Universität Rostock

Traditio et Innovatio

Management von Typhierarchien in der XML-Schemaevolution

Chris Kaping

Organisatorisches

- **Management von Typhierarchien in der XML-Schemaevolution**
- Masterarbeit, Abgabedatum: 15.10.2014
- Erstgutachter: Dr.-Ing. habil. Meike Klettke
- Zweitgutachter: Prof. Dr.-Ing. habil. Peter Forbrig
- Betreuer: Dipl.-Inf. Thomas Nösinger

Inhalt

Einleitung

- Problemstellung
- Schemaevolution, CodeX
- Thematische Grundlagen

Management von Typhierarchien

- Werkzeuge für die XML-Schema-Bearbeitung
- Visualisierung von Typen
- Manipulationen von Typen

Fazit

Problemstellung

Typhierarchien in CodeX visualisieren und Manipulationen an diesen kompensieren

- Recherche über das Typsystem von XML-Schema
- Vergleich von Editoren
- Ableiten eines eigenen Konzepts
- Implementierung in CodeX
- Evaluierung

CodeX - (Conceptual Design and Evolution for XML-Schema)

XML-Schemaevolution

- Anpassen eines XML-Schemas an veränderte Anforderungen
- Propagieren der Modifikationen an die Instanzebene

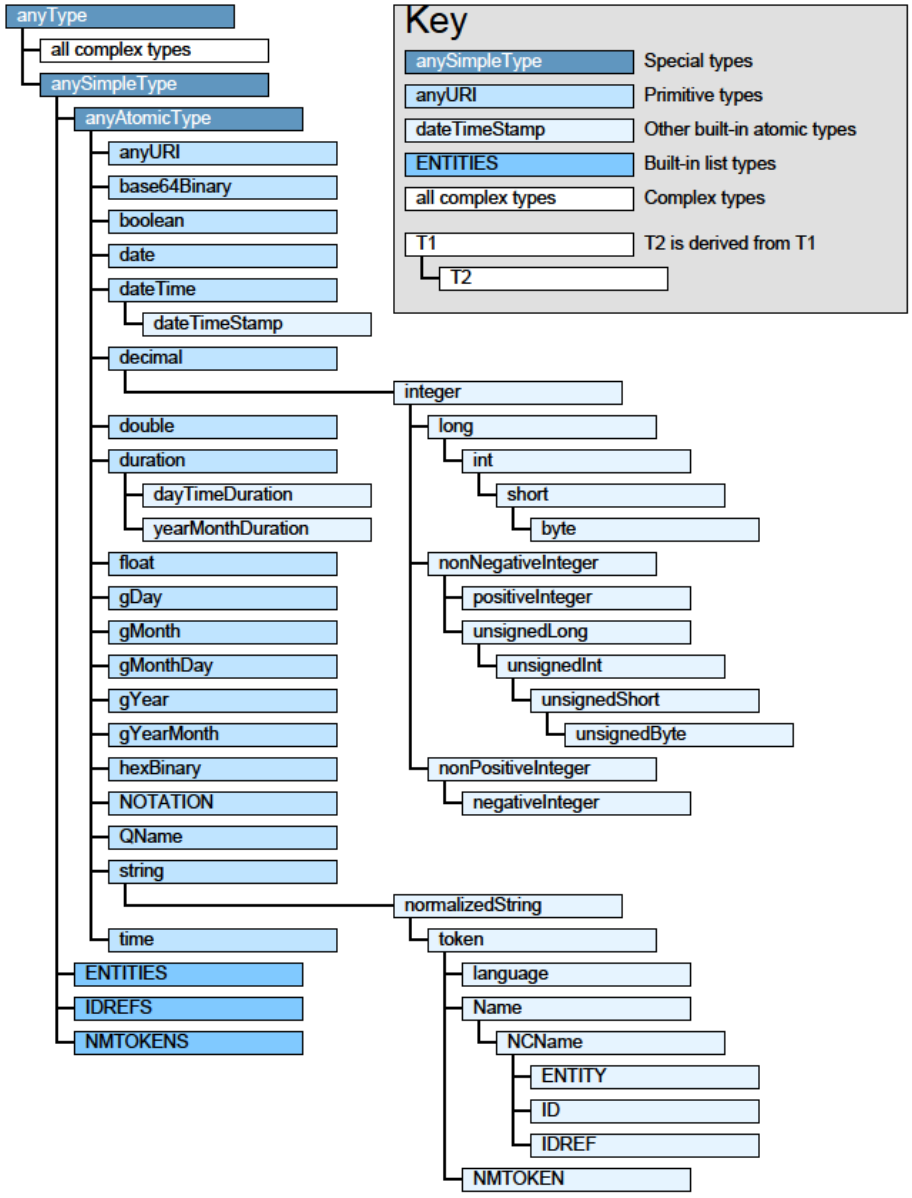
CodeX

- Editor (Prototyp) für die konzeptionelle Evolution von XML-Schemata
- Benutzt das interne Modell EMX (Entity Model for XML-Schema)
  - Abbildung von XML-Schemakomponenten auf eindeutige EMX-Entitäten
- Entwurfsschritte am EMX-Modell in Logdatei per ELaX-Statements (Evolution Language for XML-Schema) protokolliert und auf XML-Instanzen angewendet

Typsystem von XML-Schema

Einfache Typen:

- Zusammenfassung von Werten zu Wertebereich (Datentyp)
- Vordefinierte built-in Datentypen
- Ableitung nutzerdefinierter Datentypen



Typsystem von XML-Schema

Ableitung nutzerdefinierter Datentypen

- Einschränkung <xs:restriction/>
- Listenbildung <xs:list/>
- Vereinigung <xs:union/>

```
<xs:simpleType name="europaLandType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Deutschland"/>
  <xs:enumeration value="Schweiz"/>
 </xs:restriction>
</xs:simpleType>

<xs:simpleType name="amerikaLandType">
 […]
</xs:simpleType>

<xs:simpleType name="landType">
  <xs:union memberTypes="europaLandType amerikaLandType"/>
</xs:simpleType>

<xs:simpleType name="laenderType">
 <xs:list itemType="landType">
</xs:simpleType>
```

Typsystem von XML-Schema

Komplexe Typen:

- Strukturbeschreibung für Elementen
- keine vordefinierten, alle implizit oder explizit abgeleitet von xs:anyType
- Ableitung nutzerdefinierter Datentypen
  - Einschränkung <xs:restriction/>
  - Erweiterung <xs:extension/>

```
<xs:complexType name="studentType">
 <xs:complexContent>
  <xs:extension base="personType">
   <xs:sequence>
    <xs:element name="hobbies" type="hobbiesType"/>
    <xs:element name="gehalt" type="gehaltType"/>
   </xs:sequence>
   <xs:attribute name="matrikelnummer" type="xs:ID"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="USAEliteStudentType">
 <xs:complexContent>
  <xs:restriction base="studentType">
   […]
   <xs:element name="land" type="amerikaLandType"
    fixed="USA"/>
   […]
</xs:complexType>
```

Typsystem von XML-Schema

Zusammenfassung

Ableitung	simpleType	complexType	
		simpleContent	complexContent
Restriction	Facetten	Facetten Attribute	Inhaltsmodell Attribute
Extension	-	Attribute	Inhaltsmodell Attribute
List	Liste von Typwerten	-	-
Union	Vereinigung von Typwerten	-	-

Fragestellungen

- Personenschema
  - 8 komplexe, 7 einfache Typen
  - Nur grundlegende Eigenschaften modelliert (Elemente: Name, Vorname, …; Attribute: ID, …)
  - → 155 Zeilen
- Ausschweifende XML-Syntax, Orientierung schwierig bei komplexen Schemata
- Die eigentliche Modellierung rückt in den Hintergrund

…

➢ **1)** Existieren Möglichkeiten, die Aussagen und Beziehungen von Typen einfacher darzustellen?

➢ **2)** Was passiert wenn man Typen manipuliert?

→ Management von Typen und Typhierarchien

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="person" type="personType"/>
 <xs:element name="land" type="landType"/>
 <xs:element name="publikation">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="art" type="xs:string"/>
    <xs:element name="titel" type="xs:string"/>
    <xs:element name="jahr" type="xs:gYear"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:attribute name="id" type="xs:positiveInteger"/>

 <xs:complexType name="personType">
  <xs:sequence>
   <xs:element name="vorname" type="xs:string"/>
   <xs:element name="nachname" type="xs:string"/>
   <xs:element name="geburtsdatum" type="xs:date"/>
   <xs:element ref="land"/>
  </xs:sequence>
  <xs:attribute ref="id"/>
 </xs:complexType>

