

## XML Vorlesung ETHZ, Sommersemester 2006

# XML Schema Teil I

Erik Wilde

9.5.2006

<http://dret.net/lectures/xml-ss06/>

9.5.2006

XML Vorlesung ETHZ SS 2006

1

## Übersicht

- Nachteile der DTDs
- Simple Types
  - Type Restrictions mit Facets
- Complex Types
  - Model Groups
  - Attribut-Definitionen

9.5.2006

XML Vorlesung ETHZ SS 2006

2

## Nachteile von DTDs

- keine Beziehungen zwischen Elementtypen
  - keine Typ-Hierarchie der Elemente
  - zusammenhangsloses Nebeneinander
- keine Unterstützung von Wiederverwendung
  - verbreitetes Parameter Entity Design Pattern
- keine anwendungsorientierten Datentypen
- keine Unterstützung für XML Namespaces
  - "DTDs and Namespaces don't mix"
- keine XML Syntax
  - kann nicht mit XML Tools verarbeitet werden

9.5.2006

XML Vorlesung ETHZ SS 2006

3

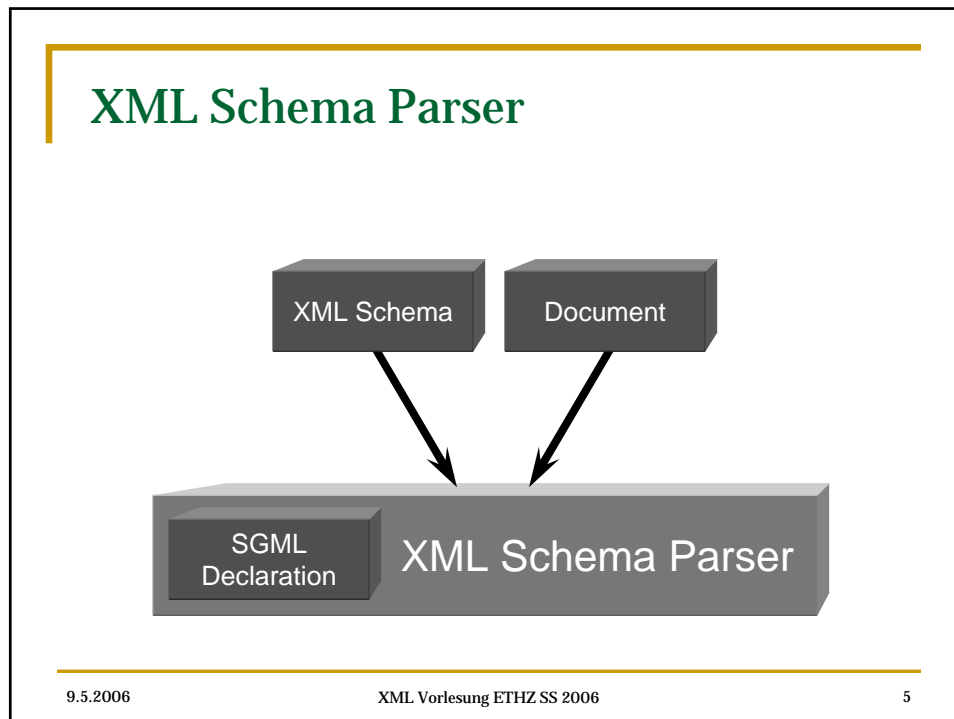
## *Valid und schema-valid XML*

- XML unterscheidet zwischen zwei "Levels"
  - *well-formed* gehorchen dem XML-Standard
  - *valid* sind *well-formed* und gehorchen einer DTD
- *well-formed* und *valid* Konzepte
  - sind direkt im XML Standard definiert
  - können mit DTD und Dokument verifiziert werden
- *schema-valid* Dokumente
  - müssen gemäss eines XML Schema validiert werden
  - gibt es nur mit XML Schema Applikationen
  - haben mehr Randbedingungen als *valid* Dokumente
  - sollten kontrolliert importiert/exportiert werden

