

Modul 5

XML Schema

XML Schema Definition Language (XSD)

Josef Altmann

<?xml?>



Der vorliegende Foliensatz basiert vorwiegend auf:

Elliotte Rusty Harold, W. Scott Means: XML in a Nutshell: A Desktop Quick Reference, 3rd Edition, O'Reilly, 2005

Priscilla Walmsley: Definitive XML Schema, 2nd Edition, Prentice Hall, 2012

Inhalt

- **Einführung**
 - Elemente und Attribute
 - Vordefinierte Datentypen
 - Benutzerdefinierte Datentypen
 - Schlüssel und Schlüsselreferenzen
 - Modularisierung und Komposition
 - Modellierungsmuster
-
- Anhang I: DTD versus XML Schema
 - Anhang II: Facetten
 - Anhang III: Entwurfsrichtlinien
 - Anhang IV: Annotationen
 - Anhang V: XML Schema 1.1 – Erweiterungen

Einführung 1/5

DTD versus XML Schema

■ Nachteile DTDs

- keine XML-Syntax
- wenige Datentypen
 - Elementinhalte nur Text
 - Wenige Attributtypen
- Keine benutzerdefinierte Datentypen
- Eingeschränkte Wiederverwendung (nur Parameter Entities, keine Vererbung)
- Nur einfache Integritätsbedingungen formulierbar
- ID, IDREF(S): Einschränkungen
- Keine Unterstützung von XML-Namensräume

➤ **zur Beschreibung von Textdokumenten ausreichend**

■ Vorteile XML Schema

- XML als Syntax
- zahlreiche vordefinierte Datentypen
 - für Elemente und
 - Attribute
- Benutzerdefinierte einfache und komplexe Datentypen
- Wiederverwendungskonzepte auf Typ-, Element- und Attributebene (strukturelle Vererbungsmechanismen)
- Komplexe Integritätsbedingungen formulierbar
- Schlüssel, Referenz: flexibles Konzept
- Unterstützung von XML-Namensräume

➤ **zur Beschreibung von Daten besser geeignet**

Einführung 2/5

DTD versus XML Schema

CourseCatalog.dtd

XML Schema

```
...
<!ELEMENT CourseCatalog (DegreeProgramme*)>
<!-- ATTLIST CourseCatalog year CDATA #REQUIRED
term (summer|winter) #REQUIRED
campus (Hagenberg|Linz|Steyr|Wels) #REQUIRED -->
...
```

CourseCatalog.xsd

XML-Dokument

```
<?xml version="1.0"?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  ...>
```

← Namensraum des XML Schema-Standards
← Namensraum des CourseCatalog Schemas
← Zielnamensraum für das Instanzdokument

Elementdeklaration

```
<xs:element name="CourseCatalog" type="cc:CourseCatalogType"/>
```

← Benutzerdefinierter komplexer Datentyp

Datentypdefinition
(komplexer Datentyp)

```
<xs:complexType name="CourseCatalogType">
  <xs:sequence>
    <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType" maxOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="term" type="cc:termType" use="required"/>
  <xs:attribute name="campus" type="cc:campusType" use="required"/>
  <xs:attribute name="year" type="xs:gYear" use="required"/>
</xs:complexType>
```

← Inhaltsmodell
← Kardinalität
← Benutzerdefinierter einfacher Datentyp
← Attributdeklaration

Datentypdefinition
durch Ableitung
(einfacher Datentyp)

```
<xs:simpleType name="termType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="summer"/>
    <xs:enumeration value="winter"/>
  </xs:restriction>
</xs:simpleType>
...
</xs:schema>
```

← Vordefinierte einfache Datentypen
← Einschränkung durch Fassetten

Einführung 3/5

XML Schema

- Unterscheidung von **Dokumentenschema** und konkreten Ausprägungen, den sog. **Instanz-Dokumenten**
- XML Schema ist Datendefinitionssprache zur Festlegung
 - der **Struktur** von Instanz-Dokumenten
 - des **Datentyps** jedes einzelnen Elements/Attributs
- XML Schema 1.0 (2004)
 - Part 0: Primer Second Edition
 - <https://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
 - Part 1: Structures Second Edition
 - <https://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>
 - Part 2: Datatypes Second Edition
 - <https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>
- XML Schema Definition Language (XSD) 1.1 (2012)
 - Part 1: Structures
 - <https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>
 - Part 2: Datatypes
 - <https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/>

Einführung 4/5

Deklaration von Namensräumen im Schema

- Namensraum für eigenes Vokabular
 - Namensraum (Namespace: NS) des zu definierenden Schemas kann über **targetNamespace** festgelegt werden.
- Namensraum des XML-Schema-Standard-Vokabulars
 - NS des XML Schema-Standards (definiert **<xs:element>**, **<xs:attribute>**,...) muss angegeben werden.
 - Weitere NS können eingebunden werden.
- Ein NS kann als Default-NS definiert werden
 - zu definierender NS, XML Schema-NS oder anderer NS
 - für alle anderen muss ein Präfix verwendet werden

CourseCatalog.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  ...
</xs:schema>
```

Einführung 5/5

Verwendung von Namensräumen im XML-Dokument

- Schema eines XML-Dokuments wird im Wurzelement durch Attribut `schemaLocation` bestimmt

1 Komponente: `targetNamespace` des Schemas

2 Komponente: Angabe der Lokation des Schema-Dokuments

```
<?xml version="1.0"?>
```

```
<xs:schema      xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
```

```
...
```

```
</xs:schema>
```

```
<?xml version="1.0"?>
```

```
<CourseCatalog xsi:schemaLocation="http://www.fh-ooe.at/CourseCatalog CourseCatalog.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.fh-ooe.at/CourseCatalog"
  year="2018" term="winter" campus="Hagenberg">
```

```
...
```

```
</CourseCatalog>
```

`xsi:noNamespaceSchemaLocation="CourseCatalog.xsd"`

CourseCatalog.xsd

CourseCatalog.xml

Inhalt

- Einführung
 - **Elemente und Attribute**
 - Vordefinierte Datentypen
 - Benutzerdefinierte Datentypen
 - Schlüssel und Schlüsselreferenzen
 - Modularisierung und Komposition
 - Modellierungsmuster
-
- Anhang I: DTD versus XML Schema
 - Anhang II: Facetten
 - Anhang III: Entwurfsrichtlinien
 - Anhang IV: Annotationen
 - Anhang V: XML Schema 1.1 – Erweiterungen

Aufbau XML Schema-Dokument

- Wurzelement `<xs:schema ...>`
 - mit Angabe eines `targetNamespace`, also dem Namensraum, in dem die Definitionen und Deklarationen gelten sollen
- Darin enthalten sind
 - Elementdeklarationen
 - Attributdeklarationen
 - Datentypdefinitionen (einfache und komplexe Datentypen)

CourseCatalog.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog" ...>

  <xs:element name="CourseCatalog" ...> ... </xs:element>           <!-- Elementdeklaration -->
  ...
  <xs:attribute name="campus" ...> ... </xs:attribute>               <!-- Attributdeklaration -->
  ...
  <xs:simpleType name="termType"> ... </xs:simpleType>               <!-- Datentypdefinition -->
  ...
  <xs:complexType name="CourseCatalogType"> ... </xs:complexType> <!-- Datentypdefinition -->
  ...
</xs:schema>
```

Element Deklaration

Einfacher oder komplexer Datentyp

Standardwert oder fixer Wert

```
<xs:element name="name" type="type" minOccurs="int" maxOccurs="int/unbounded" default/fixed="value" ...> ... </xs:element>
```

Kardinalität: Unter- und Obergrenze

- Die Anzahl der Vorkommen eines Elements innerhalb eines XML-Instanzdokuments kann durch die Attribute
 - `minOccurs="int"` (Untergrenze, Standardwert "1")
 - `maxOccurs="int|unbounded"` (Obergrenze, Standardwert "1")im `<xs:element>`-Tag festgelegt werden.
- Mit `default` kann ein Vorgabewert für das Element angegeben werden.
- Mit `fixed` kann ein fester, nicht veränderbarer Wert festgelegt werden.

Attribut Deklaration

Einfacher Datentyp

```
<xs:attribute name="name" type="simpleType" use="how-used" default/fixed="value" ...> ... </xs:attribute>
```

Häufigkeitsbeschränkung

required Attribut obligatorisch
optional Attribut optional (Standardwert)
prohibited Attribut unzulässig

Vorgabewert

default **use** muss **optional** sein
fixed nur relevant, wenn **use**
nicht gesetzt

default und **fixed** schließen sich gegenseitig aus, d.h. in einer Attribut-deklaration dürfen beide Attribute nicht gleichzeitig vorkommen.

- Attribute können nur einfache Datentypen aufnehmen

Elemente und Attribute 1/3

Lokale und globale Datentypen

- Elemente und Attribute können lokale oder globale Datentypen verwenden
 - Element/Attribut mit **lokalem** (anonymen) Datentyp

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

```
<xs:attribute name="name" use="how-used" default/fixed="value" ...>  
  <xs:simpleType> ... </xs:simpleType>  
</xs:attribute>
```

- Element/Attribut mit **globalem** (benannten) Datentyp
(global = direktes Kindelement von <xs:schema>)

```
<xs:element name="name" type="typeName" minOccurs="int" ... </xs:element>
```

```
<xs:attribute name="name" type="typeName" use="how-used" ... </xs:attribute>
```

Elemente und Attribute 2/3

Globale Elemente und Attribute

- Globale Element- und Attributdeklarationen treten als direktes Kindelement von `<xs:schema>` auf
- Globale Deklarationen sind überall im Schema sichtbar und können an beliebigen Stellen mit dem Attribut `ref` referenziert werden (Wiederverwendung).
 - Verweis auf bereits bestehendes Element bzw. Attribut

```
<xs:element ref="name" minOccurs="int" maxOccurs="int|unbounded" ... />
```

```
<xs:attribute ref="name" use="how-used" default/fixed="value" ... />
```

- Einschränkungen bei Deklaration von globalen Elemente und Attribute
 - Verwendung von `ref`-Attribut nicht erlaubt
 - Kardinalität darf nicht eingeschränkt werden

Elemente und Attribute 3/3

Global/Lokal - Beispiel

Elementdeklaration
Datentypdefinition

CourseCatalog.xsd

- Globales Element mit globalen Datentyp
- Globaler Datentyp
- Lokales Element mit globalen Datentyp
- Lokales Attribut mit globalen Datentyp
- Lokales Attribut mit vordefinierten Datentyp
- Globaler Datentyp
- Referenz auf globales Element
- Globales Element mit vordefinierten Datentyp

```
<xs:schema ... >
  <xs:element name="CourseCatalog" type="cc:CourseCatalogType"></xs:element>
  <xs:complexType name="CourseCatalogType">
    <xs:sequence>
      <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType"
        maxOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="term" type="cc:termType" use="required"/>
    <xs:attribute name="campus" type="cc:campusType" use="required"/>
    <xs:attribute name="year" type="xs:gYear" use="required"/>
  </xs:complexType>
  ...
  <xs:complexType name="CourseType">
    <xs:sequence>
      <xs:element ref="cc:Title"/>
      <xs:element name="Description" type="cc:DescriptionType"/>
    </xs:sequence>
  </xs:complexType>
  ...
  <xs:simpleType name="termType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="summer"/>
      <xs:enumeration value="winter"/>
    </xs:restriction>
  </xs:simpleType>
  ...
  <xs:element name="Title" type="xs:string"/>
</xs:schema>
```

