

# Schema als Alternative zur DTD

# Gute Gründe für DTD

- DTD ist eine einfache, ausgereifte und getestete Technologie aus dem SGML-Umfeld.
- DTD ist fester Bestandteil des XML-Standards.
- Fast alle XML-Tools unterstützen die Verwendung von DTD.
- Viele XML-Applikationen (XHTML) werden durch DTDs festgelegt.

# Nachteile der DTD

- Andere Syntax als die XML-Datei.
  - ◆ Parser und Benutzer müssen die Syntax von zwei verschiedenen Sprachen kennen.
- Sehr eingeschränkte Datentypen.
  - ◆ Es sind keine speziellen Datentypen wie Zahlen oder Datum möglich.
- Wenig Möglichkeiten eigene Datentypen zu definieren.



# Lösung: Schema

- Schema ist eine XML-Applikation zur Beschreibung der Struktur von XML-Dokumenten.
- Es werden Datentypen gängiger Programmiersprachen unterstützt.
- Vorhandene Datentypen können zu eigenen Datentypen kombiniert werden.
- Datentypen können auf der Basis vorhandener oder eigener Datentypen erweitert oder eingeschränkt werden.



# Vergleich DTD und XML-Schema

DTD	XML-Schema
Spezielle Syntax	XML-Syntax
Sehr beschränkte Datentypen (nur für Attributwerte, nicht für Elementinhalte)	Reichhaltige Datentypen: Zahlen, Boolean, Datum, Zeit, URIs etc.
Keine Unterstützung von Namensräumen	Namensräume werden unterstützt
Asymmetrien zwischen Element- und Attributtypen	Polymorphie (Substituierbarkeit) von Typen
Constraints (Schlüssel) nur mit ID und IDREF, eingeschränkter Typ (nur NAME)	Allgemeine Schlüssel- und Fremdschlüsselbedingungen
Erweiterung von Typen nur umständlich über Parameter Entities	Typen durch Benutzer definierbar
Keine Vererbung	Vererbung
	Verteilte Schemata



# Dokumentation zu Schema

- Schema wird vom W3C standardisiert.
  - ◆ *<http://www.w3.org/XML/Schema>*
- Es werden drei verschiedene Dokumentationen angeboten.
  - ◆ Part0 Primer: Gut lesbares Tutorial mit vielen Beispielen.
  - ◆ Part1 Structures: Genaue Beschreibung der Struktur einer Schema-Datei
  - ◆ Part2 Datatypes: Beschreibung der Datentypen, die in Schema verwendet werden.



# Komplexe und simple Typen

- In der Definition einer Dokumentenstruktur durch Schema werden komplexe und simple Typen unterschieden.
  - ◆ Komplexe Typen können Attribute, andere Elemente oder Text als Inhalt enthalten.
  - ◆ Simple Typen beinhalten nur Text.



# Beispieldatei für Schema

- Als Beispiel für Schema nutzen wir eine einfache XML-Datei, die Adressen speichert.
- Die Wurzel ist das Element `<adressbuch>`.
- Die Adressen werden durch das Element `<adresse>` beschrieben.
- Jedes Element `<adresse>` besteht aus den Elementen `<name>`, `<vorname>`, `<plz>`, `<ort>`.

# DTD für die Adressdatei

```
<!ELEMENT adressbuch (adresse)+>
```

```
<!ELEMENT adresse      (name,vorname,strasse,plz,ort)>
```

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

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

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

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

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

- ◆ Festlegung der Unterelemente der Elemente `<adressbuch>` und `<adresse>`. Alle Unterelemente des Elements `<adresse>` müssen erscheinen und beinhalten beliebigen Text. Eine Differenzierung des Inhalts ist nicht möglich.



# Schema-Dateien

- Die Schema-Datei ist eine XML-Datei mit dem Wurzelement `<schema>` in einem vordefinierten Namensraum.

```
<?xml version="1.0"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Inneres -->
</xsd:schema>
```

- Innerhalb des Wurzelements werden Elemente, Attribute und Typen angegeben.
- Typen werden **definiert**, Elemente und Attribute **deklariert**.