9.5.2006

XML Vorlesung ETHZ SS 2006

4



## Namespaces in XML Schema

- XML Schema Namespace (xs: oder xsd: )
  - <http://www.w3.org/2001/XMLSchema>
- XML Schema Instance Namespace (xsi : )
  - <http://www.w3.org/2001/XMLSchema-instance>
- targetNamespace des Schemas
  - der vom Schema definierte Namespace

XML Schema	Instance (Document)
benutzt xs: /xsd: definiert targetNamespace	benutzt targetNamespace benutzt xsi :

9.5.2006 XML Vorlesung ETHZ SS 2006 6

## Was sind "Simple Types"?

- die Grundbausteine von
  - XML Schema
  - und damit auch XML Dokumenten
- Inhalt von Elementen oder Attributen
  - Elemente: `<i sbn>0130655678</i sbn>`
  - Attribute: `<buch i sbn="0130655678">`
- drei Varianten von Simple Types
  - Atomic Types (kleinste Einheit, z.B. Zahlen)
  - List Types (mit Space getrennt, z.B. "3 5 7")
  - Union Types (Vereinigung anderer Simple Types)

9.5.2006

XML Vorlesung ETHZ SS 2006

7

## Named oder Anonymous?

- XML Schema Types kommen in zwei Varianten
  - Named Types
  - Anonymous Types
- eine Frage der Wiederverwendung
  - Named Types haben einen Typ-Namen
    - können (und sollten) wiederverwendet werden
    - werden immer global deklariert (im gesamten Schema)
  - Anonymous Types werden ohne Name verwendet
    - definiert an der Stelle wo sie verwendet werden
    - damit keine Wiederverwendung möglich
- eine wichtige Modellierungsfrage!

9.5.2006

XML Vorlesung ETHZ SS 2006

8

## Named Simple Types

- deklariert durch simpleType
  - ```
<xsd:simpleType name="DressSizeType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="2"/>
    <xsd:maxInclusive value="18"/>
  </xsd:restriction>
</xsd:simpleType>
```
  - Inhalt restriction, list oder union
- definiert, ob weiter abgeleitet werden darf
  - ```
<xsd:simpleType name="DressSizeType" final="#all">
```
  - kann spezifisch die Art von Ableitung verbieten
    - restriction, list, union (und #all)

