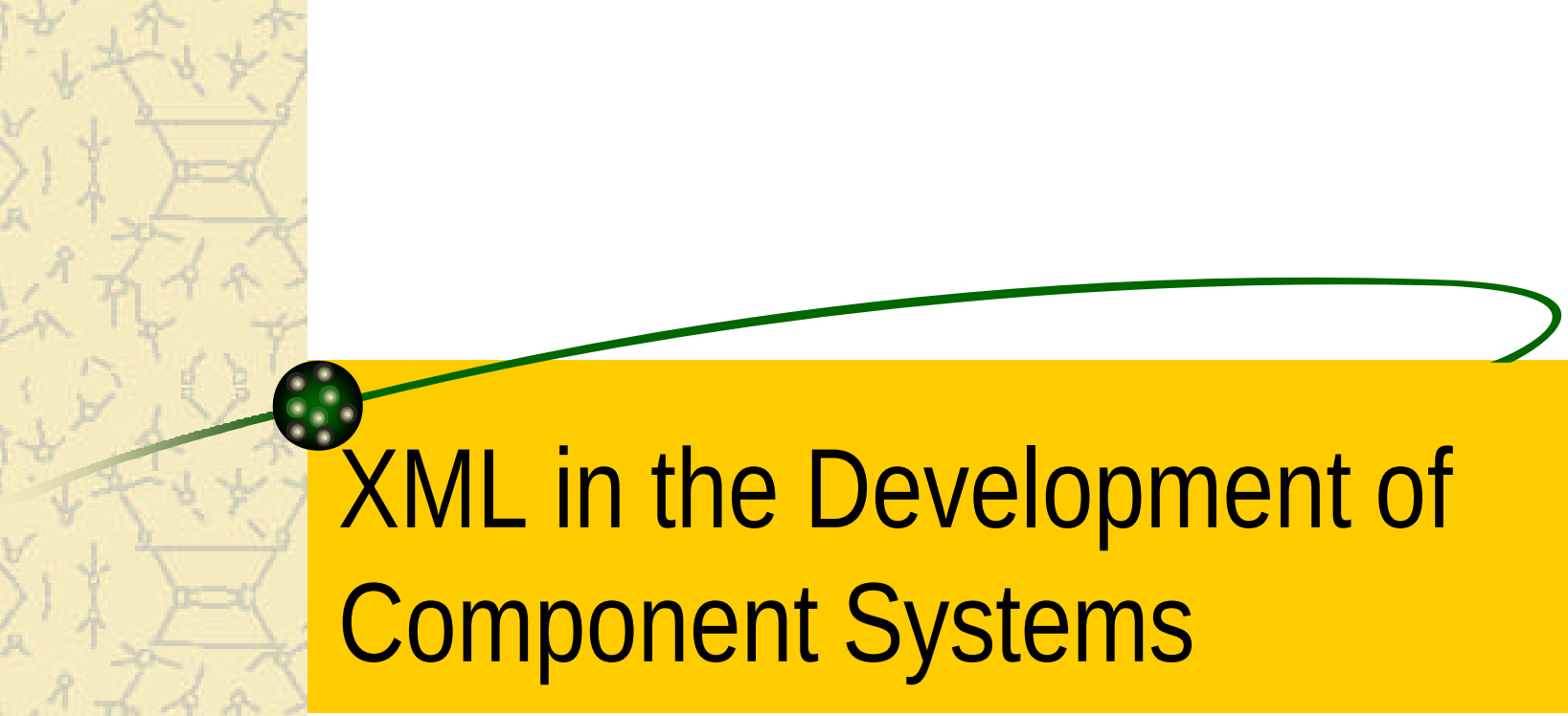

XML in the Development of Component Systems

XML Schema (Part 3)

Custom Data Types: Union

- e.g. definition of the type for the maxOccurs attribute

```
<xs:simpleType name="maxOccursType">
 <xs:union memberTypes="xs:nonNegativeInteger">
  <xs:simpleType>
   <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="unbounded"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>
```

- possible values are now 1, 2, 3, …, unbounded

Custom Data Types: Union (2)

- e.g. an extended ISBN type

```
<xs:simpleType name="extISBNType">
 <xs:union>
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:pattern value="\d-\d{3}-\d{5}-\d" />
     <xs:pattern value="\d-\d{5}-\d{3}-\d" />
     <xs:pattern value="\d-\d{2}-\d{6}-\d" />
   </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
   <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="TBD"/>
    <xs:enumeration value="NA"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>
```

Custom Data Types: Lists

```
<xs:simpleType name="ISBNListType">
  <xs:list itemType="ISBNType" />
</xs:simpleType>

<xs:simpleType name="ISBNListType10">
  <xs:restriction base="ISBNListType">
    <xs:minLength value="1" />
    <xs:maxLength value="10" />
  </xs:restriction>
</xs:simpleType>
```

