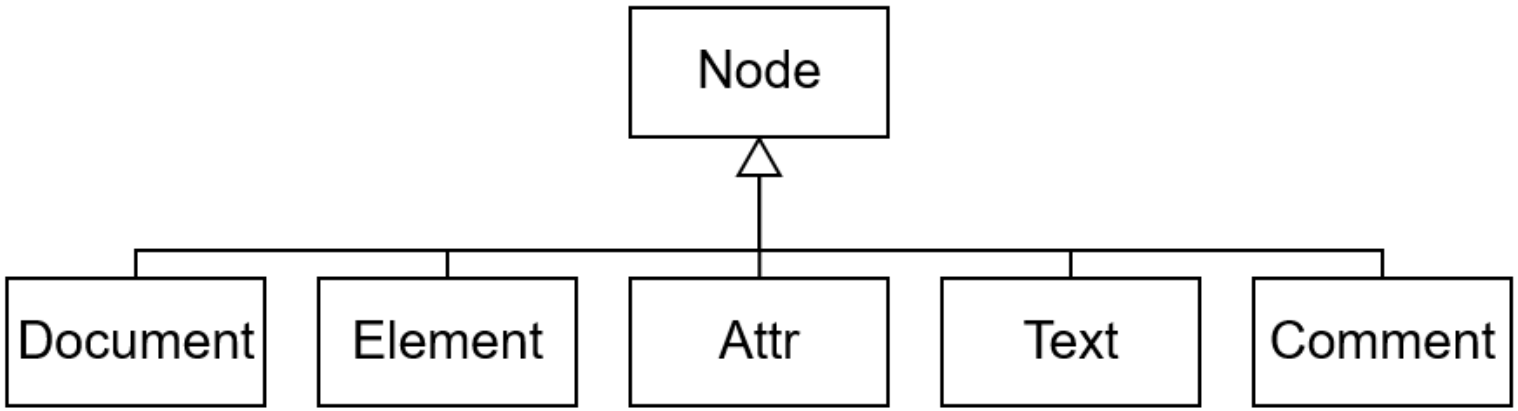
*„Das SELECT der XML-Welt“*

- Syntax zur Auswahl von bestimmten Teilen eines XML-Dokumentes
- Verwendet Pfad-Angaben
- Beinhaltet eine Bibliothek von Standard-Funktionen
- Wird massiv von XML-basierten Techniken verwendet (z.B. XSLT)

# XPath Knoten

tgm
Die Schule der Technik

Wie in DOM unterscheiden wir zwischen verschiedenen Arten von **Knoten** (Nodes)




Grundidee: Die XPath-Anfrage deklariert eine Navigation im DOM-Baum

# XPath Knoten

tgm
Die Schule der Technik

Unsere Knoten (Nodes) können wir selektieren anhand von

- Element- und Attributnamen
- Typ (Element, Attribut, Kommentar, etc.)
- Inhalt oder Wert von Knoten
- Beziehungen zwischen Knoten

Das Ergebnis einer Abfrage ist immer eine geordnete Menge von Knoten (Kontext)

# XPath Knotenbeziehungen

tgm
Die Schule der Technik

Betrachten wir folgendes Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

# XPath Knotenbeziehungen

tgm
Die Schule der Technik

Folgende Knotenbeziehungen können wir identifizieren:

- Atomare Knoten / Werte (J K. Rowling oder „en“)
- Elternknoten = parent nodes, parents (title → book)
- Kindknoten = child nodes, children (bookstore → book)
- Geschwisterknoten = siblings (title → author, year, price)
- Vorfahren = ancestors (title → book, bookstore)
- Nachfahren = descendants (bookstore → book, title, author, year, price)

# XPath Selektion

tgm
Die Schule der Technik

| **Ausdruck** | **Beschreibung** |
|---|---|
| *nodename* | Wählt alle Kindknoten vom Knoten mit *nodename* |
| / | Wählt den Dokumentknoten aus und stellt außerdem das Trennzeichen zwischen Knoten dar |
| // | Wählt alle Kindknoten des aktuellen Knotens, egal wo sie stehen (unabhängig von ihrer Verschachtelung) |
| . | Wählt aktuellen Knoten aus |
| .. | Wählt Elternknoten des aktuellen Knotens aus |
| @ | Wählt Attribute |

# XPath Selektion

tgm
Die Schule der Technik

Betrachten wir folgendes Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>
  <book>
    <title lang="eng">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="eng">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

# XPath Selektion

tgm
Die Schule der Technik

| Ausdruck | Beschreibung |
|---|---|
| bookstore | Wählt bookstore aus |
| /bookstore | Wählt bookstore über absoluten Pfad aus |
| bookstore/book | Wählt alle book-Elemente aus, die ein Kindelement von bookstore sind |
| //book | Wählt alle book-Elemente aus, egal wo sie sich befinden |
| bookstore//book | Wählt alle book-Elemente aus, die Nachfahren von bookstore sind |
| //@lang | Wählt alle Attribute aus, die lang heißen |