9.5.2006

XML Vorlesung ETHZ SS 2006

9

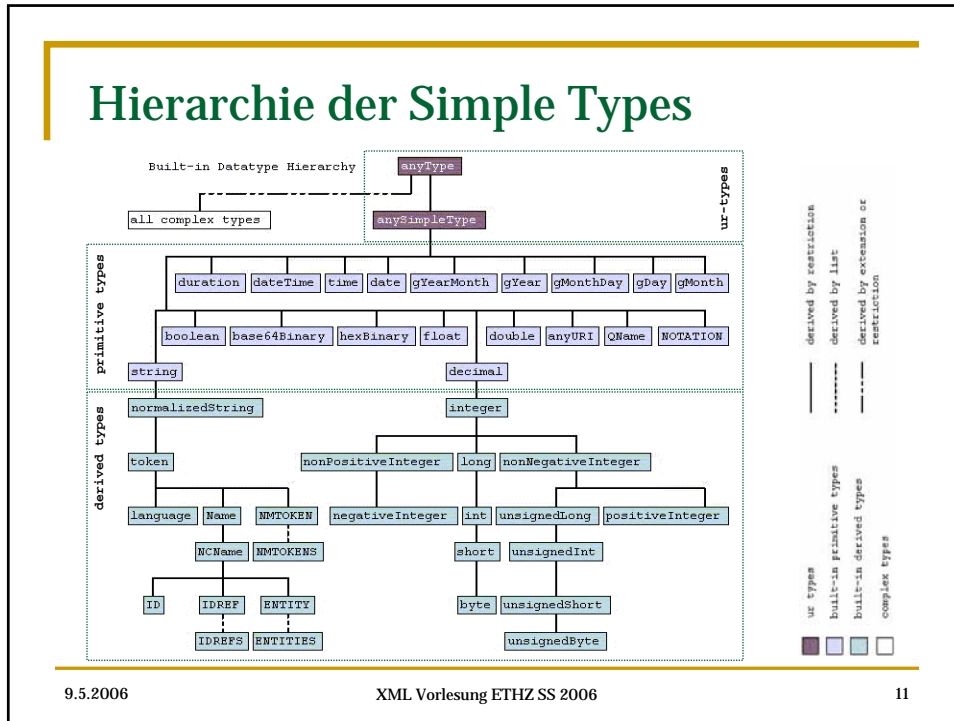
## Anonymous Simple Types

- gleiche Struktur wie Named Types
  - simpleType Element
    - aber ohne name (und u.U. final) Attribut
  - immer innerhalb von anderen Typ-Definitionen
    - element, attribute, restriction, list, union
- die Möglichkeiten sind genau die gleichen
  - ```
<xsd:attribute name="size">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="2"/>
      <xsd:maxInclusive value="18"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

10



## Schema der Built-In Types

```

<xs:simpleType name="integer">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="0" fixed="true"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeInteger">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="positiveInteger">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="1"/>
  </xs:restriction>
</xs:simpleType>
    
```

9.5.2006                      XML Vorlesung ETHZ SS 2006                      12

## Simple Type Restrictions

- Simple Types können durch *Restriction* von anderen Simple Types abgeleitet werden
  - der *Base Type* ist immer ebenfalls ein Simple Type
  - Wurzel dieser Hierarchie ist der *anySimpleType*
  - Erweiterungen sind nur möglich, indem der Simple Type zum Complex Type erweitert wird
- Restrictions enthalten *Facets*
  - Facets sind durch vorgegebene Elemente definiert
  - jede Restriction enthält 0-n Facets
    - Facets können auch wiederholt werden
  - wichtiges Werkzeug zur exakten Typ-Definition

9.5.2006

XML Vorlesung ETHZ SS 2006

13

## Facets

- definieren Einschränkungen von Wertebereichen von Simple Types
- Einteilung in zwei Klassen von Facets
  - fundamental Facets
    - grundlegende Eigenschaften
  - constraining (or non-fundamental) Facets
    - einschränkende Eigenschaften
- Facets haben einen Wert
- die meisten Facets können fixiert werden

9.5.2006

XML Vorlesung ETHZ SS 2006

14

## Constraining Facets

- die praktisch anwendbaren Facets
  - definieren Einschränkungen von Wertebereichen
  - können bei Typableitungen verschärft werden
- es gibt 12 Typen von Constraining Facets
  - length, minLength, maxLength, pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive, totalDigits, fractionDigits
  - nicht alle Facets sind für alle Typen sinnvoll
  - Verfügbarkeit richtet sich nach Primitive Type

## Primitive Types Facets (I)

|          |                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------|
| string   | length, minLength, maxLength, pattern, enumeration, whiteSpace                                                        |
| boolean  | pattern, whiteSpace                                                                                                   |
| float    | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive                              |
| double   | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive                              |
| decimal  | totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive |
| duration | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive                              |
| dateTime | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive                              |
| time     | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive                              |
| date     | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive                              |



## Primitive Types Facets (II)

|              |                                                                                          |
|--------------|------------------------------------------------------------------------------------------|
| gYearMonth   | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive |
| gYear        | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive |
| gMonthDay    | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive |
| gDay         | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive |
| gMonth       | pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive |
| hexBinary    | length, minLength, maxLength, pattern, enumeration, whiteSpace                           |
| base64Binary | length, minLength, maxLength, pattern, enumeration, whiteSpace                           |
| anyURI       | length, minLength, maxLength, pattern, enumeration, whiteSpace                           |
| QName        | length, minLength, maxLength, pattern, enumeration, whiteSpace                           |
| NOTATION     | length, minLength, maxLength, pattern, enumeration, whiteSpace                           |

9.5.2006

XML Vorlesung ETHZ SS 2006

17

## Verwendung von Facets

- Facets können fixiert werden
  - keine weitere Änderung in Subtypen erlaubt
  - wird mit dem fixied Attribut angegeben
  - nicht möglich für pattern und enumeration
- Facets werden vererbt
  - entlang der gesamten Typenhierarchie
  - wiederholte Facets müssen restriktiver sein
    - keine Erweiterung der Einschränkungen erlaubt
    - gilt auch für pattern und enumeration

9.5.2006

XML Vorlesung ETHZ SS 2006

18

## Beispiele für Facets

