7437 - EDI und E-Business Standards, 4661 – E-Business: Standards und Automatisierung

Electronic

Data

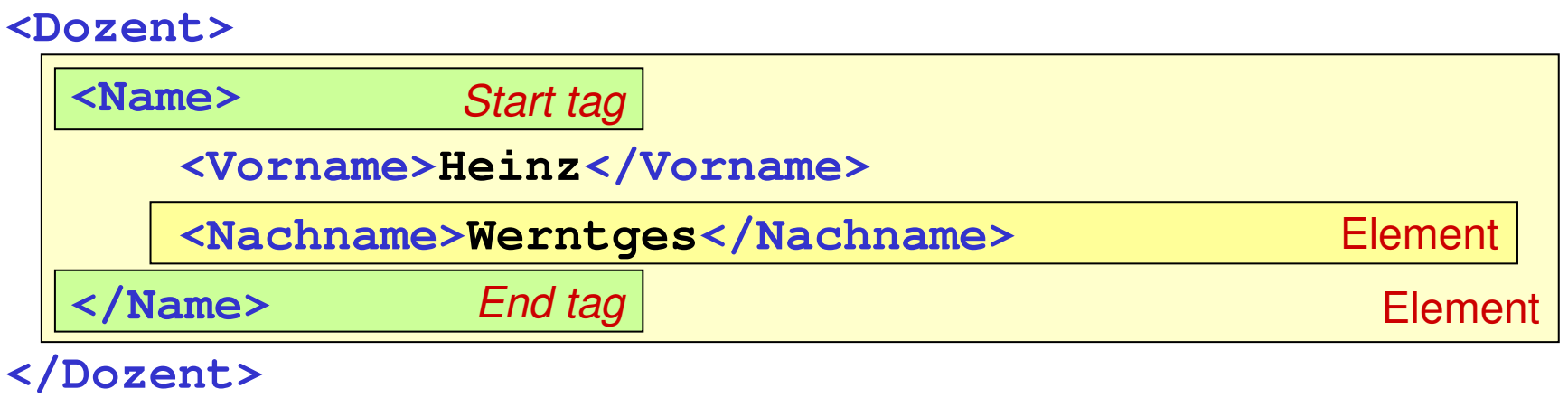
Interchange

(Elektronischer Datenaustausch)

XML-Standards

Ein Crash-Kurs für Anwender

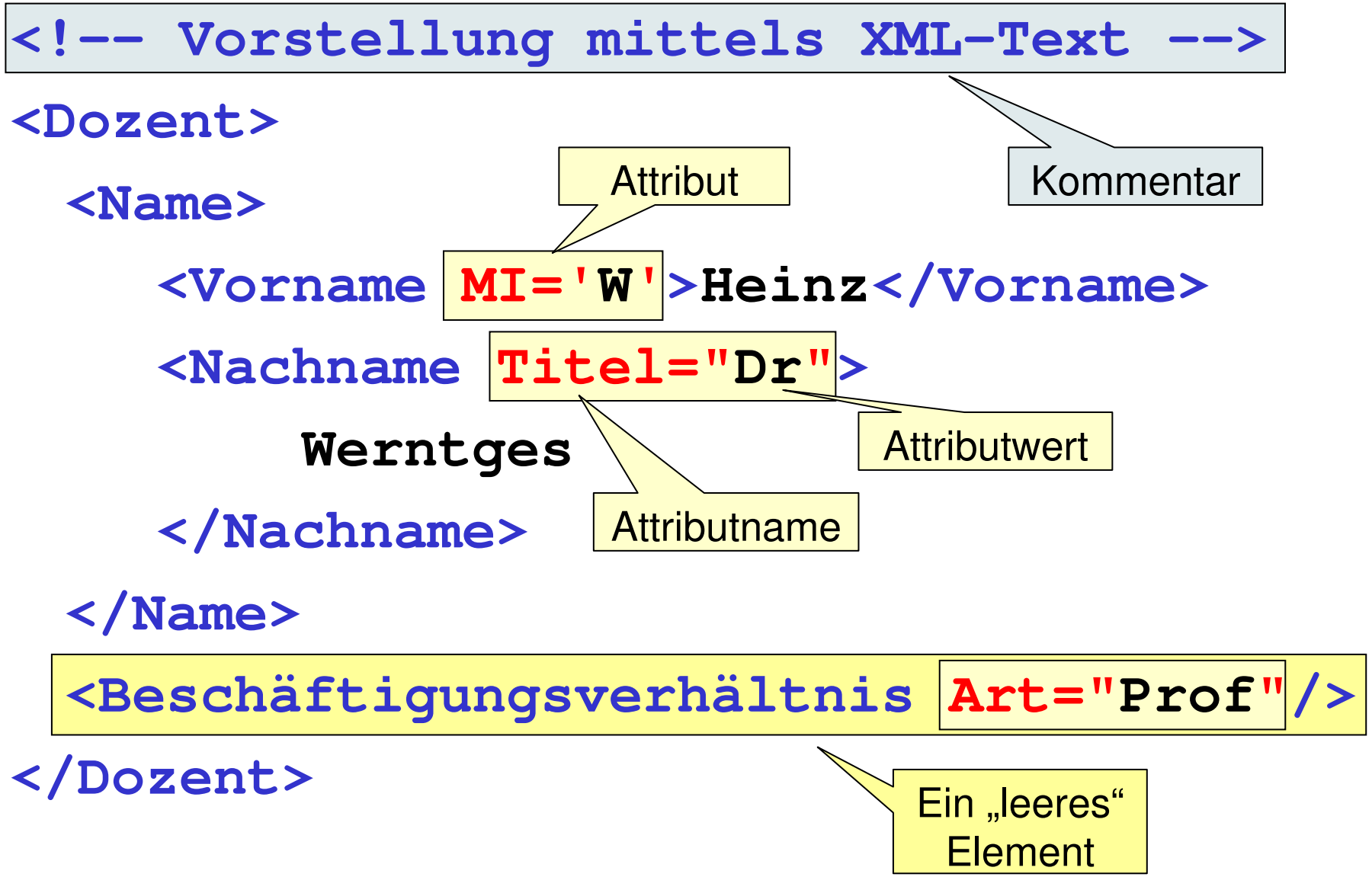
XML – ganz einfach



Ein „wohlgeformtes“ XML-Dokument

- Es besteht aus einem Element (namens „Dozent“)
- Dieses enthält Unter-Elemente usw.: Baum-Struktur!
- Elemente können auch Freitext enthalten

XML – immer noch einfach



Einschub: XML im Vergleich zu HTML

- XML sieht fast so aus wie HTML
  - Eine Auszeichnungssprache (*markup language*)
    - Elemente, "tags", Attribute
  - Grund: Gemeinsame Herkunft SGML
- Allgemeine Unterschiede
  - Klein-/Groß-Schrift unterscheiden!
  - Attribute: Immer in Anführungszeichen
    (' als auch " möglich, aber stets paarweise!)
- Elemente
  - anscheinend beliebige Element-Namen möglich
  - neu: *empty elements*
  - *Elemente sind immer zu schließen!*

Immer noch XML! - Immer noch einfach?

