
XML-Schema im Detail

Wie geht es weiter?

letzte Woche

- ☑ Definition von XML-Sprachen
- ☑ DTDs und XML-Schema anhand eines Beispiels

heutige Vorlesung

- XML-Schema
 - Datentypen
 - Element- und Attribut-Deklarationen
 - Polymorphismus
- Relax NG als Alternative?

Datentypen

Wozu Datentypen?

```
<location>  
  <latitude>  
    { $x \in \text{Float} : -90 \leq x \leq 90$ }  
  </latitude>  
  <longitude>  
    { $x \in \text{Float} : -180 \leq x \leq 180$ }  
  </longitude>  
  <uncertainty units=" {m, ft} ">  
    { $x \in \text{Float} : x \geq 0$ }  
  </uncertainty>  
</location>
```

Datentypen definieren

- z.B. gültigen Inhalt von latitude, longitude, uncertainty und units
- aber auch gültigen Inhalt von location

Können zusammen mit Parameter einer Prozedur übergeben werden (→ Web Services)

Was sind Datentypen?



- **Dokument-Typ**: gültiger Inhalt von XML-Dokumenten eines bestimmten Namenraumes
- **Datentyp**: gültiger Inhalt von Elementen oder Attributen
- Formal repräsentiert ein Datentyp eine Menge von gültigen Werten, den so genannten **Wertebereich**.

Einfache vs. komplexe Datentypen



Einfache Datentypen (*simple types*)

beschreiben *unstrukturierten* Inhalt *ohne* Elemente oder Attribute (PCDATA)

Komplexe Datentypen (*complex types*)

beschreiben *strukturierten* XML-Inhalt *mit* Elementen oder Attributen

```
<xsd:element name="BookStore">  
  <xsd:complexType>  
    Liste von Büchern  
  </xsd:complexType>  
</xsd:element>
```

- anonymer Datentyp
- lokale Definition

```
<xsd:complexType name="BookStoreType">  
  Liste von Büchern  
</xsd:complexType>
```

- benannter Datentyp
- globale Definition
- wiederverwendbar

Element- Deklarationen

Element-Deklaration: 1. Möglichkeit



- Element kann mit Datentyp deklariert werden, der woanders definiert ist:

```
<xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" type="BookType"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<BookStore>
  <Book>
    BookType
  </Book>
  ...
</BookStore>
```

Instanz

Element-Deklaration: 1. Möglichkeit



```
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int"/>
```

- **name**: Name des deklarierten Elementes
- **type**: Datentyp, der woanders definiert ist
- **minOccurs**: so oft erscheint das Element *mindestens* (nicht-negative Zahl)
- **maxOccurs**: so oft darf das Element *höchstens* erscheinen (nicht-negative Zahl oder unbounded).
- Default-Werte von minOccurs und maxOccurs jeweils 1
- Beachte: minOccurs und maxOccurs nur erlaubt, wenn Element-Deklaration Kind-Element von xsd:sequence (oder ähnlichen Konstrukten)!

Element-Deklaration: 2. Möglichkeit



- Element kann auch mit anonymen Datentyp deklariert werden:

```
<xsd:element name="BookStore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" type="BookType"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Instanz

```
<BookStore>
  <Book> ... </Book>
  <Book> ... </Book>
</BookStore>
```

Element-Deklaration: 2. Möglichkeit



- entweder ist anonymes Datentyp komplex:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
  <xsd:complexType>
    ...
  </xsd:complexType>
</xsd:element>
```

- oder einfach:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
  <xsd:simpleType>
    ...
  </xsd:simpleType>
</xsd:element>
```

Attribut- Deklarationen

Deklaration von Attributen

- ähnlich wie Elemente
- aber nur einfache Datentypen erlaubt
- entweder Deklaration mit Datentyp, der woanders definiert ist:

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

- oder Deklaration mit anonymen Datentyp:

```
<xsd:attribute name= "name">  
  <xsd:simpleType>  
    ...  
  </xsd:simpleType>  
</xsd:attribute>
```

Deklaration von Attributen