Verweis auf globale
Elementdeklaration

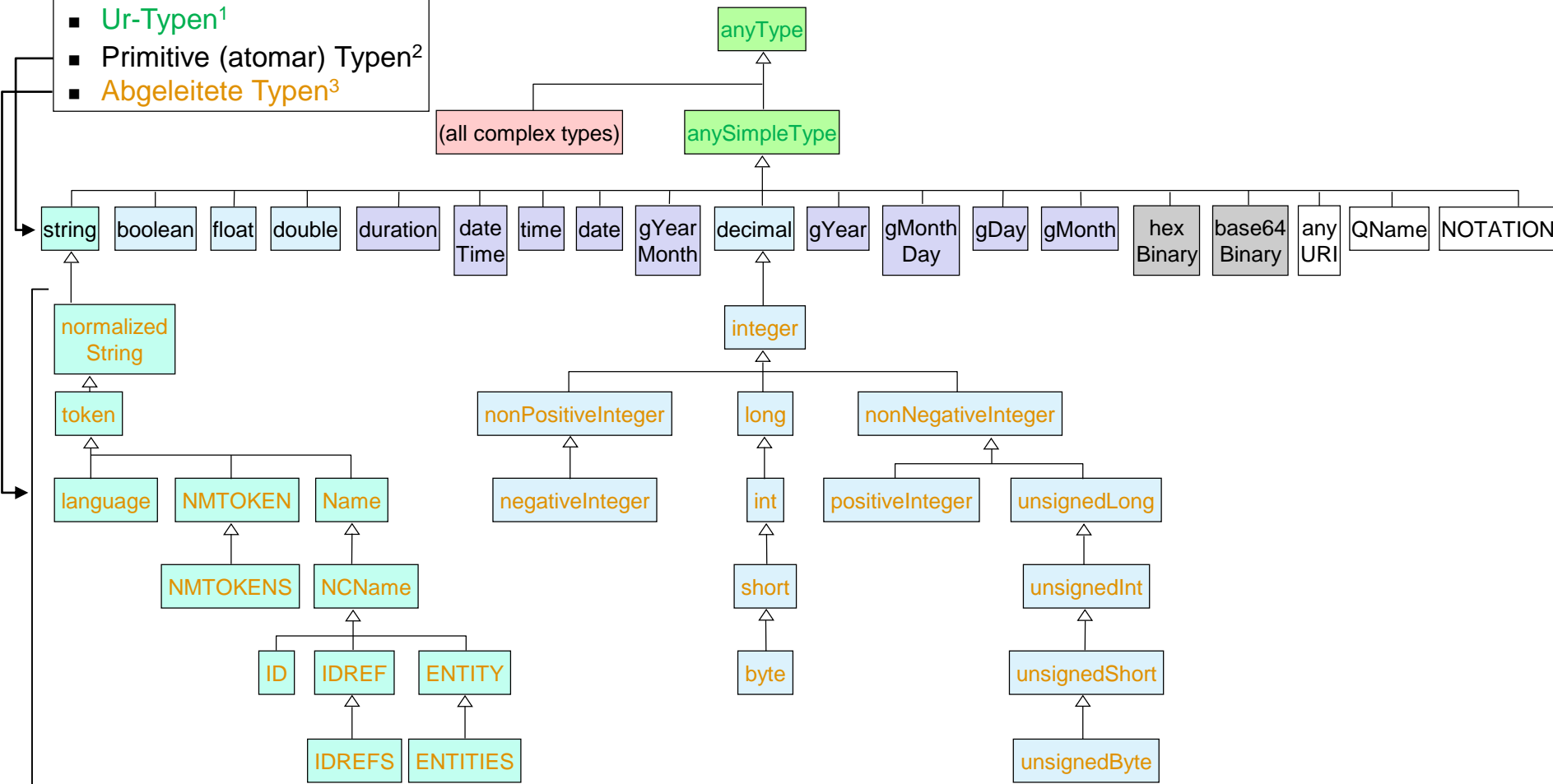
Inhalt

- Einführung
 - Elemente und Attribute
 - **Vordefinierte Datentypen**
 - Benutzerdefinierte Datentypen
 - Schlüssel und Schlüsselreferenzen
 - Modularisierung und Komposition
 - Modellierungsmuster
-
- Anhang I: DTD – XML Schema – Vergleich
 - Anhang II: Facetten – Wertebereichseinschränkung
 - Anhang III: XML Schema Entwurfsrichtlinien
 - Anhang IV: XML Schema Dokumentation
 - Anhang V: XML Schema 1.1 – Erweiterungen

Vordefinierte Datentypen 1/4

Typhierarchie von W3C XML Schema Datentypen

- Ur-Typen¹
- Primitive (atomar) Typen²
- Abgeleitete Typen³



¹ Vgl. <https://www.w3.org/TR/xmlschema-2/#built-in-datatypes>

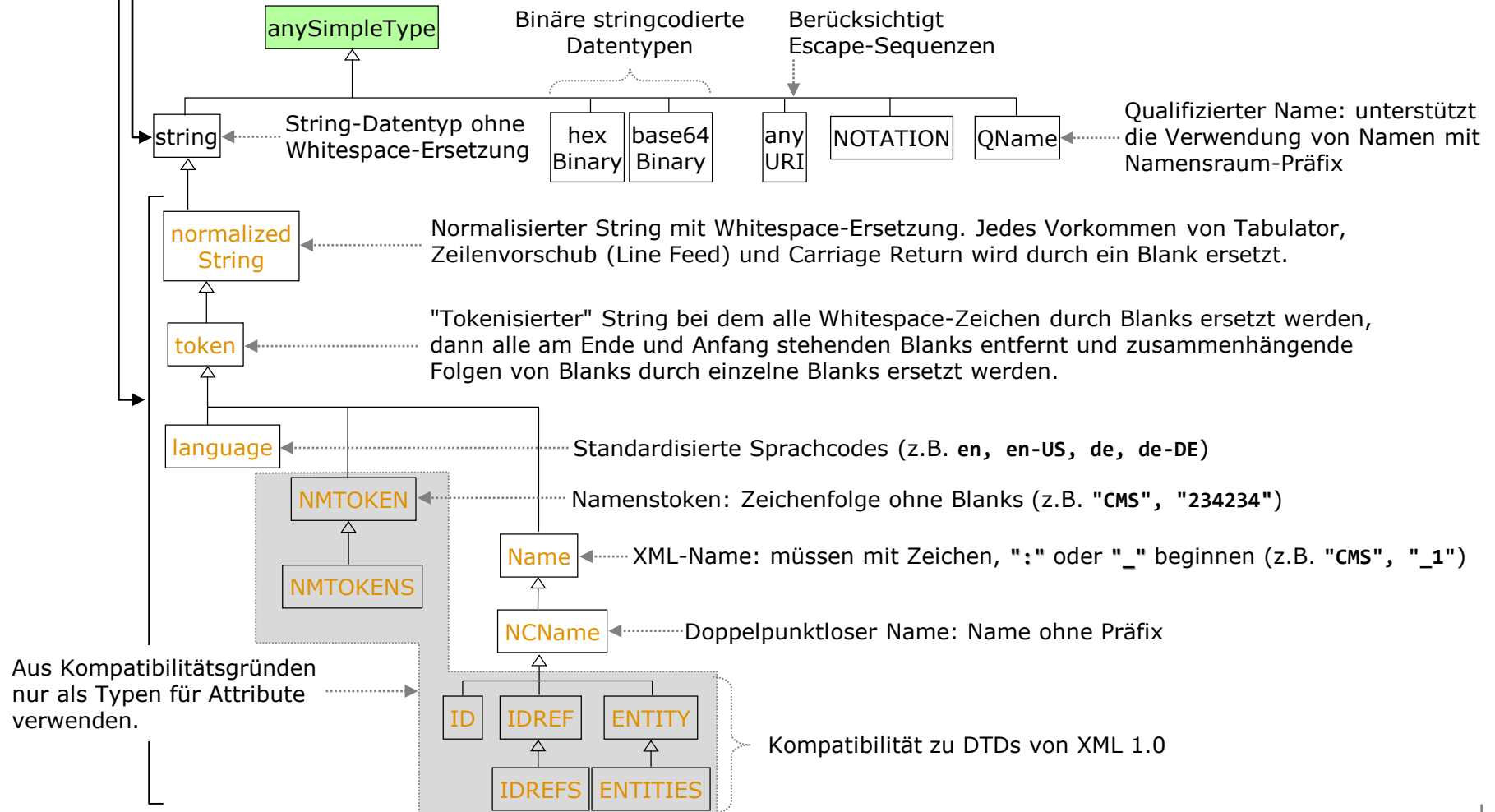
² Vgl. <https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes>