```
<xs:simpleType name="betragType">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="2"/>
    <xs:minInclusive value="0"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="titelType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="Doktor"/>
    <xs:enumeration value="Professor"/>
  </xs:restriction>
</xs:simpleType>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

19

## Was man mit Facets nicht kann...

- Facets schränken einen Simple Type ein
  - in verschiedenen Dimensionen
  - in u.U. mehreren Schritten (Typableitung)
- können sich nicht auf andere Typen beziehen
  - z.B. End-Datum muss nach dem Anfang liegen
- können keine Auswertungen vornehmen
  - Summenfeld muss Summe bestimmter Felder sein
- d.h. konkrete Instanzen sind unberücksichtigt
- falls notwendig: zusätzliche Mechanismen
  - z.B. Schematron oder programmgesteuert

9.5.2006

XML Vorlesung ETHZ SS 2006

20

## Patterns (Regular Expressions)

- Einschränkung von Simple Types
  - Restrictions mit dem pattern Element
- Beschränkung der lexikalischen Werte
- einfacher Aufbau der Ausdrücke
  - bestehen aus einem oder mehreren *Branches*
    - Branches werden mit einem | getrennt
  - diese bestehen aus einem oder mehreren *Pieces*
  - jedes Piece besteht aus ein bis zwei Teilen
    - ein *Atom*
    - ein optionaler *Quantifier*

9.5.2006

XML Vorlesung ETHZ SS 2006

21

## Beispiele für Regular Expressions

- XML NCName (non-colonized name)
  - $[\backslash i - [ : ] ] [\backslash c - [ : ] ] ^ *$
  - $\backslash i$  bezeichnet *XML initial character*
  - $\backslash c$  bezeichnet *XML name character*
- XML Schema language pattern
  - $([a-zA-Z]{2} | [i I] - [a-zA-Z]^+ | [xX] - [a-zA-Z]{1, 8}) (-[a-zA-Z]{1, 8})^*$
  - $[a-zA-Z]{2}$  bezeichnet den ISO 639 Typ
  - $[i I] - [a-zA-Z]^+$  bezeichnet den IANA Typ
  - $[xX] - [a-zA-Z]{1, 8}$  bezeichnet eigene Typen
  - $(-[a-zA-Z]{1, 8})^*$  für Erweiterungen

9.5.2006

XML Vorlesung ETHZ SS 2006

22

## Union und List Types

- Simple Types können abgeleitet werden durch
  - Restrictions (ergibt einen neuen Simple Type)
    - Verwendung von Facets zur Einschränkung
  - Definition eines List Types
    - erlaubt eine Whitespace-separierte Liste von Werten
  - Definition eines Union Types
    - erlaubt eine Kombination verschiedener Simple Types
- List Types und Union Types sind Simple Types
  - rekursive Kombination ist möglich
  - aber: Listen von Listen (auch indirekt) sind illegal

9.5.2006

XML Vorlesung ETHZ SS 2006

23

## Simple Type Derivation Resultate

|                                                             |             | Base Type |       |                     |
|-------------------------------------------------------------|-------------|-----------|-------|---------------------|
|                                                             |             | Atomic    | List  | Union               |
| <sup>1)</sup> nur enumeration und pattern<br>Facets erlaubt |             | Atomic    | List  | Union               |
| Derivation<br>by...                                         | Restriction | Atomic    | List  | Union <sup>1)</sup> |
|                                                             | List        | List      |       | List                |
|                                                             | Union       | Union     | Union | Union               |

9.5.2006

XML Vorlesung ETHZ SS 2006

24

## Zusammenfassung Simple Types

- Simple Types sind das Grundgerüst
  - Typ-spezifisch sind sie strukturiert
    - Einschränkungen über Factes, z.B. Patterns
  - aus XML Markup Sicht sind sie unstrukturiert
    - List Types sind ein Grenzfall
- sollten das Fundament von Modellierung bilden
  - gute Modellierung sollte Grundtypen definieren
    - alle Einschränkungen so genau wie möglich
    - vergleichbar Invarianten bei Programmiersprachen
  - Validierung macht der Parser