 <xs:complexType name="professorType">
  <xs:complexContent>
   <xs:extension base="personType">
    <xs:sequence>
     <xs:element name="hobbies" type="professorHobbiesType"/>
     <xs:element name="gehalt" type="gehaltType"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="europaProfessorType">
  <xs:complexContent>
   <xs:restriction base="professorType">
    <xs:sequence>
     <xs:sequence>
      <xs:element name="vorname" type="xs:string"/>
      <xs:element name="nachname" type="xs:string"/>
      <xs:element name="geburtsdatum" type="xs:date"/>
      <xs:element name="land" type="europaLandType"/>
     </xs:sequence>
     <xs:sequence>
      <xs:element name="hobbies" type="professorHobbiesType"/>
      <xs:element name="gehalt" type="gehaltEuroType"/>
     </xs:sequence>
    </xs:sequence>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="studentType">
  <xs:complexContent>
   <xs:extension base="personType">
    <xs:sequence>
     <xs:element name="hobbies" type="hobbiesType" minOccurs="0" maxO
     <xs:element name="gehalt" type="gehaltType" minOccurs="0" maxOcc
    </xs:sequence>
    <xs:attribute name="matrikelnummer" type="xs:ID"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 …
```

1) Existieren Möglichkeiten, die Aussagen und Beziehungen von Typen einfacher darzustellen?

… Verschiedene Werkzeuge als Erleichterung für die Bearbeitung von XML-Schemata und –Dokumenten durch grafische Visualisierung

Dokumenteditoren:

- Altova XMLSpy 2014
- Visual Studio 2013 XML-Schema-Designer
- Oracle JDeveloper 12c
- Eclipse WTP 3.5.2

Konzeptionelle Modellierung

- hyperModel 3.6 (UML)
- CodeX (EMX)

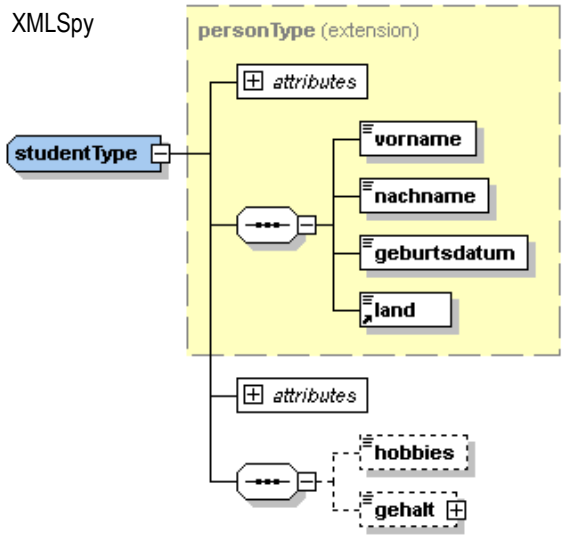
Element-zentrische Editoren

- Elemente als Diagrammbausteine, Darstellung von Typen an diese geknüpft
- Leichte Navigation entlang den Referenzbeziehungen, gute Editierbarkeit

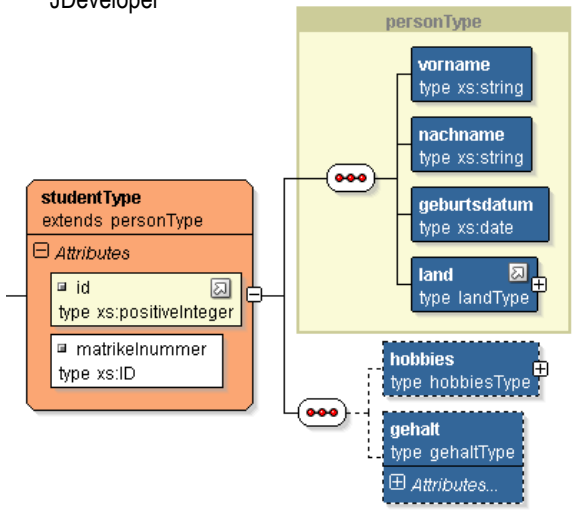
Probleme

- geringe Abstraktion von der XML-Syntax
- Navigation entlang der Typhierarchien eingeschränkt

XMLSpy



JDeveloper



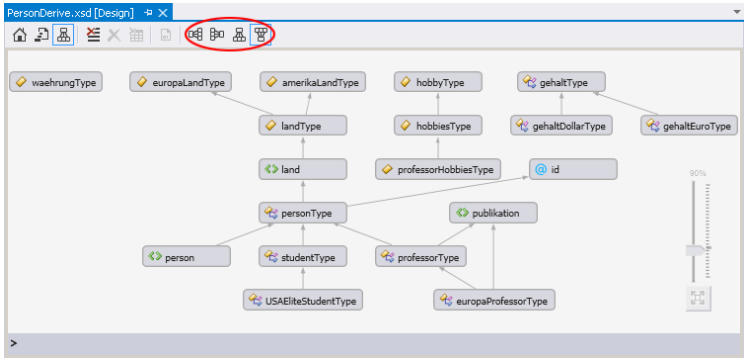
Typ-zentrische Editoren

- Typen als grundlegende Diagrammbausteine, Elemente und Attribute als Eigenschaften
- Höhere Abstraktion, Typhierarchien im Vordergrund

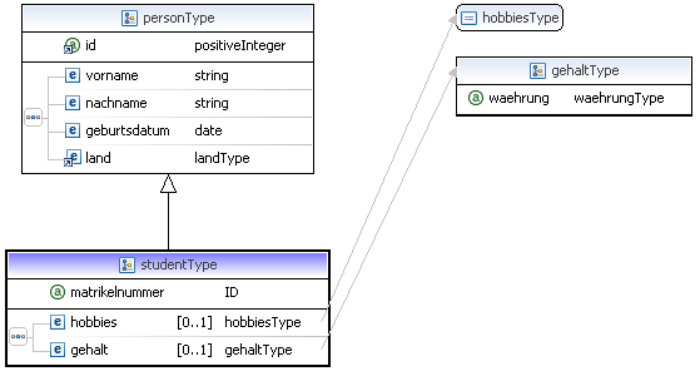
Probleme

- fehlende Eigenschaften, eingeschränkte Editierbarkeit
- Änderungen an der Definition nicht (vollständig) erkennbar
- fehlende Visualisierung von einfachen Typen, Facetten in Seitenleisten ausgelagert

Visual Studio



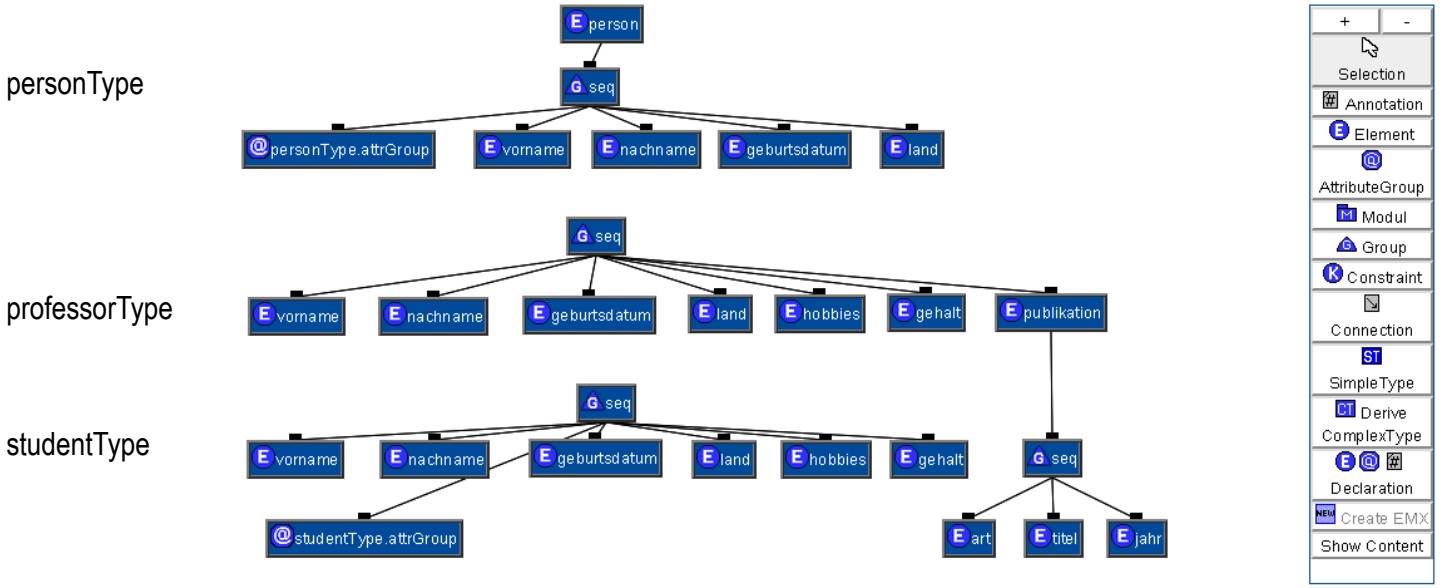
Eclipse WTP



Visualisierung von Typen in CodeX

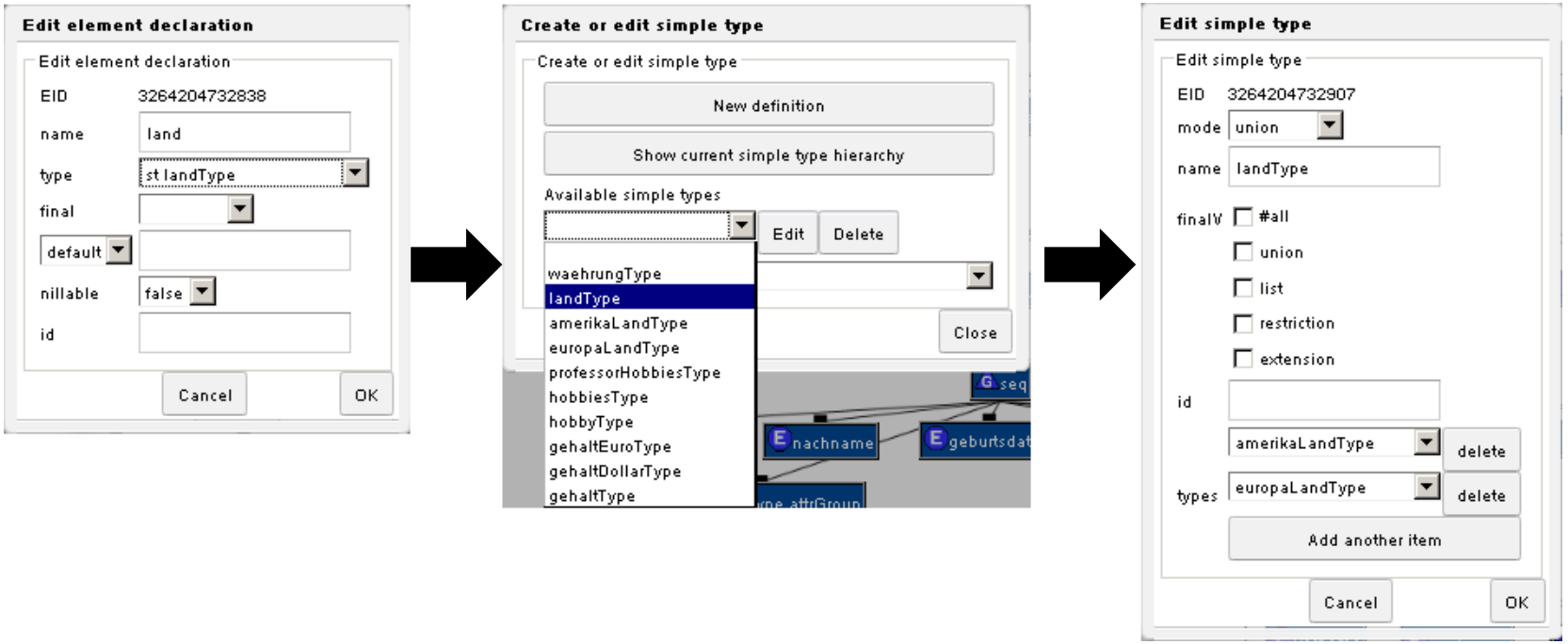
- Bisher im Fokus: Deklarationen und die Referenzbeziehungen, Element-zentrisch
- Syntaxdetails ausgelagert in Dialoge, resultierende Dokumentstrukturen erkennbar

→ sehr gute Editierbarkeit, aber nicht optimal für die Darstellung von Typhierarchien (Beispiel komplexe:)



Visualisierung von Typen in CodeX

- Einfache Typen?



- Jetzt nur noch jeweils gucken, was sich hinter den Mitgliedstypen verbirgt …

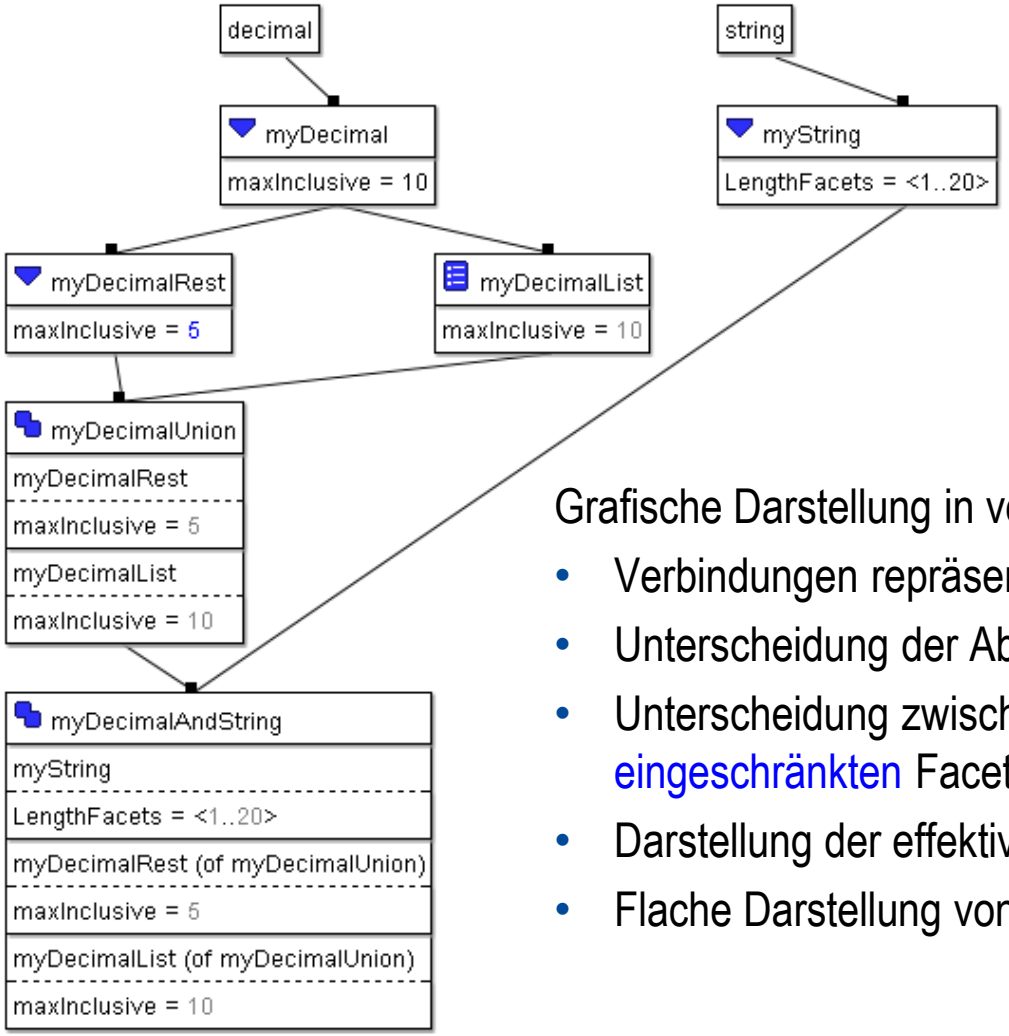
→ Navigation durch mehrere Dialogebenen erschweren das rasche Erfassen

Konsequenzen für die Visualisierung

Visualisierung der Hierarchie für je einfache und komplexe Typen als ergänzende **Sichten**

- Loslösen von der Elementzentrierung und der Dokumentstruktur
- Fokus auf das Wesentliche
- Anzeigen des Wertebereichs einfacher Typen
- Anzeigen des Inhaltsmodells komplexer Typen
- Signalisieren von Veränderungen an Typkomponenten entlang eines Hierarchiepfades

Visualisierung von einfachen Typen



Grafische Darstellung in verbundenen Diagrammen

- Verbindungen repräsentieren Beziehungen zwischen Typen
- Unterscheidung der Ableitungsart durch einfache Icons
- Unterscheidung zwischen vererbten, **neu eingeführten** und eingeschränkten Facetten in Einschränkungen
- Darstellung der effektiv wirksamen Facetten
- Flache Darstellung von Mitgliedern in Vereinigungen

Visualisierung von komplexen Typen ?

Verbindungen zwischen Diagrammen stellen Ableitungsbeziehungen dar.

- Ableitungen von komplexen Typen sind im Modell von CodeX zwar vorhanden, aber (noch) nicht vollständig implementiert
- Ein Vergleich von Eigenschaften zweier komplexer Typen führt zu einem Rückschluss über die vorliegende Beziehung

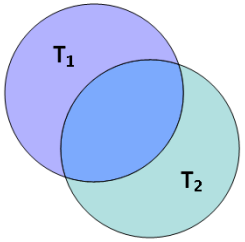
```
<xs:complexType name="abc">
 <xs:sequence>
  <xs:element name="a" type="xs:string" maxOccurs="10"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="c" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="abc1">
 <xs:sequence>
  <xs:element name="a" type="xs:string"
   maxOccurs="3"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="c" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="abc2">
 <xs:complexContent>
  <xs:restriction base="abc">
   <xs:sequence>
    <xs:element name="a" type="xs:string"
     maxOccurs="3"/>
    <xs:element name="b" type="xs:string"/>
    <xs:element name="c" type="xs:string"/>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