³ Vgl. <https://www.w3.org/TR/xmlschema-2/#built-in-derived>

Vordefinierte Datentypen 2/4

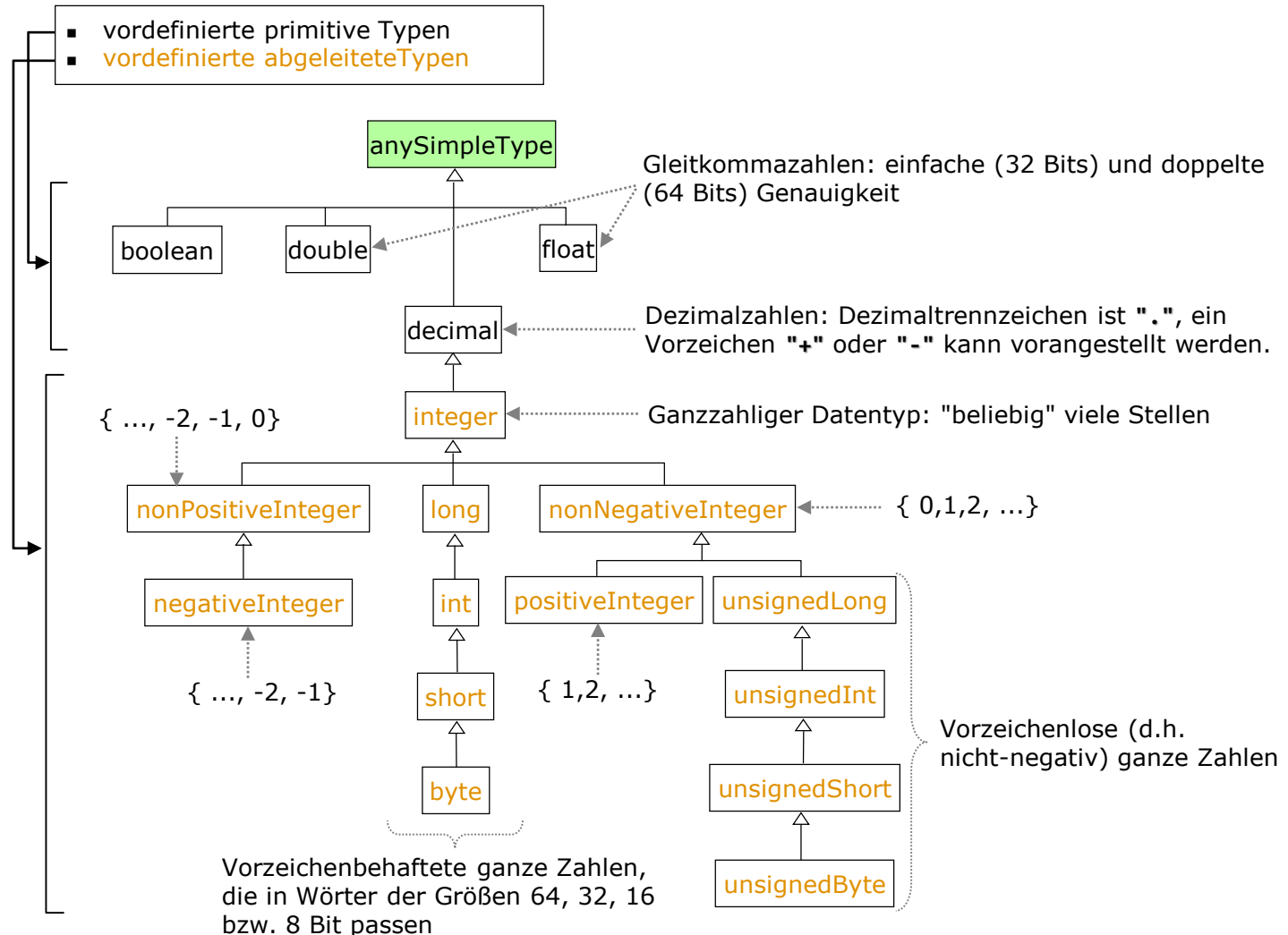
Zeichenketten-Datentypen

- vordefinierte primitive Typen
- vordefinierte abgeleitete Typen



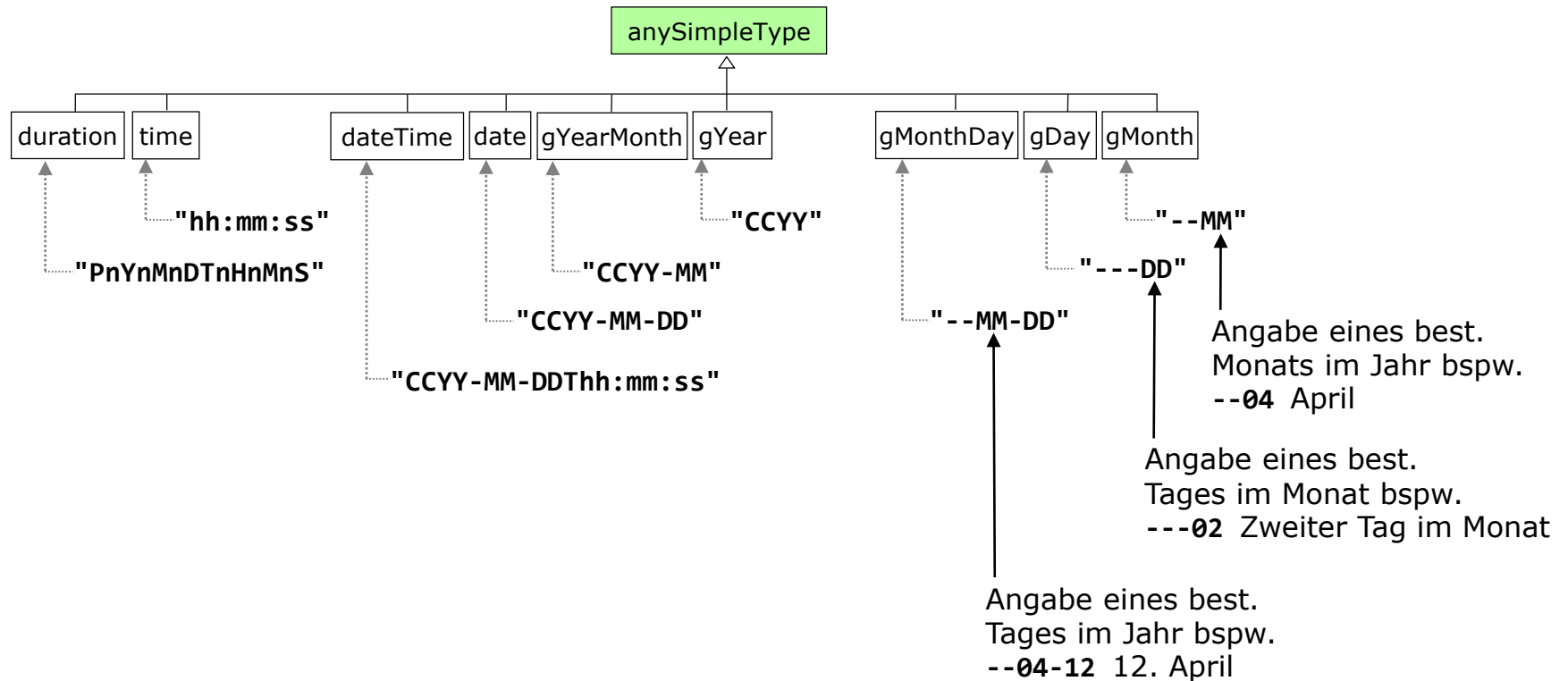
Vordefinierte Datentypen 3/4

Numerische Datentypen



Vordefinierte Datentypen 4/4

Datums- und Zeit-Datentypen

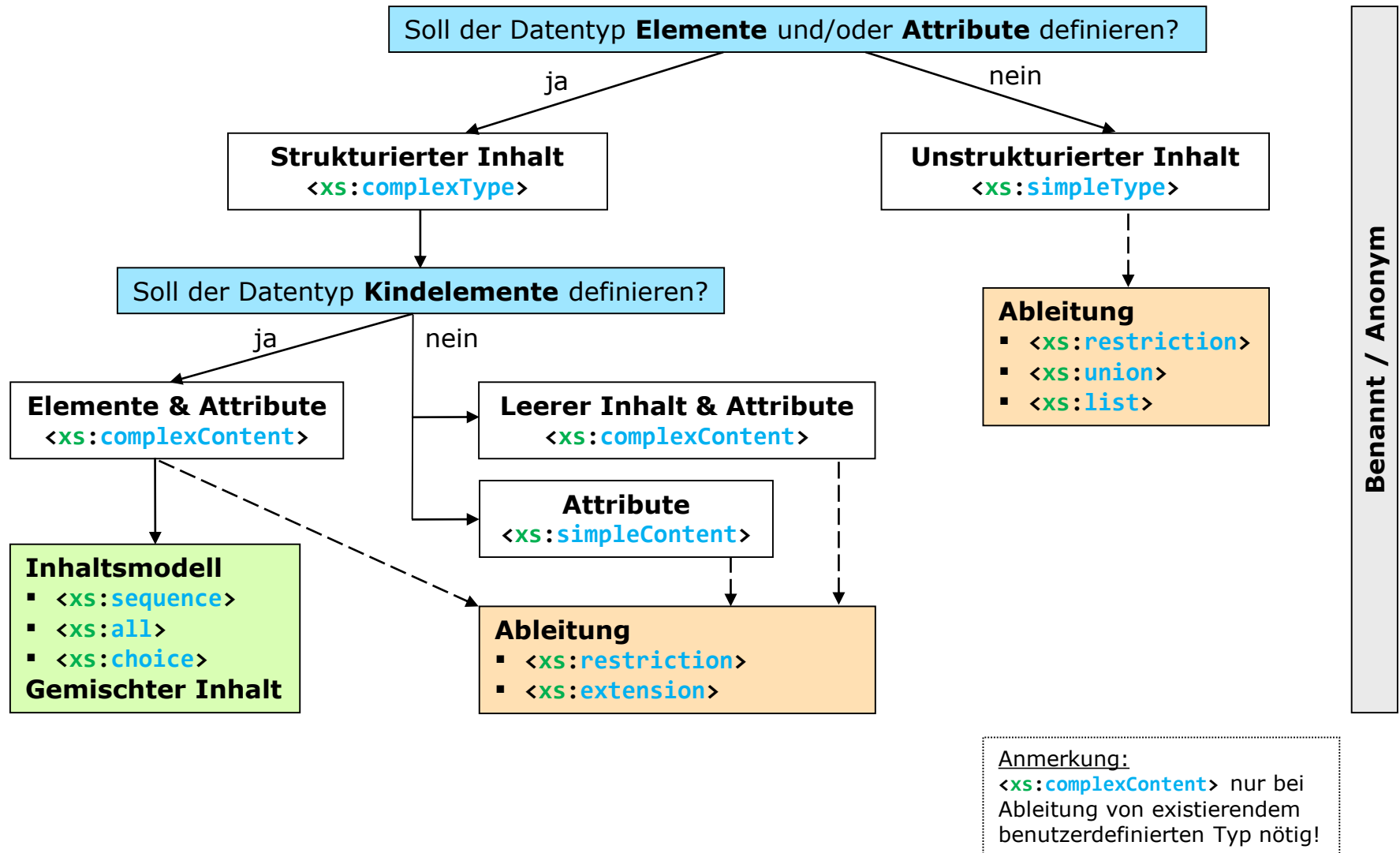


Inhalt

- Einführung
 - Elemente und Attribute
 - Vordefinierte Datentypen
 - **Benutzerdefinierte Datentypen**
 - Schlüssel und Schlüsselreferenzen
 - Modularisierung und Komposition
 - Modellierungsmuster
-
- Anhang I: DTD – XML Schema – Vergleich
 - Anhang II: Facetten – Wertebereichseinschränkung
 - Anhang III: XML Schema Entwurfsrichtlinien
 - Anhang IV: XML Schema Dokumentation
 - Anhang V: XML Schema 1.1 – Erweiterungen

Benutzerdefinierte Datentypen

Alternativen



Benutzerdefinierte Datentypen

Alternativen - Beispiele

Benutzerdefiniert
 Vordefiniert

Nicht abgeleitet

Abgeleitet

Einfach

```
<xs:decimal>
```

```
<xs:simpleType name="creditType">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="1"/>
    <xs:minInclusive value="0.5"/>
    <xs:maxInclusive value="30"/>
  </xs:restriction>
</xs:simpleType>
```

Komplex

```
<xs:complexType
  name="CourseType">
  <xs:sequence>
    <xs:element .../>
    ....
  </xs:sequence>
  <xs:attribute .../>
</xs:complexType>
```

```
<xs:complexType name="InternationalCourseType">
  <xs:complexContent>
    <xs:extension base="cc:CourseType" >
      <xs:sequence>
        <xs:element name="Prerequisites" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="languageCertificate"
        type="cc:certificateType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Benutzerdefinierte Datentypen

Abgeleitete Einfache Datentypen – `<xs:simpleType>`

- **Einschränkung** eines vordefinierten Datentyps
 - `<xs:restriction>`
 - Wertebereich wird eingeschränkt
- **Vereinigung** von vordefinierten Datentypen (Erweiterung)
 - `<xs:union>`
 - Werte des neuen Datentyps müssen zumindest einem der vereinigten Datentypen entsprechen
- **Liste** von Werten eines vordefinierten Datentyps (oder wiederum eines List-Datentyps)
 - `<xs:list>`
 - Werte sind durch *Whitespace* getrennt

Benutzerdefinierte Datentypen

Abgeleitete **Einfache Datentypen** `<xs:simpleType>` – `<xs:restriction>`

- Datentypdefinition mit
 - Attribut **base** existierenden einfachen Datentyp referenzieren
 - `<xs:restriction>` als Sub-Element des `<xs:simpleType>`-Elements neu definieren
- 12 mögliche Einschränkungen (Facetten), abhängig vom Basisdatentyp:

Facetten:

- length
- minLength
- maxLength
- pattern
- enumeration
- minInclusive
- maxInclusive
- minExclusive
- maxExclusive
- totalDigits
- fractionDigits
- whitespace
- assertion



► Anhang II

CourseCatalog.xsd

