

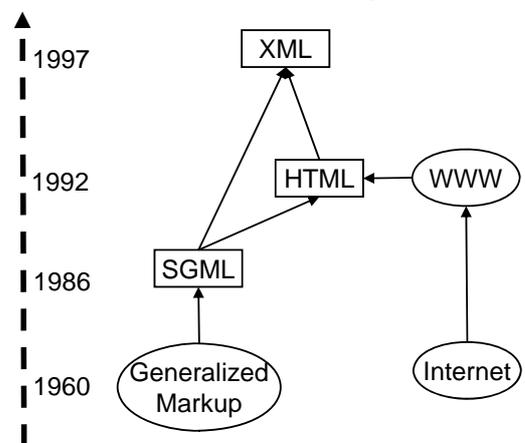
5. XML-Datenbanken: Datendefinition

- XML
- DTD
- Namensräume
- XML Schema
 - Typen
 - Deklarationen
 - Integritätsbedingungen
- Modellierungsstile (Matroschka-, Salami-, Jalousie-Design)
- Nutzung mehrerer Schemas



XML – eXtensible Markup Language

- Beschreibung **semistrukturierter Daten** / Dokumente (Text + flexible Struktur)
- Markup-Sprache: Dokumentabschnitte werden durch Marker ausgezeichnet
 - hervorgegangen aus SGML und HTML
- SGML
 - Standard Generalized Markup Language
 - Problem: sehr hohe Komplexität
- HTML: HyperText Markup Language
 - SGML-Anwendung
 - feste Menge an Auszeichnungselementen:
 - nicht erweiterbar
- XML: Empfehlung des **W3C** - World Wide Web Consortium (1998)
 - SGML-Kompatibilität
 - einfach anwendbar, klare und lesbare Dokumentformate
 - Schemaunterstützung



Quelle: Neil Bradley: The XML Companion



XML - Beispiel

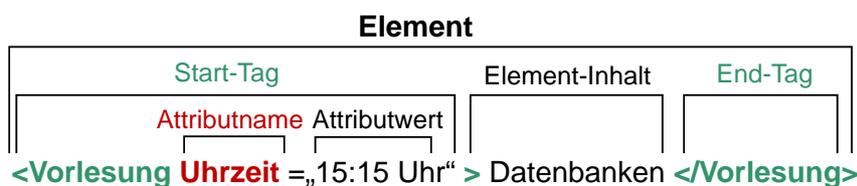
■ XML-Dokumente haben Baumstruktur

- Schachtelung von Elementen (= Baumknoten)
- Elemente können neben Unterelementen Attribute bzw. Inhalt (z.B. Text) aufweisen

```
<?XML version="1.0"?>
<!DOCTYPE Vorlesungsverzeichnis SYSTEM
"http://dbs.uni-leipzig.de/dtd/VLVerzeichnis.dtd">
<VLVerzeichnis>
  <Vorlesung Uhrzeit=„15:15 Uhr“>
    <Thema>DBS2</Thema>
    <Dozent>
      <Name>Prof. Rahm</Name>
      <Einrichtung>Uni Leipzig</Einrichtung>
    </Dozent>
  </Vorlesung>
</VLVerzeichnis>
```



XML: Elemente und Attribute



■ Elemente

- Start- und End-Tag müssen vorhanden sein.
Ausnahme: leeres Element (Bsp. <Leer />)
- feste Reihenfolge von Geschwisterelementen (Reihung ist wichtig!)
- Element-Inhalt besteht entweder nur aus weiteren Elementen (**element content**) oder aus Zeichendaten optional vermisch mit Elementen (**mixed content**)

■ Attribute

- Attributwerte können nicht strukturiert werden
- Attributreihenfolge beliebig

■ Groß-/Kleinschreibung ist relevant (gilt für alle Bezeichner in XML)

■ Weitere XML-Bestandteile (hier nicht behandelt): Entities, Processing Instructions, Kommentare



XML-Strukturdefinition: DTD

■ DTD (Document Type Definition) / Schema optional

- **wohlgeformt**: XML-Daten sind syntaktisch korrekt
- **gültig**: XML-Daten sind wohlgeformt und entsprechen einer DTD

```
<?XML version="1.0"?>
<!DOCTYPE Vorlesungsverzeichnis SYSTEM
"http://dbs.uni-leipzig.de/dtd/VLVerzeichnis.dtd">
<VLVerzeichnis>
  <Vorlesung Uhrzeit=„15:15 Uhr“>
    <Thema>DBS2</Thema>
    <Dozent>
      <Name>Prof. Rahm</Name>
      <Einrichtung>Uni Leipzig</Einrichtung>
    </Dozent>
  </Vorlesung>
</VLVerzeichnis>
```