# Beispiel komplexe Typen

```
<!-- Definition des Typen für das Element adresse -->
<xsd:complexType name="adresstype">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="vorname" type="xsd:string"/>
    <xsd:element name="strasse" type="xsd:string"/>
    <xsd:element name="plz" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>

<!-- Deklaration des Elements adresse -->
<xsd:element name="adresse" type="adresstype"/>
```



# Erklärung des Beispiels

- Die Schema-Datei deklariert den komplexen Datentyp `adresType`.

```
<xsd:complexType name="adresType">
```

```
...
```

```
</xsd:complexType>
```

- Durch die Gruppierung der Elemente mit dem Element `<xsd:sequence>` ist vorgeschrieben, dass alle Elemente genau einmal und in genau dieser Reihenfolge vorkommen müssen.
- Die einzelnen Elemente haben unterschiedliche Datentypen (`string` oder `integer`).
- Das Element `<adresse>` hat den Type `adresType`.

```
<xsd:element name="adresse" type="adresType"/>
```



# Schema mit XML-Datei verknüpfen

- Eine XML-Datei, die einem Schema entspricht, ist eine Schema-Instanz.

```
<adressbuch  
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  
  xsi:noNamespaceSchemaLocation="adresse1.xsd"  
>
```

- Alternativ kann mit dem Attribute `xsi:schemaLocation` eine Schema-Datei mit Namensraum angegeben werden.



# Schema-Datentypen

- Einfache Typen (Simple Types)
  - ◆ Eingebaute (built-in), atomare Typen (atomic types)
  - ◆ Von atomaren Typen abgeleitete Typen
    - ▶ durch Restriktionen
    - ▶ Listentypen (list types)
    - ▶ Vereinigungstypen (union types)
  - ◆ Verwendung in Element- und Attributdeklarationen
- Komplexe Typen (Complex Types)
  - ◆ Aus einfachen und komplexen zusammengesetzte Typen
  - ◆ Vererbung möglich
  - ◆ Verwendung nur in Elementdeklarationen



# Einfache Datentypen (Simple Types)

- Einfache Datentypen definieren Wertebereiche von Element-Inhalten und Attributen.

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

```
<xsd:element name="plz" type="xsd:integer"/>
```

```
<xsd:element name="gebdatum" type="xsd:date"/>
```

- Eine ganze Reihe von einfachen Datentypen sind vordefiniert.



# Einige einfache Datentypen (1)

Simple Type	Beispiel(e)
string	XML ist toll
byte	-1, 126
unsignedByte	0, 126
hexBinary	0FB7
integer	-126789, -1, 0, 1, 126789
positiveInteger	1, 126789
negativeInteger	-126789, -1
nonNegativeInteger	0, 1, 126789
int	-1, 126789675
unsignedInt	0, 1267896754
long	-1, 12678967543233
unsignedLong	0, 12678967543233
short	-1, 12678
decimal	0, 12678
boolean	true, false, 1, 0