9.5.2006

XML Vorlesung ETHZ SS 2006

25

## Was sind "Complex Types"?

- Elemente mit komplexem Typ haben Child Elements und/oder Attribute
- ein komplexer Typ hat (meist) entweder ein Attribut oder ein Element
  - Attribute haben auch einen Typ, aber nie einen *Complex Type*
- Complex Types haben entweder einen Namen (global definiert) oder sind anonym (lokal definiert)
- Elemente in einem *Complex Type* können nur dann den gleichen Namen haben, wenn sie auch vom gleichen Typ sind (alle anderen Eigenschaften dürfen aber verschieden sein, z.B. mit `nOccurs`)

9.5.2006

XML Vorlesung ETHZ SS 2006

26

## Named Complex Types

- Beispiel:

```
<xs:complexType name="referentTyp">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Vorname" type="xs:string"/>
    <xs:element name="Ti tel " type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="KursReferent" type="referentTyp"/>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

27

## Anonymous Complex Types

- Beispiel:

```
<xs:element name="KursReferent">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Vorname" type="xs:string"/>
      <xs:element name="Ti tel " type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

28

## Content von Complex Types

- der Content eines Elementes sind die Daten (Zeichenketten) und Child Elements zwischen den beiden Tags des Elementes
- es gibt folgende Arten von Content für komplexe Typen:
  - Simple Content
  - Complex Content mit verschiedenen Varianten
    - Element-only Content
    - Mixed Content
    - Empty Content

9.5.2006

XML Vorlesung ETHZ SS 2006

29

## Complex Types + Content Types

|                    |                       |                        |       |
|--------------------|-----------------------|------------------------|-------|
| <b>Simple Type</b> | <b>Complex Type</b>   |                        |       |
|                    | <i>Simple Content</i> | <i>Complex Content</i> |       |
|                    | Element only          | Mixed                  | Empty |

9.5.2006

XML Vorlesung ETHZ SS 2006

30

## Was ist ein Content Model?

- das Content Model eines komplexen Typs ist die Ordnung und Struktur des im komplexen Typ enthaltenen Inhalts
- ein Content Model wird aus einer Kombination von Model Groups (sequence, choice, all), Elementen und Wildcards gebildet

9.5.2006

XML Vorlesung ETHZ SS 2006

31

## Complex Type mit Simple Content

- keine Child Elements
- Inhalt ist eine Zeichenkette
  - d.h. ein Simple Type
- der Unterschied zwischen einem *Simple Type* und einem *Complex Type* mit *Simple Content* ist, dass der letztere Attribute haben darf

9.5.2006

XML Vorlesung ETHZ SS 2006

32



## Simple Content

- Beispiel:

```
<xs:complexType name="TeilnehmerAnzahlTyp">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="Stand" type="xs:date"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="TeilnehmerAnzahl"
  type="TeilnehmerAnzahlTyp"/>

<TeilnehmerAnzahl Stand="2002-03-12">20</TeilnehmerAnzahl >
```

9.5.2006

XML Vorlesung ETHZ SS 2006

33

## Element-only Content

- hat nur Child Elements
- Beispiel:

```
<xs:complexType name="referentTyp">
  <xs:sequence>
    <xs:element name="Vorname" type="xs:string"/>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Titel" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Referent" type="referentTyp"/>

<Referent >
  <Vorname>Andreas</Vorname>
  <Name>Steiner</Name>
  <Titel>Dr. </Titel >
</Referent>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

34

## Mixed Content

- erlaubt gleichzeitig sowohl Zeichen wie auch Child Elements
- nota bene: die Zeichen in komplexen Typen mit Mixed Content haben keinen Typ
  - einziger aus XML Schema Sicht typfreier Inhalt eines XML Dokuments
- Attribut `mixed="true"` setzen (Default ist false)

## Mixed Content

- Beispiel:

```

<xs:complexType name="anmeldeTyp" mixed="true">
  <xs:sequence>
    <xs:element name="Kurs" type="xs:string"/>
    <xs:element name="Teilnehmer" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="AnmeldungsText" type="anmeldeTyp"/>

