UNIVERSITÄT SIEGEN

Vorlesung Datenbanksysteme II

Entwurf von XML-DTD

Inhalt

- Motivation und Grundlagen
- Umsetzung von Entitätstypen
- Umsetzung von Attributen
- Umsetzung von Beziehungstypen
- Umsetzung von Typhierarchien

UNIVERSITÄT SIEGEN

Motivation und Grundlagen

Eigenschaften semistrukturierter Daten

- Struktur von Dokumenten gleichen “Typs” variiert erheblich (je nach Realwelt-Objekt nur bestimmte Elemente, Attribute etc. relevant).
- Große Teile der Dokumente ohne nutzbare Struktur („Fließtext“, „Mixed Contents“).
- Große Vielfalt der Inhalte von Attributwerten, nicht durch wenige elementare Datentypen adäquat modellierbar.
- Strukturen werden relativ häufig geändert.

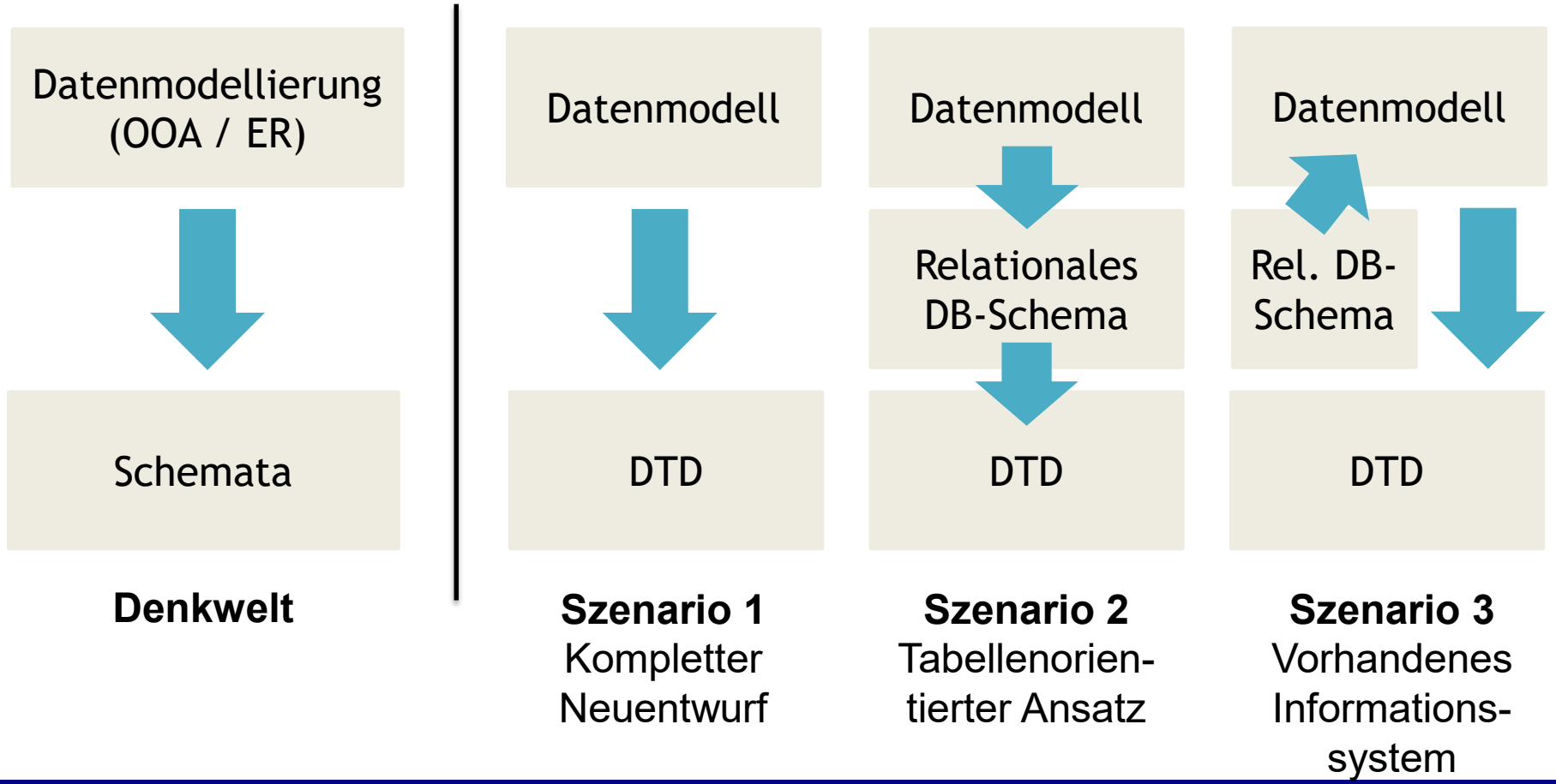
Eigenschaften semistrukturierter Daten

- Schema ist implizit in den Daten / Dokumenten enthalten.
- Keine starre Vorgabe der Strukturen durch ein zentrales Schema.
- Unscharfe Trennung zwischen Daten und Schema.
- Schemainformationen können auch als normale Daten verstanden werden.

Eigenschaften semistrukturierter Daten

Fazit: ER-Modellen / Analyseklassendiagrammen eignen sich nicht gut zur Modellierung semistrukturierter Daten, da vertiefende Behandlung semistrukturierter Daten nicht unterstützt wird.