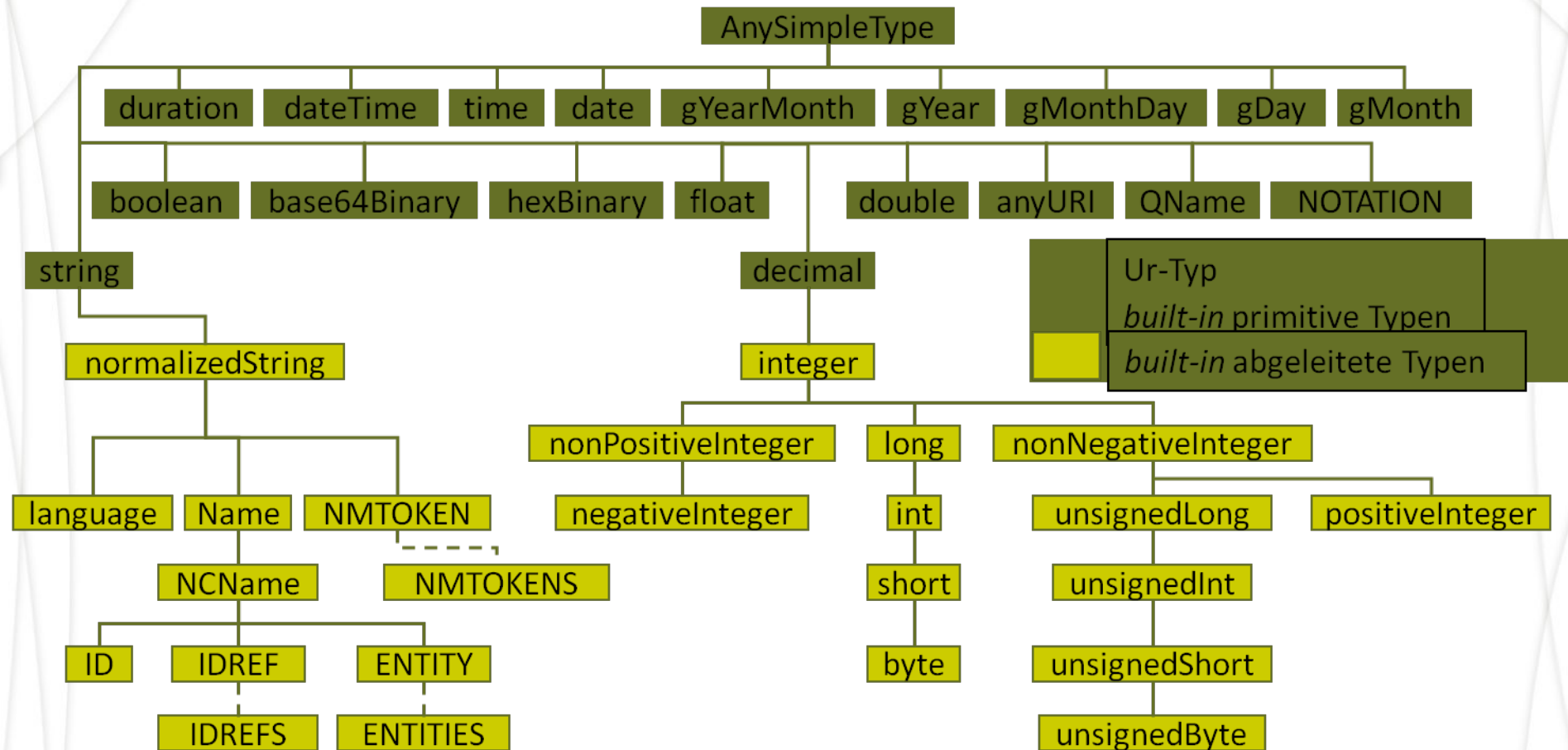


# Einige einfache Datentypen (2)

Simple Type	Beispiel(e)	Kommentar(e)
float	-INF, -1E4, -0, 0, 12.78E-2, 12	32-bit, einfache Genauigkeit
double	-INF, -1E4, -0, 0, 12.78E-2, 12	64-bit, doppelte Genauigkeit
time	13:20:00.000	
date	2009-03-12	12. März 2009
duration	P1Y2M3DT10H30M12.3s	1 Jahr, 2 Monate, 3 Tage, 10 Std., 30 Min., 12.3 Sek.
gMonthDay	--02-03	jeden 3. Februar
gYear	2005	2005
Name	shipTo	XML 1.0 Name
Qname	po:US-Adresse	NXML Namensraum Qname
NCName	USAdresse	unqualifizierter Name
anyURI	http://tutego.com	
ID		XML 1.0 ID Attribut-Typ
IDREF		XML 1.0 IDREF Attribut-Typ
ENTITY		XML 1.0 ENTITY Attribut-Typ
NOTATION		XML 1.0 NOTATION Attribut-Typ
NMTOKEN		XML 1.0 NMTOKEN Attribut-Typ



# Hierarchie der einfachen Typen



# Simple Typen auch für Attribute

- Simple Typen werden sowohl für Elemente als auch für Attribute benutzt.
- Die Syntax ist bei beiden analog.

```
<xsd:element name="plz" type="xsd:integer"/>
```

oder

```
<xsd:attribute name="plz" type="xsd:integer"/>
```

# Facetten

- Einen Wert wie `plzType` wird man von einem einfachen Type ableiten. Der einfache Typ wird i. A. eingeschränkt (Restriktion). Die Einschränkungen von Wertebereichen heißen Facetten.

```
<xsd:simpleType>
  <xsd:restriction base=Basistyp>
    <xsd:Facette value=Wert />
    ...
    <xsd:Facette value=Wert />
  </xsd:restriction>
</xsd:simpleType>
```

XML-Schema kennt 12 Facetten:  
length, pattern,  
minLength, maxLength,  
whitespace, enumeration,  
maxExclusive,  
minExclusive,  
minInclusive,  
maxInclusive, totalDigits,  
fractionDigits



# Simple Typen deklarieren (adresse2.xsd)

```
<xsd:simpleType name="plzType">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="1000"/>  
    <xsd:maxInclusive value="99999" />  
  </xsd:restriction>  
</xsd:simpleType>  
  
<xsd:element name="plz" type="plzType"/>
```

Einschränkung  
von integer auf  
Wertebereich  
1000 - 99999



# Simple Typen deklarieren

- In diesem Beispiel wurde ein simpler Typ deklariert.
- Basis war der vorgegebene Type `xsd:integer`.
- Der eigene Typ `<plzType>` hat einen eingeschränkten Wertebereich.
- Da es sich um einen simplen Typ handelt, kann er keine untergeordneten Elemente enthalten.