```

<xs:simpleType name="courseType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Lecture"/>
    <xs:enumeration value="Seminar"/>
    <xs:enumeration value="LabSession"/>
    <xs:enumeration value="PracticeSession"/>
    <xs:enumeration value="Training"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="CourseType" type="cc:courseType"/>
  
```

Diagramm zur XSD-Definition:

- Ein Pfeil weist von der Beschriftung **Basisdatentyp** auf das Attribut `base="xs:string"` im `<xs:restriction>`-Element.
- Eine Klammer fasst die fünf `<xs:enumeration>`-Elemente als **Werteliste** zusammen.
- Ein weiterer Klammer-Symbol links neben der Werteliste ist mit **Facetten** beschriftet.

CourseCatalog.xml

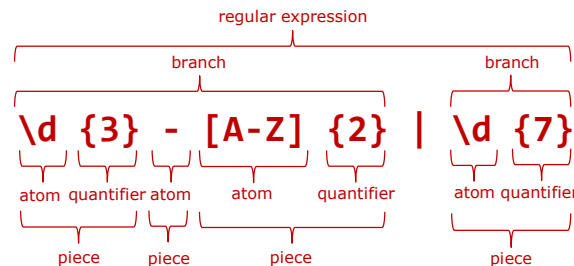
```
<CourseType>Lecture</CourseType>
```


Benutzerdefinierte Datentypen

Abgeleitete **Einfache Datentypen** `<xs:simpleType>` – `<xs:restriction>`

■ Einschränkung: `<xs:pattern>`

- regulärer Ausdruck schränkt Wertebereich ein



Struktur eines regulären Ausdrucks

CourseCatalog.xsd

```
<xs:simpleType name="instructorNumberType">
  <xs:restriction base="xs:string">
    <xs:pattern value="p[1-4]{1}[\d]{4}" />
  </xs:restriction>
</xs:simpleType>
```

Regulärer Ausdruck

```
<xs:element name="Instructor" type="cc:instructorNumberType" />
```

CourseCatalog.xml

```
<Instructor>p20621</Instructor>
```

Benutzerdefinierte Datentypen

Abgeleitete **Einfache Datentypen** `<xs:simpleType>` – `<xs:restriction>`

- Einschränkung: `<xs:fractionDigits>`
 - Anzahl der Nachkommastellen begrenzt
- Einschränkung: `<xs:minInclusive>`
 - eingeschlossene Untergrenze begrenzt Wertebereich
- Einschränkung: `<xs:maxInclusive>`
 - eingeschlossene Obergrenze begrenzt Wertebereich

CourseCatalog.xsd

```
<xs:simpleType name="creditType">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="1"/> ← Nachkommastellen
    <xs:minInclusive value="0.5"/> ← Untergrenze
    <xs:maxInclusive value="30"/> ← Obergrenze
  </xs:restriction>
</xs:simpleType>

<xs:element name="Credit" type="cc:creditType"/>
```

CourseCatalog.xml

```
<Credit>2.5</Credit>
```

Benutzerdefinierte Datentypen

Abgeleitete Einfache Datentypen – `<xs:union>`

- Ableitung durch Vereinigung von einfachen Datentypen
 - über `memberTypes`-Attribut existierende Datentypen referenzieren

CourseCatalog.xsd

<pre> <xs:simpleType name="letterGradeType"> <xs:restriction base="xs:token"> <xs:enumeration value="A"/> <xs:enumeration value="B"/> <xs:enumeration value="C"/> <xs:enumeration value="D"/> <xs:enumeration value="F"/> </xs:restriction> </xs:simpleType> </pre>	<pre> <xs:simpleType name="numericalGradeType"> <xs:restriction base="xs:integer"> <xs:minInclusive value="1"/> <xs:maxInclusive value="5"/> </xs:restriction> </xs:simpleType> </pre>
<pre> <xs:simpleType name="gradeType"> <xs:union memberTypes="cc:letterGradeType cc:numericalGradeType"/> </xs:simpleType> </pre> <p style="text-align: center;">Datentypvereinigung</p> <pre> <xs:element name="Grade" type="cc:gradeType"/> </pre>	

CourseCatalog.xml

```

<Grade>2</Grade>
<Grade>B</Grade>

```

Benutzerdefinierte Datentypen

Abgeleitete Einfache Datentypen – `<xs:list>`

- Ableitung durch Auflistung von atomaren Werten eines einfachen Datentyps
 - über `itemType`-Attribut existierenden Datentyp referenzieren

CourseCatalog.xsd

```
<xs:simpleType name="gradeType">
  <xs:union memberTypes="cc:letterGradeType cc:numericalGradeType"/>
</xs:simpleType>

<xs:simpleType name="combinedGradeType">
  <xs:list itemType="cc:gradeType"/>
</xs:simpleType>
  <simpleType> oder <union>

<xs:element name="CombinedGrades" type="cc:combinedGradeType"/>
```

CourseCatalog.xml

```
<CombinedGrade>2 B 1 F</CombinedGrade>
```

Benutzerdefinierte Datentypen

Komplexe Datentypen – `<xs:complexType>`

- Geschachtelte Elemente
 - nur innerhalb eines komplexes Datentyps möglich
- Attribute
 - nur innerhalb eines komplexes Datentyps möglich
 - unabhängig davon, ob geschachtelte Elemente vorhanden oder nicht
- Leerer Inhalt – empty content
 - weist keine geschachtelten Elemente auf
 - nur innerhalb eines komplexes Datentyps möglich
- Gemischter Inhalt – mixed content
 - Datentyp kann geschachtelte Elemente und Text enthalten
 - im Gegensatz zu DTDs sind für die geschachtelten Elemente folgende Eigenschaften spezifizierbar:
 - Reihenfolge
 - Kardinalität

Benutzerdefinierte Datentypen

<xs:complexType> – Geschachtelte Elemente

■ Sequenz – <xs:sequence>

```
<xs:complexType name="CourseType">
  <xs:sequence>
    <xs:element name="Title" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="Description" type="cc:DescriptionType" minOccurs="1" maxOccurs="1"/>
    ...
  </xs:sequence>
</xs:complexType>

<xs:element name="Course" type="cc:CourseType"/>
```

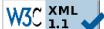
■ Auswahl – <xs:choice>

- von den angeführten Elementen darf nur eines auftreten

■ Menge – <xs:all>

- Reihenfolge der Elemente beliebig
- jedes Element erscheint maximal einmal

■ Kardinalität wird durch **minOccurs** u. **maxOccurs** ausgedrückt

- Einschränkung bei <xs:all>: **minOccurs** kann nur die Werte 0 od. 1 annehmen, **maxOccurs** muss den Wert 1 aufweisen
-  **minOccurs** und **maxOccurs** dürfen > 1 sein

Benutzerdefinierte Datentypen

<xs:complexType> – Geschachtelte Elemente und Attribute

- Attribute werden am Ende der Typ-Definition angeführt

```
<xs:complexType name="CourseType">
  <xs:sequence>
    <xs:element name="Title" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="Description" type="cc:DescriptionType" minOccurs="1" maxOccurs="1"/>
    ...
  </xs:sequence>
  <xs:attribute name="id" type="cc:idCourseType" use="required"/>
  <xs:attribute name="semesterHours" type="xs:decimal" use="required"/>
  <xs:attribute name="language" type="xs:language" use="optional"/>
  <xs:attribute name="basedOn" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="cID_[\d]{4}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
```

Benutzerdefinierte Datentypen

<xs:complexType> – Atomarer Elementinhalt mit Attribut

CourseCatalog.xsd

```
<xs:complexType name="RoomType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="roomNumber" type="xs:string" use="required"/>
      <xs:attribute name="building" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="Room" type="cc:RoomType"/>
```

CourseCatalog.xml

```
<Room roomNumber="2.020" building="FH2">karriere.at Audimax</Room>
```

Atomarer Elementinhalt

Benutzerdefinierte Datentypen

`<xs:complexType>` – Leerer Elementinhalt mit Attribut

CourseCatalog.xsd

```
<xs:complexType name="DateType">
  <xs:attribute name="startDate" type="xs:gMonthDay" use="required"/>
  <xs:attribute name="endDate" type="xs:gMonthDay" use="required"/>
</xs:complexType>

<xs:element name="Date" type="cc:DateType"/>
```

CourseCatalog.xml

```
<Date startDate="--10-03" endDate="--01-24"></Date>
```

Benutzerdefinierte Datentypen

<xs:complexType> – Gemischter Elementinhalt

CourseCatalog.xsd

```
<xs:complexType name="DescriptionType" mixed="true">
  <xs:sequence>
    <xs:element name="Content" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Exam" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="Tool" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="Description" type="cc:DescriptionType"/>
```

CourseCatalog.xml

```
<Description>Introduction of skills related to XML.<Content>Includes DTD, Schema, XPath,
XQuery, XSLT, JSON</Content><Exam>Final Exam required.</Exam>Participation without any
previous knowledge.</Description>
```

- Reihenfolge und Anzahl des Auftretens von Kindelementen wird kontrolliert!

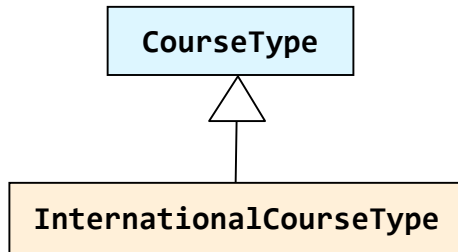
Benutzerdefinierte Datentypen

`<xs:complexType>` – Ableitung von komplexen Typen

- Erweiterung
 - `<xs:extension>`
 - zusätzliche geschachtelte Elemente und/oder Attribute
 - Einschränkung
 - `<xs:restriction>`
 - Wertebereich
 - Kardinalität
-
- Abstrakte Datentypen
 - `<xs:complexType>` mit Attribut `abstract="true"`
 - Verbot der Ableitung
 - `<xs:complexType>` mit Attribut `final`
 - mit Ausprägungen: `#all`, `restriction`, `extension`

Benutzerdefinierte Datentypen

<xs:complexType> – Ableitung durch Erweiterung



```

<xs:complexType name="CourseType">
  <xs:sequence>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Description" type="cc:DescriptionType"/>
  </xs:sequence>
  <xs:attribute name="id" type="cc:idCourseType" use="required"/>
  <xs:attribute name="semesterHours" type="xs:decimal" use="required"/>
</xs:complexType>

```

```

<xs:complexType name="InternationalCourseType">
  <xs:complexContent>
    <xs:extension base="cc:CourseType">
      <xs:sequence>
        <xs:element name="Prerequisites" type="xs:string" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="languageCertificate" type="cc:certificateType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

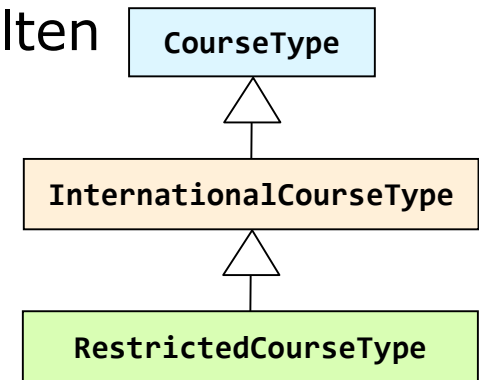
```

- Elemente werden am Ende angefügt
- Erweiterung muss innerhalb eines **<xs:complexContent>**-Elements vorgenommen werden

Benutzerdefinierte Datentypen

<xs:complexType> – Ableitung durch Einschränkung

- Die Deklarationen des Basistyps, die beibehalten werden sollen, müssen wiederholt werden!
- Einschränkung muss innerhalb eines **<xs:complexContent>**-Elements vorgenommen werden