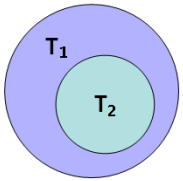
Bestimmung der Beziehung von komplexen Typen

$T_1 \neq T_2 \rightarrow PT$

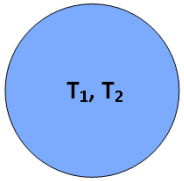


- Je nachdem, welche Element- und Attributnamen wie häufig auftreten, können Ober- bzw. Untermengen entstehen
- Nutzen eines Entscheidungssystems mit 5 Ergebnissen
- Verschiedene Aspekte von Kompositoren, Elementen und Attributen werden getrennt voneinander analysiert und bewertet
- Zusammenführung der Einzelentscheidungen zu einem der 5 Ergebnisse

$T_1 = OT, T_2 = UT$



$T_1 = T_2 \rightarrow ST$



Entscheidung	PT	ST	UT	OT	ET
PT	PT	PT	PT	PT	PT
ST	PT	ST	UT	OT	ET
UT	PT	UT	UT	PT	PT
OT	PT	OT	PT	OT	PT
ET	PT	ET	PT	PT	ET

PT – Paralleltyp
ST – Gleicher Typ
UT – Untertyp
OT – Obertyp
ET – Erweiternder Typ

Bestimmung der Beziehung von komplexen Typen

Beispiel: Vergleich von Sequenzen, es wird immer T_2 gegen T_1 getestet

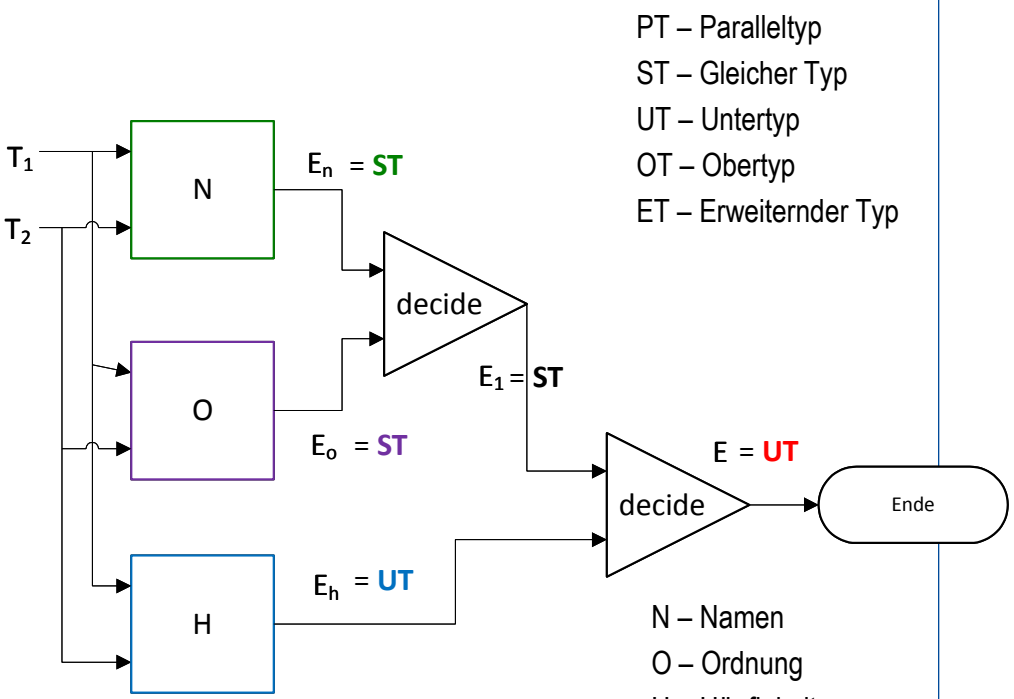
T_1:

```
<xs:complexType name="abc">
 <xs:sequence>
1<xs:element name="a" type="xs:string"
   maxOccurs="10"/>
2<xs:element name="b" type="xs:string"/>
3<xs:element name="c" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

T_2:

```
<xs:complexType name="abc1">
 <xs:sequence>
1<xs:element name="a" type="xs:string"
   maxOccurs="3"/>
2<xs:element name="b" type="xs:string"/>
3<xs:element name="c" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