Entwurf von DTD für strukturierte Daten



Kompletter Neuentwurf einer Applikation

- Anforderungsanalyse wie üblich mit ER- oder Klassendiagrammen.
- Direkte Umsetzung dieser Datenmodelle in DTD, also in das “Datenbankmodell” von XML.
- Hierbei zu beachten: Einige Unterschiede im Vergleich zu relationalen DB.

Tabellenorientierter Ansatz beim Neuentwurf

- Datenmodell zunächst in relationale Schemata (Tabellendefinitionen) umsetzen (d.h. keine Typhierarchien mehr, Beziehungstypen ggf. in Verbindungstabellen umsetzen).
- Jede Tabelle nach einem Standardverfahren in XML nachbilden.
- Immer möglich, aber insbesondere bei der Umsetzung von Beziehungstypen nicht optimal.

Vorhandenes Informationssystem mit RDB

XML wird vor allem zum Datentransport zwischen Applikationen und DB eingesetzt:

- Entweder: Alle Tabellen der vorhandenen relationalen DB-Schemata einzeln nach Standardverfahren umsetzen.
- Oder: Zuerst Reverse Engineering zu ER-Modellen und dann Szenario 1.

Umsetzung von Entitätstypen

Basis und Container-Elementtyp

Für jede Klasse / jeden starken Entitätstyp folgenden Elementtyp in der DTD definieren:

1. **Basis-Elementtyp:** Instanzen stellen genau eine Zeile der Tabelle dar, dabei Name im Singular (z.B.: „Adresse“).
2. **Container-Elementtyp:** hiervon nur eine Instanz nötig, die einen Container für alle Zeilen der Tabelle bildet, dabei Name im Plural (z.B.: „Adressen“).
3. Ggf. weitere Elementtypen für Attribute.

Basis und Container-Elementtyp

Beispiel:

```
<!ELEMENT Adressen (Adresse)* >
<!ELEMENT Adresse ( .... ) >
....
<Adressen>
   <Adresse> ..... </Adresse>
   <Adresse> ..... </Adresse>
....
</Adressen>
```

Struktur des Container-Elementtyps nach dem Schema: <!ELEMENT ContainerET (BasisET)* >

Basis und Container-Elementtypen

Alternative: Gemeinsame Container für mehrere Basis-Elementtypen

```
<!ELEMENT Pruefungsamt ( Studiengang | Pruefungsfach |
            Faecherkatalog | Pruefer | Pruefling | .... )* >
<!ELEMENT Studiengang ( .... ) >
<!ELEMENT Pruefungsfach ( .... ) >
....
```

Struktur des Container-Elementtyps nach dem Schema: <!ELEMENT ContainerET (BasisET1 | BasisET2 | ...)*

- >

Basis und Container-Elementtypen

- Instanzen der Basis-Elementtypen können unsortiert im Container-Element stehen (und verstreut im Speicher).
- Fehlende Container manchmal lästig bei Abfragen.

Zusätzlicher Wurzelelementtyp

Schema:

<!ELEMENT WurzelET (ContainerET1 , ContainerET2 , ...)>

- Falls nur eine Klasse pro Elementtyp.
- Dann setze: WurzelET := ContainerET1.

Umsetzung von Attributen

Attributarten in Dokumenten

1. Einfaches Attribut.
2. Mehrwertiges Attribut.
3. Attribut mit potentiellen „Nullwerten“.
4. Attribut ist Identifizierungsschlüssel.
5. Attribut ist Referenz / Fremdschlüssel (nur bei Umsetzung von Tabellen denkbar, sonst Beziehungstyp).
6. Wie zuvor, aber mehrwertig.

Umsetzung in Kindelement

Jede Attributinstanz wird durch ein Element repräsentiert (Kindknoten einer Instanz des Basis-Elementtyps).

Umsetzung in Kindelemente

Beispiel:

```
<!ELEMENT Adresse (Name, Vorname, Straße,
                   Hausnummer, PLZ, Ort) >
<!ELEMENT Name      (#PCDATA) >
<!ELEMENT Vorname   (#PCDATA) >
<!ELEMENT Straße    (#PCDATA) >
<!ELEMENT Hausnummer (#PCDATA) >
<!ELEMENT PLZ       (#PCDATA) >
<!ELEMENT Ort       (#PCDATA) >
```

- Struktur des Basis-Elementtyps: Sequenz der inneren Elemente.
- Struktur der inneren Elemente: (#PCDATA).

Umsetzung in Kindelemente

Beispiel: Instanz mit einem Element.

```
<Adresse>
   <Name>Meier</Name>
   <Vorname>Hans</Vorname>
   <Straße>Hauptstr.</Straße>
   <Hausnummer>5</Hausnummer>
   <PLZ>57076</PLZ>
   <Ort>Siegen</Ort>
</Adresse>
```

Mehrwertige Attribute

Wiederholungsoperator „*“ oder „+“ am jeweiligen Kindelement.

Beispiel: Mehrere Vornamen

<!ELEMENT Adresse (Name, Vorname+, Straße, Hausnummer, PLZ, Ort) >

Attribute mit Nullwert

Kindelementtyp mit Optionalitätsoperator „?“

Beispiel: Optionaler Titel

<!ELEMENT Adresse (Name, Vorname+, Titel?, Straße, Hausnummer, PLZ, Ort) >

Diskussion: Umsetzung in Kindelemente

- Für alle genannten Fälle anwendbar.
- Aber: bei Id-Attributen im Rahmen der Validitätsprüfung keine automatische Prüfung, ob alle Werte eindeutig sind.
- Und: Potentiell Probleme mit Leerraum im Inhalt der Elemente (wird nicht immer automatisch entfernt).