Achtung: Üblicherweise ist der Start-Kontext das Dokument – über die beschriebenen Selektionsmöglichkeiten wird der Kontext schrittweise während des Verarbeitens gewechselt

# XPath Prädikate

tgm
Die Schule der Technik

- Ähnlich dem „Where“-Teil eines Select-Statements
- Über Prädikate können Einschränkungen getroffen werden ohne den Kontext zu verändern
- Es können bestimmte Knoten anhand ihrer Beziehungen ausgewählt werden
- Es können Knoten mit bestimmten Inhalten ausgewählt werden
- Prädikate werden in eckigen Klammern angegeben

# XPath Prädikate

tgm Die Schule der Technik

| Ausdruck | Beschreibung |
|---|---|
| /bookstore/book[1] | Wählt das erste Kindelement von bookstore mit dem Namen book |
| /bookstore/book[last()] | Letzes Kindelement von bookstore mit dem Namen book |
| /bookstore/book[last()-1] | Vorletzes Kindelement von bookstore mit dem Namen book |
| /bookstore/book[position()<3] | Die ersten zwei Kindelemente von bookstore mit dem Namen book |
| //title[@lang] | Alle Elemente mit dem Namen title, die ein Attribut mit dem Namen lang besitzen |
| //title[@lang='eng'] | Alle Elemente mit dem Namen title, die ein Attribut mit dem Namen lang und dem Wert „en“ besitzen |
| /bookstore/book[price>35.00] | Alle Kindelemente von bookstore mit dem Namen book, deren Kindelement price einen Wert größer als 35 besitzt |
| /bookstore/book[price>35.00]/title | Das title Element von allen book-Elementen (die Kindelemente von bookstore sind), deren Kindelement price einen Wert größer als 35 besitzt |

# XPath Wildcards

tgm
Die Schule der Technik

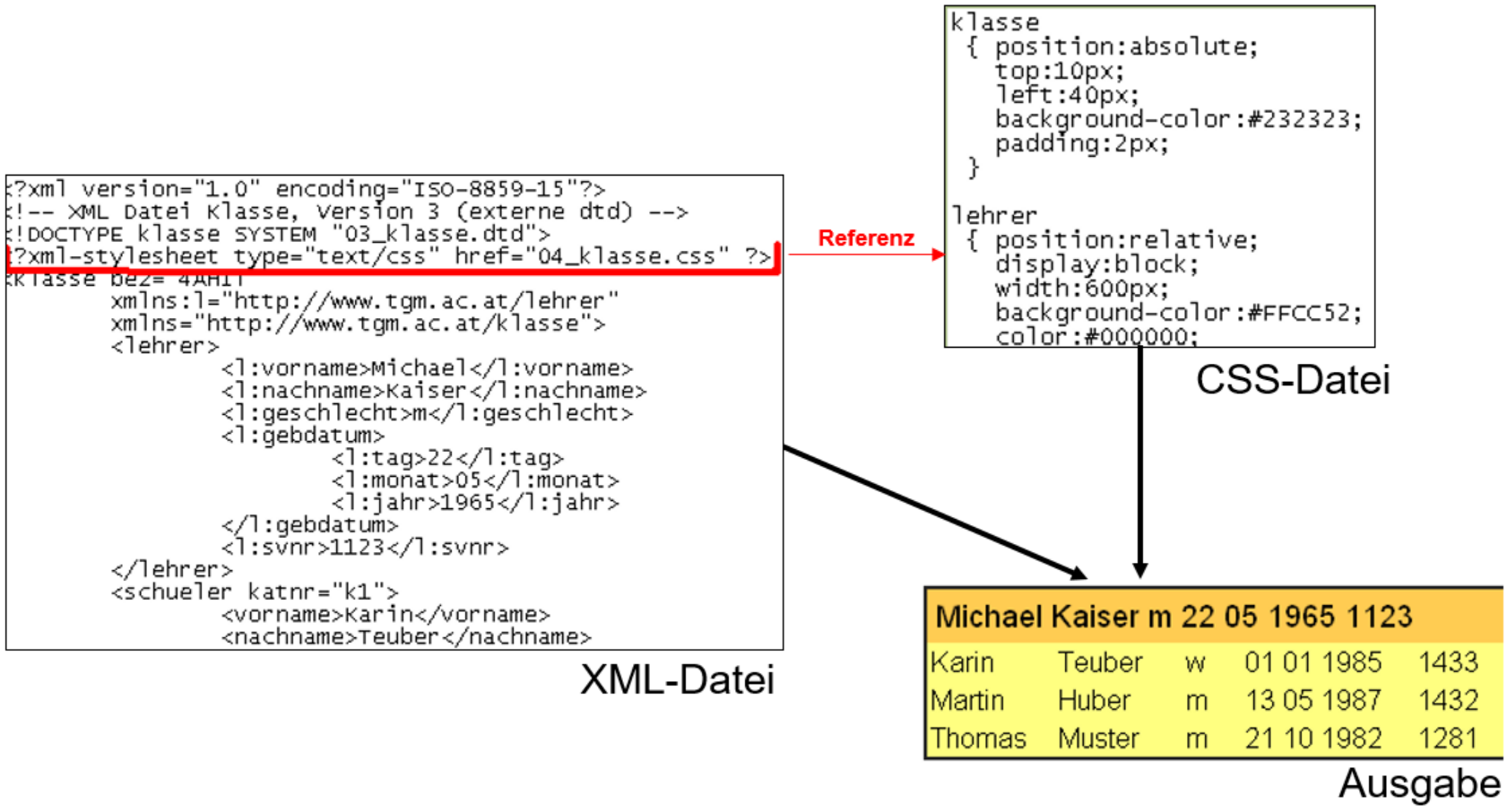
| Ausdruck | Beschreibung |
|---|---|
| * | Jedes Element |
| @* | Jedes Attribut |
| node() | Jeder Knoten, egal welchen Typs |