→ T_2 Untertyp

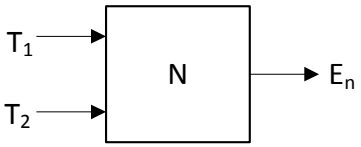


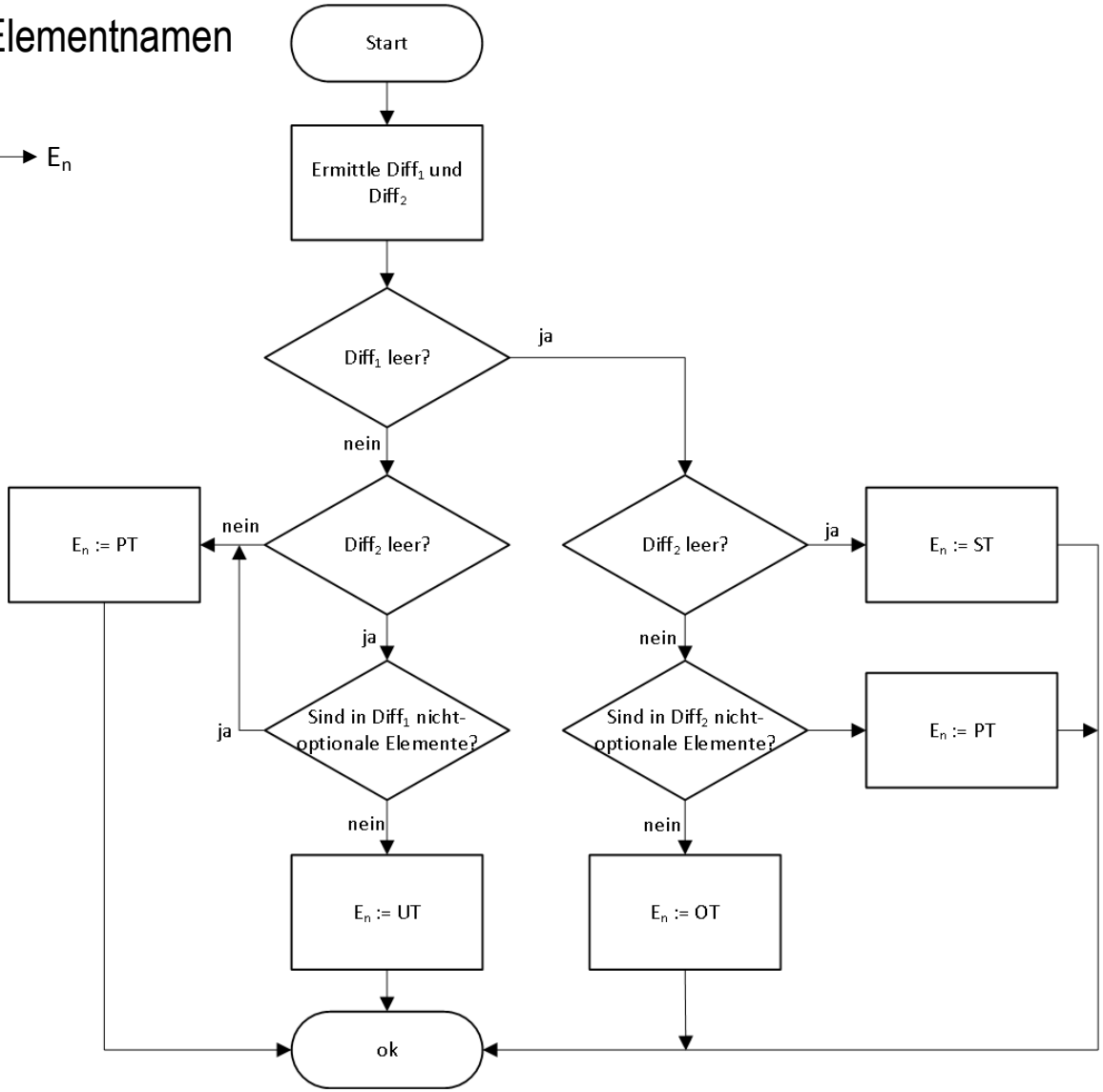
PT – Paralleltyp
ST – Gleicher Typ
UT – Untertyp
OT – Obertyp
ET – Erweiternder Typ

N – Namen
O – Ordnung
H – Häufigkeit
E – Entscheidung

Bestimmung der Beziehung von komplexen Typen

Vergleich der Elementnamen

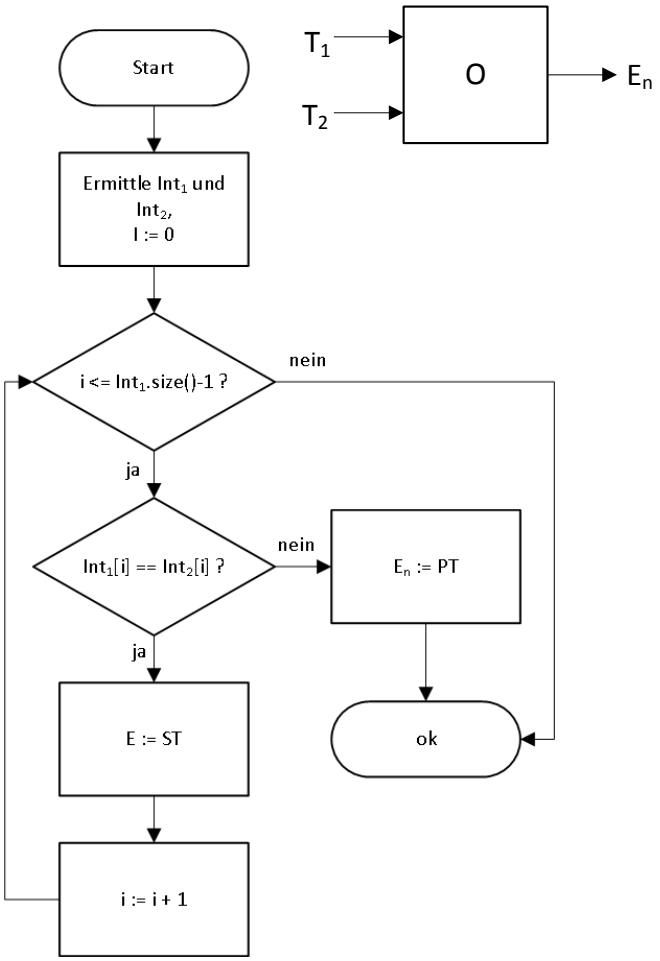




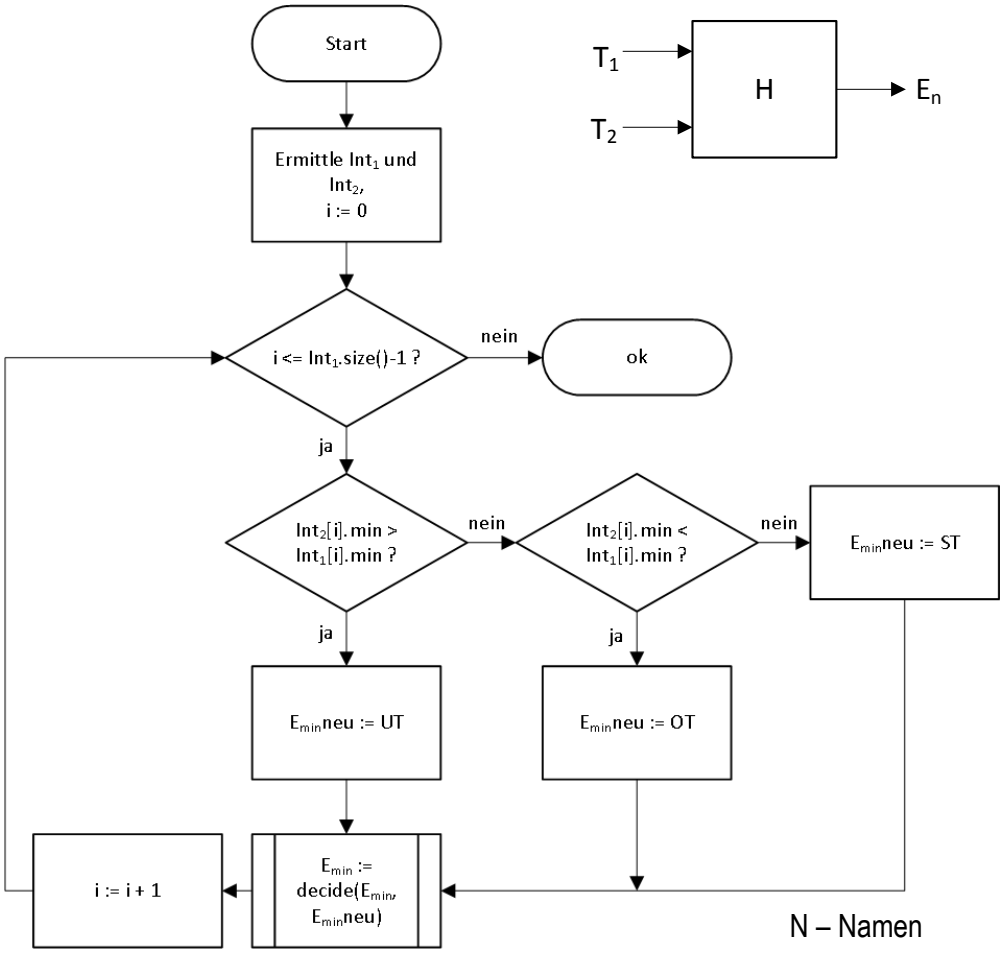
N – Namen
O – Ordnung
H – Häufigkeit
E – Entscheidung
Diff – Differenzmenge

Bestimmung der Beziehung von komplexen Typen

Vergleich der Ordnung



Vergleich der Häufigkeit (auch für Kompositoren)