VLVerzeichnis.dtd:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT VLVerzeichnis (Vorlesung)* >
<!ELEMENT Vorlesung (Thema, Dozent) >
<!ATTLIST Vorlesung
  Uhrzeit CDATA #REQUIRED >
<!ELEMENT Thema (#PCDATA) >
<!ELEMENT Dozent (Name, Einrichtung?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Einrichtung (#PCDATA)>
```



DTD – Document Type Definition

- beschreibt Dokumentstruktur und legt damit einen Dokumenttyp fest
- im Dokument verweist *Dokumenttyp-Deklaration* auf DTD

- interne DTD

```
<!DOCTYPE Vorlesungsverzeichnis [<!ELEMENT VLVerzeichnis (Vorlesung)*>...]>
```

- externe DTD

```
<!DOCTYPE Vorlesungsverzeichnis SYSTEM „http://dbs.uni-leipzig.de/dtd/VLVerzeichnis.dtd“>
```

■ Definition von Elementen in einer DTD

- Sequenz: (A , B) - vorgegebene Reihenfolge

```
<!ELEMENT Vorlesung (Thema, Dozent)
```

- Alternative: (A | B) - entweder A oder B (XOR)

```
<!ELEMENT Adresse (PLZ, Ort, (Str, Nr) | Postfach)>
```



DTD (2)

■ Element-Wiederholung / Kardinalität:

- A? - 0..1 Mal

```
<!ELEMENT Dozent (Name, Einrichtung?)>
```

- A+ - 1..n Mal

```
<!ELEMENT Name (Vorname+, Nachname)>
```

- A* - 0..n Mal

```
<!ELEMENT VLVerzeichnis (Vorlesung)*>
```

■ Mixed Content:

- (#PCDATA / A / B)*

```
<!ELEMENT Text (#PCDATA | Link)*>
```

Elemente A, B und Text treten in beliebiger Reihenfolge und Anzahl auf

■ Leeres Element (kein Element-Inhalt)

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



DTD (3)

■ Definition von Attributen in einer DTD

```
<!ATTLIST Elementname (Attributname Typ Auftreten)* >
```

■ Attribute gehören zu einem Element

■ Jedes Attribut hat Namen, Typ und Auftretensangabe

■ Mögliche Attribut-Typen

- CDATA
- ID
- IDREF/IDREFS
- Aufzählung möglicher Werte (wert1 | wert2 | ...)
- NMTOKEN/NMTOKENS, ENTITY/ENTITYS

■ ID- und IDREF-Attribute ermöglichen Querverweise innerhalb eines Dokumentes (Graphstruktur)



DTD (4)

■ Auftretensangabe eines Attributs

- Obligatheit: #REQUIRED - das Attribut muss angegeben werden
- Defaultwert-Regelung: #IMPLIED - es gibt keinen Defaultwert
defaultwert - wird angenommen, wenn Attribut nicht angegeben wird
- #FIXED defaultwert - Attribut kann nur den Defaultwert als Wert besitzen

■ Beispiele:

```
<!ATTLIST Entfernung Einheit CDATA #FIXED „km“>
```

```
<!ATTLIST Karosse Farbe („rot“ | „gelb“ | „blau“) „blau“>
```

```
<!ATTLIST Artikel id ID #REQUIRED>
```

```
<!ATTLIST Rechnungsposten Artikel IDREF #REQUIRED>
```



Elemente vs. Attribute

- Datenmodellierung oft durch Element als auch durch Attribut möglich
- Einsatzkriterien unter Verwendung einer DTD

	Element	Attribut
Inhalte	komplex	nur atomar
Ordnungserhaltung	ja	nein
Quantoren / Kardinalität	1 / ? / * / +	REQUIRED / IMPLIED
Alternativen	ja	nein
Identifikation	nein	ID / IDREF / IDREFS
Defaultwerte	nein	ja
Aufzählungstypen	nein	ja

- bei XML Schema anstelle DTD können Elemente alle Eigenschaften von Attributen erhalten; trotzdem Attribute zu bevorzugen wenn:
 - Aufzählungstypen mit atomaren Werten und Defaultwert zu modellieren sind
 - es sich um 'Zusatzinformation' handelt (Bsp. Währung, Einheit)
 - das Dokument effizient verarbeitet (geparsed) werden soll



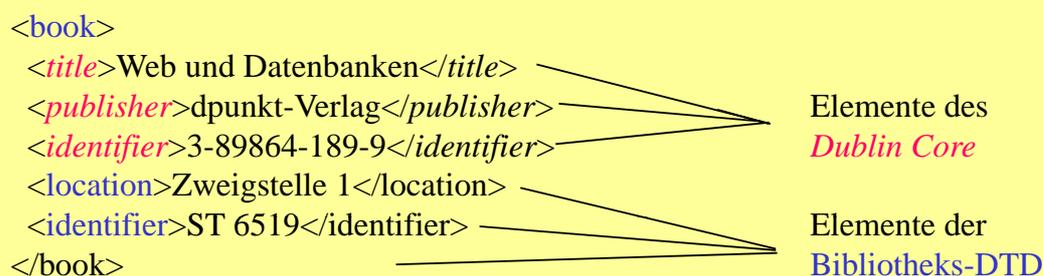
XML Schemabeschreibungssprachen

- Schemainformationen wichtig für
 - Korrektheit von Daten/Dokumenten: Validierung von Struktur und Datentypen
 - effiziente DB-Speicherung und Verarbeitung von Anfragen
 - Interoperabilität in verteilten Umgebungen
- Interoperabilität: Feste Formatierungen für Datenaustausch
 - Schnittstellen für Web-Services
 - Aufbau von Dokumenten (Bestellungen, Produktbeschreibungen ...)
 - Unterstützung vieler (Teil-) Schemas erforderlich
- Defizite der DTD:
 - kaum Typisierung von Elementinhalt (Text) und Attributwerten möglich
 - zusätzliche Syntax (kein XML-Format) -> extra Parser notwendig
 - unzureichendes Schlüssel/Fremdschlüsselkonzept (id/idref)
 - mehrere DTDs pro Dokument sind nicht erlaubt
 - keine Unterstützung von Namensräumen
 - Unzureichende Unterstützung der Interoperabilität und Wiederverwendbarkeit



Nutzung mehrerer Schemas

- Ziel: Verwendung unterschiedlicher Schemas innerhalb eines Dokumentes
 - z. B. Bibliotheksdaten, die durch Metadatenelemente des Dublin Core ausgezeichnet sind
- Probleme:
 - Zuordnung der Elemente zu Schemas
 - Namenskollisionen (im Beispiel identifizier-Elemente)
- Lösung:
 - Namen eines Schemas werden einem Namensraum zugeordnet
 - Verwendung mehrerer Namensräume pro XML-Dokument / Schema



Namensräume (namespaces)

- Namespaces bilden eigene W3C-Recommendation
www.w3.org/TR/REC-xml-names/
- Namensraum hat global eindeutige URI (Uniform Resource Identifier)
- Bezugnahme im Dokument über eindeutigen Präfix
 - Namensraumdefinition durch xmlns-Attribut
 - gilt für alle untergeordneten Elemente (Neudefinition auf tieferer Ebene ist möglich)
 - Namen mit Präfix sind 'qualifizierte Namen'
 - Namensraumdefinition ohne Präfixangabe bezeichnet **Default-Namensraum**
 - Default-Namensraum gilt für alle Elemente ohne Präfix