```
<xsd:attribute name="name" type="type" use="use"
  default="value" />
```

- **use="optional"** Attribut optional
- **use="required"** Attribut obligatorisch
- **use="prohibited"** Attribut unzulässig
- Beachte: Wenn nichts anderes angegeben, ist das Attribut optional!
- **default:** Standard-Wert für das Attribut

Globale vs. lokale Attribute



```
<xsd:schema ...>
  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence>
        ...
      </xsd:sequence>
      <xsd:attribute name="local-attribute" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:attribute name="global-attribute" type="xsd:string"/>
</xsd:schema>
```

lokal: gehört
zu root

global: gehört
zu *allen*
Elementen

global: Deklaration Kind von `xsd:schema`

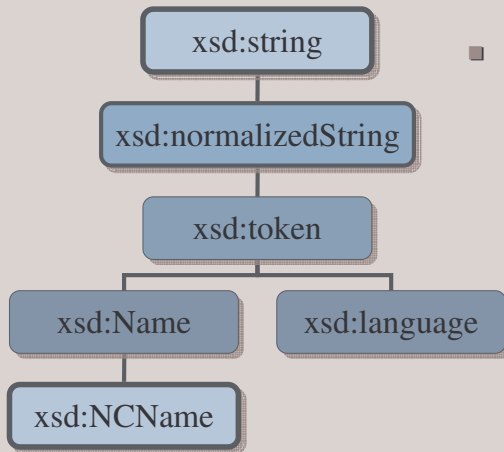
lokal: Deklaration *kein* direktes Kind von `xsd:schema`

Einfache Datentypen

Einfache Datentypen

einfache Datentypen (*simple types*)

- beschreiben *unstrukturierten* Inhalt ohne Elemente oder Attribute (PCDATA)
- Schema der Schemata definiert 44 einfache Datentypen
- eigene einfache Datentypen können definiert werden



+ 38 weitere einfache Datentypen

- **xsd:normalizedString**: string ohne Wagenrücklauf (CR), Zeilenvorschub (LF) und Tabulator.
- **xsd:token**: normalizedString ohne zwei aufeinander folgende Leerzeichen und ohne Leerzeichen am Anfang und Ende.
 - **xsd:Name**: token, der Namenskonvention von XML entspricht (mit oder ohne Präfix)
 - **xsd:NCName**: Name ohne Präfix.
 - **xsd:language**: Bezeichner für Sprache, wie z.B. „EN“

Abgeleitete Datentypen (*derived types*)

auf Basis von anderen Datentypen definiert, z.B. durch Einschränkung oder Erweiterung

Primitive Datentypen (*primitive types*)

nicht von anderen Datentypen abgeleitet

	primitiv	abgeleitet
einfach	xsd:string	<pre><xsd:simpleType name="longitudeType"> <xsd:restriction base="xsd:integer"> <xsd:minInclusive value="-180"/> <xsd:maxInclusive value="180"/> </xsd:restriction> </xsd:simpleType></pre>
komplex	<pre><xsd:complexType> <xsd:sequence> ... </xsd:sequence> </xsd:complexType></pre>	<pre><xsd:complexType name="BookTypeWithID"> <xsd:complexContent> <xsd:extension base="BookType"> <xsd:attribute name="ID" type="xsd:token"/> </xsd:extension> </xsd:complexContent> </xsd:complexType></pre>

Abgeleitete einfache Datentypen



1. Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines einfachen Datentyps

2. Vereinigung

Vereinigung der Wertebereiche mehrerer einfacher Datentypen

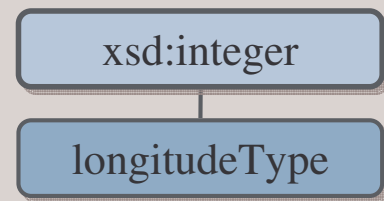
3. Listen

Liste als String, wobei Elemente durch Leerzeichen getrennt sind

1. Einschränkung (Teilmenge)