Umsetzung in XML-Attribute

Umsetzung in CDATA-Attribute: Ein (ER-) Attribut wird in ein XML-Attribut mit Typ CDATA umgesetzt.

Umsetzung in XML-Attribute

Beispiel: Umsetzung in CDATA-Attribute

```
<!ATTLIST Adresse
          Name          CDATA #REQUIRED
          Vorname       CDATA #REQUIRED
          Straße        CDATA #IMPLIED
          Hausnummer    CDATA '0'
          PLZ           CDATA #IMPLIED
          Ort           CDATA #IMPLIED >
```

- Typ der Attribute zumeist CDATA (andere Attributarten s.u.).
- Dazu: Optionalität und Vorgabewerte nach Bedarf.

Umsetzung in XML-Attribute

Beispiel: Instanz mit einem Element

```
<Adresse
   Name="Meier" Vorname="Hans" Straße="Hauptstr."
   Hausnummer="5" PLZ="57076" Ort="Siegen">
</Adresse>
```

Diskussion: Umsetzung in XML-Attribute

- Attributwerte dürfen kein Markup enthalten, d.h. „<„ und „&“ müssen umcodiert werden.
- Nicht geeignet für mehrwertige Attribute:
  - Werte müssten gemäß einer individuellen Syntax als ein einziger Wert codiert werden.
  - Einzelwerte für Abfragesprache nicht erreichbar.
- Attribute mit Nullwert meist problemlos codierbar, z.B. indem eine bestimmte Zeichenkette als Nullwert reserviert wird.

Umsetzung in ID-Attribute

Sonderfall: ER/OOA–Attribut ist Identifizierungsattribut (d.h. bildet alleine einen Identifizierungsschlüssel) und kann somit direkt als XML-ID-Attribut umgesetzt werden.

Umsetzung in ID-Attribute

Beispiel:

```
<!ATTLIST Adresse
          PersNummer   ID #REQUIRED
          Name         CDATA #REQUIRED
          Vorname      CDATA #REQUIRED
          .... >
```

Beispiel: eine Instanz mit einem Element.

```
<Adresse PersNummer="A25"
   Name="Meier" Vorname="Hans" .... ></Adresse>
```

Diskussion: Umsetzung in ID-Attribute

- Für maximal ein Attribut pro Tabelle (für weitere ID-Attribute ggf. künstliche Kindelemente bilden).
- Werte des Attributs müssen Syntax eines XML-Namens erfüllen (mit Buchstaben anfangen, also keine Nummern erlaubt).
- Werte in ID-Attributen müssen global eindeutig sein, nicht nur in der Menge der Instanzen dieses Elementtyps.

Umsetzung in IDREF-Attribute

Sonderfall für Tabellen als Ausgangsbasis: Tabellen-Attribut ist Fremdschlüssel auf eine andere Tabelle ist und kann somit direkt als XML-IDREF-Attribut umgesetzt werden.

Umsetzung in IDREF-Attribute

Beispiel:

```
<!ATTLIST Rechnung
          RgNummer      ID #REQUIRED
          PersNummer    IDREF #REQUIRED
          RgDatum       CDATA #REQUIRED .... >
```

Beispiel: eine Instanz mit einem Element.

```
<Rechnung RgNummer="R2004 176"
          PersNummer="A25" RgDatum="2004-04-14" .... />
```

Diskussion: Umsetzung in IDREF-Attribute

- Dieser Sonderfall ist bei OOA als Ausgangsbasis nicht möglich.
- Statt des Attributs müsste hier ein Beziehungstyp vorhanden sein.

Diskussion: Vergleich der Alternativen

Für jedes einzelne Attribut freie Auswahl möglich.

- Kindelemente brauchen etwas mehr Platz (maximal Faktor 2).
- Attribute können Vorgabewerte haben, Kindelemente nicht.
- ID-Attribute: Eindeutigkeit sichergestellt.
- IDREF-Attribute: referentielle Integrität sichergestellt.

Diskussion: Vergleich der Alternativen

- Reihenfolge von XML-Attributen beliebig, Kindelemente haben feste Reihenfolge.
- Attribute können keinen Text mit eingebetteten Elementen enthalten, Attributwerte sind immer Blätter im Syntaxbaum und können keine Kindknoten haben.

UNIVERSITÄT SIEGEN

Umsetzung von Beziehungstypen

Beziehungen zwischen Elementen

- Realwelt-Entitäten werden in XML-Datenbeständen in Form von Elementen repräsentiert.
- Wie können Beziehungen zwischen Realwelt-Entitäten in XML-Datenbeständen codiert werden?

Beziehungen: Grundlegende Ansätze

Wertebasierte Implementierung:

- Gleicher Datenwert (bzw. Bezeichner) irgendwo innerhalb der XML-Elemente (analog zu relationalen DB).
- Im Prinzip alle Speicherungsvarianten für Attribute dafür nutzbar.

