

XML Schema Teil III

Erik Wilde

15.5.2006

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

15.5.2006

XML Vorlesung FHA SS 2006

1

Übersicht

- Reusable Groups
 - Named Model Groups
 - Attribute Groups
- Type Substitution
- Modellierungstechniken in XML Schema
- Namensgebung in Schemas
 - qualified vs. unqualified Names
- Modularisierung von XML Schema
 - Verwendung mehrerer Schemas
 - Erweiterung und Versionierung von Schemas

15.5.2006

XML Vorlesung FHA SS 2006

2

Was sind Reusable Groups?

- zwei Arten von Reusable Groups
 - einmalige Deklaration und mehrfache Verwendung
 - benannte Gruppierungen von XML Inhalt
 - Named Model Groups für Elementgruppen
 - Attribute Groups für Attributgruppen
- Referenzierung über Gruppennamen
 - in Element Content oder Attributdeklarationen
- entspricht Parameter Entities in DTDs
 - in DTDs eher eine Notlösung
 - in XML Schema ein eingebautes Konzept

15.5.2006

XML Vorlesung FHA SS 2006

3

Named Model Groups

- Content-Model Strukturen (Gruppe von Elementen), die mehrmals in einem Schema auftreten, werden am besten als *Named Model Group* deklariert
- Beispiel: Elemente *Beschreibung* und *Kommentar* werden in jedem Objekt zur Dokumentation verwendet
- die Gruppe von Element-Deklarationen wird benannt
- überall im Schema, wo die Gruppe von Elementen in einem Content Model verwendet wird, kann sie – anstelle einer Neu-Definition – über den Namen referenziert werden
- Named Model Groups dürfen andere Named Model Groups referenzieren

15.5.2006

XML Vorlesung FHA SS 2006

4

Named Model Groups

- Beispiel:

```
<xs:group name="dokumentation">
  <xs:sequence>
    <xs:element name="Beschreibung" type="xs:string"/>
    <xs:element name="Kommentar" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:group>

...
<xs:group ref="dokumentation"/>
...
```

15.5.2006

XML Vorlesung FHA SS 2006

5

Attribute Groups

- Gruppen von Attribute, die mehrmals in einem Schema auftreten, werden am besten als *Attribute Group* deklariert
- Beispiel: Attribute *ID*, *Name* und *Version*
- die Gruppe von Attribut-Deklarationen wird benannt
- überall im Schema, wo die Gruppe von Attributen in einem Schema verwendet wird, kann sie – anstelle einer Neu-Definition – über den Namen referenziert werden
- Attribute Groups dürfen andere Attribute Groups referenzieren

15.5.2006

XML Vorlesung FHA SS 2006

6

Attribute Groups

- Beispiel:

```
<xs:attributeGroup name="identifizier">
  <xs:attribute name="Name" type="xs:string" use="optional"/>
  <xs:attribute name="ID" type="xs:integer"/>
</xs:attributeGroup>
<xs:attributeGroup name="versioning">
  <xs:attribute name="Version" type="xs:integer"/>
  <xs:attributeGroup ref="identifizier"/>
</xs:attributeGroup>
...
<xs:attributeGroup ref="versioning"/>
...
```

15.5.2006

XML Vorlesung FHA SS 2006

7

Vorteile von Reusable Groups

- Konsistenz über Schema-Komponenten kann erhöht werden
- Änderungen können an einem zentralen Ort gemacht werden
- Zusammenhänge der einzelnen Schema-Komponenten werden offensichtlicher (Teilen von gemeinsamen Strukturen)
- Schema wird übersichtlicher

15.5.2006

XML Vorlesung FHA SS 2006

8

Was ist Type Substitution?

- ein Element eines abgeleiteten Typen kann in einer Instanz ein Element des entsprechenden Basis-Typs ersetzen
- Type Substitution Varianten:
 - der Typ eines Elementes muss dann mit Attribut `xsi:type` in der Element-Instanz deklariert werden
 - der Typ wird per *Substitution Group* durch den Elementnamen signalisiert
- die Type Substitution kann mehrstufig sein

15.5.2006

XML Vorlesung FHA SS 2006

9

Beispiel Type Substitution (I)

- Beispiel: 1. Typ ableiten

```
<xs:complexType name="kochkursTyp">
  <xs:complexContent>
    <xs:extension base="kursTyp">
      <xs:sequence>
        <xs:element name="Kueche" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

15.5.2006

XML Vorlesung FHA SS 2006

10

Beispiel Type Substitution (II)

- Beispiel: 2. Type Substitution