```
<xsd:simpleType name="longitudeType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-180"/>
    <xsd:maxInclusive value="180"/>
  </xsd:restriction>
</xsd:simpleType>
```



hier konjunktiv
verknüpft!

- longitudeType = { n aus xsd:integer: $n \geq -180, n \leq 180$ }
- Für jeden einfachen Datentyp bestimmte **zulässige Einschränkungen** (*constraining facets*) festgelegt.
- Z.B. sind xsd:minInclusive und xsd:maxInclusive zulässig für xsd:integer, nicht jedoch für xsd:string.

Zulässige Einschränkungen



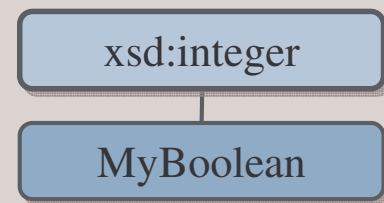
- enumeration: Zählt erlaubte Werte explizit auf
- maxExclusive: <
- maxInclusive: ≤
- minExclusive: >
- minInclusive: ≥
- fractionDigits: max. Anzahl von Stellen hinter dem Komma
- length: Anzahl von Zeichen oder Listenelemente
- minLength: min. Anzahl von Zeichen oder Listenelemente
- pattern: Zeichenketten als reguläre Ausdrücke
- whiteSpace: legt fest, wie Formatierungen behandelt werden

Für bestimmte Datentypen nur bestimmte
Einschränkungen zulässig!

Beispiel xsd:enumeration



```
<xsd:simpleType name="MyBoolean">
  <xsd:restriction base="xsd:integer">
    <xsd:enumeration value="0"/>
    <xsd:enumeration value="1"/>
  </xsd:restriction>
</xsd:simpleType>
```



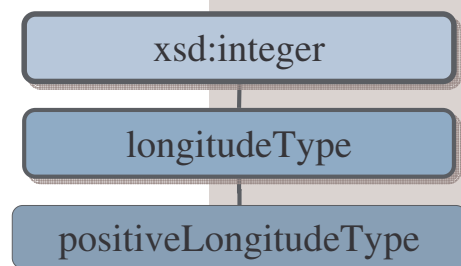
hier disjunktiv
verknüpft!

- MyBoolean = { n aus xsd:integer: $n = 0$ oder $n = 1$ }
- **xsd:enumeration**: zählt alle Elemente des Wertebereiches explizit auf.
- auch für xsd:string zulässig

Vererbung zulässiger Einschränkungen



```
<xsd:simpleType name="longitudeType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-180"/>
    <xsd:maxInclusive value="180"/>
  </xsd:restriction>
</xsd:simpleType>
```



```
<xsd:simpleType name="positiveLongitudeType">
  <xsd:restriction base="longitudeType">
    <xsd:minInclusive value="0"/>
  </xsd:restriction>
</xsd:simpleType>
```

longitudeType erbt zulässige
Einschränkungen von xsd:integer.

2. Vereinigung



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

```
<xsd:union>
```

```
<xsd:simpleType>
```

```
<xsd:restriction base="xsd:integer"/>
```

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

```
<xsd:simpleType>
```

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

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

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

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

```
</xsd:union>
```

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

MyInteger

xsd:integer

{ unknown }

MyInteger =

xsd:integer U

{ s aus xsd:string: s = unknown }

Struktur von xsd:simpleType



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

```
<xsd:union>
```

```
<xsd:simpleType>
```

```
<xsd:restriction base="xsd:integer"/>
```

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

```
<xsd:simpleType>
```

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

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

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

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

```
</xsd:union>
```

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

```
<xsd:simpleType>
```

```
xsd:integer
```

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

Beachte: simpleType muss immer restriction, union oder list als Kind-Element haben.

3. Listen



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

- IntegerList ist Liste von Integern.
- Elemente der Liste durch *beliebige* unsichtbare Zeichen (*white space*) getrennt
- gültige Werte von IntegerList z.B.:

```
108 99 205 23 0
```

```
108 99
```

```
205 23 0
```

```
108 99      205 23 0
```

Unstrukturierte Listen



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

- Beachte: IntegerList ist *einfacher* Datentyp, beschreibt also *unstrukturierten* Inhalt (PCDATA):