Beziehungen: Grundlegende Ansätze

Positionsbasierte Implementierung:

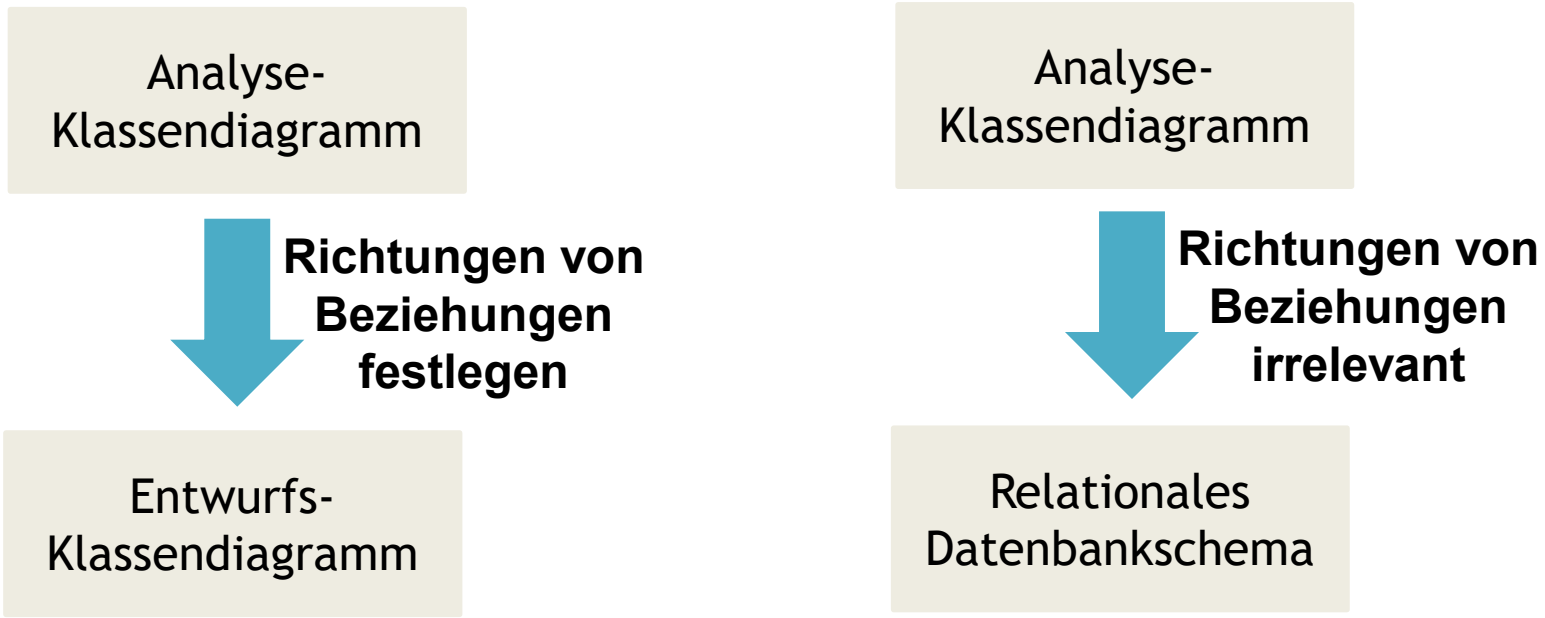
- Elemente, die eine Rolle repräsentieren, haben in der Struktur des Syntaxbaums eindeutig definierte, relative Positionen zueinander

Aus beiden Ansätzen ergeben sich eine Vielzahl an Kombinationen.

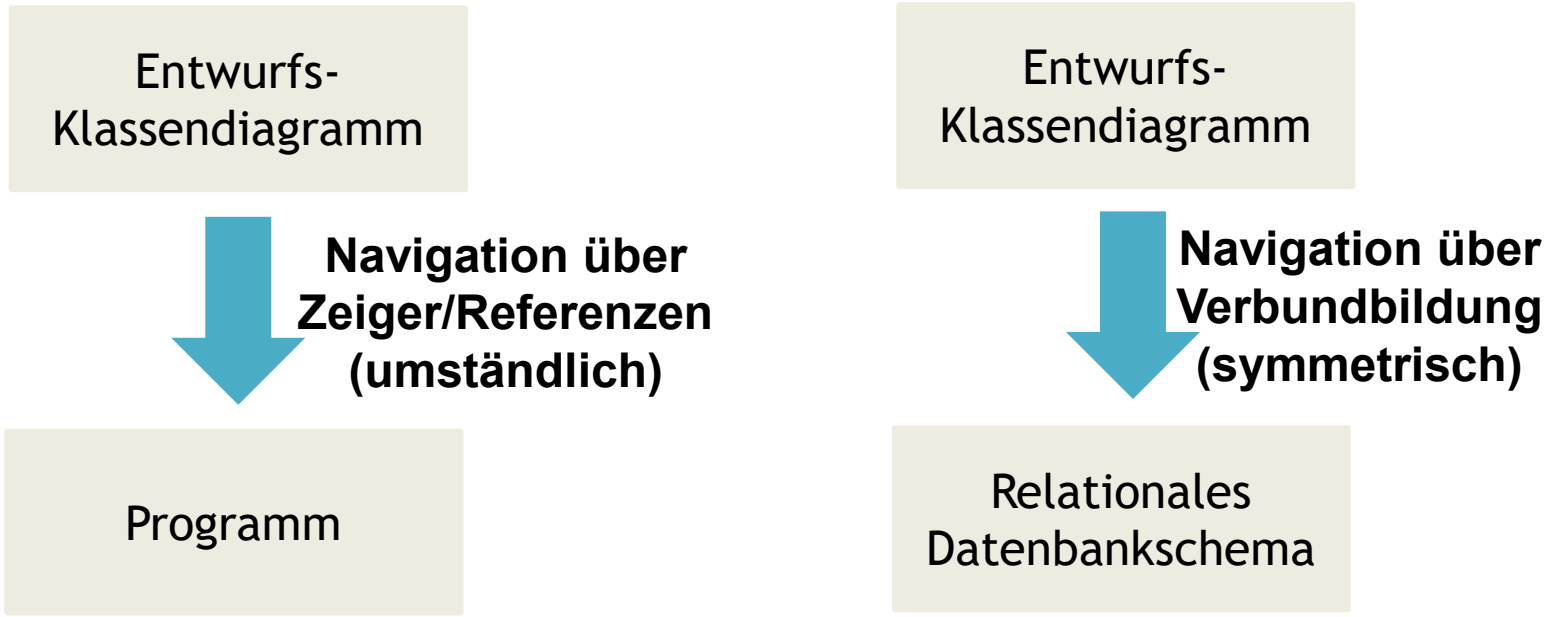
Bewertungskriterien für Umsetzungsalternativen