```

<xs:complexType name="RestrictedCourseType">
  <xs:complexContent>
    <xs:restriction base="cc:InternationalCourseType">
      <xs:sequence>
        <xs:element name="Title" type="xs:string"/>
        <xs:element name="Description" type="cc:DescriptionType"/>
      </xs:sequence>
      <xs:attribute name="id" type="cc:idCourseType" use="required"/>
      <xs:attribute name="semesterHours" type="xs:decimal" use="required"/>
      <xs:attribute name="languageCertificate" type="cc:certificateType" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
  
```

Benutzerdefinierte Datentypen

`<xs:complexType>` – Typsubstitution im Instanzdokument

- Statisch
- Dynamisch
 - Festlegung des abgeleiteten Datentyps im XML-Dokument über Attribut **type** aus dem XML Schema Instance (**xsi**) Namensraum

CourseCatalog.xml

```

...
<Course id="cID_7555" semesterHours="1" language="en" semester="6">
  <Title>Intercultural Communications</Title>
  <Description>Different types of fake news.</Description>
</Course>

<Course xsi:type="InternationalCourseType" id="cID_8840" semesterHours="2"
  language="en" semester="2" languageCertificate="TOEFL">
  <Title>Data modelling and database design</Title>
  <Description>Introduction to database design</Description>
  <Prerequisites>At least 15 ECTS in CS required</Prerequisites>
</Course>
...

```

Element **Course** mit Datentyp
CourseType

Hinweis für Schema-Prozessor

Element **Course** mit Datentyp
InternationalCourseType
Erweiterung um **Prerequisites**
und **languageCertificate**

CourseCatalog.xsd

```
<xs:element name="Course" type="cc:CourseType"/>
```

Inhalt

- Einführung
 - Elemente und Attribute
 - Vordefinierte Datentypen
 - Benutzerdefinierte Datentypen
 - **Schlüssel und Schlüsselreferenzen**
 - Modularisierung und Komposition
 - Modellierungsmuster
-
- Anhang I: DTD versus XML Schema
 - Anhang II: Facetten
 - Anhang III: Entwurfsrichtlinien
 - Anhang IV: Annotationen
 - Anhang V: XML Schema 1.1 – Erweiterungen

Schlüssel und Schlüsselreferenzen 1/2

- Eigenschaften eines **Schlüssels** `<xs:key>`
 - Wert(kombination) muss eindeutig sein
 - Wert muss vorhanden sein
- Als **Schlüsselkomponenten** können definiert werden `<xs:field>`
 - Elemente (nur einfache Datentypen)
 - Attribute
 - Kombinationen von Elementen u. Attributen
- **Gültigkeitsbereich** kann definiert werden `<xs:selector>`
- **Referenz** auf Schlüssel `<xs:keyref>`
- Weiters können Elemente, Attribute bzw. Kombinationen davon als eindeutig spezifiziert werden `<xs:unique>`
 - Wert(kombination) muss eindeutig sein
 - Wert muss nicht vorhanden sein

Schlüssel und Schlüsselreferenzen 2/2

Beispiel: `<xs:key>`

CourseCatalog.xsd

```
<xs:element name="CourseCatalog" type="cc:CourseCatalogType">
  <xs:key name="courseIdKey">
    <xs:selector xpath="cc:DegreeProgramme/cc:Course"/>
    <xs:field xpath="@id"/>
  </xs:key>
  <xs:keyref name="refCourseIdKey" refer="cc:courseIdKey">
    <xs:selector xpath="cc:DegreeProgramme/cc:Course"/>
    <xs:field xpath="@basedOn"/>
  </xs:keyref>
</xs:element>
```

Schlüssel innerhalb von
`<CourseCatalog>` eindeutig!

Definition von Schlüssel und
Referenz müssen immer
gemeinsam und lokal zu
einem Element erfolgen!

CourseCatalog.xml

```
<CourseCatalog ... >
  <DegreeProgramme code="0307" name="Software Engineering" ...>
    <Course id="cID_7540" ...>
      ...
    </Course>
    <Course id="cID_7541" basedOn="cID_7540" ...>
      ...
    </Course>
  </DegreeProgramme>
</CourseCatalog>
```

Inhalt

- Einführung
 - Elemente und Attribute
 - Vordefinierte Datentypen
 - Benutzerdefinierte Datentypen
 - Schlüssel und Schlüsselreferenzen
 - **Modularisierung und Komposition**
 - Modellierungsmuster
-
- Anhang I: DTD versus XML Schema
 - Anhang II: Facetten
 - Anhang III: Entwurfsrichtlinien
 - Anhang IV: Annotationen
 - Anhang V: XML Schema 1.1 – Erweiterungen

Modularisierung und Komposition 1/8

Innerhalb eines Schemas

- Wurzelement `<xs:schema>` eines XML Schemas enthält alle Schema-Komponenten (Datentypen, Elemente, Attribute,...) als Kindelemente
- Globale versus lokale Deklaration von Datentypen, Elementen und Attributen
- Beziehungen: lokal geschachtelte Elemente versus über Referenzen (`<xs:keyref>` oder `<xs:ref>`) realisierte Beziehungen
- Gruppierung von Elementen und Attributen
 - Zweck: Modularisierung, Wiederverwendung



Modularisierung und Komposition 2/8

Innerhalb eines Schemas

■ Elementgruppe

- Zusammenfassung von Elementen zu einer Elementgruppe
- Referenzierung über Gruppennamen
- Einschränkung: keine rekursiven Bezüge erlaubt!

```
<xs:schema ...>
  <xs:group name="CourseDescriptionGroup">
    <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Description" type="cc:DescriptionType"/>
      <xs:element name="Credit" type="cc:CreditType"/>
      <xs:element name="CourseType" type="cc:CourseType"/>
    </xs:sequence>
  </xs:group>
  <xs:complexType name="CourseType">
    <xs:sequence>
      <xs:group ref="cc:CourseDescriptionGroup" minOccurs="1" maxOccurs="1"/>
      <xs:element name="Date" type="cc:DateType"/>
      ...
    </xs:sequence>
  </xs:complexType>
  ...
</xs:schema>
```

Modularisierung und Komposition 3/8

Innerhalb eines Schemas

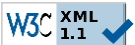
■ Attributgruppe

- Zusammenfassung von Attributen zu einer Attributgruppe
- Referenzierung über Gruppennamen
- Bessere Wiederverwendbarkeit

```
<xs:schema ...>
  <xs:attributeGroup name="IdentifierDegreeProgrammeGroup">
    <xs:attribute name="code" type="cc:codeType" use="required"/>
    <xs:attribute name="type" type="xs:string" use="required"/>
    <xs:attribute name="abbreviation" type="xs:string" use="optional"/>
  </xs:attributeGroup>
  ...
  <xs:complexType name="DegreeProgrammeType">
    <xs:sequence> ... </xs:sequence>
    <xs:attributeGroup ref="cc:IdentifierDegreeProgrammeGroup"/>
  </xs:complexType>
  ...
</xs:schema>
```

Modularisierung und Komposition 4/8

Aufbau von Schema-Bibliotheken

- Einbindung anderer Schemata durch
 - `<xs:include>`
 - `<xs:redefine>`
 - `<xs:import>`
 -  `<xs:override>` ► Anhang V
- `<xs:include>`, `<xs:redefine>` und `<xs:import>` Elemente müssen als Subelemente von `<xs:schema>` vor anderen Deklarationen angeführt werden

Modularisierung und Komposition 5/8

Schema-Inklusion

■ Inkludieren eines Schemas - `<xs:include>`

- Inkludiertes Schema muss den gleichen Namensraum wie das inkludierende Schema oder keinen Namensraum haben
- Komponenten des inkludierten Schemas können so verwendet werden, als wären sie direkt im inkludierenden Schema deklariert worden

Catalog.xsd

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog" ...>

  <xs:include schemaLocation="Course.xsd"/>

  <xs:element name="CourseCatalog" .../>
  ...
  <xs:complexType name="DegreeProgrammType">
    <xs:sequence>
      <xs:element name="Course" type="cc:CourseType"/>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Course.xsd

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog" ...>

  ...
  <xs:complexType name="CourseType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      ...
    </xs:sequence>
  </xs:complexType>
  ...
</xs:schema ...>
```

Modularisierung und Komposition 6/8

Schema-Inklusion mit Ableitung

■ Inkludieren eines Schemas - `<xs:include>`

- Ableitung durch Erweiterung
- Ableitung durch Einschränkung

Catalog.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
            targetNamespace="http://www.fh-ooe.at/CourseCatalog" ...>
```

```
<xs:include schemaLocation="Course.xsd"/>
```

```
<xs:complexType name="InternationalCourseType">
  <xs:complexContent>
    <xs:extension base="cc:CourseType">
      <xs:sequence>
        <xs:element name="Prerequisites" type="xs:string" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="languageCertificate" type="cc:certificateType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
</xs:schema>
```


Modularisierung und Komposition 7/8

Schema-Inklusion mit Redefinition

■ Inkludieren u. Redefinieren eines Schemas - `<xs:redefine>`

- Gleiche Funktionalität wie `<xs:include>`
- Zusätzlich können inkludierte Komponenten
- `<xs:simpleType>` (Einschränkung)
- `<xs:complexType>` (Einschränkung und Erweiterung)
- `<xs:group>` `<xs:attributeGroup>` (Einschränkung und Erweiterung)

neu definiert werden

Catalog.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog" ...>
  <xs:redefine schemaLocation="Course.xsd"/>
    <xs:simpleType name="dayType">
      <xs:restriction base="cc:dayType">    <!-- restrict MON to SAT -->
        <xs:enumeration value="MON"/>
        ...
        <xs:enumeration value="SAT"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:redefine>
  <xs:element name="CourseCatalog" .../>
  ...
</xs:schema>
```

Modularisierung und Komposition 8/8

Schema-Import

- Importieren eines Schemas - `<xs:import>`
 - Importiertes Schema kann einen beliebigen Namensraum (ungleich dem aktuellen Namensraum) oder keinen Namensraum haben

Catalog.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  xmlns:cr="http://www.fh-ooe.at/CourseReservation"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog" ...>
```

```
<xs:import
  namespace="http://www.fh-ooe.at/CourseReservation"
  schemaLocation="CourseReservations.xsd">
</xs:import>
```