```
108 99 205 23 0
```

- *Strukturierte* Liste könnte hingegen so aussehen:

```
<element>108</element>
<element>99</element>
<element>205</element>
<element>23</element>
<element>0</element>
```

Komplexe Datentypen

Komplexe Datentypen

Komplexe Datentypen (*complex types*)

- beschreiben *strukturierten* XML-Inhalt mit Elementen oder Attributen
- drei Möglichkeiten, komplexe Datentypen zu beschreiben:
 1. Sequenz
 2. Menge
 3. Auswahl

1. Sequenz



```
<xsd:complexType name="BookType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded" />
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Publisher" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
```

- Reihenfolge festgelegt
- Elemente erscheinen so häufig, wie mit minOccurs/maxOccurs festgelegt.
- sequence selbst kann minOccurs und maxOccurs spezifizieren

```
<Title>String</Title>
<Author>String</Author>
<Author>String</Author>
...
<Date>String</Date>
<ISBN>String</ISBN>
```

2. Menge



```
<xsd:complexType name="BookType">
  <xsd:all>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
  </xsd:all>
</xsd:complexType>
```

- Reihenfolge der Elemente beliebig
- Jedes Element erscheint genau einmal.

```
<Author>String</Author>
<Title>String</Title>
<Date>String</Date>
<Publisher>String</Publisher>
<ISBN>String</ISBN>
```

Menge: minOccurs und maxOccurs



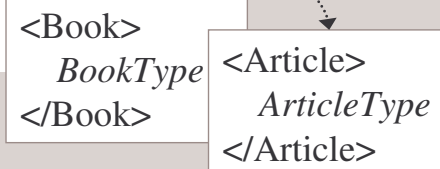
```
<xsd:complexType name="BookPublication">
  <xsd:all>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string" minOccurs="0"/>
  </xsd:all>
</xsd:complexType>
```

- Anzahl der Elemente mit minOccurs und maxOccurs
- allerdings mit Einschränkungen:
- minOccurs: nur "0" oder "1"
- maxOccurs: nur "1"

3. Auswahl



```
<xsd:complexType name="PublicationType">
  <xsd:choice>
    <xsd:element name="Book" type="BookType"/>
    <xsd:element name="Article" type="ArticleType"/>
  </xsd:choice>
</xsd:complexType>
```



- Inhalt besteht aus *genau einem* der aufgezählten Elemente.
- hier also: entweder Book- oder Article-Element
- choice selbst kann minOccurs und maxOccurs spezifizieren

Verschachtelungen



- sequence, choice, all und Rekursion können verschachtelt werden:

```
<xs:element name="Chap" type="ChapType"/>
<xs:complexType name="ChapType">
  <xs:sequence>
    <xs:element name="Title" type="TitleType"/>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="Text" type="TextType"/>
      <xs:element name="Chap" type="ChapType"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

xs:all hier nicht erlaubt wegen Kind-Element xs:choice!

entspricht:

```
<!ELEMENT Chap (Title, (Text | Chap)+)>
```

Gemischter Inhalt



Instanz

```
<Book>
  <Title>My Life and Times</Title>
  <Author>Paul McCartney</Author>
  <Date>July, 1998</Date>
  <ISBN>94303-12021-43892</ISBN>
  Dies ist unzulässiger Text...
  <Publisher>McMillin Publishing</Publisher>
</Book>
```