- Resultierende Effizienz und Programmieraufwand von bzw. für Funktionen, die über Beziehungen navigieren oder die Daten zusammenführen, die durch Beziehungen verbunden sind (sind Beziehungen gerichtet?).
- Unterstützung der Konsistenzerhaltung von Beziehungen (generell schwach).

Exkurs: Gerichtete Beziehungen



Exkurs: Gerichtete Beziehungen

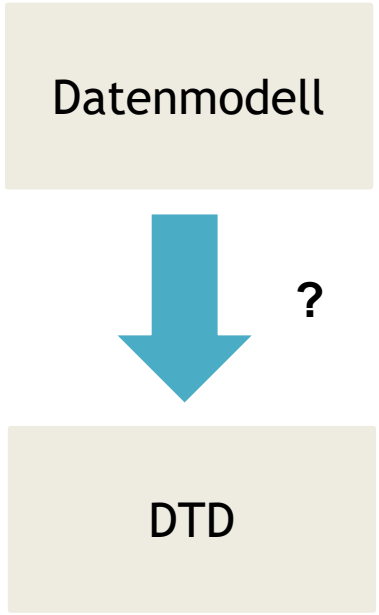


Exkurs: Gerichtete Beziehungen

In Datenverwaltungssystemen implementierte Beziehungen sind gerichtet, wenn

- die Programmierkonstrukte zum Navigieren über die Beziehungen sich je nach Richtung deutlich unterscheiden (d.h. in einer Richtung wesentlich komplizierter sind) und/oder
- die benötigten Laufzeiten zum Navigieren über die Beziehungen sich je nach Richtung deutlich unterscheiden (d.h. in einer Richtung wesentlich höher sind).

Beziehungsrichtungen in XML



- Sind Referenzen in XML gerichtet?
- Müssen Beziehungsrichtungen beim DTD-Entwurf berücksichtigt werden?

Beziehungsrichtungen in XML

- XML selbst definiert keine Operationen, deshalb sind Zugriffsverfahren und Navigationsrichtungen nicht relevant.
- Operationen für Zugriffsverfahren erst im Kontext von XPath, XQuery, XSLT, ... relevant.
- Beispiel XPath: Verbundbildung nur sehr eingeschränkt möglich und nur ungerichtet.

Wertbasierte Implementierung von Beziehungen

Welche Knotentypen (Elemente vs. XML-Attribute) eignen sich für die Implementierung von 2-/n-stelligen ER/OOA-Beziehungstypen?

Wertbasierte Implementierung von Beziehungen

Vorüberlegung: Bezeichner treten in den XML-Elementen in zwei Rollen auf:

1. Als **eindeutige Identifizierer** des Realweltobjekts, das durch das XML-Element repr¨asentiert wird.
2. als **Referenz** (d.h. gerichtete 2-stellige Beziehung) auf ein (anderes?) Realweltobjekt bzw. das repräsentierende XML-Element.

Speicherung von Identifizierern

- Datenwerte, die die Rolle von Identifiziern spielen (d.h. Identifizierungsattribute) können im Prinzip auch unabhängig von Beziehungen auftreten.
- Somit: Realisationsalternativen wie bei identifizierenden ER/OOA-Attributen

Speicherung von Referenzen

Mögliche Speicherung von Bezeichnern, die als Referenz dienen:

- Im Inhalt eines Kindelements.
- In einem CDATA-Attribut.
- In einem IDREF-Attribut.
- In einem IDREFS-Attribut.

Speicherung von Referenzen

- Restriktionen für die beiden ersten Fälle: siehe jeweilige Abschnitte.
- Restriktionen bei IDREF(S)-Attributen:
  - Bezeichner ist gültiger XML-Namen.
  - Die Identifizierungen müssen an den Zielelementen in einem ID-Attribut gespeichert sein.
  - Ein IDREF-Attribut kann nicht zugleich ID-Attribut sein.