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Vorstellung mittels XML-Text -->
<?xml-stylesheet href="demo.css" type="text/css"?>
<!DOCTYPE Dozent SYSTEM "demo.dtd" [
<!ENTITY % ISO-Latin1-chars SYSTEM "iso8859-1.ent">
%ISO-Latin1-chars;
<!ENTITY Abk "Besch&auml;ftigungsverh&auml;ltnis">
]>
<Dozent>
   <Name>
       <Vorname MI='W'>Heinz</Vorname>
       <Nachname Titel="Dr">Werntges</Nachname>
   </Name>
   <&Abk; Art="Prof"/> <!-- *SYNTAXFEHLER* -->
</Dozent>
```

XML im Alltag: Beispiele

- WWW
  - XHTML, SVG, MathML, WML, …
- Systemprogrammierung
  - Konfigurationsdateien
- Office-Suites
  - Neu: OpenDocument
- 2D-Grafik, Animationen
  - SGV, SMIL
- Laborautomation
  - NIST: AniML
- EDI, E-Business
  - ebXML, Web Services, SOA, …
- Newsticker
  - RSS

Heute wird XML praktisch „überall“ eingesetzt.

XML-Technologie durchdringt alle Anwendungen strukturierten, plattform- und sprachunabhängigen Datenaustauschs, ähnlich wie TCP/IP die Computervernetzung eroberte.

XML-Tech.: Langfristig stabiles Basiswissen für Informatiker

XML-Spezifikationen des W3C: Übersicht

Einordnung	Spezifikation / Standard
Anwendungs-sprachen	XQuery, **XSLT**, **XSL-FO**; **SVG**; **XHTML**
Neue XML-Grundlagen	**XML Schema**; XPointer/**XLink**, XBase, **XInclude**; **XPath**
Hilfs-Spezifikationen	**Namensräume**, **Stylesheet-Einbindung**, XML Infoset, **CSS2**
Fundament	**XML 1.0**, XML 1.1 (schließt **DTD**s ein)
Grundlagen	**Unicode** incl. UTF-8, UTF-16; **ISO-Codes** für Länder, Sprachen

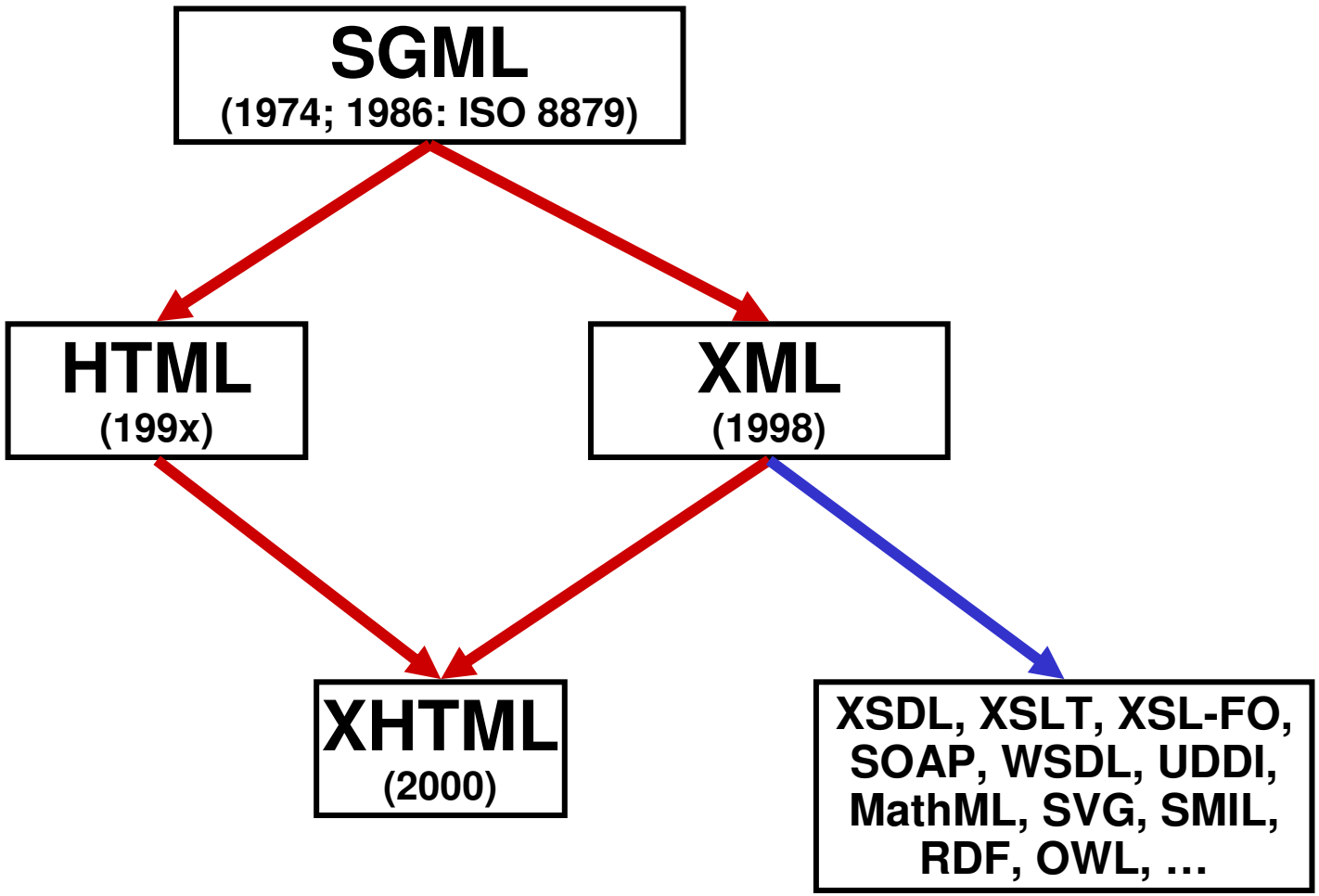
XML-Technologien: Übersicht, Einordnung

Zweck	Spezifikation / Standard
Verfassen von Dokumenten	DocBook, TEI; **XML**
Strukturierung (Dokumenten-Def.), Validierung	**DTD**, **XML Schema**, Relax NG
Darstellen	**CSS**, DSSSL, XSL-FO, **XHTML**
Zeichnen (2D), Szenendef. (3D)	**SVG**, X3D (VRML-Nachfolger)
Transformieren	**XSLT**
Assoziieren	RDF, XTM (Topic Maps)
Verlinken	**XBase**, **XLink**, XInclude
Suchen, Auswählen	**XPointer**, **XPath**, XQuery
Programmieren	**SAX, DOM**

Anmerkungen:

- Frei nach http://xml.oreilly.com/news/dontlearn_0701.html
- **Fett** gedruckte Standards sind Themen des XML-Kurses (Liste KAT)

Von HTML zu XHTML



Von HTML zu XHTML

- XML-Dokument
  - Abstrakte Sicht: Bewerteter Graph in Baumform
  - Modell: XML Information Set (http://www.w3.org/TR/infoset)
  - Übliche Darstellung: XML-Syntax (http://www.w3.org/TR/xml10)
    - Bild eines Dokumentenbaumes: Siehe unten, Bild zum Beispiel
      - Ein Dokumentenknoten (root)
      - Ein Dokumentenelement
      - Kindelemente, Text/Char-Elemente
      - Werte: Mengen (von Attributen), Verweise, etc.

- Dokumenttyp
  - Eine Menge von Regeln, die präzise beschreibt, wie Dokumente dieses Typs aufzubauen sind **(welche Elemente sind wo wie oft zulässig, welche Attribute und Datentypen besitzen sie, etc.)**.
  - Definition per „DTD“, W3C XML Schema, RELAX NG, …

Validierung

- Analogie
  - Dokumententyp ⟷ Klasse
  - Dokument, D.-Exemplar ⟷ Objekt

- Validierung – Qualitätssicherung von Dokumenten
  - Prüfung einer Dokumenteninstanz gegen ihr(e) zugrunde liegende(s) DTD/Schema
    - Werden nur zugelassene Elemente verwendet?
    - Stimmt die Elementreihenfolge und Häufigkeit?
    - Werden nur zugelassene Attribute verwendet?
    - Bei Schema: Stimmen die Inhalte mit den Datentypen überein?
  - Werkzeuge:
    - DTD- und/oder Schemavalidierer wie nsgmls, Xerces, **xmllint**, …
    - Speziell für (X)HTML-Dokumenttypen: http://validator.w3.org
    - Eingebaut in der „Europa“-Ausgabe von Eclipse

- E-Business Standards
  - XML-basierte E-Business-Standards werden heute meist als XML-Schemata oder DTDs veröffentlicht. Deren Beherrschung ist daher sehr wichtig.

Struktur eines XHTML-Dokuments

- Einfaches XHTML 1.1-Beispiel

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<?xml-stylesheet href="hello.css" type="text/css"?>
<html
  xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
  <head>
    <title>Kleines XHTML-Beispiel</title>
  </head>
  <body>
    <p>Hallo, Welt!</p>
    <!-- Kommentar: Hier ergänzen! -->
  </body>
</html>
```