- Text (PCDATA) *zwischen* Elementen normalerweise *nicht* erlaubt
- kann aber im Schema als zulässig erklärt werden

```
<xsd:complexType name="BookType" mixed="true">
  <xsd:sequence>
    <xsd:element name="Title" type="xsd:string"/>
    <xsd:element name="Author" type="xsd:string"/>
    <xsd:element name="Date" type="xsd:string"/>
    <xsd:element name="ISBN" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

- **mixed= "true"**: Text (PCDATA) zwischen Kind-Elementen zulässig

Abgeleitete komplexe Datentypen

1. Erweiterung

Datentyp wird durch zusätzliche Attribute und Elemente erweitert.

2. Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines Datentyps

Abgeleitete einfache Datentypen



1. Einschränkung (Teilmenge)

Einschränkung des Wertebereiches eines einfachen Datentyps

2. Vereinigung

Vereinigung der Wertebereiche mehrerer einfacher Datentypen

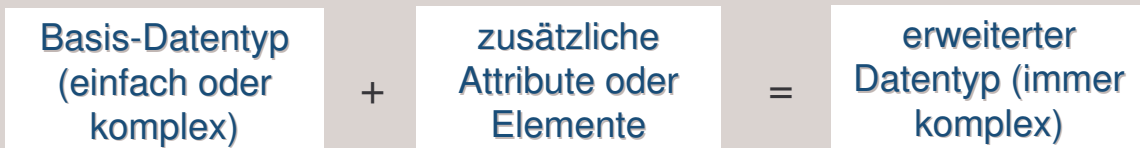
3. Listen

Liste als String, wobei Elemente durch Leerzeichen getrennt sind

1. Erweiterung



- Datentyp kann durch zusätzliche Attribute und Elemente erweitert werden.
- Sowohl einfache als auch komplexe Datentypen können erweitert werden.
- Ergebnis *immer* komplexer Datentyp



Beispiel



```
<xsd:complexType name="StringWithLength">  
  <xsd:simpleContent>  
    <xsd:extension base="xsd:string">  
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>  
    </xsd:extension>  
  </xsd:simpleContent>  
</xsd:complexType>
```

xsd:string + Attribut length = StringWithLength

Basis-Datentyp
(einfach) + zusätzliches
Attribut = erweiterter
Datentyp
(komplex)

xsd:string + Attribut ?



```
<xsd:complexType name="StringWithLength">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

- Nur Elemente können Attribute haben.
- Unstrukturierter Inhalt wie `xsd:string` kann aber *keine* Attribute haben.

Wie ist also diese Erweiterung zu verstehen?

Aha!



```
<xsd:complexType name="StringWithLength">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

- Datentypen *keine* eigenständige Objekte, sondern beschreiben immer Inhalt von Elementen oder Attributen
- Attribut-Werte immer unstrukturiert
- Komplexer Datentyp (wie `StringWithLength`) beschreibt daher immer Inhalt eines Elementes.
- Zusätzliches Attribute `length` wird diesem Element zugeordnet.

Beispiel



```
<xsd:element name="Abstract" type="StringWithLength"/>
<xsd:complexType name="StringWithLength">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Instanz

```
<Abstract length="4">
  Text (StringWithLength)
</Abstract>
```

- Element Abstract hat Inhalt vom Typ StringWithLength.
- Attribut length von StringWithLength wird Element Abstract zugeordnet.

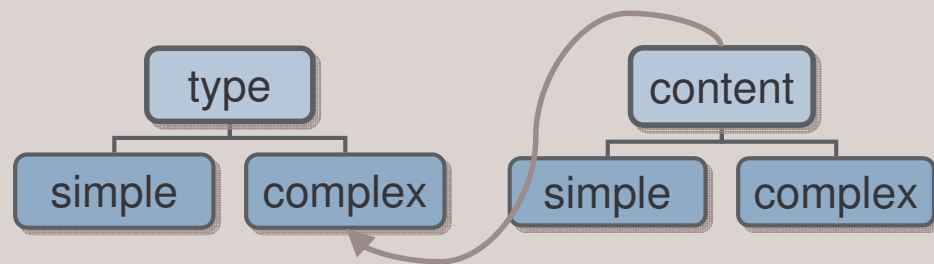
simpleContent vs. complexContent



```
<xsd:complexType name="StringWithLength">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="length" type="xsd:nonNegativeInteger"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

- **simpleContent**: *unstrukturierter* Inhalt (PCDATA) mit Attributen.
- **complexContent**: *strukturierter* Inhalt (mit Elementen).

- wird verlangt, obwohl eigentlich redundant
- erleichtert aber Parsen

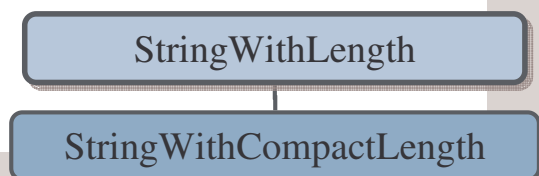


<u>Elemente:</u>	nein	ja	nein	ja
<u>Attribute:</u>	nein	ja	ja	ja

- simpleContent und complexContent dienen zur Unterscheidung komplexer Datentypen:
- strukturierter Inhalt (complexContent) vs. unstrukturierter Inhalt mit Attributen (simpleContent)

2. Einschränkung (Teilmenge)