```
<kursliste>
  <kurs letzteAenderungen="2002-04-06">
    <kursname>XML Schema Kurs</kursname>
    ...
  </kurs>
  <kurs xsi:type="kochkursTyp"
        letzteAenderungen="2002-04-06">
    <kursname>Fein Kochen</kursname>
    ...
    <kueche>Asiatisch</kueche>
  </kurs>
</kursliste>
```

15.5.2006

XML Vorlesung FHA SS 2006

11

Kontrolle Type Substitution

- Kontrolle der Ersetzung von Typen
 - Attribut `block` in `complexType`-Deklaration:
 - `#all`: Typ darf in Instanzen weder durch erweiterten Typ noch durch eingeschränkten Typ ersetzt werden
 - `restriction`: Typ darf nicht durch eingeschränkten Typ ersetzt werden
 - `extension`: Typ darf nicht durch erweiterten Typ ersetzt werden
 - Attribut `abstract` in `complexType`-Deklaration:
 - Typ darf nicht instanziiert werden

15.5.2006

XML Vorlesung FHA SS 2006

12

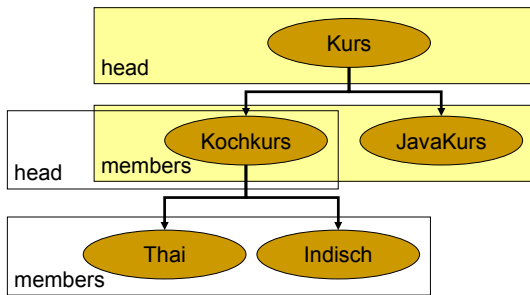
Was sind Substitution Groups?

- erlauben das Ersetzen von Elementen (!) durch andere Elemente in XML Schemas
- Beispiel: das Element Kurs könnte in einer Liste von Kursen ersetzt werden durch Elemente wie Kochkurs, JavaKurs etc
- machen zum Beispiel heterogene Aufzählungen möglich (Liste von Elementen unterschiedlicher Typen in einer Schema-Instanz)
- Substitution Groups machen solche Listen lesbarer

Was sind Substitution Groups?

```
<kursliste>
  <kurs letzteAenderungen="2002-03-14">
    <kursname>XML Schema</kursname>
    ...
  </kurs>
  <kochkurs letzteAenderungen="2002-03-14">
    <kursname>Fein Kochen</kursname>
    ...
    <kueche>Asiatisch</kueche>
  </kochkurs>
  <javakurs letzteAenderungen="2002-03-14">
    <kursname>Java Einführung</kursname>
    ...
  </javakurs>
  ...
</kursliste>
```

Substitution Group Hierarchien



Substitution Group Hierarchien

- eine Substitution Group besteht aus einem *Head* und einem oder mehreren *Members*
- überall wo das *Head* Element in einem Content Model referenziert wird, kann eines der entsprechenden *Member* Elemente verwendet werden
- Elemente können nur zu einer Substitution Group gehören
- eine Substitution Group Hierarchie kann aber mehrere Stufen haben

Substitution Group Deklaration

- Beispiel:

```
<xs:element name="kochkurs" substitutionGroup="kurs">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="kursTyp">
        <xs:sequence>
          <xs:element name="kueche" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Substitution Group Deklaration

- Beispiel:

```
<xs:element name="kursliste">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="kurs" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Substitution Group Deklaration

- Beispiel:

```
<kursliste>
  <kurs letzteAenderungen="2002-03-14">
    <kursname>XML Schema</kursname>
    ...
  </kurs>
  <kochkurs letzteAenderungen="2002-03-14">
    <kursname>Fein Kochen</kursname>
    <anmeldeTermin>2002-06-12</anmeldeTermin>
    <referenten>
      <name>Bossi</name>
      <vorname>Betti</vorname>
    </referenten>
    ...
    <kueche>Asiatisch</kueche>
  </kochkurs>
</kursliste>
```

15.5.2006

XML Vorlesung FHA SS 2006

19

Substitution Group Deklaration

- der *Head* einer Substitution Group wird nicht speziell gekennzeichnet, ausser dass der *Head* als globales Element deklariert werden muss
 - sonst ist er nicht über einen globalen Namen referenzierbar
- Members* haben ein Attribut `substitutionGroup` in der Element-Deklaration, welches das entsprechende *Head*-Element referenziert
 - Members* müssen ebenfalls global deklariert sein
- falls das *Member*-Element ohne Typ deklariert wird, hat es automatisch den gleichen Typ wie das *Head* Element
- Member*-Elemente müssen immer entweder den gleichen Typ wie das *Head*-Element haben, oder aber einen davon abgeleiteten Typ

15.5.2006

XML Vorlesung FHA SS 2006

20

Substitution Group Deklaration

- Kontrolle über die Verwendung von Substitution Groups in Elementen mit `final`, `block` und `abstract` möglich
 - Attribut `final`: Element darf nicht als Head einer Substitution Group verwendet werden
 - Attribut `block`: verhindert die Bildung von Instanzen, die Substitution Groups verwendet (erlaubt aber Deklaration)
 - Attribut `abstract`: Head Element dient nur als Platzhalter, *Members* müssen deklariert und instanziiert werden

15.5.2006

XML Vorlesung FHA SS 2006

21

Substitution Group Alternativen

- "heterogene Listen" können auch mit `choice` Groups und *Type Substitution* modelliert werden
- `choice` Group:
 - Liste der möglichen Elemente wird in `choice` Group deklariert
 - feste Verdrahtung dieser Aufzählung in der entsprechenden Gruppe (vgl. *Substitution Group*: Verdrahtung über Referenz, flexibler, Kombination von Schemas)
- Type Substitution:
 - Verwenden des gleichen Element-Namens (z.B. *Kurs*)
 - Element-Typen als *Derived Types* deklarieren
 - Verwendung des Attributes `xsi:type` im Element-Tag
 - Vorteil: leichtere Verarbeitung (Typ ist direkt benannt)
 - ist aber nach Validierung auch über das Infoset verfügbar

15.5.2006

XML Vorlesung FHA SS 2006

22

Type Derivation vs. Reusable Groups

- Type Derivation, wenn
 - der wiederverwendbare Teil des Content Models am Anfang des Content Models steht (Subtypen bilden)
 - der komplexe Datentyp ähnliche Objekte beschreibt (Person, Teilnehmer, Referent etc)
- Reusable Groups, wenn
 - der wiederverwendbare Teil in verschiedenen komplexen Datentypen vorkommt, die typmässig nichts miteinander zu tun haben (Personen, Kurse, Autos, etc.)

15.5.2006

XML Vorlesung FHA SS 2006

23

Modellierungstechniken

- DTD sind "einfacher"
 - keine Typenkonzept, also kein Typenmodell
 - oft Typenmodellierung ausserhalb, dann Mapping auf DTD Mechanismen
- XML Schema unterstützt Typenmodellierung
 - aber muss man es deshalb immer so machen?
 - Varianten der Verwendung von XML Schema
 - Typen ausserhalb modellieren
 - XML Schema zur Typmodellierung verwenden
 - XML Schema braucht Planung vor der Verwendung

15.5.2006

XML Vorlesung FHA SS 2006

24

Russian Doll Design

- Elementdeklarationen ineinander schachteln
 - vermeidet globale Typdefinitionen
- Vorteile des Russian Doll Designs
 - Struktur sehr offensichtlich
 - Vermeidung Doppelspurigkeit Typen/Elemente
- Nachteile des Russian Doll Designs
 - u.U. sehr tiefe Schachtelungstiefe der Elemente
 - keine Wiederverwendung von Deklarationen
 - Rekursionen brechen das Design-Modell
 - Wiederverwendungen brechen das Design-Modell

15.5.2006

XML Vorlesung FHA SS 2006

25

Venetian Blinds Design

- prinzipiell globale Typdeklarationen
 - Verwendung der Typen per Referenz
- Vorteile des Venetian Blinds Designs
 - Rekursion wird einfach ermöglicht
 - generell: Wiederverwendung von Typen ist einfach
 - zu jedem Element und Attribut gibt es einen Named Type
- Nachteile des Venetian Blinds Designs
 - grosse Menge an (globalen) Typen
 - XML Schema Code wird noch grösser und unübersichtlicher

15.5.2006

XML Vorlesung FHA SS 2006

26

Praxisorientierte Designmodelle

- in der Praxis oft kombinierte Modelle
 - Russian Doll für restriktive Strukturen
 - Struktur der Instanz stark durch das Schema vorgegeben
 - Venetian Blinds für permissive Strukturen
 - Struktur der Instanz kann sehr stark schwanken
- viele Schemas sind eine Kombination
 - z.B. in HTML: Tabellenstruktur vs. Tabelleninhalt
- Modellierung nicht immer in XML Schema
 1. Abbildungsvorschriften nach XML Schema