```
<xs:element name="Reservations" type="cr:CourseReservationsType"/>
```

```
<xs:element name="CourseCatalog" .../>
...
</xs:schema>
```

Inhalt

- Einführung
 - Elemente und Attribute
 - Vordefinierte Datentypen
 - Benutzerdefinierte Datentypen
 - Schlüssel und Schlüsselreferenzen
 - Modularisierung und Komposition
 - **Modellierungsmuster**
-
- Anhang I: DTD versus XML Schema
 - Anhang II: Facetten
 - Anhang III: Entwurfsrichtlinien
 - Anhang IV: Annotationen
 - Anhang V: XML Schema 1.1 – Erweiterungen

Modellierungsmuster 1/8

Beziehungen / Global vs. Lokal / Element vs. Typ

- Beziehungen
 - Realisierung durch geschachtelte Elemente oder über Referenzen
- Globale Element/Attribut-Deklarationen
 - Voraussetzung für Wiederverwendung in gleichem/anderem Schema
 - Wurzelement muss immer global sein
- Lokale Element/Attribut-Deklarationen
 - falls Deklaration nur im Zusammenhang mit deklarierten Typ sinnvoll
- Lokale Elementdeklarationen
 - können mit unterschiedlicher Struktur aber gleichem Namen in unterschiedlichen Typen auftreten
- Lokale Attributdeklarationen
 - sinnvoll, da Attribute meist eng an Elemente gekoppelt sind
- **Entwurfsmuster**
 - Russische Matroschka (Russian Doll)
 - Salamischeiben (Salami Slice)
 - Jalousien (Venetian Blind)
 - Garten Eden (Garden of Eden)

Vgl. Roger Costello: Schema structure patterns, www.xfront.com/GlobalVersusLocal.pdf

Modellierungsmuster 2/8

- Entwurfsmuster unterscheiden sich in der Sichtbarkeit der Elementdeklarationen und Typdefinitionen

		Elementdeklarationen	
		Lokal	Global
Typdefinitionen	Anonym/Lokal	<i>Russian Doll</i>	<i>Salami Slice</i>
	Benannt/Global	<i>Venetian Blind</i>	<i>Garden of Eden</i>

- Globale Elementdeklarationen und Typdefinitionen
 - Direkt unter dem Wurzelement `<xs:schema>`
 - Wiederverwendung in anderen Schemata durch `<xs:include>`, `<xs:redefine>` und `<xs:import>` möglich
- Lokale Elementdeklarationen und Typdefinitionen
 - Elementdeklarationen und Typdefinitionen sind verschachtelt
 - Wiederverwendbarkeit ist nur sehr eingeschränkt gegeben



Modellierungsmuster 3/8

Russische Matryoschka (Russian Doll Design)

- Elementdeklarationen ineinander schachteln
 - Ein einziges globales Element
 - sonst nur lokale Deklarationen
 - vermeidet globale Typdefinitionen
- Vorteile
 - Struktur offensichtlich (entspricht Struktur des XML-Dokuments)
 - Vermeidung von Seiteneffekte
 - restriktive Strukturen möglich
- Nachteile
 - tiefe Schachtelungstiefe der Elemente (Redundanzen)
 - keine Wiederverwendung von Deklarationen und Typen
 - keine Erweiterbarkeit (Ableitung)
 - nur eine XML-Schema-Datei möglich

CourseCatalog.xsd (Ausschnitt)

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="CourseCatalog">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DegreeProgramme" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Course" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Title" type="xs:string"/>
                  </xs:sequence>
                  <xs:attribute name="semester" type="xs:decimal" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="code" use="required">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:pattern value="[d]{4}"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="abbreviation" type="xs:string" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="term" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="summer"/>
            <xs:enumeration value="winter"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



Modellierungsmuster 4/8

Salamischeiben-Stil (Salami Slice)

- Globale Elementdeklarationen
 - Verwendung globaler Elemente per Referenz (**ref**-Attribut)
 - jedes globale Element kann Wurzelement sein
- Lokale Typdeklarationen
- Vorteile
 - Wiederverwendung von globalen Elementdeklarationen
 - mehrere Wurzelemente möglich
- Nachteile
 - große Menge an globalen Elementen (ev. unübersichtlicher)
 - Seiteneffekte bei Änderungen möglich
 - keine Erweiterbarkeit (Ableitung)

CourseCatalog.xsd (Ausschnitt)

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="CourseCatalog">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cc:DegreeProgramme" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="cc:term" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="DegreeProgramme">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="cc:Course" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="cc:code" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="Course">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Title" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="semester" type="xs:decimal" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:attribute name="term">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="summer"/>
        <xs:enumeration value="winter"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

  ...
</xs:schema>
```



Modellierungsmuster 5/8

Jalousien-Design (Venetian Blinds Design)

- Globale Typdeklarationen
 - Elemente sind lokal deklariert (Ausnahme Wurzelement)
- Vorteile
 - Wiederverwendung von Typen
 - zu jedem Element und Attribut existiert ein benannter Typ
 - Typen können aus anderen Schemata importiert werden
 - Erweiterbarkeit (Ableitung und **<xs:redefine>**)
- Nachteil
 - große Menge an globalen Typen (ev. unübersichtlicher)

CourseCatalog.xsd (Ausschnitt)

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="CourseCatalog" type="cc:CourseCatalogType"/></xs:element>

  <xs:complexType name="CourseCatalogType">
    <xs:sequence>
      <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="term" type="cc:termType" use="required"/>
  </xs:complexType>

  <xs:complexType name="DegreeProgrammeType">
    <xs:sequence>
      <xs:element name="Course" type="cc:CourseType" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="code" type="cc:codeType" use="required"/>
  </xs:complexType>

  <xs:complexType name="CourseType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="semester" type="xs:decimal" use="required"/>
  </xs:complexType>

  <xs:simpleType name="termType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="summer"/>
      <xs:enumeration value="winter"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="codeType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[ld]{4}"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```


Modellierungsmuster 6/8

Vergleich

► Anhang III Entwurfsrichtlinien

- *Russian Doll* für **restriktive Strukturen**
 - Struktur der Instanz stark durch Schema vorgegeben
- *Salami Slice* für **variable Strukturen**
 - Struktur der Instanz kann stark schwanken, da aus verschiedenen Wurzelementen ausgewählt werden kann
- *Venetian Blinds* ebenfalls für **variable Strukturen**
 - Struktur der Instanz kann schwanken, falls Typvererbung genutzt wird
- In der Praxis Mischformen, bspw. **Garden of Eden**

Modellierungsmuster 7/8

Mischform – Garten Eden (Garden of Eden)



■ Einflüsse

- *Venetian Blinds* → alle Typdefinitionen global
- *Salami Slice* → alle Elementdefinitionen global

■ Jedes Element wird unter dem Wurzelement definiert

■ Vorteile

- Schemata sind stark wiederverwendbar, da alle Elemente und Typen global definiert wurden.
- Sinnvoll vor allem bei der Entwicklung von Bibliotheken mit umfangreichen Anwendungsbereich (oder wenn der Anwendungsbereich vorab noch nicht genau bekannt ist)

■ Nachteile

- Viele unterschiedliche Wurzelemente möglich
- Datenkapselung durch die globalen Elemente/Typen schwierig
- Oft schwierig zu lesen und zu interpretieren

Modellierungsmuster 8/8

Mischform – Garten Eden (Garden of Eden)

CourseCatalog.xsd (Ausschnitt)



```

<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:cc="http://www.fh-ooe.at/CourseCatalog"
  targetNamespace="http://www.fh-ooe.at/CourseCatalog"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="CourseCatalog" type="cc:CourseCatalogType"/></xs:element>

  <xs:complexType name="CourseCatalogType">
    <xs:sequence>
      <xs:element ref="cc:DegreeProgramme" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="term" type="cc:termType" use="required"/>
  </xs:complexType>

  <xs:element name="DegreeProgramme" type="cc:DegreeProgrammeType"/>

  <xs:complexType name="DegreeProgrammeType">
    <xs:sequence>
      <xs:element name="Course" type="cc:CourseType" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="cc:code" use="required"/>
  </xs:complexType>

  <xs:complexType name="CourseType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="semester" type="xs:decimal" use="required"/>
  </xs:complexType>

  <xs:simpleType name="termType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="summer"/>
      <xs:enumeration value="winter"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:attribute name="code">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="[d]{4}" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

</xs:schema>

```

Inhalt

- Einführung
 - Elemente und Attribute
 - Vordefinierte Datentypen
 - Benutzerdefinierte Datentypen
 - Schlüssel und Schlüsselreferenzen
 - Modularisierung und Komposition
 - Modellierungsmuster
-
- Anhang I: DTD versus XML Schema
 - Anhang II: Facetten
 - Anhang III: Entwurfsrichtlinien
 - Anhang IV: Annotationen
 - Anhang V: XML Schema 1.1 – Erweiterungen

Anhang I

DTD versus XML Schema

Gegenüberstellung

Vergleich DTD – XML Schema 1/6

Allgemeines

	DTD	XML Schema
Syntax	eigene Syntax	benutzt XML Syntax
Struktur	relativ einfache Struktur	komplexe Struktur
Namensräume	✗	✓

Vergleich DTD – XML Schema 2/6

Elemente

	DTD	XML Schema
Defaultwerte	✗	✓
Definition des Inhalts	Text, Elemente, gemischter Inhalt (Text und Elemente)	einfache Typen, komplexe Typen
Reihenfolge	mittels ", " definierbar	<xs:sequence>
Keine Reihenfolge	✗	<xs:all>
Alternative	mittels " " definierbar	<xs:choice>
Kardinalität	"?", "*", "+"	minOccurs und maxOccurs (flexibler)

Vergleich DTD – XML Schema 3/6

Attribute

	DTD	XML Schema
Defaultwerte	✓	✓
Optionalität	✓	✓

Vergleich DTD – XML Schema 4/6

Datentypen

	DTD	XML Schema
Vordefinierte Datentypen	wenige Datentypen; nur String-Datentypen, z.B. <code>CDATA</code> , <code>ID</code> , ...	zahlreiche Datentypen; Vielfalt von Datentypen, z.B. <code>string</code> , <code>integer</code> , ...
Benutzerdef. Datentypen	✗	✓
Wertebereiche	durch Aufzählen aller Werte (nur für Attribute)	verschiedenste Möglichkeiten <code><xs:length></code> , ...
Muster für Datentypen	eingeschränkt u. kompliziert realisierbar (z.B. durch Kardinalitätsspezifikation)	mittels <code><xs:pattern></code> möglich

Vergleich DTD – XML Schema 5/6

Vererbung

	DTD	XML Schema
Ableiten von vordef. und einfachen Datentypen	x	mittels <code><xs:base></code>
Ableiten von komplexen Datentypen (Erweiterung)	x	mittels <code><xs:base></code> und <code><xs:extension></code>
Ableiten von komplexen Datentypen (Einschränkung)	x	mittels <code><xs:base></code> und <code><xs:restriction></code>

Vergleich DTD – XML Schema 6/6

- Die wichtigsten Vorteile von DTD's:
 - schnell und einfach zu erstellen
 - zur Erstellung einfacher Dokumente gut geeignet
- Die wichtigsten Vorteile von XML Schema:
 - zahlreiche vordefinierte Datentypen
 - eigene Datentypen definierbar (Vererbungshierarchie)
 - integrieren Namensräume
 - keine eigene Syntax, sondern selbst XML-Sprache
 - zum Modellieren komplexer Dokumente gut geeignet

Anhang II

Facetten

Wertebereichseinschränkung bei
einfachen Datentypen

Facetten 1/2

Einschränkung des Wertebereiches

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

Facetten 2/2

Einschränkung des Wertebereiches

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

Anhang III

Entwurfsrichtlinien

Entwurfsrichtlinien 1/6

- Verwende globale und lokale Elementdeklarationen
 - Globale Elementdeklarationen können mithilfe von Verweisen in anderen Schemateilen oder anderen Schemadokumenten wieder verwendet werden.
 - Globale Elementdeklarationen können in Ersetzungsgruppen (Substitution Group) verwendet werden.
 - Lokale Elementdeklarationen können mit gleichen Namen in unterschiedlichen Typen auftreten.
 - Lokale Elementdeklarationen sollten dann eingesetzt werden, wenn die Elementdeklaration nur im Zusammenhang mit dem deklarierten Typ sinnvoll ist.

Entwurfsrichtlinien 2/6

- Verwende globale und lokale Attributdeklarationen
 - Globale Attributdeklarationen können mithilfe von Verweisen in anderen Schemateilen oder anderen Schemadokumenten wieder verwendet werden.
 - Lokale Attributdeklarationen sollten dann eingesetzt werden, wenn die Elementdeklaration nur im Zusammenhang mit dem deklarierten Typ sinnvoll ist.
 - Lokale Attributdeklarationen sind vorzuziehen, da Attribute gewöhnlich eng an die ihnen übergeordneten Elemente gekoppelt sind.