XML-Deklaration

Zeichensatz-Code!

Dokumententyp-Deklaration

Stylesheet-PI

Namensraum-URI

Globales Attribut

Von DTD gefordert!

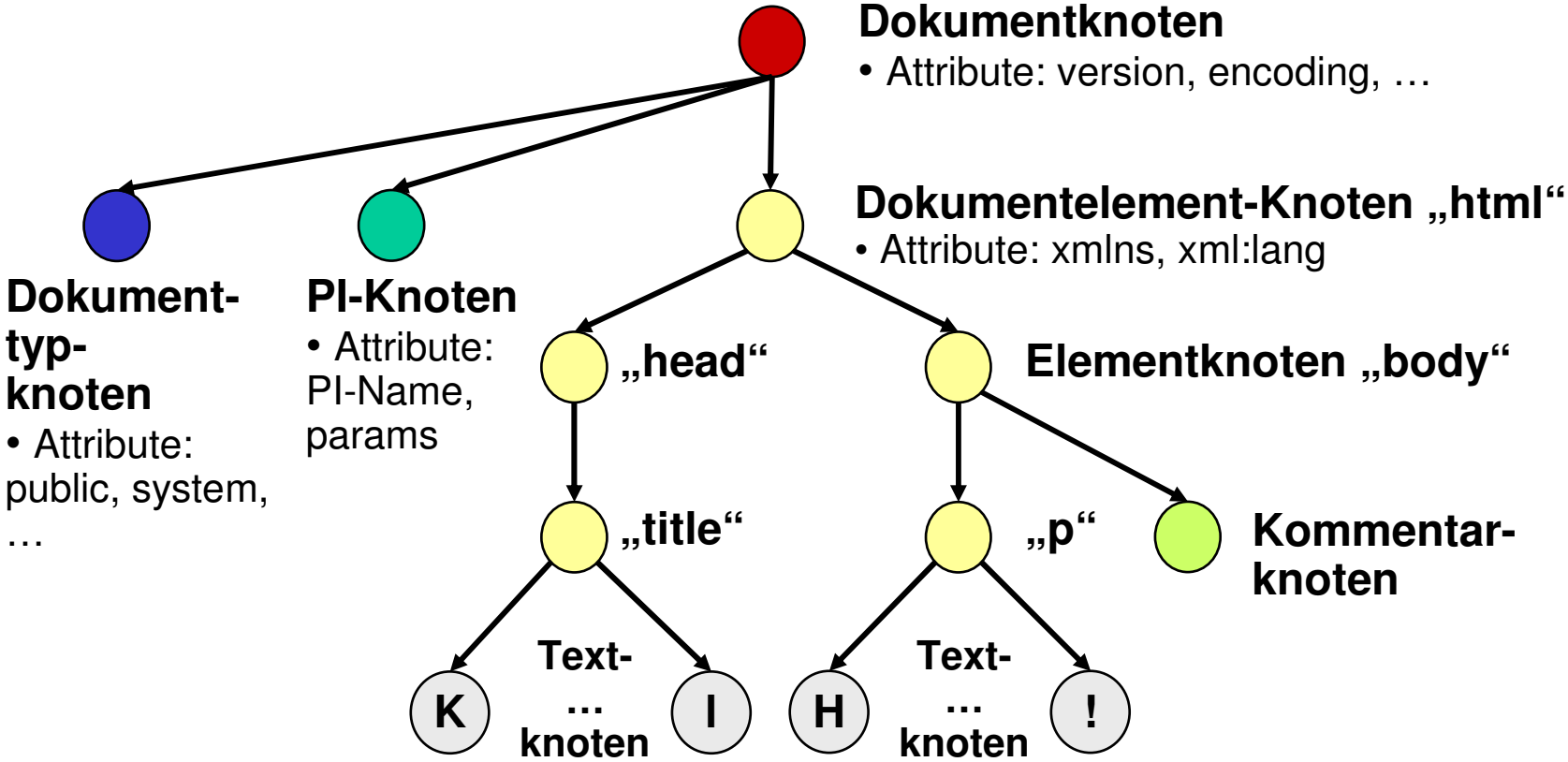
Nun im Zeichensatz enthalten

DTD

- Gemeinsame Entwicklung einer DTD für das kleine XHTML-Beispiel
  - Erweiterungen:
    - „h1“ in „body“
    - „b“ und „i“ im Text
    - „a“ mit Attributen „id“ und „href“

Struktur eines XHTML-Dokuments

- XML-Dokumente:
  - Markierte (attributierte), baumartige Graphen!
    Besitzen verschiedene Knotentypen:



Von HTML zu XHTML

- HTML **– Hypertext Markup Language**
  - Auszeichnungssprache zum einfachen Aufbau vernetzter Dokumente
  - Basis: SGML (Standard Generalized Markup Language)
  - Ca. 1990 von Tim Berners-Lee am CERN (Genf) entwickelt
  - Verschiedene Versionen, teils konkurrierend
  - Explosives Wachstum seitdem, Grundlage des WWW

- XML **– Extensible Markup Language**
  - Eine präzise Spezifikation zur Definition und Verwendung beliebiger *Dokumenttypen*. Ein Dokumenttyp entspricht dabei einer konkreten Auszeichungssprache.
  - XML 1.0 (1998, Update in 2000), XML 1.1 (2004)
  - Basis: Ebenfalls SGML

- XHTML
  - Der HTML-Dokumententyp, übertragen auf den XML-Standard
  - Präziser, modularer, flexibler, zukunftsweisender als HTML

Von HTML zu XHTML

- Warum nach XHTML wechseln?
  - Präzision einer formalen Sprache
    - Traditionelle HTML-Entwicklung:
      - Testen, was „geht“; mehrere Internet-Browser vergleichen – aufwändig und unzuverlässig!
    - Idealtypische Entwicklung:
      - Autoren „validieren“ Dokumente gegen die Regeln des Standards
      - Browser-Hersteller halten Spezifikationen des Standards genau ein
      - Möglich erst dank der Präzision einer formalen Sprache!
  - Bessere Trennung von Inhalt und Layout
    - HTML vermischt Auszeichnung der Inhalte und deren Gestaltung!
    - XHTML steuert dagegen: CSS für die Gestaltung, keine *Frames* etc. mehr …
  - Modularisierung
    - HTML 4.01 ist recht komplex. XHTML-Modularisierung ermöglicht maßgeschneiderte, kompakte Lösungen (z.B. für Handhelds)
  - Mischdokumente
    - Gestaltung (CSS), Animation (SMIL), API & Scripting (DOM): Einheitlich für XHTML-Elemente und eingebettete Dokumente (Grafiken, Formeln etc.)!