```
<book xmlns="urn:de.uni-leipzig.bibl"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Web und Datenbanken</dc:title>
  <dc:publisher>dpunkt-Verlag</dc:publisher>
  <dc:identifier>3-89864-189-9</dc:identifier>
  <location>Zweigstelle 1</location>
  <identifier>ST 6519</identifier>
</book>
```



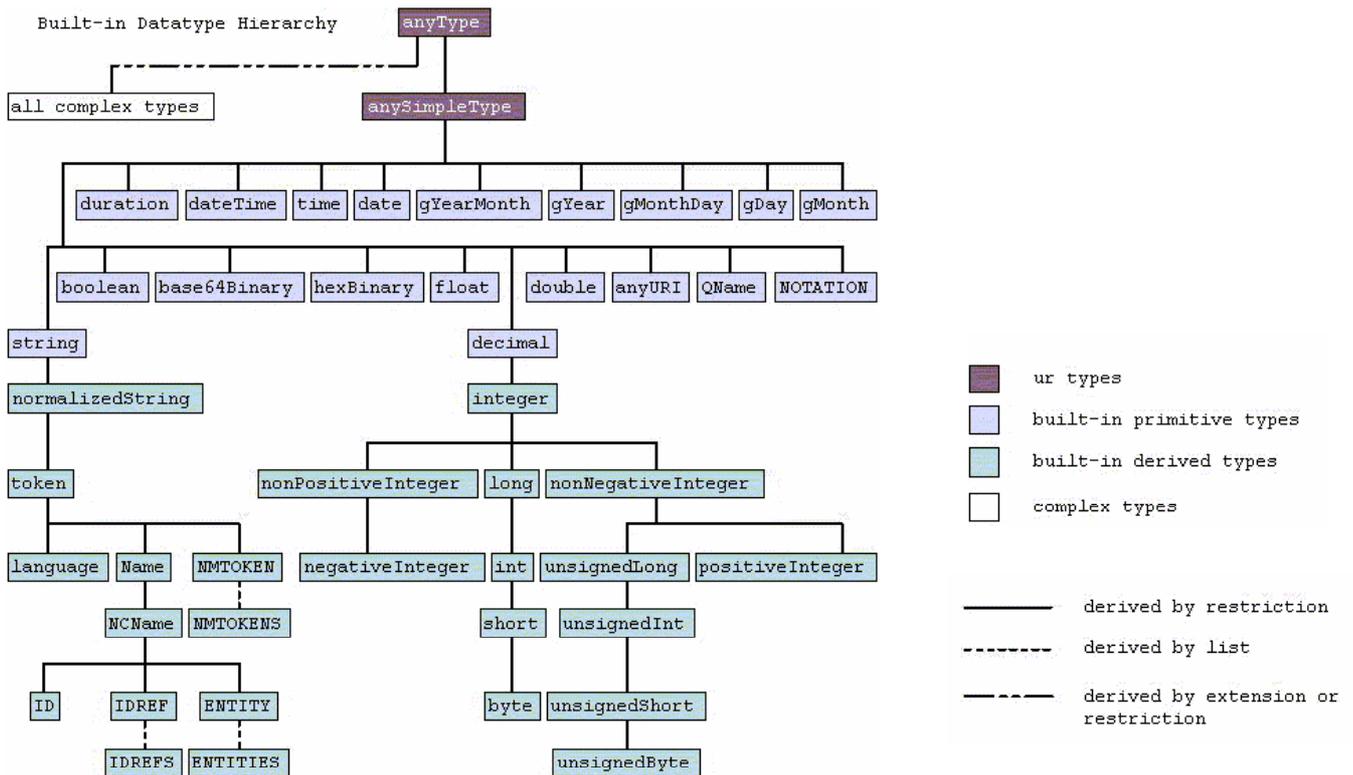
XML Schema*

- XML Schemabeschreibungssprache, entwickelt durch das W3C
 - Recommendation seit 2001
 - Teil 1: Strukturbeschreibung (ersetzt Strukturbeschreibung der DTD)
 - Teil 2: Datentypen
- Eigenschaften
 - verwendet XML-Syntax
 - Unterstützung von Namespaces und verteilter Schemas
 - erweiterbares Typsystem
 - vielfältige vordefinierte Datentypen
 - Ableitung neuer Datentypen durch Restriktionen oder Erweiterungen
 - Simple Types
 - Complex Types
 - erlaubt Definition von Integritätsbedingungen (unique, key)
 - nur eingeschränkte Unterstützung von Vererbung (kein Verhalten)
 - komplexer und weniger kompakt gegenüber DTD

* <http://www.w3c.org/XML/Schema>



XML Schema: Vordefinierte Typhierarchie



XML Schema: simpleType

- **simpleType**: benutzerdefinierter, einfacher Datentyp
 - kann keine Attribute oder Kindelemente enthalten
 - geeignet zur Definition von XML-Attributwerten und Elementinhalten
 - ableitbar durch Angabe von einschränkenden Eigenschaften (facets), Listenbildung oder Vereinigung
- **Beispiel: simpleType-Definition mit length-Einschränkung**
 - Präfix **xs:** bezeichne standardisierten XML-Schema-Namensraum

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

- **Anonyme vs. benannte Typen**: ohne / mit name-Attribut (ermöglicht Mehrfach/Wiederverwendung)



XML Schema: simpleType (2)

■ Möglichkeiten der Einschränkung

- Längenbeschränkung für Zeichenketten sowie Listen (genau, min, max): `length`, `minLength`, `maxLength`
- Längenbeschränkung für decimal-Beschränkungen: `totalDigits`, `fractionDigits`
- Unter/Obergrenzen geordneter Wertebereiche:
`minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive`
- Aufzählung zulässiger Werte: `enumeration`
- Muster: `pattern` value = <regulärer Ausdruck>

```
<xs:simpleType name="OktalDigit">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="7" />
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="BibId">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z][1-9][0-9]*" />
  </xs:restriction>
</xs:simpleType>
```



XML Schema: simpleType (3)

■ Listenbildung

- definiert leerzeichenseparierte Liste von Werten eines *simpleType*-Typs):

```
<xs:simpleType name="BookList">
  <xs:list itemType="BibId"/>
</xs:simpleType>
```

```
<xs:simpleType name="Lottozahlen">
  <xs:list>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="1" />
        <xs:maxInclusive value="49" />
      </xs:restriction>
    </xs:simpleType>
  </xs:list>
</xs:simpleType>
```

■ Vereinigung (Typ erlaubt mehrere Wertebereiche):