Implementierung mit autarken Beziehungselementen

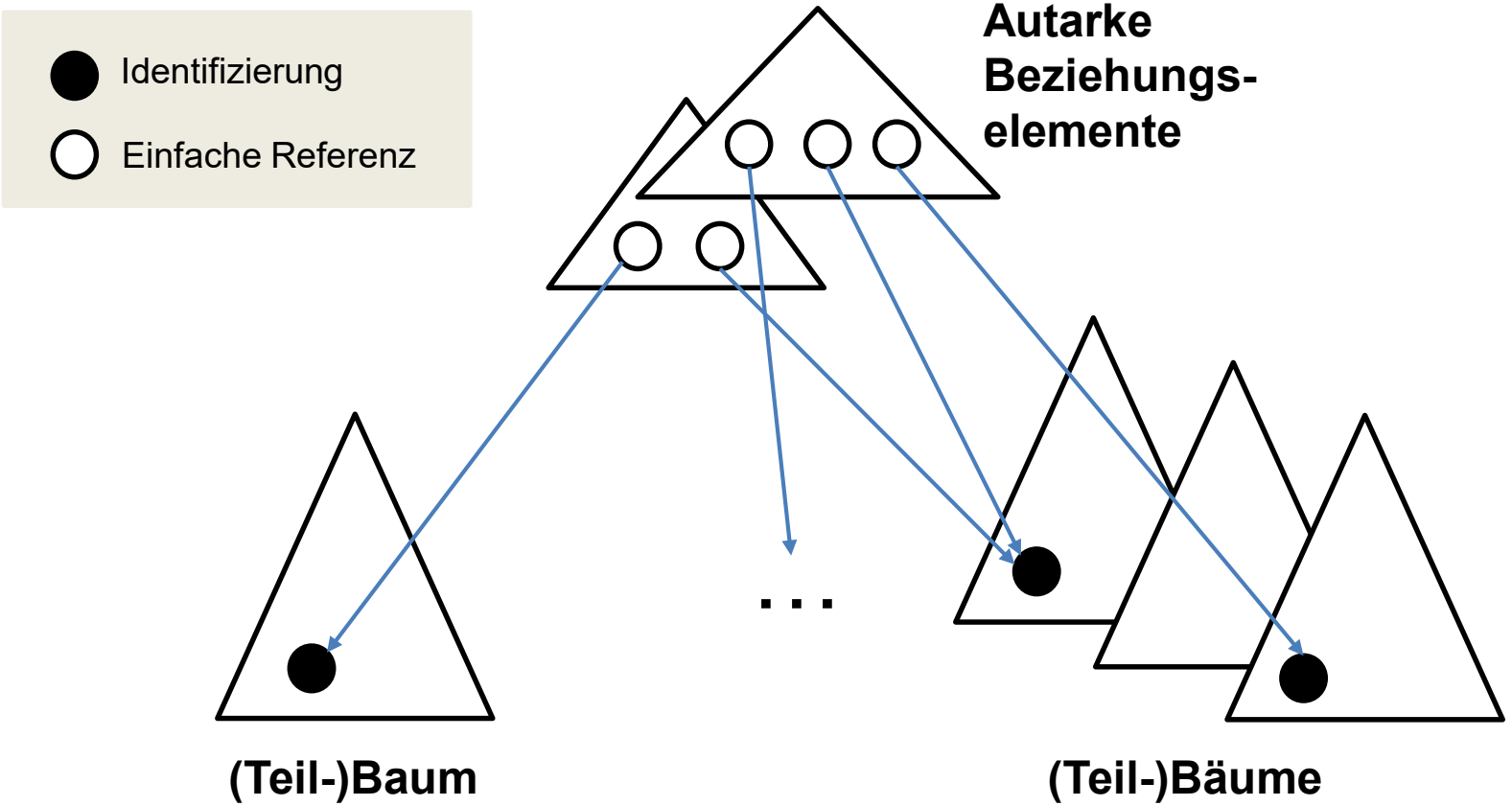
Problem: Für eine n-stellige Beziehung müssen n Referenzen auf die durch die Beziehung verbundenen Elemente gespeichert werden.

Implementierung mit autarken Beziehungselementen

Autarke Beziehungselemente:

- Jede einzelne Beziehung wird durch ein Element repräsentiert, das Referenzen auf die in Beziehung stehenden Elemente enthält.
- Entspricht einer Verbindungstabelle.