Die Module von XHTML 1.1

- Strukturmodul
  - body, head, html, title
- Textmodul
  - abbr, acronym, address, blockquote, br, cite, code, dfn, div, em, h1, h2, h3, h4, h5, h6, kbd, p, pre, q, samp, span, strong, var
- Hypertextmodul
  - a
- Listmodul
  - dl, dt, dd, ol, ul, li
- Objektmodul
  - object, param
- Präsentationsmodul
  - b, big, hr, i, small, sub, sup, tt
- Edit-Modul
  - del, ins
- *Bidirectional Text*-Modul
  - bdo
- Formularmodul
  - button, fieldset, form, input, label, legend, select, optgroup, option, textarea
- Tabellenmodul
  - caption, col, colgroup, table, tbody, td, tfoot, th, thead, tr
- *Image*-Modul
  - img
- *Client-side Image Map*-Modul
  - area, map
- *Server-side Image* Map-Modul
  - Attribute ismap on img
- *Intrinsic Events*-Modul
  - Events attributes
- Metainformationsmodul
  - meta
- Scriptingmodul
  - noscript, script
- *Stylesheet*-Modul
  - style element
- *Style Attribute*-Modul *Deprecated*
  - style attribute
- *Link*-Modul
  - link
- *Base*-Modul
  - base
- *Ruby Annotation*-Modul
  - ruby, rbc, rtc, rb, rt, rp

Durch CSS-Anweisungen zu ersetzen!

Bem.: Farbig markierte Elemente werden im Praktikum Einf. in d. Inf. verwendet.

XHTML und CSS

- HTML war ursprünglich zur inhaltlichen, abstrakten Strukturierung von Dokumenten entworfen worden.
  Über die Art der Darstellung entschied der Browser.
  - Beispiel:

```
<h1>Überschrift</h1>
<p>Geben Sie <kbd>Strg-C</kbd> an, um ein
      Programm abzubrechen.</p>
```

- Spätere Sprachelemente ergänzten Darstellungsaspekte – und verletzten dadurch das Prinzip „Trennung von Inhalt & Darstellung“!
  - Beispiele:

```
<b>Fett</b> und <i>kursiv</i> gedruckte Wörter.
<p align="center">Ein zentrierter Absatz.</p>
```

- Cascading Stylesheets (CSS)
  - dienen ausschließlich der Darstellung von XHTML- und XML-Inhalten
  - sollen XHTML von Darstellungselementen wieder befreien.

Entity- und Zeichenreferenzen

- XHTML, XML und Unicode
  - Beliebige Unicode-Zeichen können in allen XML-Texten per Zeichenreferenz eingebunden werden. Beispiel:

```
Dies kostet <Preis>50 &#x20AC;</Preis>
```

Unicodewert für €

  - Die fünf für Markup reservierten Zeichen: < > & " ' lassen sich über folgende in XML vordefinierte Entity-Referenzen als normale Zeichen verwenden:
    `< > & &quot; &apos;`

```
<Relation> a &lt; b </Relation>
```

a < b

  - In XHTML sind ferner zahlreiche Sonderzeichen aus Unicode über Entity-Referenzen verfügbar:

```
<p>Au&szlig;erdem m&ouml;chte ich betonen, dass… </p>
```

ß **ö**

Von HTML zu XHTML

- Wer HTML 4.01 gewöhnt ist, beachte bei Umstellung auf XHTML 1.1:
  - Alle Elemente und Attribute klein schreiben: <html><body>…
  - Elemente stets schließen:

```
<p>Text Text</p>  <p>Mehr Text …</p>
<p>Zeilenumbruch<br/>Neue Zeile…</p>
```

  - Immer Attributwerte angeben

```
<td nowrap="nowrap"> … </td>
```

  - XML-Deklaration und Dokumententyp-Deklaration stets angeben
  - Namensraum definieren: **<html xmlns="…">**
  - Sprachkennzeichen: Nur noch mit globalem Attribut **xml:lang**
  - Anker nicht mehr mit name, sondern mit id setzen (analog: map):

```
<h2>Ein Sprungziel<a id="ziel123"/></h2>
```

  - Zeichensatz in XML-Deklaration festlegen statt in „head“
  - CSS-Datei per Stylesheet-PI einbinden statt in „head“, CSS nutzen!
  - Basis ist „strict“ - „transitional“ und „frames“-Elemente entfallen!

XML Schema: Strukturen und Datentypen

http://www.w3.org/TR/xmlschema-1

http://www.w3.org/TR/xmlschema-2

Warum reichen DTD nicht?

- Attribute
  - Keine selbständigen Objekte, nur lokal einem Objekt zugeordnet
  - Keine Gruppenbildung möglich

- Elemente
  - Keine Defaultbelegung möglich, Inhalt nicht validierbar
  - Keine Wiederholungsfaktoren
  - Gruppenbildung nur indirekt möglich
  - Nur global wirksame Deklarationen möglich

- Beide
  - Typisierung nicht ausreichend
  - Keine benutzerdefinierten Typen
  - Syntax erfordert speziellen Markup
  - Namespace-Konzept nicht integriert
  - Komplexe Strukturen, objektorientiertes Vorgehen schlecht unterstützt

Schema - welches Schema?

- XML DTD
  - Seit langer Zeit die gemeinsame Grundlage
  - Herkunft SGML

- XDR (XML-Data Reduced)
  - Microsoft-Standard, älter als W3C XML Schema
  - z.B. in MSXML 3.0, BizTalk, SQL 2000
  - wird nun zunehmend verdrängt von W3C XML Schema

- Schematron
  - Regelbasierter Ansatz, z.B. zur Abbildung komplexer Abhängigkeiten zwischen Elementen. Gut mit XPath und XSLT vereinbar.
  - Gut kombinierbar mit W3C XML Schema
  - Standardisierungsprozess:
    - ISO/IEC 19757 - DSDL Document Schema Definition Language - Part 3: Rule-based validation - Schematron
  - Siehe auch: http://www.ascc.net/xml/resource/schematron/

Schema - welches Schema?

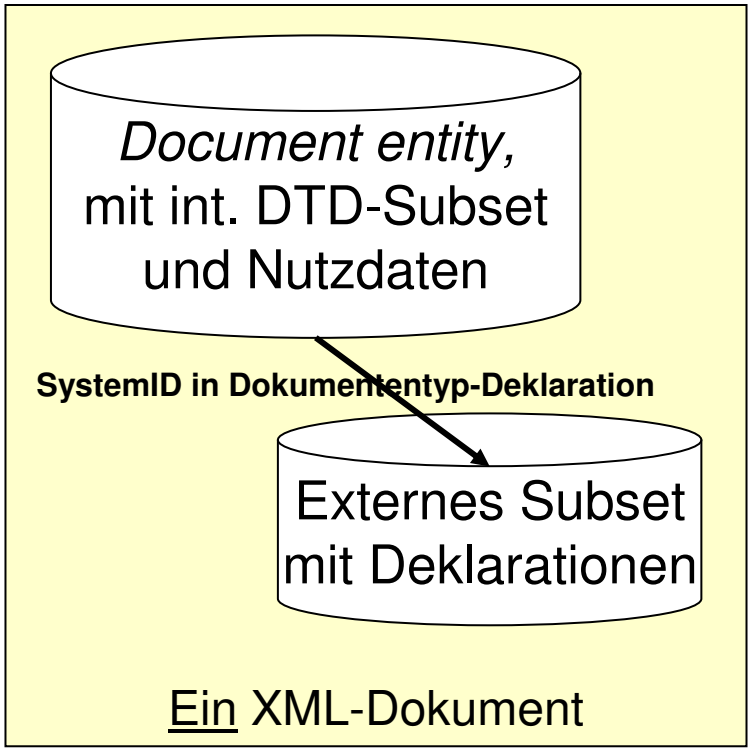
- Examplotron
  - Einfacher, aber wirksamer Ansatz - allerdings mit nur eingeschränkten Möglichkeiten
  - Ausgehend von „Beispielinstanzen mit Zusätzen“
  - Diese werden nach RELAX NG zur Validierung übersetzt
  - Siehe auch: http://examplotron.org
- RELAX NG
  - Zusammenfassung zweier Schema-Sprachen: RELAX und TREX
  - Große Ähnlichkeit zu W3C XML Schema, z.B. XML Syntax
  - Formaler (im math. Sinn), frei von einigen komplizierten Eigenschaften von W3C XML Schema
  - Erwartet die Definition zulässiger Elemente und Attribute in den Dokumentinstanzen
  - Datentypen von W3C XML Schema können verwendet werden
  - Siehe auch: http://relaxng.org
- **W3C XML Schema** – unser Thema im Folgenden!