```
<xs:simpleType name="Kontakt">
  <xs:union memberTypes="TelefonTyp EmailTyp"/>
</xs:simpleType>
```



XML Schema: complexType

- komplexe Datentypen dienen der Beschreibung von Elementinhalten
 - Beschränkung (restriction) oder Erweiterung (extension) bestehender Typen
 - einfaches oder komplexes Inhaltsmodell
 - Default: complexContent, Basistyp anyType
- simpleContent
 - für Elemente ohne Subelemente (Blätter im XML-Baum),
 - nur Attribute
- Elementstrukturierung bei complexContent
 - Reihung: sequence
 - Auswahl/Alternative: choice
 - Gruppe: all (jedes Element kann max. 1-mal auftreten; reihenfolgeunabhängig)
- Kardinalitätsrestriktionen: minOccurs, maxOccurs
 - Defaultwert: 1
 - unbeschränkte Anzahl: "unbounded"



XML Schema: complexType (2)

- Definition durch **Beschränkung (restriction)**

```
<xs:complexType name="DozentTyp">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:element name="Name"
          type="xs:string"/>
        <xs:element name="Einrichtung"
          type="xs:string"
          minOccurs="0"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

≅

```
<xs:complexType name="DozentTyp">
  <xs:sequence>
    <xs:element name="Name"
      type="xs:string"/>
    <xs:element name="Einrichtung"
      type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```



XML Schema: complexType (3)

■ Definition durch **Erweiterung (extension)**:

```
<xs:complexType name="UniDozentTyp">
  <xs:complexContent>
    <xs:extension base="DozentTyp">
      <xs:sequence>
        <xs:element name="Adresse" type="AdressTyp"/>
        <xs:choice>
          <xs:element name="Mail" type="xs:string"/>
          <xs:element name="URL" type="xs:string"/>
        </xs:choice>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```



XML Schema: Schemaaufbau

- Schema in *schema*-Element eingeschlossen
 - Namensraum *http://www.w3.org/2001/XMLSchema* für Schemaelemente
 - Spezifikation eines „target namespace“, dem das Schema zugeordnet wird
- Schemakomponenten, die direkt unter *schema*-Element liegen, sind *global*; tiefer liegende Komponenten sind *lokal*

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="...">
  <xs:import ... />
  <xs:complexType name="...">...</xs:complexType>
  <xs:attribute name="..." ... />
  <xs:element name="..." >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="..." ... />
    </xs:sequence>
    <xs:attribute name="..." ... />
  </xs:complexType>
</xs:element>
</xs:schema>
```

benannt ———— <xs:complexType name="...">...</xs:complexType> ———— global

anonym ———— <xs:complexType> ———— lokal



XML Schema: Elementdeklaration

■ verschiedene Möglichkeiten der Elementdeklaration

- **direkte Deklaration** des Elementinhalts

```
<xs:element name=„Einrichtung“ type=„xs:string“ minOccurs=„0“ />  
<xs:element name=„Dozent“>  
  <xs:complexType> ... </xs:complexType>  
</xs:element>
```

- Verweis auf **komplexen Datentyp**

```
<xs:element name=„Dozent“ type=„DozentTyp“ />
```

- Verweis auf **globale Elementdeklaration**

```
<xs:element ref=„Dozent“ />
```



Vergleich DTD - XML Schema Elementdeklaration

DTD:

```
<!ELEMENT Dozent (Titel?, Vorname+, Name, (Vorlesung | Seminar)*)>
```

XML Schema:

```
<xs:element name=„Dozent“>  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name=„Titel“ type=„xs:string“ minOccurs=„0“ />  
      <xs:element name=„Vorname“ type=„xs:string“ maxOccurs=„unbounded“ />  
      <xs:element name=„Name“ type=„xs:string“ />  
      <xs:choice minOccurs=„0“ maxOccurs=„unbounded“>  
        <xs:element name=„Vorlesung“ type=„xs:string“ />  
        <xs:element name=„Seminar“ type=„xs:string“ />  
      </xs:choice>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```



Vergleich DTD - XML Schema (2)

Mixed Content:

```
<!ELEMENT Text (#PCDATA | Link)*>
```

```
<xs:element name="Text">  
  <xs:complexType mixed="true">  
    <xs:choice minOccurs="0" maxOccurs="unbounded">  
      <xs:element name="Link" type="LinkType"/>  
    </xs:choice>  
  </xs:complexType>  
</xs:element>
```

Leeres Element:

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

```
<xs:element name="br">  
  <xs:complexType></xs:complexType>  
</xs:element>
```