- Values are space-separated items
  - e.g. “3-251-00452-2 3-89721-286-2 0-09-920191-7"

Derivation of Complex Types

- Derivation by extension: new elements or attributes
- Derivation by restriction: removal of possible values

Derivation by Extension

- Example: addresses

```
<xs:complexType name="Adresse">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />
    <xs:element name="Straße" type="xs:string" />
    <xs:element name="Ort" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Derivation by Extension (2)

- Derivation to DEAdresse

```
<xs:complexType name="DeAdresse">
  <xs:complexContent>
    <xs:extension base="Adresse">
      <xs:sequence>
        <xs:element name="PLZ" type="xs:decimal" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Derivation by Extension (3)

- Derivation to USAAdresse

```
<xs:complexType name="USAAdresse">
 <xs:complexContent>
  <xs:extension base="Adresse">
   <xs:sequence>
    <xs:element name="State" type="USBundesstaat" />
    <xs:element name="ZIP" type="xs:decimal" />
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

- Additional elements are appended at the end
  - Extending choices is not possible
  - Inserting elements at the start is not possible

Derivation by Extension (4)

- Polymorphism: actual type can vary from declared type
  - xsi:type indicates actual type (xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance")
- e.g. <xs:element name="Lieferadresse" type="Adresse" /> can be instantiated as

```
<Lieferadresse xsi:type="USAAdresse">
    <Name>Alice Smith</Name>
    <Straße>42 Walnut Alley</Straße>
    <Ort>Old Town</Ort>
    <State>CA</State>
    <ZIP>90342</ZIP>
</Lieferadresse>
```

Derivation by Restriction

- Consider the base type

```
<xs:complexType name="rezeptTyp">
 <xs:sequence>
  <xs:element ref="titel" />
  <xs:element ref="zutaten" minOccurs="0" />
  <xs:element ref="zubereitung" />
  <xs:element ref="idee" minOccurs="0" maxOccurs="unbounded" />
 </xs:sequence>
</xs:complexType>
```

Derivation by Restriction (2)

- Require that *zutaten* must occur, and idee can occur at most 4 times

```
<xs:complexType name="rezeptTyp2">
 <xs:complexContent>
  <xs:restriction base="rezeptTyp">
   <xs:sequence>
    <xs:element ref="titel" />
    <xs:element ref="zutaten" />  <!-- no minOccurs, i.e. 1 -->
    <xs:element ref="zubereitung" />
    <xs:element ref="idee" minOccurs="0" maxOccurs="4" />
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

Derivation by Restriction (3)

- Removed attributes can be expressed through use="prohibited"

Base type:

```
<xs:complexType name="personTyp">
        <xs:sequence>...</xs:sequence>
        <xs:attribute name="gehalt" type="xs:integer" use="optional" />
</xs:complexType>
```

Derived Type:

```
<xs:complexType name="studentTyp">
  <xs:complexContent>
    <xs:restriction base="personTyp">
        <xs:sequence>...</xs:sequence>
        <xs:attribute name="gehalt" type="xs:integer" use="prohibited" />
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Further Derivation Features

- abstract types: set abstract="true" on the type
  - Instance must use a derived type, and declare that through xsi:type
- final types: set final="restriction" | "extension" | "#all"
  - Type is final with respect to restriction, extension, or both
- blocked types: set block="restriction" | "extension" | "#all"
  - Further derivation is allowed, but derived types cannot replace the base type

Uniqueness

- Attributes declared of ID type in DTD must be unique
  - Uniqueness applies to the entire document
  - ID values must follow Name production
  - not applicable to element content
  - not suitable for structured values
- Schema generalizes IDs to the xs:unique constraint

Uniqueness (2)

```
<xs:unique name="name of uniqueness constraint">
  <xs:selector xpath="base set" />
  <xs:field xpath="unique value" />
</xs:unique>
```

- xs:selector specifies the set with unique identification
- xs:field specifies identification within each element of the set
- multiple xs:field can be provided for a xs:unique, forming a combined key

Uniqueness (3)

```
<xs:element name="rezept">
 <xs:complexType>
  <!– content model for rezept -->
 </xs:complexType>
 <xs:unique name="zutatenIDs">
  <xs:selector xpath="zutaten/zutat" />
  <xs:field xpath="@id" />
 </xs:unique>
</xs:element>
```

- in the context of *rezept*, the *id* attributes of *zutaten/zutat* must be unique
  - no longer have to specify type of id as xs:ID

Foreign Keys

- In DTDs, IDREF(s) attributes allow referral to ID attributes
- Schema generalizes this with xs:key and xs:keyref
- xs:key specifies the primary key
  - similar to xs:unique
  - selected key values must be present
- xs:keyref specifies the foreign key
  - refers to primary keys specified elsewhere
  - in the instance, the value of the keyref must be present as primary key

Foreign Keys (2)