Implementierung mit autarken Beziehungselementen

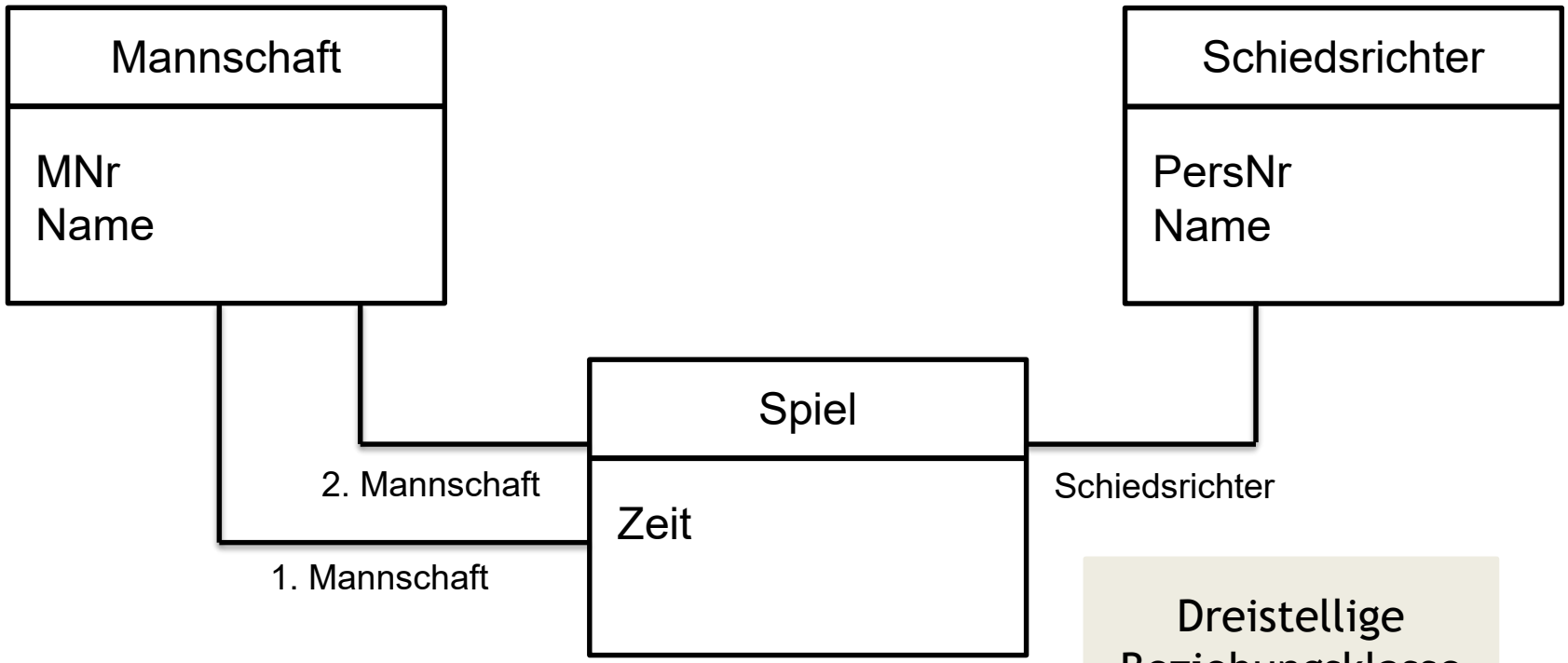


Implementierung mit autarken Beziehungselementen

Speicherung der Referenzen im Beziehungselement:

- Kindelemente, CDATA-Attribute oder IDREF-Attribute sind nutzbar.
- IDREFS-Attribute hingegen nicht sinnvoll.

Beispiel: Analyse-Klassendiagramm „Spielplan“



Beispiel: DTD „Spielplan“

```
<!ELEMENT Mannschaft ( ..... )    >
<!ATTLIST Mannschaft
                MNr    ID          #REQUIRED
                Name CDATA         #REQUIRED ..... >
<!ELEMENT Schiedsrichter EMPTY >
<!ATTLIST Schiedsrichter
                PersNr ID          #REQUIRED
                Name CDATA         #REQUIRED ..... >
<!ELEMENT Spiel EMPTY >
<!ATTLIST   Spiel MNr1             IDREF #REQUIRED
                  MNr2             IDREF #REQUIRED
                  PersNr           IDREF #REQUIRED
                  Zeit             CDATA .... >
```

Referenzen als IDREF-Attribute implementiert.

Beispiel: XML-Nutzdaten „Spielplan“

```
.....

<Mannschaft      MNr = "M123"     Name="FC Kick" ... />
<Schiedsrichter  PersNr = "p007"  Name="Richter" ... />
<Spiel MNr1="M123" MNr2="M17" PersNr="p007„ Zeit="15:30" />

.....
```

Beispiel: Diskussion

- Identifizierungen bei **allen** Elementtypen, die eine Rolle in der Beziehung spielen.
- Geeignet für **n-stellige** und **attributierte** Beziehungen.
- Beziehungselemente können **unabhängig** von den verbundenen Elementen angelegt und gelöscht werden.
- Beziehungselemente brauchen **separate** Mengenverwaltung.

Implementierung mit eingebetteten Beziehungselementen

- Die Daten, die die Beziehung repräsentieren, werden in **eines der Elemente** eingebettet, die in der Beziehung eine Rolle spielen.
- Vorteil: Referenz auf das einbettende Element nicht mehr nötig (implizit vorhanden).

Beispiel: DTD „Spielplan“

```
<!ELEMENT Mannschaft ( ..... ) >
<!ATTLIST Mannschaft .... s.o. ..... >

<!ELEMENT Schiedsrichter (Spiel*) >
<!ATTLIST Schiedsrichter .... s.o. ..... >

<!ELEMENT Spiel EMPTY >
<!ATTLIST Spiel   MNr1 IDREF #REQUIRED
                  MNr2 IDREF #REQUIRED
                  Zeit   CDATA .... >
```

Beziehungen des Typs „Spiel“ werden in die „Schiedsrichter“-Elemente eingebettet.

Beispiel: XML-Nutzdaten „Spielplan“

```
.....

<Mannschaft      MNr="M123"      Name="FC Kick" ... />
<Schiedsrichter  PersNr="p007"   Name="Richter" ... />
   <Spiel MNr1="M123" MNr2="M17" Zeit="15:30" />
</Schiedsrichter>
.....
```

Varianten des Ansatzes

Je nach “Kardinalität” der Rolle, also der Anzahl der Beziehungen, in der das einbettende Element diese Rolle spielen kann:

- Mehrfach (z.B. Schiedsrichter): Beziehungs-unterelemente mit Wiederholungsoperator „*“ definieren.
- „0“ oder „1“: mit Operator „?“optional definieren oder ohne Operator, falls immer genau „1“.

Varianten des Ansatzes

- Bei „1“ kann man auch auf die Klammer durch das explizite Kindelement verzichten und die Attribute und inneren Elemente direkt dem einbettenden Elementtyp zuordnen.
- Bei „0“ analog, aber innere Elemente als optional definieren und Nullwerte bei Attributen vereinbaren.