XML Schema: Attributdeklaration

■ Attributdeklarationen

- können **global** erfolgen **oder lokal** am Ende einer ComplexType-Definition stehen
- können wie Elemente direkt deklariert werden oder sich per type bzw. ref auf andere Definitionen beziehen
- **nur SimpleType-Datentypen** erlaubt

■ Beispiel

```
<xs:element name="Vorlesung">  
  <xs:complexType>  
    <xs:sequence>...</xs:sequence>  
    <xs:attribute name="Uhrzeit" type="xs:time" use="required"/>  
  </xs:complexType>  
</xs:element>
```



Attributdeklarationen (2)

■ Attribute für Attribute:

- Name, Type
- Use (optional/required)
- Default, Fixed

■ Beispiele

```
<xs:attribute name="Einheit" type="xs:string" fixed="km"/>
```

```
<xs:attribute name="Farbe" default="blau">  
  <xs:simpleType>  
    <xs:restriction base="xs:NMTOKEN">  
      <xs:enumeration value="rot" />  
      <xs:enumeration value="gelb" />  
      <xs:enumeration value="blau" />  
    </xs:restriction>  
  </simpleType>  
</xs:attribute>
```

```
<xs:attribute name="id" type="xs:ID" use="required"/>
```

```
<xs:attribute name="Artikel" type="xs:IDREF" use="required"/>
```



XML Schema: Schlüsselbedingungen

■ Eindeutigkeitsbedingung: unique

- Selector-Ausdruck: Teilobjekte eines Dokuments, auf die Bedingung anzuwenden ist (XPath-Ausdruck)
- field-Ausdruck: Element- bzw. Attributwert, bzgl. dessen (Eindeutigkeit) getestet werden soll

```
<xs:element name="VLVerzeichnis">  
  <xs:complexType>...</xs:complexType>  
  <xs:unique name="UniqueBed">  
    <xs:selector xpath="./Vorlesung" />  
    <xs:field xpath="Thema/text()" />  
  </xs:unique>  
</xs:element>
```

■ Schlüsselbedingung/-referenz: key / keyref

```
<xs:key name="ArtikelKey">  
  <xs:selector xpath="./Produkte/Artikel" />  
  <xs:field xpath="@aID" />  
</xs:key>  
  
<xs:keyref name="ArtikelKeyRef" refer="ArtikelKey">  
  <xs:selector xpath="/Verzeichnis/Artikel" />  
  <xs:field xpath="@artikelID" />  
</xs:keyref>
```



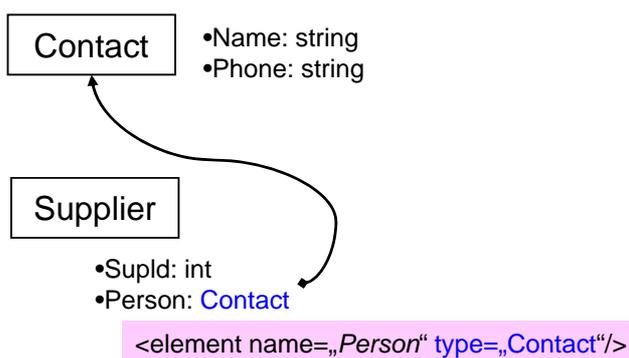
XML Schema: Modellierungsstile

- verschiedene Modellierungskonzepte für Entwurf eines Schemas
- Beschränkung auf bestimmte Konstrukte beeinflusst Sichtbarkeit und Wiederverwendbarkeit von Element/Typ-Deklarationen
 - nur globale Elemente und benannte Typen können wiederverwendet werden
- wichtige Unterscheidungsmerkmale für Schema-Design sind
 - Ableitung neuer Typen: Aggregation vs. Spezialisierung
 - Verwendung lokaler vs. globale Elemente
 - Verwendung benannter Typen vs. direkter Definition der Typen in Elementdeklaration
 - Nutzung verteilter Schemas
- Unterscheidung von 3 grundlegenden Modellierungsstilen
 - Lokale Elementmodellierung: Matroschka-Design (inline-Definition von Komponenten)
 - Globale Elementmodellierung: Salami-Design (Element-Referenzierung für gemeinsame Komponenten)
 - Typ-Modellierung: Jalousie-Design (Typ-Referenzierung für gemeinsame Komp.)