Von der DTD zum Schema

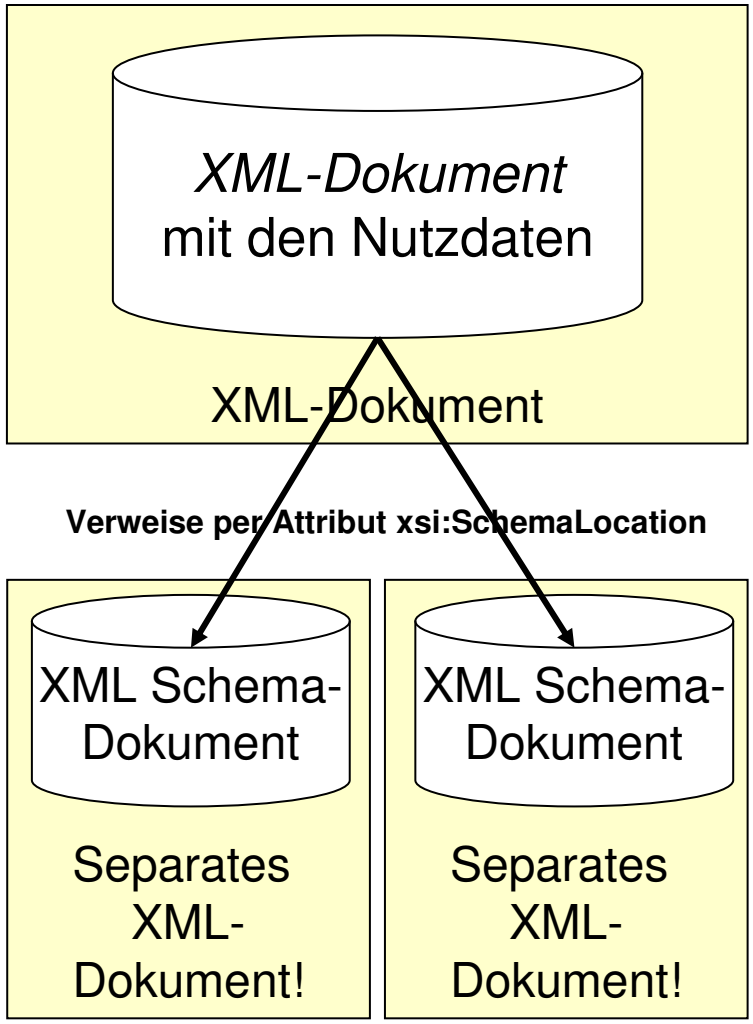
Ein beispiel-orientierter „Einstieg„
(Auszüge aus der LV „XML-Tech.“)

Von DTD zu Schema

- Arbeiten mit DTD



- Arbeiten mit Schema



Dokumententyp-Deklaration

- Die Dokumententyp-Deklaration entfällt bzw. kann entfallen!
- Statt dessen vergibt man „*hints*" (Hinweise) mittels spezieller globaler Attribute an einen Schema-Validierer:
- Bisher: DTD-Einbindung

```
<!DOCTYPE Dozent SYSTEM "dozent.dtd"[ … ]>
```

- Nun (auch zusätzlich): XML Schema-Einbindung

```
<Dozent
   xmlns="http://fbi.fh-wi.de/~werntges/ns/dozent"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=
     "http://fbi.fh-wi.de/~werntges/ns/dozent dozent.xsd">
```

  - Man beachte die paarweise Auflistung von Namespace-URI und URL in xsi:schemaLocation.
  - Grundsätzlich lassen sich auch mehrere solche Schema-Paare angeben - alle in *einem* Attributwert!

Anwendung auf XHTML + SVG

- Übung zur Selbstkontrolle:
  - Erweitern Sie das folgende XHTML 1.1-Dokument mit SVG-Anteilen so, dass es mit den Schemata für beide Standards validiert werden kann.
    - Nehmen Sie an, dass die Schemata für XHTML 1.1 in Datei `xhtml11.xsd` und für SVG in Datei `svg10.xsd` im Arbeitsverzeichnis vorliegen.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<html
   xmlns="http://www.w3.org/1999/xhtml" xml:lang="de"
   xmlns:svg="http://www.w3.org/2000/svg">
   <head>
     <title>Text z.B. für den Fensterbalken</title>
   </head>
   <body>
     <p>
       SVG Quellcode, im XHTML-Quellcode eingebettet:
      </p>
    <svg:svg width="280" heigth="280">
       <!-- SVG-Inhalt! -->
    </svg:svg>
   </body>
</html>
```

Aufbau eines Schema-Dokuments

- Schema-Dateien sind eigenständige XML-Dokumente, und zwar Exemplare des Dokumenttyps „schema" aus einem reservierten Namensraum.

- Sie sind KEINE externen *entities* der beschriebenen XML-Dokumentexemplare!

- XML Schema-Rahmen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace=
     "http://fbi.fh-wi.de/~werntges/ns/dozent"
   xmlns:target="http://fbi.fh-wi.de/~werntges/ns/dozent"
   elementFormDefault="qualified">
  <!-- <xsd:element>, <xsd:attribute>, <xsd:group> ... -->
</xsd:schema>
```

Erläuterungen

- **`<?xml … ?>`**
  - Die normale XML-Deklaration (optional)
- **`xmlns:xsd = "…"`**
  - Eine verbreitete Konvention zur Bezeichnung des Namensraums von XML Schema. Siehe auch vereinfachtes Beispiel weiter unten.
- **`targetNamespace = "…"`**
  - Der Namensraum, für den das im Folgenden definierte „Vokabular" bestimmt ist, i.d.R. der Ihrer Dokumentinstanz
- **`xmlns:target = "…"`**
  - Ein lokal definiertes Namensraum-Präfix, das benötigt wird, um in der Schema-Datei auf hier deklarierte Elemente verweisen zu können.
- **`elementFormDefault = "qualified"`**
  (default wäre `"unqualified"`)
  - Bewirkt „normales" Namensraum-Verhalten, verhindert die gemischte Verwendung von Elementen mit und ohne Namensraum