Analog für maxOccurs

N – Namen
O – Ordnung
H – Häufigkeit
E – Entscheidung

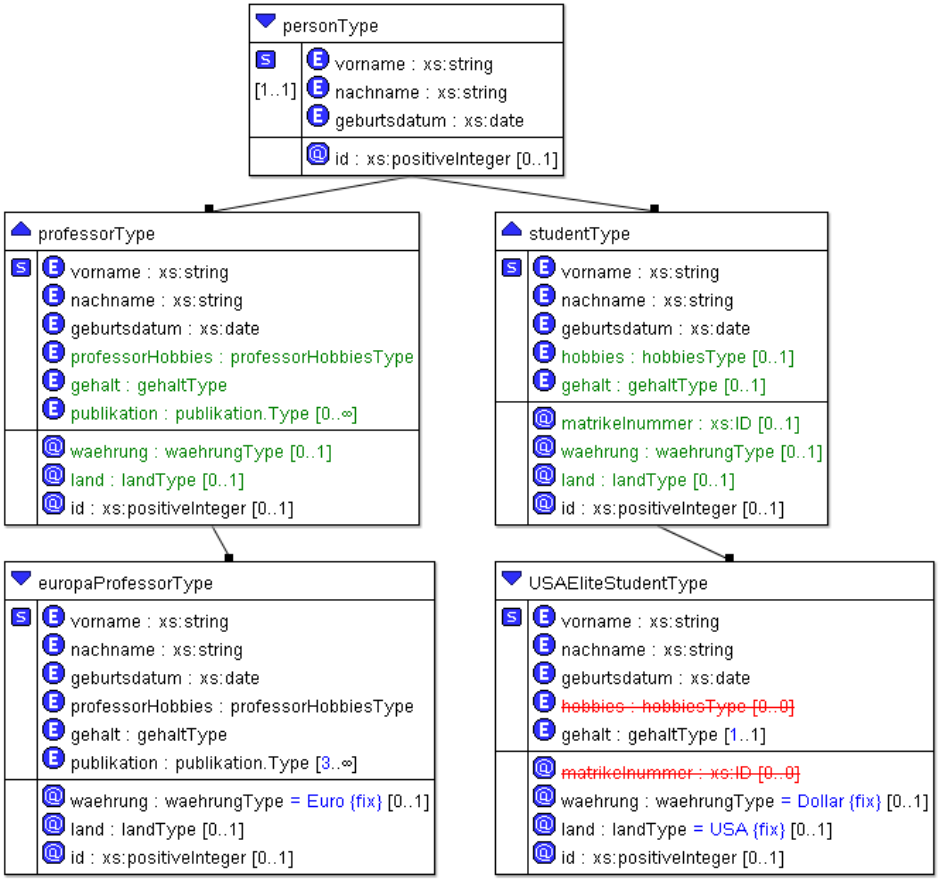
Visualisierung von komplexen Typen

restriction

extension

Grafische Darstellung in verbundenen Diagrammen

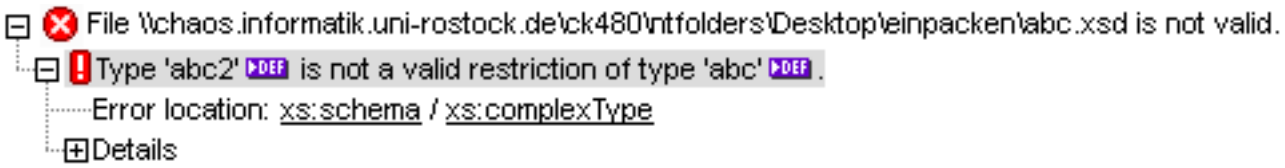
- Verbindungen repräsentieren **berechnete** Beziehungen zwischen Typen
- Hervorhebung von
  - neuen Bestandteilen
  - entfernten Bestandteilen (auch implizit)
  - Einschränkungen durch Änderung der Häufigkeit und Wertefixierung



2) Was passiert, wenn man Typen manipuliert?

Verletzung der Gültigkeit

- Fehler durch Angaben, die nicht konform zum Datenmodell sind …
- Strukturen mit vielen Abhängigkeiten wie Typhierarchien anfällig hierfür



- Signalisierung in den untersuchten Werkzeugen oft erst beim Auslösen einer Validierung
- Fehlerursache und –ort werden zwar angezeigt, der Nutzer muss diese aber nachträglich beseitigen

→ Operationen in CodeX vor Umsetzung auf Fehler überprüfbar

Manipulation von Typen

Informationsverlust

```
<xs:simpleType name="hobby">
  [...]
</xs:simpleType>
```

```
<xs:simpleType name="hobby" final="list">
  [...]
</xs:simpleType>
```

```
<xs:simpleType name="hobbies">
 <xs:list itemType="hobby">
</xs:simpleType>
```

```
<xs:element name="hobbies"
type="hobbies"/>
```

? ?

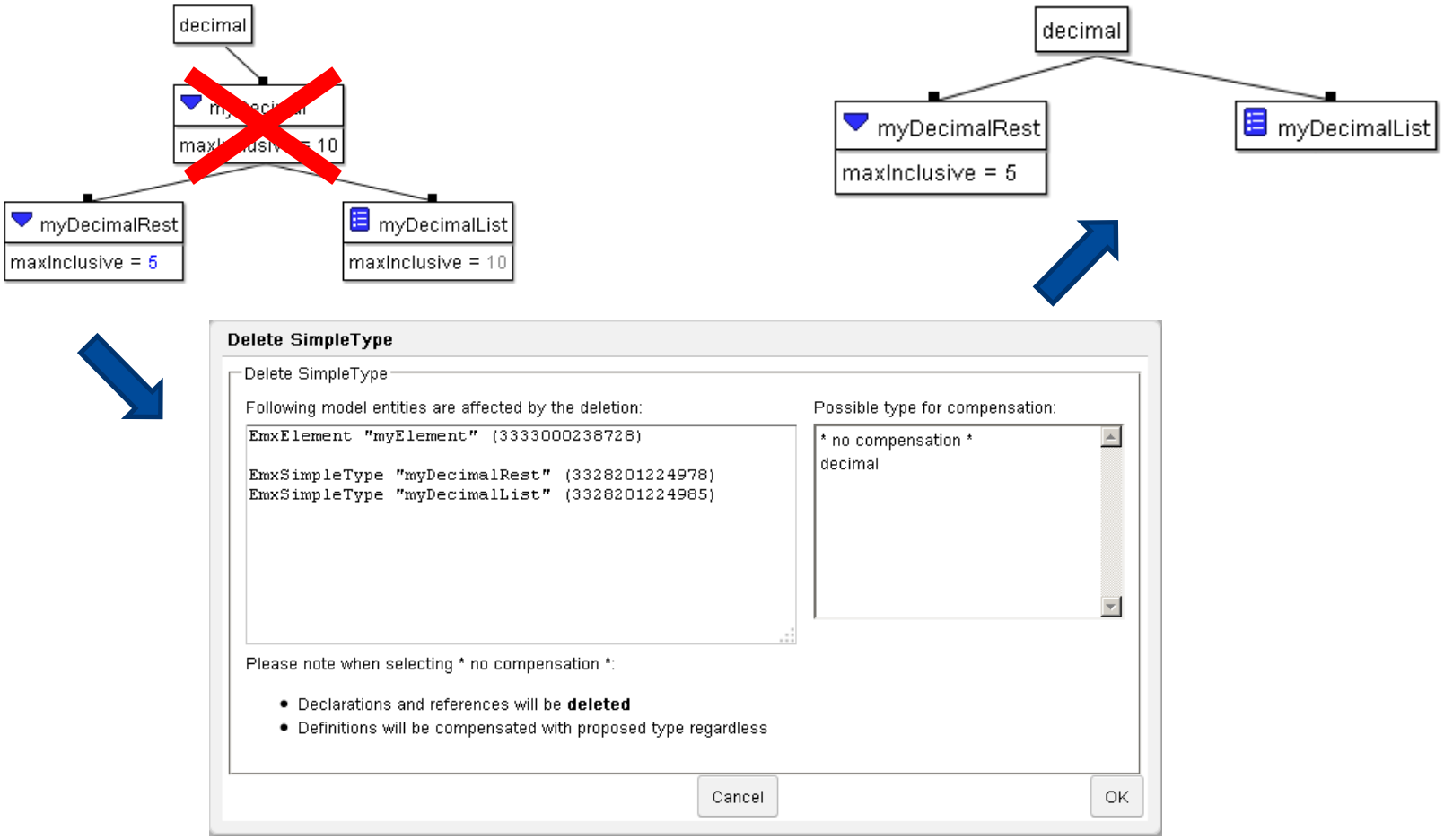
Löschen ? Und was ist mit den XML-Dokumenten?

- Abhängigkeiten in den Typhierarchien erkennen (Untertypen und referenzierende Deklarationen)
- Eine **Kompensation** anbieten, wenn Abhängigkeiten durch Operationen zu Ungültigkeiten führen würden

→ Gültigkeitsverletzungen und Informationsverlust vermeiden

Manipulation von einfachen Typen

Kompensation nach dem Löschen



Manipulation von einfachen Typen

Auswirkung auf die Schemaevolution:

- Erzeugung von ELaX-Statements für die getätigten Änderungen am Modell

```
00:09:28 Project/noName.emx: delete simpletype name '3328201224971' ;
00:09:28 Project/noName.emx: update simpletype name '3328201224971' change mode 'restriction' of '3328200852370' remove 'maxInclusive' '10' fixed
'false' id '' at '3328201224972' ;
00:09:28 Project/noName.emx: update simpletype name '3328201224985' change mode 'list' of '3328200852370' ;
00:09:28 Project/noName.emx: update simpletype name '3328201224978' change mode 'restriction' of '3328200852370' ;
00:09:28 Project/noName.emx: update element name '3333000238728' change type '3328200852370' ;
```

Weitere Manipulationen: **Updates** von

- Ableitungsart
- Basistypen
- final-Attribut
- Facetten

Manipulation von komplexen Typen

- Keine Abhängigkeiten durch Untertypen
- Was bleibt, sind referenzierende **Elemente**