<AnmeldungsText>Ich melde mich hiermit für den <Kurs>XML Schema
Kurs</Kurs> an. Mit freundlichen Grüßen, <Teilnehmer>Max
Frosch</Teilnehmer></AnmeldungsText>

```

## Empty Content

- erlaubt weder Zeichen noch Child Elements
- Elemente von komplexem Typ mit Empty Content haben oft Attribute
- aber auch ohne Attribute haben solche Elemente ihren Sinn:
  - Beispiel: `<br/>` in XHTML steht für eine neue Zeile
- ein komplexer Typ mit Empty Content wird nur durch seine Struktur gekennzeichnet (und nicht durch ein spezielles Attribut wie z.B. `mixed`)

9.5.2006

XML Vorlesung ETHZ SS 2006

37

## Empty Content

- Beispiel:

```
<xs:complexType name="IetzterStand">
  <xs:attribute name="Datum" type="xs:date"/>
</xs:complexType>

<xs:element name="Version" type="IetzterStand"/>

<Version Datum="1967-08-13"/>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

38

## Elemente in komplexen Typen

- komplexe Typen können drei verschiedene Arten von Element-Deklarationen enthalten:
  - lokale Element-Deklarationen
  - Referenzen zu globalen Element-Deklarationen
  - Wildcards

## Lokale Element-Deklaration

- ein komplexer Typ kann lokale Element-Deklarationen enthalten:
  - alle vorherigen Beispiele von komplexen Typen enthalten lokale Element-Deklarationen
- Beispiel:

```
<xs:complexType name="referentTyp">
  <xs:sequence>
    <xs:element name="Vorname" type="xs:string"/>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Ti tel " type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

## Referenz auf ein Element

- ein komplexer Typ kann Referenzen auf globale Element-Deklarationen enthalten
- anstelle der Attribute name und type wird Attribut ref im komplexen Typ verwendet
- nota bene: Bedingungen mit minOccurs und maxOccurs können nur in Deklarationen von komplexen Typen (und nicht in der Deklaration von globalen Elementen) erscheinen

## Referenz auf ein Element

- Beispiel:

```
<xs:element name="Name" type="xs:string"/>
<xs:element name="Vorname" type="xs:string"/>
<xs:element name="Titel" type="xs:string"/>

<xs:complexType name="referentTyp">
  <xs:sequence>
    <xs:element ref="Name"/>
    <xs:element ref="Vorname"/>
    <xs:element ref="Titel" minOccurs="0" maxOccurs="2"/>
  </xs:sequence>
</xs:complexType>
```

## Model Groups

- es gibt drei unterschiedliche Model Groups:
  - sequence Group
  - choice Group
  - all Group
- abgesehen vom *komplexen Typ mit Empty Content* hat jeder komplexe Typ genau ein Model Group Child (das weitere Model Groups beinhalten darf)

9.5.2006

XML Vorlesung ETHZ SS 2006

43

## Model Groups: Sequence

- sequence Group:
  - bereits in den vorherigen Beispielen verwendet
  - spezifiziert in einem komplexen Typ eine Liste von Elementen und ihre Reihenfolge
  - jedes Element in der Liste muss in der Instanz vorkommen, ausser es wird als *optional* gekennzeichnet (mit  `minOccurs="0"`)
- DTD Syntax
  - `<!ELEMENT demo (e1, e2, e3) >`

9.5.2006

XML Vorlesung ETHZ SS 2006

44

## Model Groups: Choice

- choice Group:
  - spezifiziert eine Liste von Elementen
  - nur genau ein Element der Liste darf in der Instanz erscheinen
  - für eine choice Group kann über minOccurs und maxOccurs die Anzahl ihrer Vorkommnisse spezifiziert werden
  - sequence und choice Groups dürfen beliebig verschachtelt werden
- DTD Syntax
  - `<!ELEMENT demo (e1 | e2 | e3) >`

9.5.2006

XML Vorlesung ETHZ SS 2006

45

## Sequence und Choice (Schema)