Entwurfsrichtlinien 3/6

- Definiere **elementFormDefault** immer als **qualified**
 - Lokale Elemente im XML-Dokument sind somit auch dem Zielnamensraum zuzuordnen (→ Dokumentation)
- Verwende Attributgruppen und Elementgruppen
 - Benannte Auflistung von Attributen/Elementen können an einer einzigen Stelle deklariert werden und ein oder mehrere Schemata können dann darauf verweisen.
- Verwende integrierte einfache Typen (44 Datentypen)
 - Schränke die verwendeten Typen auf eine Menge ein, die bewältigt werden kann.
- Bevorzuge für Identitätseinschränkungen **<xs:key>**, **<xs:keyref>** und **<xs:unique>** gegenüber **ID/IDREF**

Entwurfsrichtlinien 4/6

■ Verwende komplexe Typen

- Benannte komplexe Typen ermöglichen Typableitung und Wiederverwendung (im internen und externen Schemadokumenten)
- Anonyme Typen sollten nur dann verwendet werden, wenn Verweise auf den Typ nicht außerhalb der Elementdeklaration benötigt werden und keine Typableitung gebraucht wird.

■ Vermeide Standard- oder feste Werte

- Durch Standard- und feste Werte werden neue Daten nach der Prüfung in das XML-Dokument eingefügt und dadurch die Daten verändert.
- Das bedeutet, dass ein XML-Dokument mit einem Schema mit Standardwerten, das nicht geprüft wurde, nicht vollständig ist.

Entwurfsrichtlinien 5/6

- Verwende Einschränkungen von einfachen Typen
- Verwende Erweiterungen von komplexen Typen
 - Wiederverwendung durch Erweiterung ist eine leistungsstarke Funktion und entspricht den Konzepten der objektorientierten Programmierung.
- Verwende Einschränkungen von komplexen Typen mit Vorsicht
 - Eine Vielzahl von Nuancen der Ableitung durch Einschränkung in komplexen Typen führt oft zu Programmierfehler.
 - Ableitung durch Einschränkung von komplexen Typen entspricht nicht den Konzepten der objektorientierten Programmierung.

Entwurfsrichtlinien 6/6

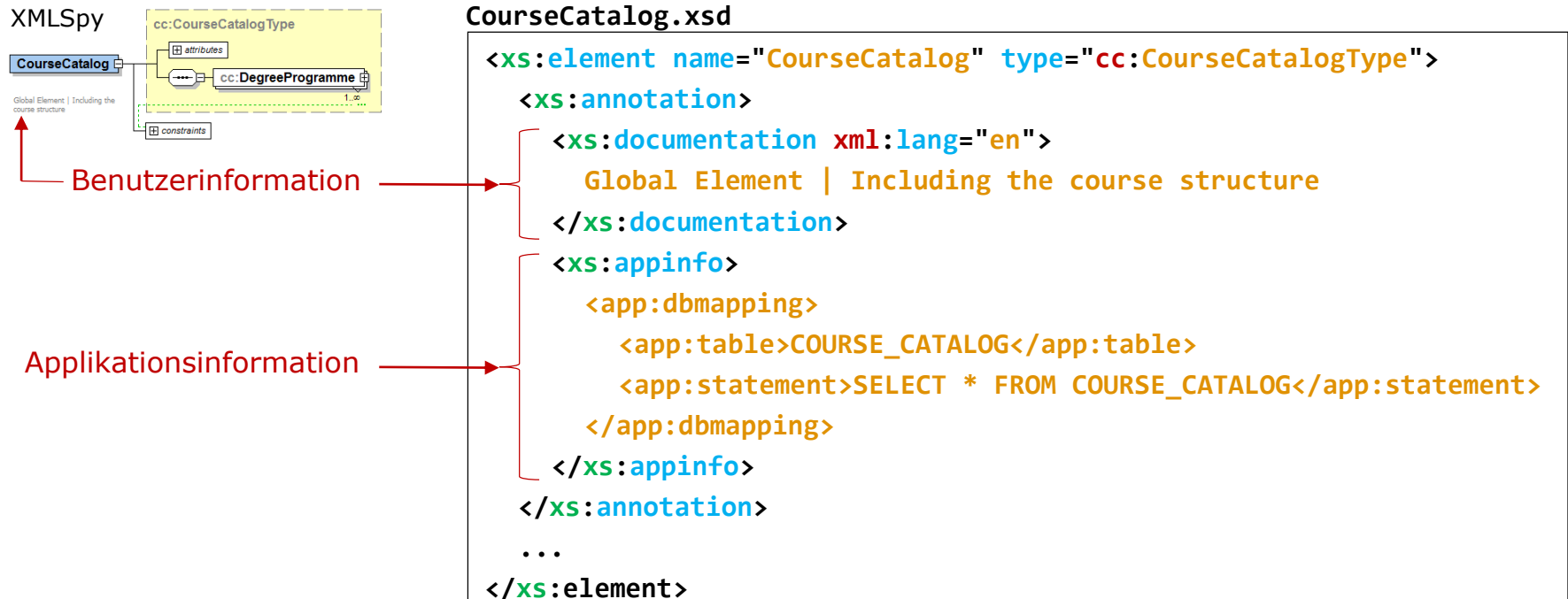
- Verwende Platzhalter **any/anyAttribute**, um fest definierte Punkte für die Erweiterbarkeit bereitzustellen
- Vermeide Typen- und Gruppenneudefinition mit **<xs:redefine>**
 - Alle Verweise auf den ursprünglichen Typ oder Gruppe in beiden Schemata verweisen auf den neu definierten Typ, während die ursprüngliche Definition verdeckt wird.
 - Verwende **<xs:override>** (XML Schema 1.1), um Element- und Attributdeklarationen, Typdefinitionen sowie Element- und Attributgruppen zu überschreiben/ersetzen
- Mache Typnamen erkennbar
 - Typ(e) Anhang oder andere Schreibweise
- Validiere Schemata mit mehreren Schemaprozessoren
- Versuche nicht, XML Schema meisterhaft zu beherrschen
 - Das würde Monate dauern!

Anhang IV

Annotationen

Annotationselement

- `<xs:annotation>` kann allen XML Schema-Elementen als erstes Kindelement hinzugefügt werden und enthält die optionalen Elemente
 - `<xs:documentation>` für menschenlesbare Dokumentation - Benutzer
 - `<xs:appinfo>` für maschinenlesbare Zusatzinformation - Applikation (z.B. Metadaten, Verarbeitungsanweisungen, Programmteile)



Anhang V

XML Schema Definition Language (XSD) 1.1

Erweiterungen

XML Schema 1.0 – Schwachstelle

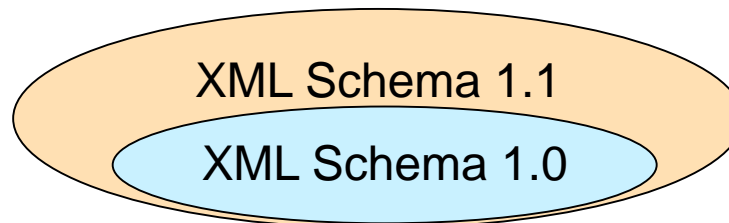
Co-Occurrence Constraints / Co-Constraints

- Unterstützung von besseren Beschränkungen (constraints) bzw. Zusicherungen (assertions)
 - Komplexere Beschränkungen und Zusicherungen (die mehr als ein Element betreffen) mussten in der jeweiligen Applikationslogik implementiert werden (XML Schema 1.0 bietet hier nur grundlegende Möglichkeiten an).
 - Vergleiche: Schematron und RelaxNG (weitere Schema-Sprachen) – Möglichkeiten deutlich ausgeprägter.

```
<xs:complexType name="intRange">  
  <xs:attribute name="min" type="xs:int"/>  
  <xs:attribute name="max" type="xs:int"/>  
  <xs:assert test="@min <= @max"/> <!-- co-constraint -->  
</xs:complexType>
```

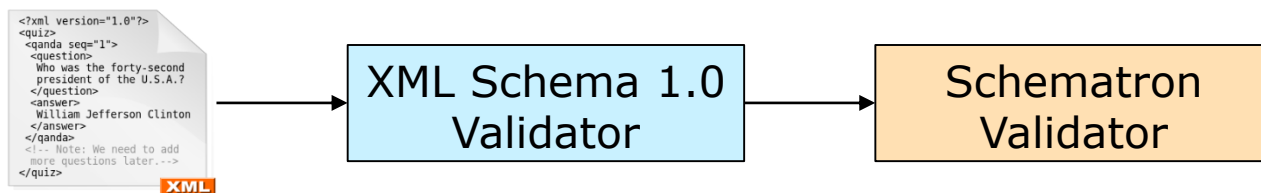
XML Schema 1.1

- W3C Recommendation seit April 2012
- Neuerungen (Auswahl)
 - Regel-basierte Validierung - Zusicherungen `<xs:assert>` und `<xs:assertion>`
 - Bedingte Typisierung / Datentyp-Alternativen `<xs:alternative>`
 - Standard-Attributgruppen und Schemaweite Attribute
 - Offener Inhalt `<xs:openContent>` `<xs:defaultOpenContent>`
 - Attribute an Kindelemente vererben (inheritable)
 - Schemata wiederverwenden über `<xs:override>` und `<xs:error>`
 - Aufweichung der Reihenfolge von Elementen (all)
 - Ersetzungsgruppen für Wörter (substitution)
- XML Schema 1.1 baut auf XML Schema 1.0 auf



XML Schema 1.1 - Regel-basierte Validierung

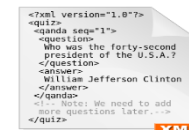
- XML Schema 1.0 unterstützt die Definition und Validierung von Grammatik-Regeln
 - Werden die richtigen Elemente / Attribute verwendet?
 - Werden die richtigen Attribut-Werte verwendet?
 - ...
- XML Schema 1.0 unterstützt keine Definition und Validierung von Geschäftsregeln
 - Zusicherungen innerhalb eines XML-Instanzdokumentes, die vom jeweiligen Geschäftsfall abhängig sind
 - Dafür ist mit XML Schema 1.0 ein weiterer Verarbeitungsschritt notwendig!