Schema-Dokument mit *default*-Namensraum

- Häufig wird der Namensraum von XML Schema zum *default* in Schemainstanzen erklärt, um so zahlreiche Präfixes zu vermeiden.
- Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace =
    "http://fbi.fh-wi.de/~werntges/ns/dozent"
   xmlns:target =
    "http://fbi.fh-wi.de/~werntges/ns/dozent"
   elementFormDefault="qualified">
   <!-- <element>, <attribute>, <group> ... -->
</schema>
```

Präfix entfällt!

Elementtyp-Deklaration

- #PCDATA
  (nur Freitext, häufiger Spezialfall von Mixed)
  - DTD:

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

  - XML Schema:

```
<xsd:element name="title" type="xsd:string"/>
```

  - Bemerkungen:
    - Der eingebaute Datentyp „`string`" kommt der Bedeutung von #PCDATA sehr nahe.
    - Neu: **Datentyp-Konzept** !

Elementtyp-Deklaration

- ANY (beliebige Inhalte, eher „pathologisch")
  - DTD:

```
<!ELEMENT Container ANY>
```

  - XML Schema:

```
<xsd:element name="Container">
  <xsd:complexType>
    <xsd:any namespace="##any"
       processContents="lax"
       minOccurs="0"
       maxOccurs="unbounded"/>
  </xsd:complexType>
</xsd:element>
```

Elementtyp-Deklaration

- EMPTY (nur Attribute)
  - DTD:

```
<!ELEMENT Beschäftigungsverhältnis EMPTY>
<!ATTLIST Beschäftigungsverhältnis Art … >
```

  - XML Schema:

```
<xsd:element name="Beschäftigungsverhältnis">
  <xsd:complexType>
    <xsd:attribute name="Art" type="…"/>
  </xsd:complexType>
</xsd:element>
```

  - Bemerkungen:
    - Kurzschreibweise! Ausgelassen (vor *attribute*) wurde:

```
<xsd:complexContent>
<xsd:restriction base="xsd:anyType">
```

    - Siehe auch: *XML Schema Tutorial, „2.5.3 Empty Content“*

Elementtyp-Deklaration

- Children (hier nur direkte Unterelemente)
  - DTD:

```
<!ELEMENT html-mini (head?, body)>
```

  - XML Schema:

```
<xsd:element name="html-mini">
 <xsd:complexType>
    <xsd:sequence>
       <xsd:element ref="target:head minOccurs="0"/>
       <xsd:element ref="target:body"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

  - Bemerkungen:
    - Elemente „Vorname“ und „Nachname“ werden separat deklariert.

Elementtyp-Deklaration

- Mixed, Choice
  - DTD:

```
<!ELEMENT p (#PCDATA|b|i|a)*>
```

  - XML Schema:

```
<xsd:element name="p">
 <xsd:complexType mixed="true">
   <xsd:choice minOccurs="0"
               maxOccurs="unbounded">
     <xsd:element ref="target:b"/>
     <xsd:element ref="target:i"/>
     <xsd:element ref="target:a"/>
   </xsd:choice>
 </xsd:complexType>
</xsd:element>
```

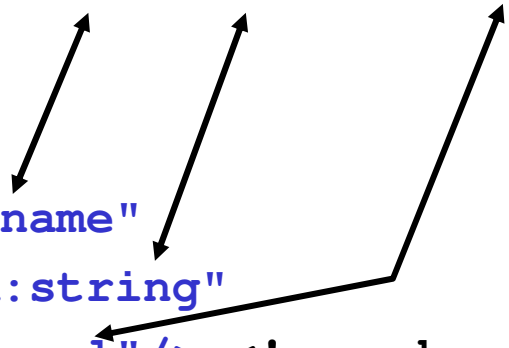
Attributtyp-Deklaration

- StringType
  - DTD:

```
<!ATTLIST elem attname CDATA #IMPLIED>
```

  - XML Schema:

```
<xsd:attribute
        name="attname"
        type="xsd:string"
        use="optional"/> <!-- oder: "required" -->
```



- Bemerkungen:
  - Attribute in XML Schema können ähnlich wie Elemente lokal oder global eigenständig deklariert werden.
  - Ihre Zuordnung zu Elementen erfolgt über den Kontext ihrer Einbindung in einen `complexType`.

Vorgriff: <complexType>

Erweiterung eines nicht-leeren Elements mit einfachem Datentypen um ein Attribut:

```
<xs:element name="width">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="xs:nonNegativeInteger">
      <xs:attribute name="unit" type="xs:NMTOKEN"/>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

Anwendung:

```
<width unit="cm">25</width>
```

Attributtyp-Deklaration

- Ableitung eines eigenen Datentypen, hier: Auswahlliste
  - DTD:
    - `<!ATTLIST Vorlesung Wochentag (Montag|Dienstag|…|Sonntag) #IMPLIED>`
  - XML Schema:

```
<xsd:simpleType name="WochentagTyp">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Montag"/>
      ...
    <xsd:enumeration value="Sonntag"/>
  </xsd:restriction>
</xsd:simpleType>
```

Attributtyp-Deklaration

- Default-Deklarationen in XML-Schema

a) #REQUIRED, #IMPLIED:

- `<xsd:attribute>` kennt das Attribut **`use`**
- Zulässige Werte: "`required`", "`optional`", ferner "`prohibited`" (etwa zum gezielten Blockieren / Reservieren)

b) „Echte" Defaultwert-Belegung, optional mit „#FIXED":

- `<xsd:attribute>` kennt die Attribute **`default`** und **`fixed`**
- Diese werden einfach – alternativ - mit dem gewünschten Defaultwert belegt. Also: Nie gleichzeitig „default“ und „fixed“ verwenden!

Attributtyp-Deklaration

- Default-Deklarationen in XML-Schema, Beispiel:
  - DTD:

```
<!ATTLIST elem attname1 CDATA "myDefaultValue"
               attname2 CDATA #FIXED "fixedValue">
```

  - XML Schema:

```
<xsd:attribute
        name="attname1"
        type="xsd:string"
        default="myDefaultValue"/>
<xsd:attribute
        name="attname2"
        type="xsd:string"
        fixed="fixedValue"/>
```

Beispiel (Demo)

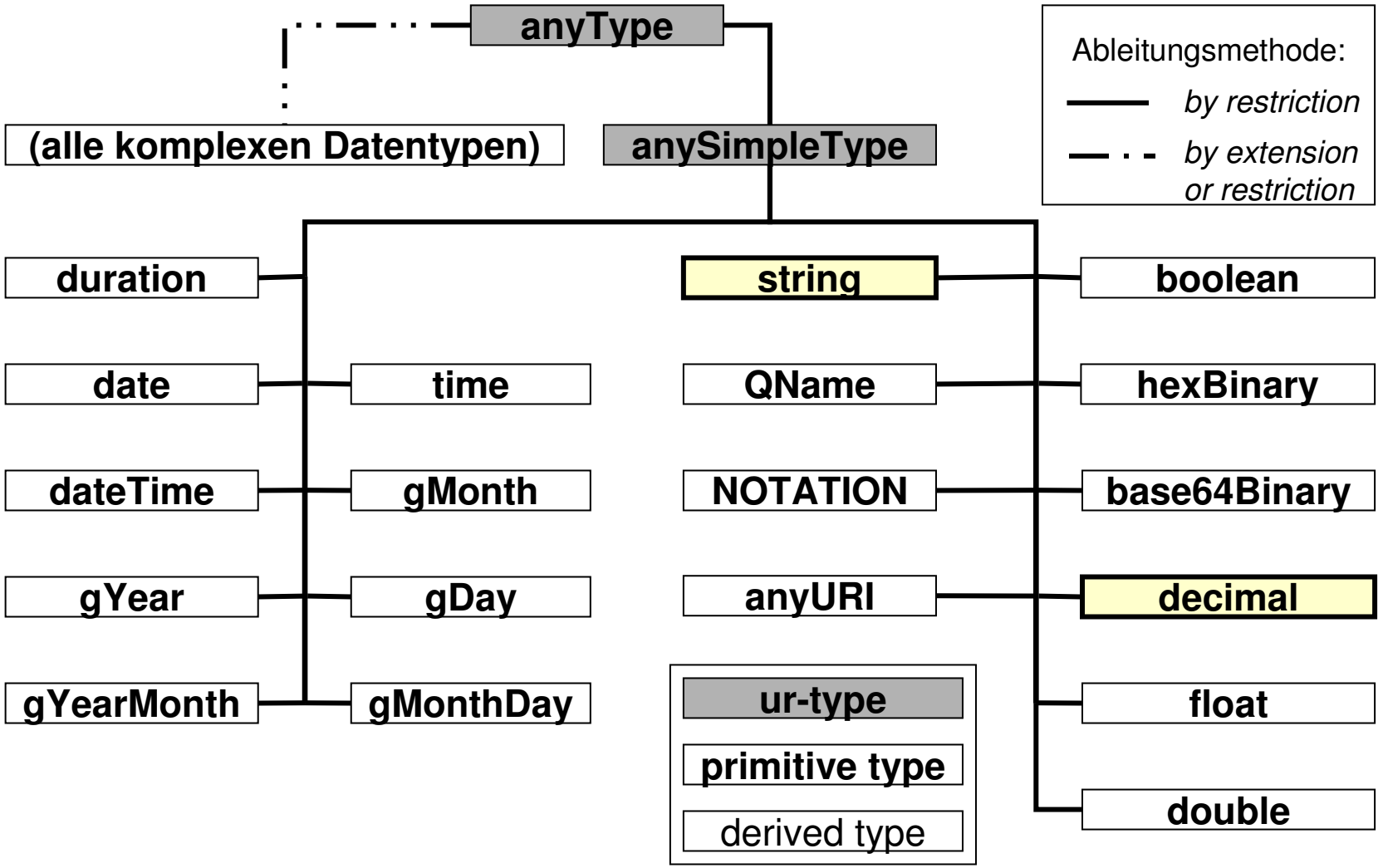
- Umsetzung der Mini-XHTML-DTD in ein XML-Schema
  - XML-Dokumentendatei:
    - `xhtml-with-schema.xml`
  - DTD:
    - `xhtml11-mini2.dtd`
  - XML Schema:
    - `xhtml11-mini.xsd`
  - Demo, einschließlich Validierung mit xmllint:
    - ```
      xmllint --valid --noout \
        --schema xhtml11-mini.xsd xml-with-schema.xml
      ```
  - Bemerkungen
    - Angabe der Schemadatei sollte überflüssig sein!
    - Fehlermeldung zu xml:lang ist anzuzweifeln.
      ```

Datentypen in XML Schema

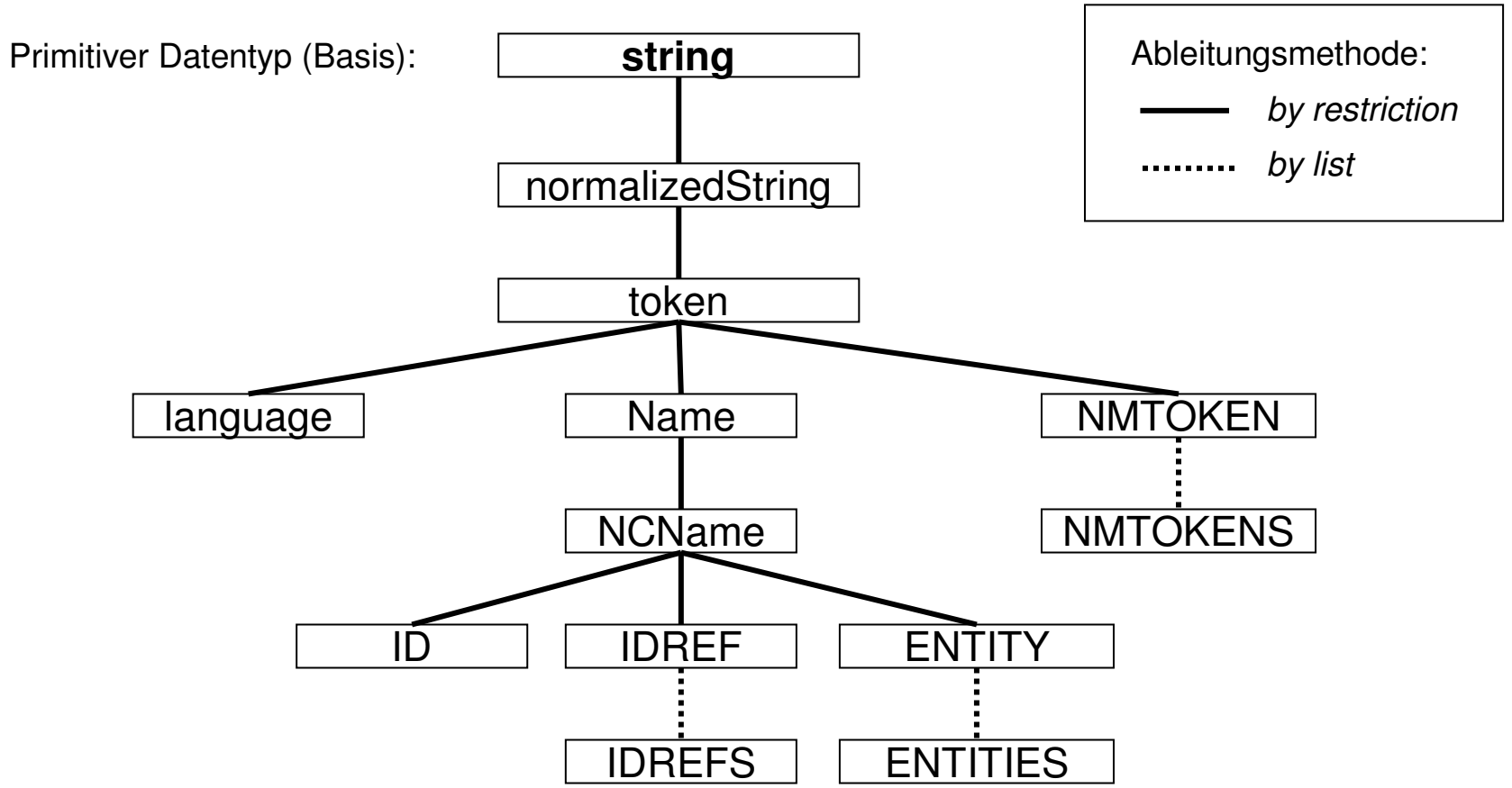
Vordefinierte Datentypen

Ableitung eigener Datentypen

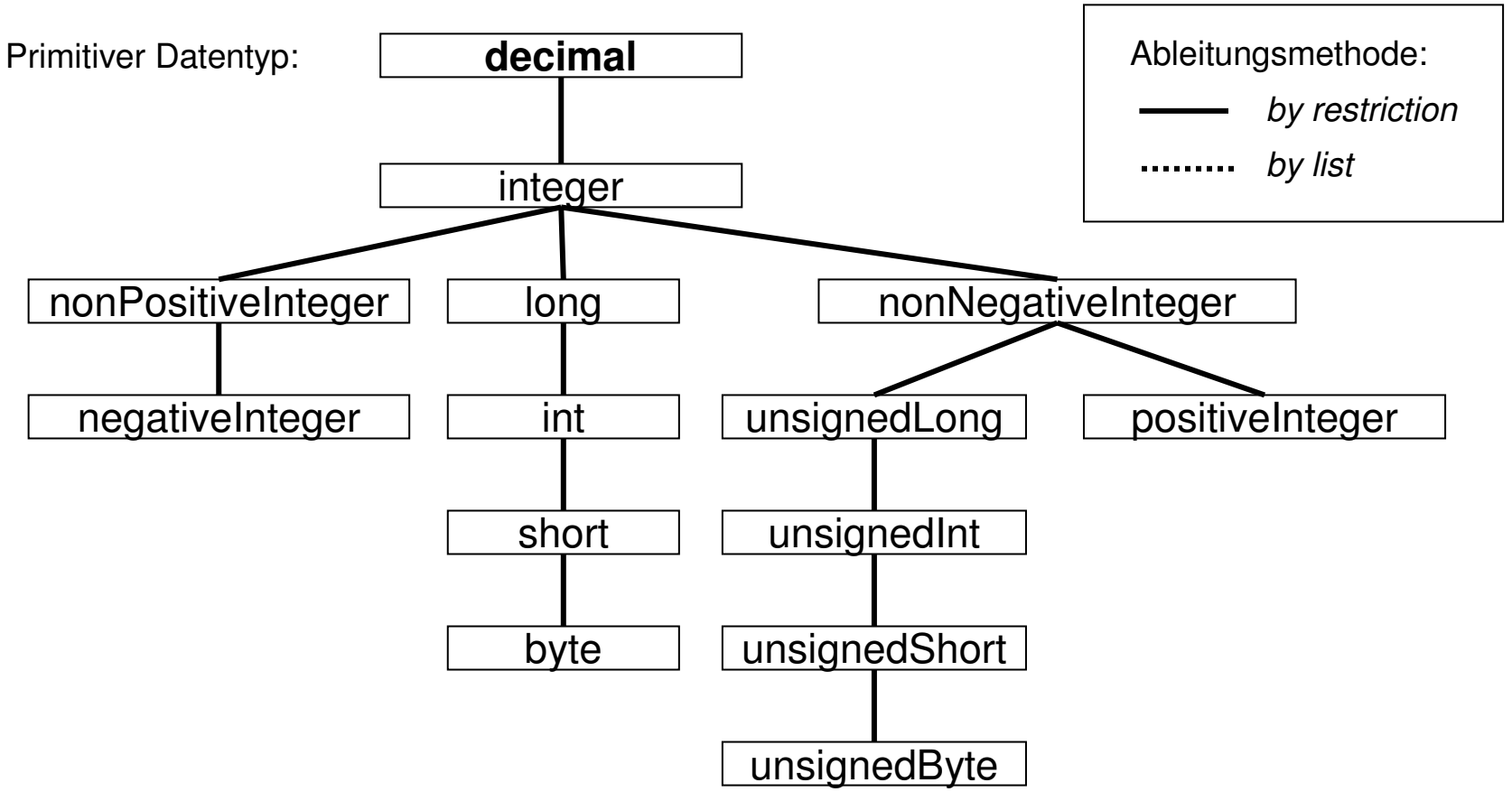
Abstammung der primitiven Datentypen



Vordefinierte Datentypen



Vordefinierte Datentypen



Vordefinierte Datentypen

- Benutzung, Beispiel:
  - Datentypen-Information direkt aus dem Instanzdokument an die Anwendung, ohne Schema-Validierung:

```
<doc xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd=
    "http://www.w3.org/2001/XMLSchema-datatypes">
  <mynum xsi:type="xsd:decimal">409</mynum>
  <mystr xsi:type="xsd:string">
    This is my string</mystr>
</doc>
```

Anwendungen:

Etwa beim Aufbau dynamischer Datenstrukturen auch ohne Schema, bei Verwendung der Datentypen durch andere Schemasprachen, im Rahmen von SOAP (!), …

Ableitung eigener Datentypen

- Die drei Methoden der Ableitung
  - ***by list***
    - Ein Element des Listentyps ist eine Folge (*sequence*) von Elementen der zugrundeliegenden Wertemenge des *itemType.*
  - ***by union***
    - Vereinigungsmenge (von W bzw. L) bilden
  - ***by restriction***
    - Die 12 Facetten (in 6 Kategorien) der Einschränkung:
      - Länge: ***length, minLength, maxLength***
      - Muster: ***pattern***
      - Aufzählung: ***enumeration***
      - *Whitespace*: ***whitespace***
      - Intervall (*range*): ***minInclusive, minExclusive, maxExclusive, maxInclusive***
      - Dezimalstellen: ***totalDigits, fractionDigits***

Ableitung *by list*

- Beispiel:
  - Eine Liste von Größenangaben mit dem Basistyp `decimal`

```
<simpleType name='sizes'>
        <list itemType='decimal'/>
</simpleType>
```

  - Anwendung dann:

```
<cerealSizes xsi:type='sizes'> 8 10.5 12
</cerealSizes>
```

- Neu im Beispiel:
  - `<list>` mit Attribut „`itemType`“

- Vorsicht:
  - Listenelemente werden mit *whitespace* separiert. Enthält der `itemType` *whitespace* als zulässige Zeichen, lässt sich die Liste nicht immer in ihre korrekten Bestandteile zerlegen!

Ableitung *by union*

- Beispiel:
  - Die Menge Z^+ (alle ganzen Zahlen außer Null)

```
<simpleType name='z-plus'>
 <union>
    <simpleType>
          <restriction base="positiveInteger"/>
    </simpleType>
    <simpleType>
          <restriction base="negativeInteger"/>
    </simpleType>
  </union>
</simpleType>
```

- Neu im Beispiel:
  - `<union>` unterhalb vom zu definierenden simpleType

Ableitung *by restriction*

- Typische Konstruktion beim Ableiten:

```
<simpleType name='myRestrictedType'>
  <restriction base='baseType'>
          facet 1 ...
          ...
          facet n ...
  </restriction>
</simpleType>
```

- Neu im Beispiel:
  - `<restriction>` mit Attribut „`base`“
  - Diverse Facetten-Elemente

Ableitung *by restriction*

<enumeration>

- Reduzierung der Wertemenge des Basistyps auf die explizit gelisteten Werte-Elemente. Praktisch immer möglich außer bei `boolean`.

```
<simpleType name='unbeweglicheFeiertage'>
  <restriction base='gMonthDay'>
    <enumeration value='--01-01'/>
    <enumeration value='--05-01'/>
    <enumeration value='--10-03'/>
    <enumeration value='--12-24'>
      <annotation><documentation>
           Halber Tag!</documentation></annotation>
    </enumeration>
    <enumeration value='--12-25'/>
    <enumeration value='--12-26'/>
    <enumeration value='--12-31'> … </enumeration>
  </restriction>
</simpleType>
```

Ableitung *by restriction*

<pattern>

- Sehr flexible und mächtige Ableitungsmethode, basierend auf „regulären Ausdrücken“. Mit allen einfachen Datentypen verwendbar.

```
<simpleType name='Bankleitzahl'>
 <restriction base='nonNegativeInteger'>
  <pattern value='\d{8}'/>
</restriction>
</simpleType>

<simpleType name='KFZ-Kennzeichen'>
 <restriction base='token'>
  <pattern
    value='[A-ZÄÖÜ]{1,3}-[A-ZÄÖÜ]{1,2} [1-9]\d{0,3}'/>
  <maxLength value='10'/>
 </restriction>
</simpleType>
```

Ableitung *by restriction*

Beispiel:

- Definition und Anwendung des Attributs „my_lang“ in Element „html-mini“
  - Validierung mit xmllint
  - Test: Verhalten bei abweichendem Ländercode