 2. Modellierung im bevorzugten Tool/Formalismus
 3. XML Schema generieren aus externem Modell

15.5.2006

XML Vorlesung FHA SS 2006

27

Namensgebung in Schemas

- Namen gehorchen den XML-Regeln
 - beginnen mit Buchstaben oder _
 - enthalten Buchstaben, Ziffern und _ : - .
 - immer NCNames verwenden (keine Doppelpunkte)
- Trennungsregeln definieren und befolgen
 - mit Separatoren trennen: _ - .
 - CamelCase benutzen
 - erster Buchstabe GrossBuchstabe oder kleinBuchstabe?
- Typnamen sollten erkennbar sein
 - Typ(e) Anhang oder andere Schreibweise

15.5.2006

XML Vorlesung FHA SS 2006

28

Namespaces und Schemas

- XML Schema unterstützt Namespaces
 - Namespace URI wird im Schema angegeben
 - definiert auf diese Weise einen Namespace
- kein Namespace notwendig
 - bei kleineren Experimenten und im Entwicklungsstadium
 - bei rein lokalen Verwendungen
- Namespace sollte verwendet werden
 - verteilte Verwendung eines Schemas
 - Erweiterung oder Versionierung eines Schemas
- Tip: RDDL hinter der Namespace URI ablegen

15.5.2006

XML Vorlesung FHA SS 2006

29

Qualified/Unqualified Names

- global Names in Instanzen müssen immer qualified Names verwenden
 - natürlich nur, wenn es einen Namespace gibt
- Kontrolle bei local Names mit zwei Attributen
 - `elementFormDefault` für Elemente
 - Default-Wert ist `unqualified`
 - `attributeFormDefault` für Attribute
 - Default-Wert ist `unqualified`
- `unqualified` bei Elementen kann irritieren
 - Instanzen müssen wissen, ob ein Element-Typ lokal oder global deklariert ist

15.5.2006

XML Vorlesung FHA SS 2006

30

Unqualified Names + Namespaces

- Unqualified Names sind in keinem Namespace
 - definiert durch `elementFormDefault`
- oft gibt es Default Namespaces
 - praktisch für kompaktere Dokumente
 - `xmlns=""` im Document Element
 - wird auf alle unqualified Names angewendet
 - kann zu Problemen führen
- gute Planung unerlässlich
 - qualified Names für Elemente empfehlenswert
 - wenige Gründe für den Default-Wert (unqualified)

15.5.2006

XML Vorlesung FHA SS 2006

31

Qualified Names für Attribute

- Attribute gehören meistens zu Elementen
 - damit sind unqualified Names sinnvoll
 - der XML Schema Default kann belassen werden
- globale Attribute sollten qualified Names haben
 - das haben sie in jedem Fall
 - `attributeFormDefault` ist nur für local Names
 - Beispiele finden sich an verschiedenen Stellen
 - `xml:lang` ist ein solches Attribut aus XML
 - `xsi:type` ein weiteres aus XML Schema
- der Default ist in fast allen Fällen sinnvoll

15.5.2006

XML Vorlesung FHA SS 2006

32

Sprachabhängige Instanzen

- nicht direkt unterstützt von XML Schema
- Substitution Groups können verwendet werden
 - Abbildung von Elementnamen auf andere Sprachen
 - `<x:element name="nummer" substitutionGroup="number"/>`
- Nachteile dieser Methode
 - funktioniert nur für Elemente
 - keine Substitution Groups für Attribute
 - alle Elemente müssen global deklariert sein
- u.U. besserer Ansatz: XSLT
 - Transformationen in beide Richtungen definieren

15.5.2006

XML Vorlesung FHA SS 2006

33

Modularisierung von Schemas

- XML Schema unterscheidet 3 Mechanismen
- `include` ist wie textuelles Einbinden
 - muss global angegeben werden
 - bezieht sich immer auf den gleichen Namespace
- `redefine` erweitert die `include` Methode
 - macht alles, was `include` macht
 - ermöglicht die Neudefinition von Komponenten
 - nur möglich für Complex Types, Simple Types, Named Model Groups, Attribute Groups
- `import` verwendet andere Namespaces
 - Referenzen auf Namen aus anderen Namespaces

15.5.2006

XML Vorlesung FHA SS 2006

34

Erweiterungsmöglichkeiten

- Erweiterungen wenn möglich evolutionär
 - weniger Kompatibilitätsprobleme
- XML Schema bietet diverse Mechanismen
 - Wildcards (verfügbar für Elemente und Attribute)
 - Type Derivation
 - Substitution Groups
 - Type Redefinition
 - verlangt Kontrolle über das zu erweiternde Schema
 - Reusable Group Redefinition

15.5.2006

XML Vorlesung FHA SS 2006

35

Versionierung von Schemas

- Versionierung oft notwendig
 - Spezialfall der Erweiterung von Schemas
 - die gleichen Mechanismen sind verwendbar
- Frage der Kompatibilität zwischen Versionen
 - minor Versions müssen voll kompatibel sein
 - major Versions dürfen inkompatibel sein
 - u.U. Fallback-Mechanismen vorsehen
 - Ziel sollte Kompatibilität in beide Richtungen sein
- Kompatibilität über Erweiterungen
 - alte Versionen müssen Untermengen sein

15.5.2006

XML Vorlesung FHA SS 2006

36

Zusammenfassung

- Reusable Groups:
 - Named Model Groups
 - Attribute Groups
- Substitution Groups
 - erlauben heterogene Aufzählungen
 - Head und Member Elemente deklarieren
 - ähnliche Konzepte:
 - Choice Group
 - Type Substitution