→ Updates und **Löschen** wirken isoliert auf die Typdefinitionen

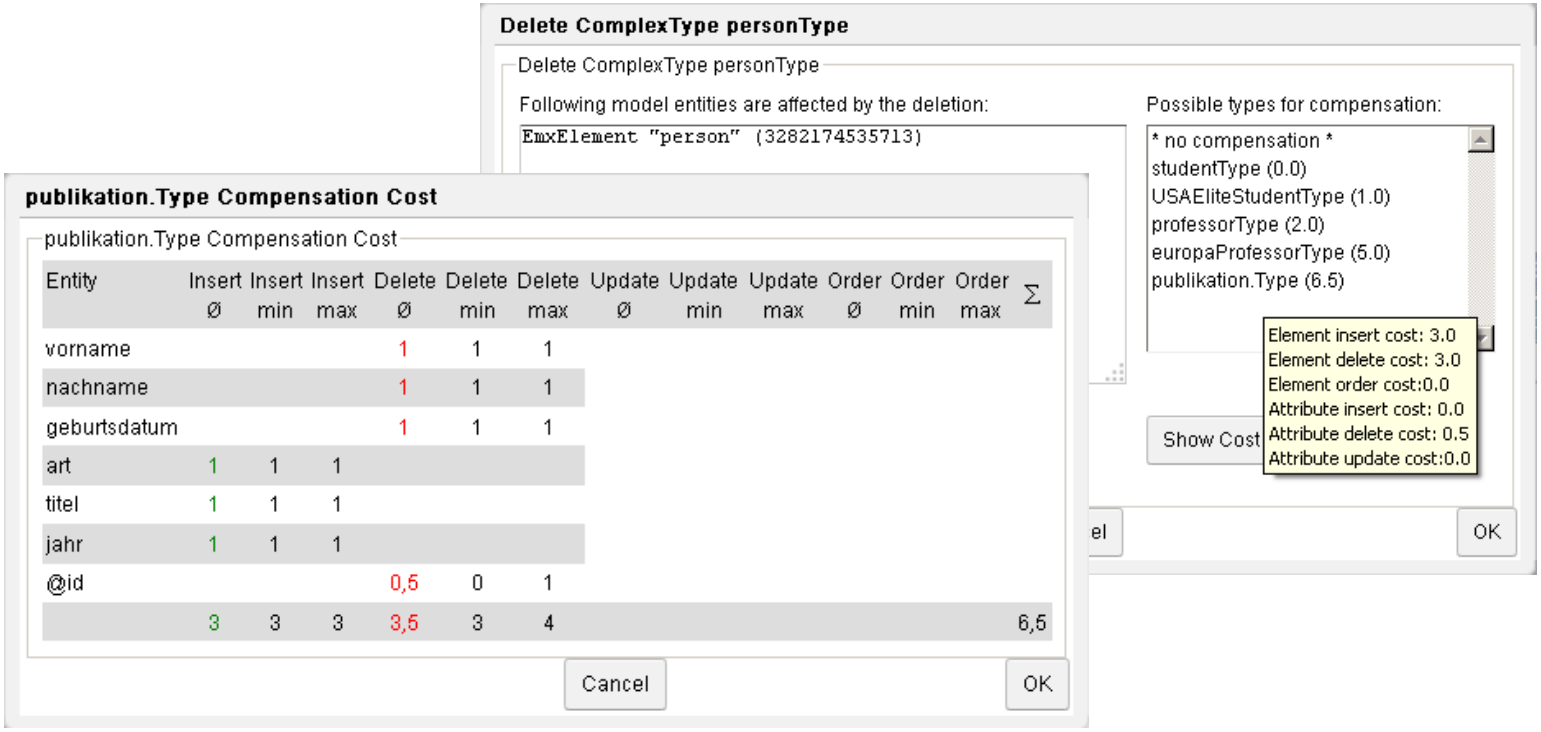
Problem: für die **Kompensation** steht kein Obertyp zur Verfügung

Lösung: Bestimmen der **Ähnlichkeit** des gelöschten Typen zu den restlichen

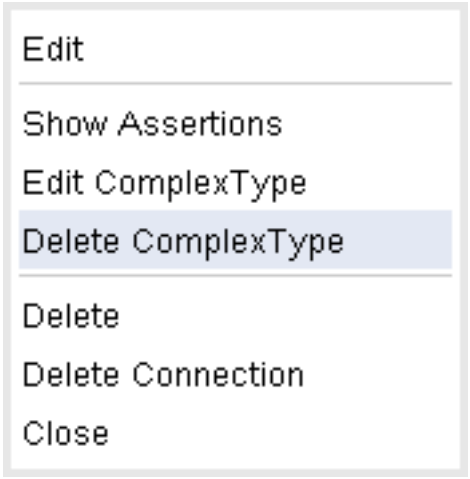
- Resultat einer Kostenschätzung
- Kosten = Aufwand / Operationen um Elemente in XML-Dokumenten an das neue Inhaltsmodell anzupassen
- Ähnliches Vorgehen wie beim Bestimmen der Beziehung
  - Statt qualitativem Ergebnis nun ein quantitatives
  - Angepasst an die beteiligten Kompositoren
  - Wiederholt für die minimale und maximale Ausprägung des gelöschten Typen

Löschen von komplexen Typen

- Für alle Typen im Schema, Vorschläge nach den berechneten Kosten sortiert
- Je geringer die (Lösch-)Kosten, desto weniger Informationsverlust ist zu erwarten



Live-Präsentation



Fazit

- Typen und Typhierarchien
- Sichtung von Werkzeugen für die XML-Schemabearbeitung
- Erweiterung von CodeX mit Typ-zentrischen Sichten
  - Typbeziehungen auch ohne Hierarchieinformationen
- Kompensation bei Manipulationen von Typhierarchien
  - Ähnlichkeit von komplexen Typen durch Kostenschätzung

Ausblick

- Verbindung beider Sichten, interaktive Diagramme
- Verfeinerung der Kompensation
  - einfache Typen: betrachten von anderen Typen
  - komplexe Typen: miteinbeziehen von Wildcards

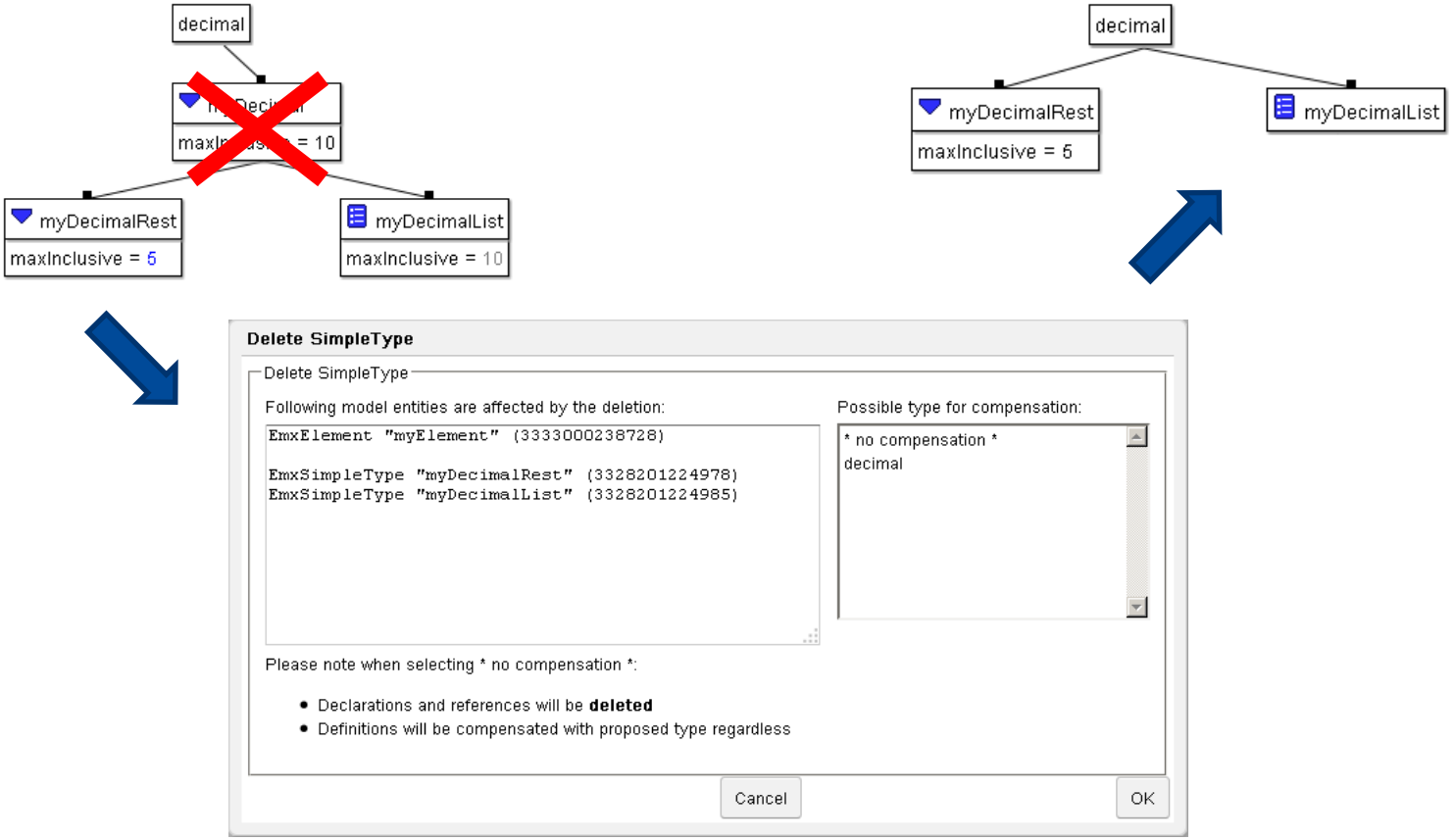
Umbenennen statt Löschen und Einfügen

Vielen Dank für Ihr Interesse!

Gibt es weitere Fragen?

Manipulation von einfachen Typen

Kompensation nach dem Löschen



00:09:28 Project/noName.emx: delete simpletype name '3328201224971' ;
00:09:28 Project/noName.emx: update simpletype name '3328201224971' change mode 'restriction' of '3328200852370' remove 'maxInclusive' '10' fixed 'false' id '' at '3328201224972' ;
00:09:28 Project/noName.emx: update simpletype name '3328201224985' change mode 'list' of '3328200852370' ;
00:09:28 Project/noName.emx: update simpletype name '3328201224978' change mode 'restriction' of '3328200852370' ;
00:09:28 Project/noName.emx: update element name '3333000238728' change type '3328200852370' ;

Löschen von komplexen Typen

- Einfügen, Löschen und Updates von Elementen und Attributen
- Je geringer die (Lösch-)Kosten, desto weniger Informationsverlust ist zu erwarten

Bestimmung der Kosten

- Eigentlich nötig: Analyse der Instanzbasis, in CodeX aber nicht verfügbar
- Stattdessen Nutzen von Schemainformationen (→ Schätzung)
- Ähnliches Vorgehen wie beim Bestimmen der Beziehung
  - Statt qualitativem Ergebnis nun ein quantitatives
  - Angepasst an die beteiligten Kompositoren
  - Wiederholt für die minimale und maximale Ausprägung des gelöschten Typen

Löschen von komplexen Typen

Bestimmung der Kosten

- Eigentlich nötig: Analyse der Instanzbasis, in CodeX aber nicht verfügbar
- Stattdessen Nutzen von Schemainformationen (→ Schätzung)
- Ähnliches Vorgehen wie beim Bestimmen der Beziehung
  - Statt qualitativem Ergebnis nun ein quantitatives
  - Angepasst an die beteiligten Kompositoren
  - Wiederholt für die minimale und maximale Ausprägung des gelöschten Typen

T_1:

```
<xs:complexType name="abc">
 <xs:sequence>
  <xs:element name="a" type="xs:string"
   maxOccurs="10"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="c" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

T_2:

```
<xs:complexType name="abc1">
 <xs:sequence>
  <xs:element name="a" type="xs:string"
   maxOccurs="3"/>
  <xs:element name="b" type="xs:string"/>
  <xs:element name="c" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

→ minimal keine Kosten, maximal 7 Löschkosten für Element "a"

Manipulation von einfachen Typen

Update

- Ableitungsart
  list → restriction
- Basistypen
  base="newType"
- final-Attribut
  final="#all"
- Facetten
  maxLength="2"
  minLength="3„

Löschen

Kompensation

- Umsetzen des Basistypen

Konzept für das Management von Typhierarchien

Operationen für das Bearbeiten von Typen (in einer Hierarchie):

- Einfügen
- Bearbeiten
- Löschen

Folgen und Risiken:

- Informationszuwachs
- Informationsverlust

} u. U. Anpassungen der XML-Instanzen

- Verletzung der Gültigkeit des XML-Schemas

Konzept für das Management von Typhierarchien

Oberstes Ziel: Informationsverlust und Gültigkeitsverletzung vermeiden

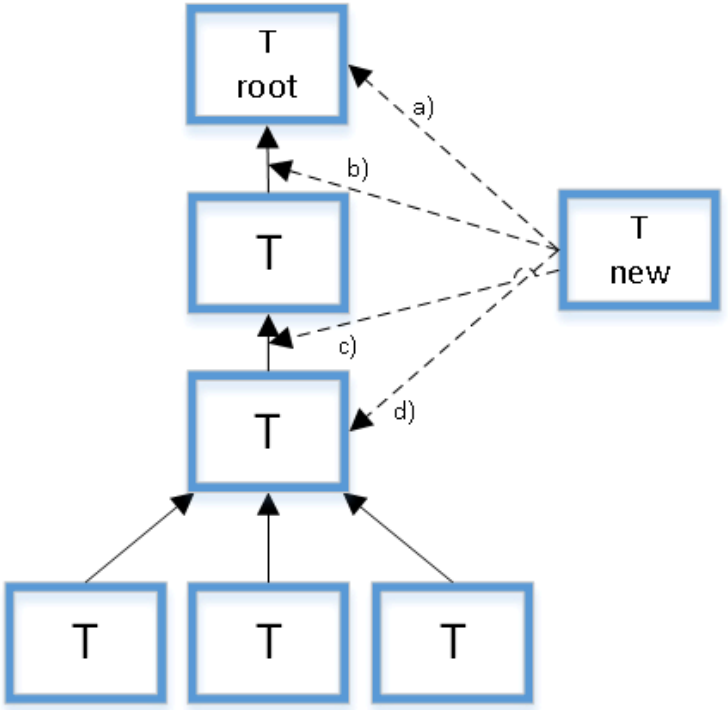
- Blockieren / Nutzer befragen bei entsprechenden Updateoperationen
- Kompensation bei Löschoperation so, dass möglichst wenig Anpassungen der Instanzen notwendig sind

- Systematisieren der Operationen durch Fallunterscheidung
- Ermitteln der Bedingungen (Gegebenheiten in der Typhierarchie) für das Ausführen oder Ablehnen der Operation

Einfügen von Typen

als …

a) Parallelhierarchie
b) Wurzel
c) Inneren Knoten
d) Blatt



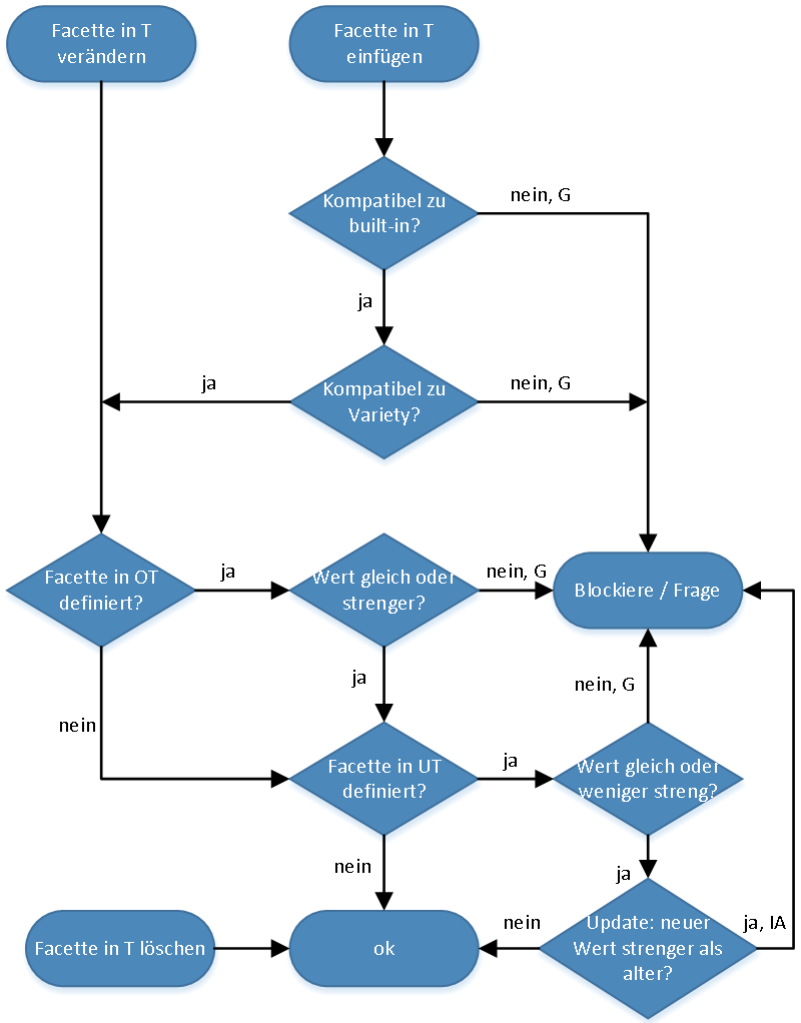
Was passiert mit dem alten Typ bei b) und c)?

- Neuer Typ soll diesen ersetzen
  → **Löschen** des alten Typen
  Update dessen Untertypen
- Neuer Typ soll neuer Obertyp sein
  → **Update** des alten Typen

Ändern von einfachen Typen

Facetten

- Einfügen
- Bearbeiten
- Löschen



G – Gültigkeitsverletzung
IA – Instanzanpassung
OT – Obertyp (rekursiv)
UT – Untertyp (rekursiv)