```
<xsd:complexType name="StringWithCompactLength">
  <xsd:simpleContent>
    <xsd:restriction base="StringWithLength">
      <xsd:attribute name="length" type="xsd:unsignedShort"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```



- nur echte Teilmenge erlaubt:
Resultierender Datentyp darf nur gültige Werte des ursprünglichen Datentyps enthalten.
- hier OK, da `xsd:unsignedShort` Teilmenge von `xsd:nonNegativeInteger`

Polymorphismus

Polymorphismus

- Voraussetzung: XML-Schema S leitet Datentyp t' von Datentyp t ab:
entweder mit `xsd:extension` oder `xsd:restriction`
- Betrachten wir eine Instanz von S .

Typsubstitution

- An jeder Stelle in der Instanz, wo S den Datentyp t verlangt, kann auch t' verwendet werden.
- Verwendete Datentyp t' muss mit `xsi:type` explizit angegeben werden.

Beispiel



```
<xsd:complexType name="NameType">
  <xsd:sequence>
    <xsd:element name="first" type="xsd:string"/>
    <xsd:element name="middle" type="xsd:string"/>
    <xsd:element name="last" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="ExtendedNameType">
  <xsd:complexContent>
    <xsd:extension base="target:NameType">
      <xsd:attribute name="gender" type="xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Datentyp t

Datentyp t' mit
zusätzlichem
Attribut gender

Beispiel einer Typsubstitution



Schema

```
<xsd:element name="name" type="NameType">
```

Instanz

```
<name title="Mr.">
  <first>...</first>
  <middle>...</middle>
  <last>...</last>
</name>
```

oder

Instanz

```
<name title="Mr." gender="male" xsi:type="ExtendedNameType">
  <first>...</first>
  <middle>...</middle>
  <last>...</last>
</name>
```

t' Einschränkung (restriction) von t

- Laut Schema S müssen Anwendungen sowieso mit *allen* gültigen Werten von t rechnen, also auch mit t' .

⇒ unproblematisch

t' Erweiterung (extension) von t

- Laut Schema S müssen Anwendungen mit *allen* möglichen Werten von t rechnen, nicht aber mit *zusätzlichen* Attributen und Elementen

⇒ evtl. problematisch

⇒ Typsubstitution für Erweiterungen unterdrücken:

```
<xsd:element name="name" type="NameType" block="extension">
```

Relax NG

Beispiel



```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
      <optional>
        <element name="note">
          <text/>
        </element>
      </optional>
    </element>
  </zeroOrMore>
</element>
```

entspricht folg. DTD

```
<!ELEMENT addressBook (card*)>
<!ELEMENT card (name, email, note?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT note (#PCDATA)>
```

Relax NG



- kein W3C-, sondern OASIS-Standard
- integriert Datentypen von XML-Schema:



```
<element name="number">
  <data type="integer"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
</element>
```

- Konverter Relax NG → XML-Schema:
<http://thaiopensource.com/relaxng/trang.html>
- mehr Informationen: <http://www.relaxng.org/>

- + einfacher, lesbarer und kompakter
- + theoretisch fundiert
- + integriert XML-Schema-Datentypen
- + es gibt Konverter: Relax NG \Leftrightarrow XML-Schema
- Konverter funktionieren schlecht
- nicht etabliert

Wie geht es weiter?

heutige Vorlesung

- XML-Schema
 - Datentypen
 - Element- und Attribut-Deklarationen
 - Polymorphismus
- Relax NG als Alternative

Übung heute und nächste Woche

- Beispiel: XML-Schema für Bäume