XML Schema 1.1 - Regel-basierte Validierung

`<xs:assert>` / `<xs:assertion>`

- Element- oder Attributwerte können mithilfe von XPath 2.0-Ausdrücken validiert werden.
 - Ähnlich zu XML-Schemasprache *Schematron* oder *RelaxNG*
 - Beispiel: `<xs:assertion test="xpath"/>`
 - Attributwert für `test` muss ein gültiger XPath-2.0-Ausdruck sein, der `true` oder `false` zurückgibt.
 - Spezielle Variable `$value`, um auf den zu prüfenden `simpleContent`-Wert zugreifen zu können
 - Evaluierung erfolgt im Kontext des Elternknotens
 - Zugriff nur auf Nachfahren eines Elementes (u.a. wegen Kompatibilität zu „streaming validation“ – SAX)
 - Zugriff auf andere Dokumente nicht erlaubt (kein `doc(...)`)
- Geschäftsregeln werden dadurch Teil des XML-Schemas und müssen nicht mehr in der jeweiligen Anwendung implementiert werden
 - Vorteil für Lesbarkeit und Wartung



XML Schema 1.1
Validator

Grammatik und
Geschäftsregeln

XML Schema 1.1 - Regel-basierte Validierung

`<xs:assert>` / `<xs:assertion>`

■ Assertion für einfachen Datentyp

- `<xs:assertion>`-Facette
- Zugriff auf den Wert des einfachen Datentyps über `$value`

```
<xs:simpleType name="SizeType">  
  <xs:restriction base="xs:integer">  
    <xs:assertion test="$value != 0"/>  
  </xs:restriction>  
</xs:simpleType>
```

■ Assertion für komplexen Datentyp

- `<xs:assert>`-Element für Zusicherungen über Element- und Attributwerte
- Zugriff auf Elemente / Attribute über deren Namen

```
<xs:complexType name="PointType">  
  <xs:attribute name="X" type="xs:integer"/>  
  <xs:attribute name="Y" type="xs:integer"/>  
  <!-- XPath-2.0-Ausdruck -->  
  <xs:assert test="(@X eq @Y) or (@X lt @Y)"/>  
</xs:complexType>
```

XML Schema 1.1 - Regel-basierte Validierung

<xs:assert> – Beispiele

```
<xs:complexType name="ProductType">
  <xs:sequence>
    <xs:element name="number" type="xs:integer"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="size" type="SizeType"/>
  </xs:sequence>
  <xs:attribute name="dept" type="xs:string"/>
  <xs:assert test="
    ((@dept eq 'ELEC' and number gt 500) or
    (string-length(@dept) lt 4))"/>
</xs:complexType>
```

<xs:complexType>
<xs:assert>-Element

Vererbung möglich – alle
test-Ausdrücke müssen
true sein

Gültig, da test-Ausdruck true ergibt

```
<Product dept="ELEC">
  <number>501</number>
  <name>iPhone XS</name>
  <size>10</size>
</Product>
```

Nicht gültig, da string-length(@dept) >= 4

```
<Product dept="ELECTRONICS">
  <number>200</number>
  <name>iPhone XS</name>
  <size>10</size>
</Product>
```

XML Schema 1.1 - Regel-basierte Validierung

<xs:assertion> – Beispiele

```
<xs:simpleType name="DepartmentCodeType">
  <xs:restriction base="xs:token">
    <xs:length value="3"/>
    <xs:assertion test="not(contains($value,'X'))"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="RestrictedDepartmentCodeType">
  <xs:restriction base="DepartmentCodeType">
    <xs:assertion test="substring($value,2,2) != '00'"/>
    <!-- Assertion-Facette für "simpleType" -->
  </xs:restriction>
</xs:simpleType>
```

→ <xs:simpleType>
 <xs:assertion>-Facette

Vererbung möglich → alle
test-Ausdrücke müssen
true sein

Gültig

<Department>K20</Department>

Nicht gültig

<Department>X20</Department>

XML Schema 1.1 - Bedingte Typisierung

`<xs:alternative>`

- Elementtyp kann, je nach Attribut-Werten im Instanz-Dokument, dynamisch zugewiesen werden
- Elementdeklaration wird um `<xs:alternative>`-Sequenz erweitert
 - Attribut `test`: Bedingung, die zutreffen muss
 - Attribut `type`: Typ, der dem Element dynamisch zugewiesen wird
- Bedingte Typisierung kann auch über Zusicherungen nachgebaut werden
 - Dabei werden keine vordefinierten Datentypen ausgewählt, sondern bestimmte Nachfahren über XPath-Ausdrücke zugelassen/verboten.

XML Schema 1.1 - Bedingte Typisierung

<xs:alternative> – Beispiele

```
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="Vorname" type="xs:string"/>
    <xs:element name="Nachname" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="kind" type="xs:string"/>
</xs:complexType>
```

```
<xs:complexType name="TeacherType">
  <xs:complexContent>
    <xs:extension base="PersonType">
      <xs:sequence>
        <xs:element name="PersonalNumber" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:element name="Persons">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="Person" type="PersonType">
        <xs:alternative test="@kind eq 'teacher'" type="TeacherType"/>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<Person kind="teacher">
  <Vorname>Julian</Vorname>
  <Nachname>Haslinger</Nachname>
  <PersonalNumber>P22080</PersonalNumber>
</Person>
```

Typ abhängig von
test-Auswertung

XML Schema 1.1 - Standard-Attribute und Attributgruppen

- Grundidee: Bestimmte Attribute oder Attributgruppen sollen in einem XML-Dokument bei allen/vielen Elementen verwendet werden.
- Möglichkeiten in XML Schema 1.0:
 - Erstellung einer Attributgruppe und explizite Modellierung zu jedem komplexen Datentypen
 - Alle Datentypen erben von einem „Grunddatentyp“, der nichts außer die Attribute definiert
- Lösung in XML Schema 1.1: Schemaweite Attribute
 1. `<xs:attributeGroup>` definieren
 2. `<xs:schema ... defaultAttributes="Attributgruppe">`
 3. Attribute werden automatisch Teil aller `<xs:complexType>`-Definitionen
- `defaultAttributesApply="false"` zum Deaktivieren für einzelne Elemente

XML Schema 1.1 - Standard-Attribute und Attributgruppen – Beispiele

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning" xmlns:fh="www.fh-ooe.at/xmlschema/xml11"
targetNamespace="www.fh-ooe.at/xmlschema/xml11" elementFormDefault="qualified"
defaultAttributes="fh:DefaultAttributes" vc:minVersion="1.1">
```

```
<xs:attributeGroup name="DefaultAttributes">
  <xs:attribute name="CreatedAt" type="xs:gYear"/>
  <xs:attribute name="CreatedBy" type="xs:string"/>
</xs:attributeGroup>
```

```
<xs:complexType name="ProductType" defaultAttributesApply="false">
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>
```

```
<xs:complexType name="OrderType">
  <xs:sequence>
    <xs:element name="OrderName"/>
  </xs:sequence>
  <xs:attribute name="OrderNr" type="xs:integer" use="required"/>
</xs:complexType>
```

```
<Order OrderNr="12" CreatedAt="2017" CreatedBy="JH">
  <OrderName>Order_1</OrderName>
</Order>
```

```
<Product>iPhone XS</Product>
```

Element mit
Default-Attributen

Element mit deaktivierten
Default-Attributen

XML Schema 1.1 - Offener Inhalt

`<xs:openContent>` / `<xs:defaultOpenContent>`

- Grundidee: Erhöhung der XML-Schema-Flexibilität
- XML Schema 1.0: Geänderte Anforderungen an das Schema können oftmals schwierig ohne Hilfe der Schema-Entwickler eingearbeitet werden.
 - starre Schemata
- XML Schema 1.1: Während der Entwicklung eines Schemas bereits „offenen Inhalt“ (Sub-Elemente, die nicht im Inhaltsmodell definiert wurden) einplanen. Die Instanz-Dokumente können dann sofort geändert werden und das Schema kann (muss aber nicht) später nachgezogen (um die neuen Elemente) erweitert werden.
 - Definition auf Schema-Ebene oder für einzelne komplexe Datentypen
 - Definition von offenem Inhalt beinhaltet eine „Element-Wildcard“, z.B.

○ `<xs:any namespace="##any" processContents="skip"/>`

NS, aus welchem die Elemente stammen dürfen
z.B. `##any`, `##other`, `##local`, `##targetNamespace`, explizite Angabe

Validierung: `strict`, `lax`, `skip`

XML Schema 1.1 - Offener Inhalt

`<xs:openContent>` / `<xs:defaultOpenContent>`

- Offener Inhalt in komplexen Datentypen (**openContent**)
 - `<xs:openContent>`-Element als Kindelement oder als Teil von `<xs:restriction>` / `<xs:extension>`
 - Attribut: **mode**
 - **interleave** - offener Inhalt kann überall im `complexType` vorkommen
 - **suffix** - offener Inhalt darf nur am Ende einer Sequenz vorkommen
 - **none** - `complexType` verwendet **defaultOpenContent** nicht
- Offener Inhalt im gesamten XML-Schemadokument (**defaultOpenContent**)
 - Oftmals Anforderung, offenen Inhalt für viele komplexe Datentypen zuzulassen.
 - `<xs:defaultOpenContent>` als Kindelement vom Schema definieren

XML Schema 1.1 - Offener Inhalt

<xs:openContent> – Beispiele

```
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="Vorname" type="xs:string"/>
    <xs:element name="Nachname" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="kind" type="xs:string"/>
</xs:complexType>
```

Offener Inhalt für Elemente vom Typ **TeacherType**

interleave: Offener Inhalt kann im ganzen Element vorkommen

```
<xs:complexType name="TeacherType">
  <xs:complexContent>
    <xs:extension base="PersonType">
      <xs:openContent mode="interleave">
        <xs:any namespace="##any" processContents="skip"/>
      </xs:openContent>
      <xs:sequence>
        <xs:element name="PersonalNumber" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:element name="Person" type="PersonType">
```

```
<Person>
  <Vorname>Julian</Vorname>
  <MittlererName>Paul</MittlererName>
  <Nachname>Haslinger</Nachname>
  <PersonalNumber>p22080</PersonalNumber>
</Person>
```

Offener Inhalt: Neues Element **<MittlererName>**

XML Schema 1.1 - Offener Inhalt

<xs:defaultOpenContent> – Beispiele

```
<xs:schema ...
  <xs:defaultOpenContent mode="suffix" appliesToEmpty="true">
    <xs:any/>
  </xs:defaultOpenContent>
...
```

Offener Inhalt:
Kann in jedem Element
vorkommen;
auch in Elementen, die „leer“
definiert
wurden (**appliesToEmpty**).

```
<Persons>
  <Teacher>
    <Vorname>FirstName Teacher</Vorname>
    <Nachname>LastName Teacher</Nachname>
    <PersonalNummer>P22080</PersonalNummer>
  </Teacher>

  <Student>
    <Vorname>FirstName Student</Vorname>
    <Nachname>LastName Student</Nachname>
    <NeuesElement attribute="Wert"></NeuesElement>
  </Student>
</Persons>
```

suffix: Offener Inhalt darf nur
am Ende eines Elements
vorkommen (default: **interleave**)

XML Schema 1.1 – Ersetzungsgruppen

Erweiterung zu XML Schema 1.0

- Grundidee: Element kann durch ein anderes Element (Namen) ersetzt werden
 - Beispiel: Element `<Book>` soll im Instanzdokument durch mehrere andere Element ersetzt werden können.
 - Beschreibung von beiden bzw. mehreren Elementen
 - Angabe der Elemente in `substitutionGroup`-Attribut von `<Book>`
 - Beispiel: Angabe von mehreren Ersetzungen

```
<xs:element name="Buch"/>
<xs:element name="Livre"/>

<xs:element name="Book" substitutionGroup="Buch Livre"/>
...
</xs:element>
```

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

`<पुस्तक>...</पुस्तक>`

`<Livre>...</Livre>`

XML Schema 1.1 - Wiederverwendung

`<xs:override>` und Datentyp `<xs:error>`

- XML Schema 1.0: Über `<xs:redefine>` konnten Teile von bestehenden Schemata übernommen und modifiziert werden.
 - Globale Datentypen aus anderem Schema erweitern / einschränken
- XML Schema 1.1: Neues Element `<xs:override>`
 - Global deklarierte Items aus anderen Schemata können im eigenen Schema überschrieben / ersetzt werden.
 - Elemente, Attribute, Datentypen (`<xs:simpleType>`, `<xs:complexType>`), Element- und Attributgruppen
 - Über Datentyp `<xs:error>` werden Teile des importierten Schemas von der weiteren Verwendung ausgeschlossen