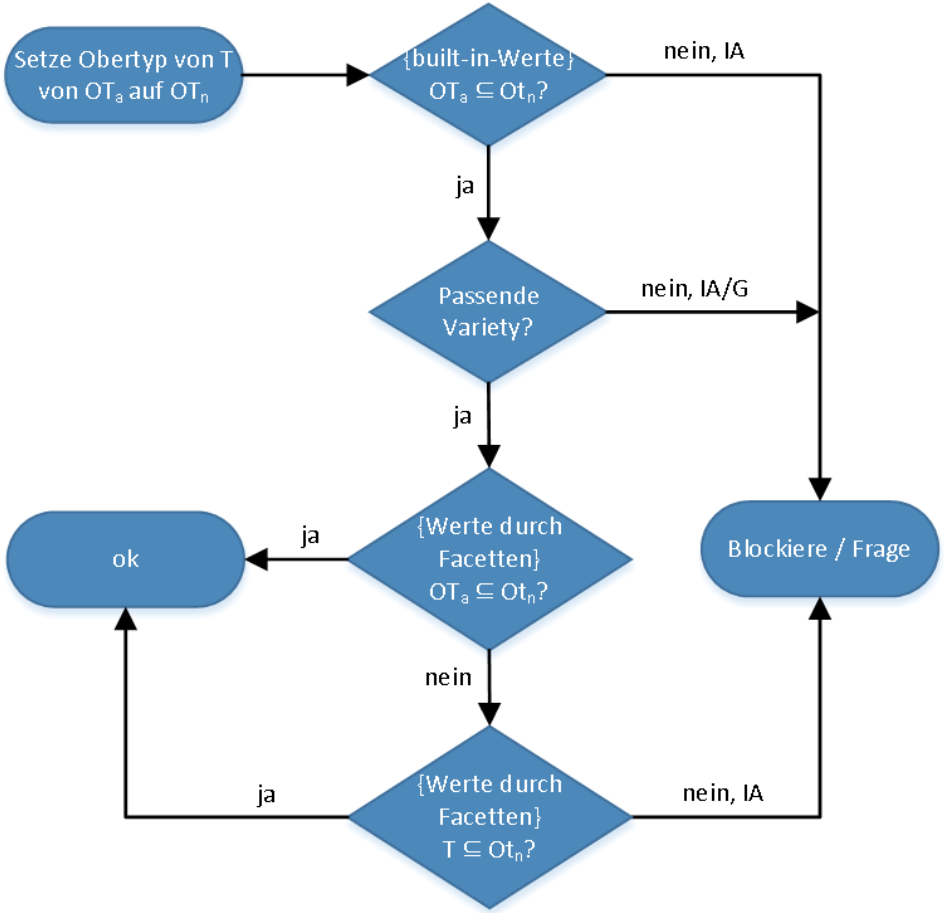
Ändern von einfachen Typen

Obertyp

- Bearbeiten

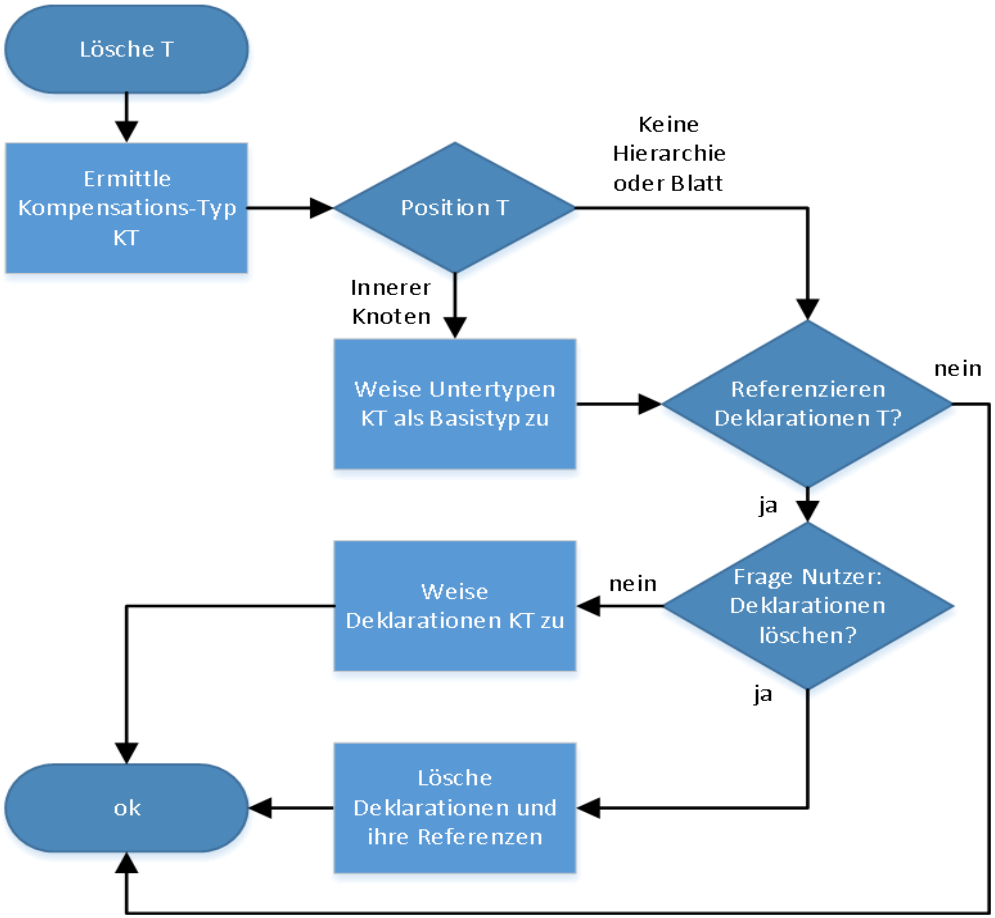
Variety

- atomar -> list
  atomar -> union
  list -> atomar
  list -> union
  union -> atomar
  union -> list



ültigkeitsverletzung
nstanzanpassung
- alter Obertyp
- neuer Obertyp

Löschen von Typen



KT – Kompensationstyp

Kompensation für einfache Typen

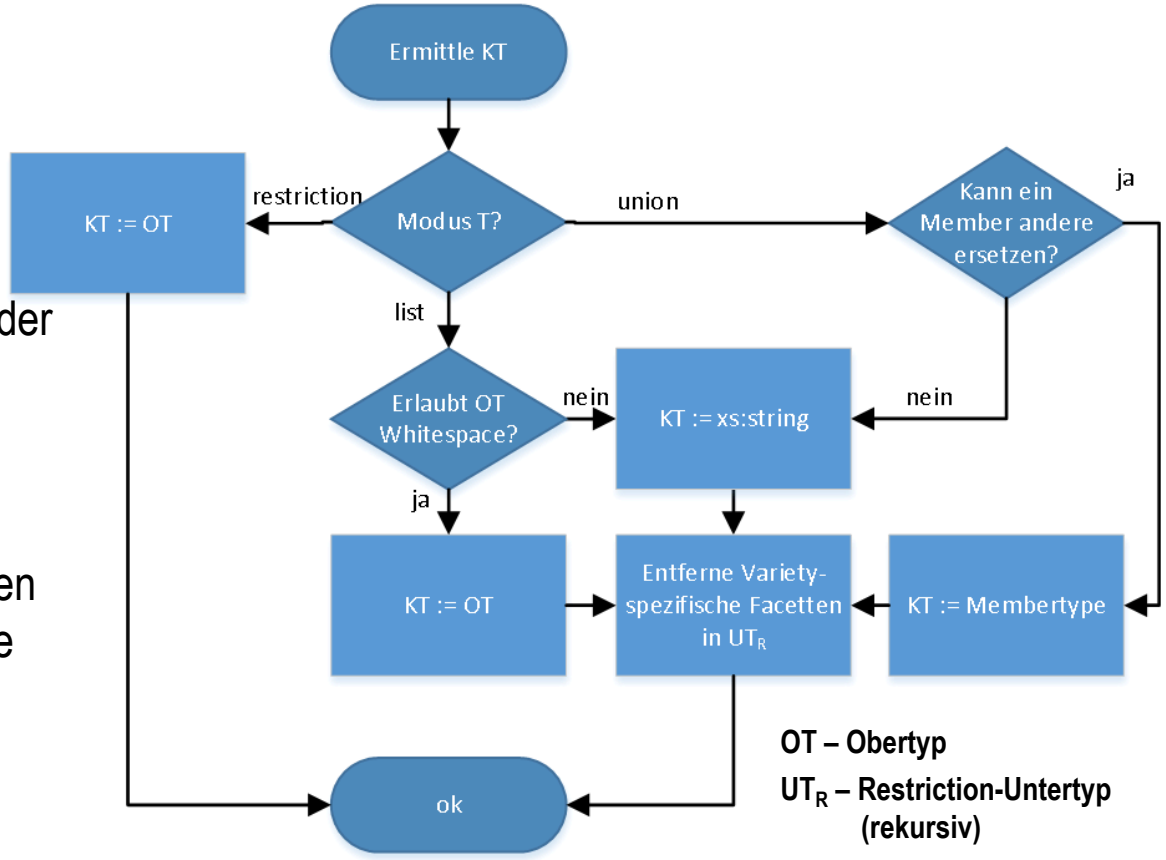
Warum Facetten löschen?

Gelöschter Listentyp wird durch atomaren Typ kompensiert:

→ <xs:length/> und Co ungültig oder andere Bedeutung

Außerdem <xs:assertion/>:

Test-Ausdrücke mit Listenfunktionen schlagen fehl, obwohl Instanzwerte eigentlich ok



OT – Obertyp

UT_R – Restriction-Untertyp (rekursiv)

KT – Kompensationstyp

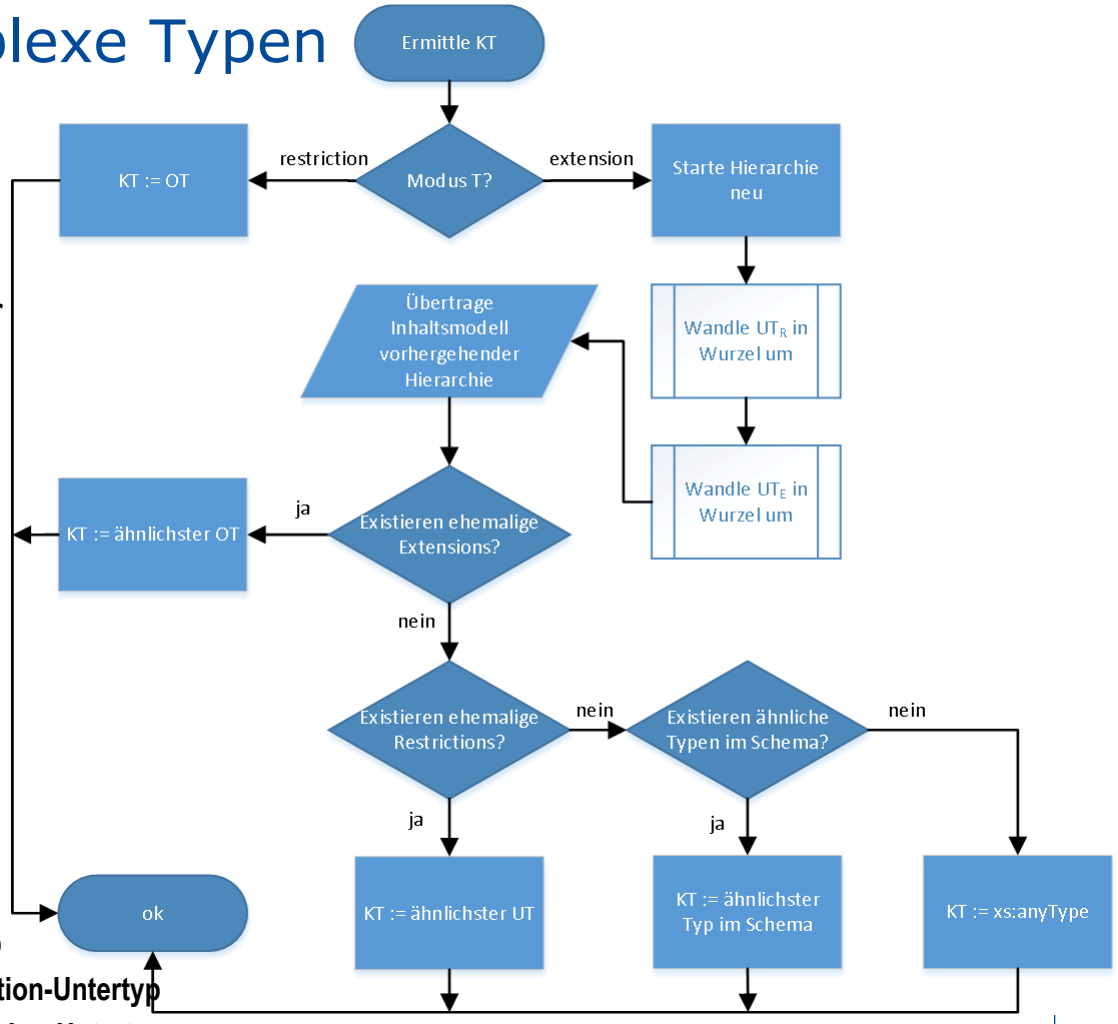
Kompensation für komplexe Typen

Warum Hierarchieneustart?

Alternative: Beseitigen der "Spuren" der Extension in allen Restriction-Untertypen (sonst Schema ungültig)

→ Löschen des erweiterten Teil des Inhaltsmodells

Neben Anpassungen von Elementen die T realisieren, auch Anpassungen von Elementen die Untertypen realisieren



OT – Obertyp
UT – Untertyp
UT_R – Restriction-Untertyp
UT_ET – Extension-Untertyp

Umsetzung in CodeX

Gegebenheiten

- Erzeugung von ELaX-Statements für die erfolgten Änderungen
- EMX sieht keine Abbildung von XML-Schemakomponenten zur Einschränkung und Erweiterung von komplexen Typen vor
- Wegfall des klassischen Hierarchiebegriffs für diese

Folgen

- Existierende komplexe Typen auf Ähnlichkeit überprüfen, daraus konzeptionelle Hierarchie ableiten
- Einfügen und Aktualisieren: Hierarchiebedingungen müssen nicht beachtet werden
- Löschen: kein „verlässlicher“ Obertyp für die Kompensation verfügbar
    - Stattdessen: einen ähnlichen Typen verwenden, Instanzanpassungen nicht ausgeschlossen