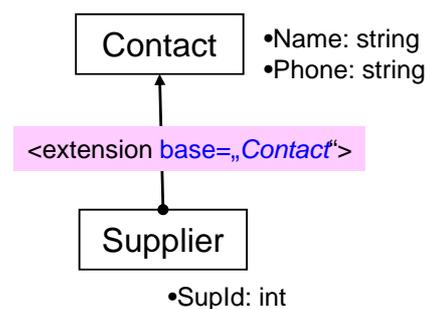


Aggregation vs. Spezialisierung

- unterschiedliche Modellierungsvarianten zur Ableitung neuer Typen aus bestehenden
 - Aggregation / Komposition: Referenzierung oder direkte Einbettung von Komponententypen
 - Spezialisierung (Sub-Classing): Nutzung der Restriction / Extension-Mechanismen
- rekursive Anwendung führt zu Aggregations- bzw. Spezialisierungshierarchien / -graphen



A) Aggregation

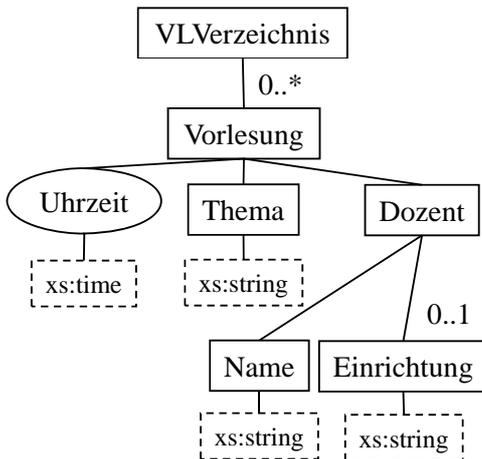


B) Spezialisierung



Lokale Elementmodellierung (Matroschka-Design)

- nur lokale Deklarationen (außer Wurzelement)
- Redundanz, wenn gleiche Elemente, Attribute oder Elementinhalte an verschiedenen Stellen benötigt werden



```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="VLVerzeichnis">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Vorlesung" minOccurs="0"
          maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Thema" type="xs:string"/>
              <xs:element name="Dozent">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="Einrichtung" type="xs:string"
                      minOccurs="0"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Uhrzeit" type="xs:time" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>

```



Globale Elementmodellierung (Salami-Design)

- globale Elementdeklarationen (ähnlich DTD)
- Elementänderungen wirken sich überall aus, wo Element referenziert ist
- flexible Dokumentausgestaltung (keine Beschränkung auf 1 Wurzelement)
- Elementnamen müssen eindeutig sein
- Elementnamen können bei Verwendung nicht angepasst werden

```

<xs:schema xmlns:xs="...">
  <xs:element name="VLVerzeichnis">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Vorlesung" minOccurs="0"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Vorlesung">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Thema" />
        <xs:element ref="Dozent" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```

  <xs:attribute name="Uhrzeit" type="xs:time"
    use="required" /> </xs:complexType>
</xs:element>

  <xs:element name="Dozent">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name" />
        <xs:element ref="Einrichtung" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Thema" type="xs:string" />
  <xs:element name="Name" type="xs:string" />
  <xs:element name="Einrichtung" type="xs:string" />
</xs:schema>

```



Typ-Modellierung (Jalousie-Design)

- globale (benannte) Typ-Deklarationen
- lokale Elementdeklarationen (außer Wurzelement)
 - Nur rekursive Elemente müssen global definiert werden (ref-Verweise)
- hohe Wiederverwendbarkeit
- Sichtbarkeit kann durch Verwendung globaler Typ-Deklarationen gesteuert werden
- Modularisierbarkeit (Aufbau globaler Typbibliotheken) -> verteilte Schemas

```
<xs:schema xmlns:xs="http://www.w3.org/...">
<xs:element name="VLVerzeichnis"
  type="VLVerzeichnisTyp">
<xs:complexType name="VorlesungTyp">
  <xs:sequence>
    <xs:element name="Thema" type="xs:string" />
    <xs:element name="Dozent" type="DozentTyp" />
  </xs:sequence>
  <xs:attribute name="Uhrzeit" type="xs:time"
    use="required" />
</xs:complexType>
</xs:element>
</xs:schema>
```

```
<xs:complexType name="VLVerzeichnisTyp">
  <xs:sequence>
    <xs:element name="Vorlesung" type="VorlesungTyp"
      minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DozentTyp">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />
    <xs:element name="Einrichtung" type="xs:string"
      minOccurs="0" />
  </xs:sequence>
</xs:complexType>
</xs:schema>
```



XML Schema und Namensräume

- XML Schema baut auf und unterstützt Namensräume
- bisherige Schemabeispiele enthielten 'unqualifizierte' Definitionen
- Namensraumzuordnung im Schema und im Dokument erforderlich
 - Attribut targetNamespace im Schema
 - Verweise im Schema auf definierte Elemente müssen qualifizierte Namen verwenden