```

<xs:complexType name="kursTyp2">
  <xs:sequence>
    <xs:element name="Kursname" type="xs:string"/>
    <xs:choice maxOccurs="3">
      <xs:sequence>
        <xs:element ref="Vorname"/>
        <xs:element ref="Name"/>
        <xs:element ref="Titel"/>
      </xs:sequence>
      <xs:element ref="Name"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

```

9.5.2006

XML Vorlesung ETHZ SS 2006

46

## Sequence und Choice (Instanz)

```
<xs:element name="Kurs2" type="kursTyp2" />
```

```
<Kurs2>  
  <Kursname>XML Schema Kurs</Kursname>  
  <Name>Steiner</Name>  
  <Vorname>Erik</Vorname>  
  <Name>Wilde</Name>  
  <Titel>Dr. </Titel>  
</Kurs2>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

47

## Model Groups: All

- all Group:
  - spezifiziert eine Liste von Elementen
  - alle Elemente in der Liste müssen in einer Instanz erscheinen, höchstens einmal (sie können auch optional sein)
  - die Reihenfolge der Elemente ist beliebig
- eine all Group muss die einzige Model Group in einem komplexen Typ sein
- für Elemente in einer all Group kann minOccurs nur 0 oder 1 sein und maxOccurs nur 1 sein (optional)
- für die all Group selber kann minOccurs nur 0 oder 1 sein und maxOccurs nur 1 sein (optional)

9.5.2006

XML Vorlesung ETHZ SS 2006

48



## Model Groups: All (Schema)

- Beispiel: komplexer Typ mit all Group

```
<xs:complexType name="referentTyp">
  <xs:all>
    <xs:element ref="Vorname"/>
    <xs:element ref="Name"/>
    <xs:element ref="Ti tel "/>
  </xs:all>
</xs:complexType>
```

## Model Groups: All (Schema)

- Beispiel: Element

```
<xs:element name="Kurs">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Kursname" type="xs:string"/>
      <xs:element name="Referent" type="referentTyp"
        maxOccurs="5"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

## Model Groups: All

- Beispiel: Instanz

```
<Kurs>
  <Kursname>XML Schema Kurs</Kursname>
  <Referent>
    <Name>Steiner</Name>
    <Vorname>Andreas</Vorname>
    <Titel>Dr. </Titel >
  </Referent>
  <Referent>
    <Titel>Dr. </Titel >
    <Vorname>Erik</Vorname>
    <Name>Wilde</Name>
  </Referent>
</Kurs>
```

## Verwendung von Attributen

- wie bei Element-Typen können Attribute in komplexen Typen folgendermassen deklariert werden:
  - als lokale Deklarationen
  - als Referenzen zu globalen Deklarationen
  - als *Wildcards*
  - durch *Attribute Group* Referenzen
- Attribut Deklarationen müssen in komplexen Typen nach dem Content Model erscheinen
- die Attribut-Reihenfolge spielt keine Rolle
- Attributnamen müssen innerhalb eines Elements eindeutig sein

## Attribut: Lokale Deklaration

- nur innerhalb des komplexen Typs sichtbar
- Beispiel:

```
<xs:complexType name="kursTyp">
  <xs:sequence>
    <xs:element name="Kursname" type="xs:string"/>
    ...
  </xs:sequence>
  <xs:attribute name="letzteAenderungen"
    type="xs:date"/>
</xs:complexType>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

53

## Attribut: Referenz

- Referenz auf globale Attribut-Deklaration
- Beispiel:

```
<xs:attribute name="letzteAenderungen" type="xs:date"/>
<xs:complexType name="kursTyp">
  <xs:sequence>
    <xs:element name="Kursname" type="xs:string"/>
    ...
  </xs:sequence>
  <xs:attribute ref="letzteAenderungen"
    default="2002-01-01"/>
</xs:complexType>
```

9.5.2006

XML Vorlesung ETHZ SS 2006

54

## Zusammenfassung Complex Types

- Complex Types definieren Elementtypen
  - Content Model als erlaubter Inhalt
  - erlaubte Attribute
- Model Groups definieren Inhaltsmodelle
  - sequence und choice wie in DTDs
  - all als neues Konstrukt
  - feinere Steuerung mit minOccurs
- Attribute sind separate Teile
  - komplette Trennung von den Content Models
  - bekannte Möglichkeiten sind optional oder mit Default