| Ausdruck | Beschreibung |
|---|---|
| bookstore/* | Jedes Kindelement von bookstore |
| //* | Jedes Element im Dokument |
| //title[@*] | Alle title-Elemente, die ein Attribut haben |

# XML CSS

XML-Dokumente können mithilfe von CSS formatiert werden

Die Schule der Technik



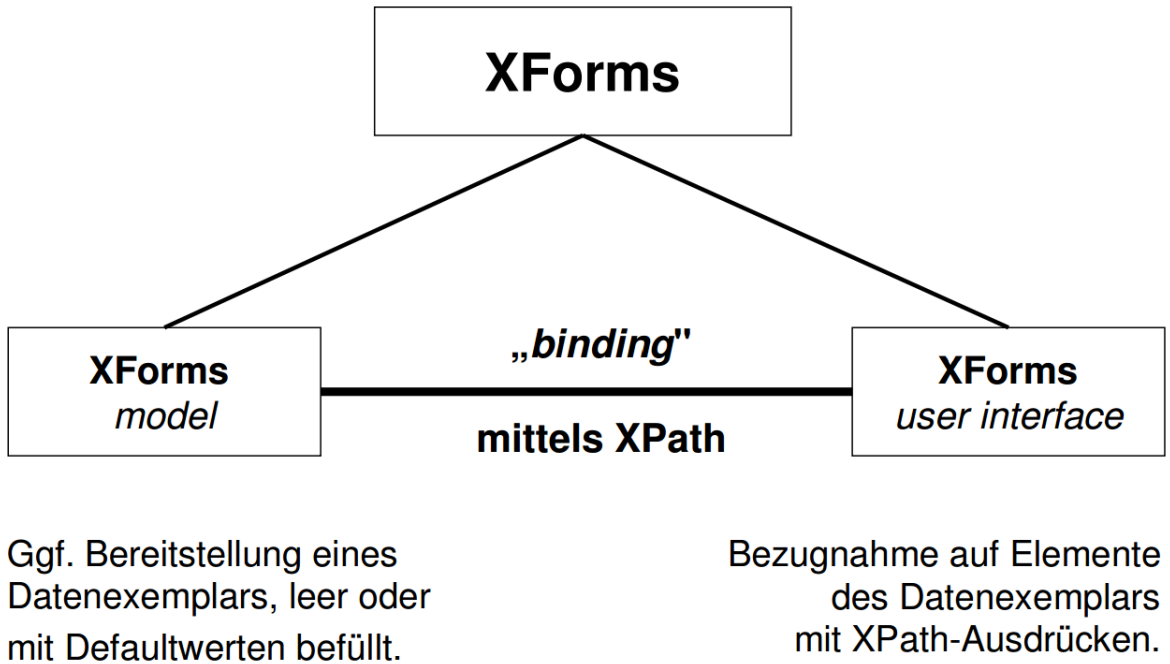

Nachteile: Keine Attribute, Filterung und Sortierung

# XSLT

- eXtensible Stylesheet Language Transformation
- XML-Files können über XSLT in andere Formate umgewandelt werden (HTML-Seiten, anderes XML-Format mit anderen Schema, beliebige andere Textformate)
- XSL bezeichnet den Standard zur Formatierung von XML, XSLT kann als Programmiersprache gesehen werden
- XSLT-Dokumente sind XML-Dateien
- Sie verwenden XPath zur Navigation
- Sie benötigen einen Prozessor (moderne Webbrowser, XALAN)
- Beispiele: https://www.w3schools.com/xml/xsl_examples.asp

# XForms

Idee: XML-Forumlare statt HTML-Formulare, Trennung von Daten und Darstellung, Geräteunabhängigkeit, weniger Skripte, Berechnungen, Prüfungen




Nachteile: Plugins nötig, keine generelle Unterstützung, relativ neu

# Zusammenfassung

tgm
Die Schule der Technik

## XPath, XSLT und XForms

- XPath dient der Navigation und Filterung innerhalb von XML-Dokumenten
- Mithilfe von XSLT können XML-Dokumente in andere XML-Dokumente, HTML-Seiten und beliebige Textdateien transformiert werden
- Die XML-Familie umfasst weitere Standards wie XForms und wird ständig erweitert