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://uni-leipzig.de/VLV"
  xmlns:v="http://uni-leipzig.de/VLV" >
<element name="VLVerzeichnis">
  <complexType>
    <sequence>
      <element ref="v:Vorlesung"
        minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>
<element name="Vorlesung"> ... </element>
<element name="Thema" type="string" /> ...
```

```
<?XML version="1.0"?>
<VLVerzeichnis xmlns="http://uni-leipzig.de/VLV">
  <Vorlesung Uhrzeit="15:15 Uhr">
    <Thema>DBS2</Thema>
    <Dozent>
      <Name>Prof. Rahm</Name>
      <Einrichtung>Uni Leipzig</Einrichtung>
    </Dozent>
  </Vorlesung>
</VLVerzeichnis>
```



Nutzung mehrerer Schemas

■ import-Anweisung: Einbindung anderer Schemas / Namensräume

- Zugriff per Referenz auf globale Komponenten (Elemente, Attribute, Typen) des importierten Schemas
- Referenz muss qualifizierten Namen verwenden

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:de.uni-leipzig.bibl"
  xmlns:bib="urn:de.uni-leipzig.bibl"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
<import namespace="http://purl.org/dc/elements/1.1/" />
...
<element name="book">
  <complexType>
    <sequence>
      <element ref="dc:title" />
      <element ref="dc:publisher" />
      <element ref="dc:identifier" />
      <element name="location" type="string" />
      <element name="identifier" type="string" />
    </sequence>
  </complexType>
</element> ...
</schema>
```

```
<book xmlns="urn:de.uni-leipzig.bibl"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Web und Datenbanken</dc:title>
  <dc:publisher>dpunkt-Verlag</dc:publisher>
  <dc:identifier>3-89864-189-9</dc:identifier>
  <location>Zweigstelle 1</location>
  <identifier>ST 6519</identifier>
</book>
```



Nutzung mehrerer Schemas (2)

■ any: Platzhalter für beliebige Elemente

- kann auf globale (Wurzel-) Elemente eines anderen Schemas begrenzt werden
- feste Position im nutzenden Schema
- flexible Nutzung von Fremdschemas; Flexibilität gegenüber Schemaänderungen

■ anyAttribute: Platzhalter für Attribute (anderer Schemas/Namensräume)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:de.uni-leipzig.bibl"
  xmlns:bib="urn:de.uni-leipzig.bibl" >
...
<element name="book">
  <complexType>
    <sequence>
      <any namespace="http://purl.org/dc/elements/1.1"
        minOccurs="0" maxOccurs="unbounded"/>
      <element name="location" type="string" />
      <element name="identifier" type="string" />
    </sequence>
  </complexType>
</element>
...
</schema>
```

```
<book xmlns="urn:de.uni-leipzig.bibl"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>Web und Datenbanken</dc:title>
  <dc:publisher>dpunkt-Verlag</dc:publisher>
  <dc:identifier>3-89864-189-9</dc:identifier>
  <location>Zweigstelle 1</location>
  <identifier>ST 6519</identifier>
</book>
```



Nutzung mehrerer Schemas (3)

- Schema kann über mehrere Dateien / Dokumente verteilt gespeichert werden (z.B. separate Speicherung von Typ-Definitionen)
- **Include-Anweisung**: Zusammenfügung von Dokumenten zu einem Schema / Namensraum
 - Begrenzte Umdefinitionen möglich (REDEFINE)

purchaseorder.xsd

```
<schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="purchaseorder.xsd"
        xmlns="purchaseorder.xsd" >
  <xs:include schemaLocation="PartyType.xsd"/>
  <xs:include schemaLocation="NameType.xsd"/>
  <xs:element name="Supplier" type="PartyType"/>
  <xs:element name="Buyer" type="PartyType"/>
  ...
</schema>
```

PartyType.xsd

```
<xs:complexType name="PartyType">
  <xs:element name="Name" type="NameType"/>
</xs:complexType>
```

NameType.xsd

```
<xs:simpleType name="NameType">
  <xs:restriction base="string"/>
</xs:simpleType>
```



Zusammenfassung

- XML:
 - flexibles Format für strukturierte und semistrukturierte Daten
 - dominierendes Austauschformat zwischen Web-Anwendungen
- DTD nur eingeschränkt zur Beschreibung von Schemas geeignet (mangelhafte Typisierung, keine Namensräume)
- XML Schema
 - mächtigere Strukturbeschreibung als DTD
 - ermöglicht Typisierung von Elementinhalten und Attributwerten
 - bietet verschiedene Modellierungsstile
 - eignet sich durch Integritätsbedingungen zur Abbildung von relationalen Datenbankschemas
- Namensräume erlauben die gemeinsame Nutzung verschiedener Schemas / globaler Typen in einem Dokument