# Aufzählungen

- Für erlaubte Werte wird man eine Facette `enumeration` definieren.

```
<xsd:attribute name="monat">
```

```
<xsd:simpleType name="monatsTyp">
```

```
<xsd:restriction base="xsd:string">
```

```
<xsd:enumeration value="Januar"/>
```

```
<xsd:enumeration value="Februar"/>
```

```
<xsd:enumeration value="März"/>
```

```
...
```

```
</xsd:restriction>
```

```
</xsd:simpleType>
```

```
</xsd:attribute>
```

Einschränkung  
von `string` auf  
explizit spezifi-  
zierte Zeichen-  
ketten



# Häufigkeitsbeschränkungen

- Für Elemente kann festgelegt werden, wie häufig diese erscheinen.
- Dazu wird dem Element eines der beiden Attribute **maxOccurs** oder **minOccurs** zugeordnet.
- Beide Element können gleichzeitig verwendet werden.
- Mögliche Werte sind ganze Zahlen größer oder gleich Null.
- Der Wert **unbounded** sagt aus, dass es keine Begrenzung gibt.



# Listentypen

- Der Listentyp ermöglicht eine Liste von Werten, die durch ein XML Whitespace (Leerzeichen, Tabulator, Zeilenende) getrennt werden.

```
<xsd:simpleType name="zahlenliste">  
  <xsd:list itemType="xsd:integer" />  
</xsd:simpleType>
```

Definition eines  
Listentyps

- Beispiel für ein Element vom Typ <meineZahlenliste>  

```
<intListe>1 5 89 3048 -6784375</intListe>
```
- Listen von Listen sind unzulässig!



# Attribute

- Neben dem Inhalt der Elemente spielen Attribute eine entscheidende Rolle beim Design von XML-Datenstrukturen.
- Attribute können nur komplexen Typen zugeordnet werden.
- Der Inhalt eines Attributs muss ein simpler Typ sein.

# Attribute (adresse4.xsd)

```
<xsd:complexType name="adressType">
  <xsd:sequence>
    ...
  </xsd:sequence>
  <xsd:attribute name="anrede" type="anredeType"/>
</xsd:complexType>
```

```
<xsd:simpleType name="anredeType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Frau"/>
    <xsd:enumeration value="Herr"/>
  </xsd:restriction>
</xsd:simpleType>
```



Übung 6.2



# Attribute beschränken

- Ein Attribut kann nur einmal oder gar nicht auftreten.
- Das Element `<xsd:attribute>` steuert dies durch das Attribut `use`:
  - ◆ `optional`: Das Attribut muss nicht gesetzt sein.
  - ◆ `prohibited`: Das Attribut ist verboten.
  - ◆ `required`: Das Attribut muss gesetzt werden.
- Das Attribut `default` setzt einen Standardwert.
- Das Attribut `fixed` legt den Wert fest.



Übung 6.3

# Datentypen zusammenfassen

- Bislang wurde nur gezeigt, wie Datentypen eingeschränkt werden können.
- Mehrere Datentypen können aber auch zu einem Datentyp zusammengefasst werden.
- Mit dem Element `<xsd:union>` und dem Attribut `memberTypes` werden Typen zusammengefasst.



Übung 6.4

# Zusammenfassung Schema

- Die beiden Beispiele sollen die Möglichkeiten der Strukturbeschreibung durch Schema aufzeigen.
- Die Beschreibung ist länger als in einer DTD aber strukturierter und detaillierter.
- Es stehen die gängigsten Typen zur Verfügung.
- Eigene Typen können die vorhandenen Typen einschränken und erweitern.
- Dabei können auch die Eigenschaften von mehreren Typen zusammen gefasst werden.



# Links

- [\*http://www.zvon.org/xxl/XMLSchemaTutorial/Output/series.html\*](http://www.zvon.org/xxl/XMLSchemaTutorial/Output/series.html)