```
<xs:element name="rezept">
  <xs:complexType>
    <!– content model for rezept -->
  </xs:complexType>
  <xs:key name="zutatenIDs">
    <xs:selector xpath="zutaten/zutat" />
    <xs:field xpath="@id" />
  </xs:key>
  <xs:keyref name="zubereitungREF" refer="zutatenIDs">
    <xs:selector xpath="zubereitung/zutat" />
    <xs:field xpath="@ref" />
  </xs:keyref>
</xs:element>
```

Documentation

- Entire document, every type, every element etc. can have explicit documentation
  - Usage of XML comments is discouraged, as processor may remove comments

```
<xs:annotation>
 <xs:documentation>
  This is where documentation for a human reader goes.
 </xs:documentation>
 <xs:appinfo>
  This is where control information for a machine goes.
 </xs:appinfo>
</xs:annotation>
```

- Both xs:documentation and xs:appinfo are optional.
- A source= attribute can specify a URI for further information

```
<xs:complexType name="siegertyp">
 <xs:annotation>
  <xs:documentation source="http://www.loser.com/how/to/win">
   Dieser Typ lässt Sie nicht im Regen stehen.
  </xs:documentation>
  <xs:appinfo source="http://www.loser.com/schematron/check.sch">
   <sch:assert xmlns:sch="http://www.ascc.net/xml/schematron"
            test="count(sieg) &gt; count(niederlage)">
    Wir fordern einen anderen Schiedsrichter!
   </sch:assert>
  </xs:appinfo>
 </xs:annotation>
 <xs:choice maxOccurs="unbounded">
  <xs:element ref="sieg" minOccurs="0" />
  <xs:element ref="niederlage" minOccurs="0">
   <xs:annotation>
    <xs:documentation>
     Darüber schweigen wir lieber.
    </xs:documentation>
   </xs:annotation>
  </xs:element>
 </xs:choice>
</xs:complexType>
```

Namespaces

- Typically, all global elements in a schema are part of a namespace
- Specified through targetNamespace= attribute of xs:schema
  - Applies to all global elements, attributes, types, and groups
  - also applies to local elements if elementFormDefault="qualified"
  - also applies to local attributes if attributeFormDefault="qualifies"
- All references can use qualified names
  - type, ref, base, itemType, refer
- Each schema specifies only a single namespace

Namespaces (2)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema“
           targetNamespace="http://www.beispiel.de/musik“
           xmlns="http://www.beispiel.de/musik“
           elementFormDefault="qualified">
    <xs:element name="band" type="xs:string" />
    <xs:element name="konzert">
              ...
    <xs:element ref="band" />
     ...
    </xs:element>
</xs:schema>
```

- Specifying the default namespace is necessary to support “band” (unprefixed) in ref=
  - Could have used xmlns:m="http://www.beispiel.de/musik“ instead, would require ref=“m:band”

Namespaces (3)

- To include elements, attributes, … of a different namespace, use xs:import

<xs:import namespace="imported namespace"

schemaLoction="URI of schema"/>

- Still need to declare namespace prefix to refer to imported elements
- Only global definitions are imported

Namespaces (4)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema“
           xmlns:h="http://www.w3.org/1999/xhtml“
           targetNamespace="http://www.beispiel.de/musik">
 <xs:import namespace="http://www.w3.org/1999/xhtml“
    schemaLocation="xhtml.xsd" />
 <xs:element name="info">
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="h:p" maxOccurs="unbounded" />
       </xs:sequence>
      </xs:complexType>
 </xs:element>
 ...
</xs:schema>
```

Wildcard Content

<xs:any namespace="what namespace"
processContents="validation constraints"/>

- Possible values for namespace=:
  - ##any – elements from an arbitrary namespace
  - ##other – elements from an namespace except target namespace
  - ##local – elements without a namespace
  - ##targetNamespace – short for target namespace
  - list of namespace URIs
- Possible values for processContents=:
  - strict – validate content
  - lax – validate only if schema is available
  - skip – do not validate; skip content

Wildcard Content (2)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.beispiel.de/musik">
 <xs:import namespace="http://www.w3.org/1999/xhtml"
    schemaLocation="xhtml.xsd" />
 <xs:element name="info">
       <xs:complexType>
        <xs:sequence>
          <xs:any namespace="http://www.w3.org/1999/xhtml“
                  processContents="lax“
                  maxOccurs="unbounded" />
        </xs:sequence>
       </xs:complexType>
 </xs:element>
     ...
</xs:schema>
```

Instance Creation

- Case 1: Elements are not in a namespace

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

Instance Creation (2)

- Case 2: Elements are in a namespace

```
<konzert
   xmlns="http://www.beispiel.de/musik"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.beispiel.de/musik musik.xsd">
  <band>
    ...
  </band>
</konzert>
```

Schema Validation

- Three inputs:
  - document
  - schema
  - schema of XML schema
- Resulting documents are called “schema-valid”