Diskussion

- Erfordert Identifizierungen bei allen Elementtypen, die eine Rolle in der Beziehung spielen, mit Ausnahme des einbettenden Elementtyps.
- Für n-stellige Beziehungen und alle „Kardinalitäten“ geeignet.
- Offene Frage: Auswahlkriterien für den Elementtyp, dessen Instanzen die Beziehungen einbetten sollen?

Diskussion

- Beziehungen nicht unabhängig von den einbettenden Elementen erzeugbar und löschbar.
- Attribute von Beziehungen bzw. deren XML-Realisierungen werden im einbettenden Element angeordnet, obwohl sie nicht die Entität beschreiben, die durch das einbettende Element repräsentiert wird.

Implementierung mit eingebetteten Referenzen

- Speziell im Fall von 2-stelligen Beziehungen muss nur eine Referenz dargestellt werden.
- Erlaubt eine besonders kompakte Realisierungen.
- Mehr Varianten möglich.

Beispiel

- Klassen bzw. Elementtypen „Rechnung“ und „Kunde“, Beziehungstyp „wirdBezahltVon“ zwischen „Rechnung“ und „Kunde“.
- Rechnungselements mit einem eingebetteten Beziehungselement vom Typ „wirdBezahltVon“

```
<Rechnung RgNummer = "R2004 176" RgDatum = "2004-04-14„ ... >
   <wirdBezahltVon PersNummer="P125" />
</Rechnung>
```

Beispiel

```
<!ELEMENT Kunde ( .... ) >
<!ATTLIST    Kunde
                PersNummer          ID #REQUIRED .... >
<!ELEMENT Rechnung ( .... ) >
<!ATTLIST    Rechnung
                RgNummer            ID #REQUIRED
                wirdBezahltVon      IDREF #REQUIRED
                RgDatum             CDATA #REQUIRED .... >
```

IDREF-Attribut direkt bei Rechnung-Element anordnen.

Diskussion

- Referenz auf Rolle “Rechnung” wird implizit dargestellt, indem die Beziehungen innerhalb von „Rechnung“-Elementen gespeichert werden.
- Referenz auf Kunden als IDREF-Attribut gespeichert.

Diskussion

- Wegen Kardinalität „1“ (Rechnung wird von genau „1“ Kunden bezahlt) wäre nur ein inneres Element vorhanden; das entfällt und das IDREF-Attribut wird direkt dem einbettenden Element (hier vom Typ Rechnung) zugeordnet.
- Für die Speicherung der Referenzen kann zwischen allen Alternativen gewählt werden (hier: IDREF-Attribut nur ein Beispiel)

Implementierung zweistelliger Beziehungen ohne Attribute

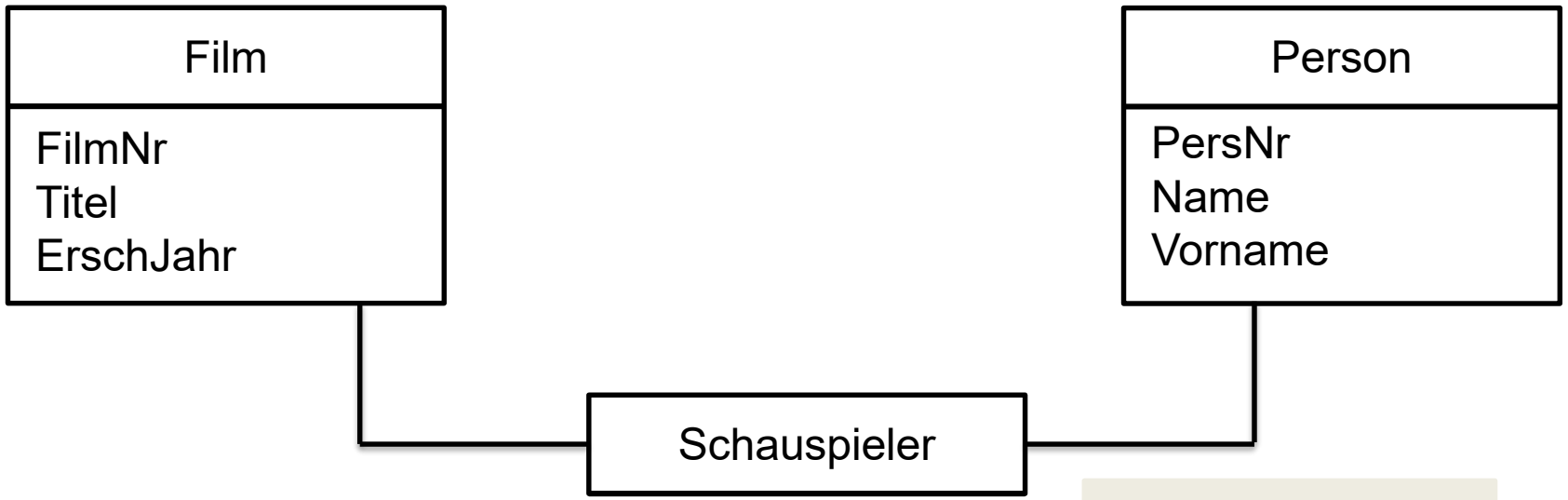
Kardinalität „0:1“:

- Beziehung kann durch eine einzige eingebettete Referenz repräsentiert werden.

Kardinalität „*“:

- Beziehung kann durch eine eingebettete mehrwertige Referenz repräsentiert werden.
- Referenz falls möglich in IDREFS-Attribut speichern.

Beispiel: Analyse-Klassendiagramm “Filmdatenbank“



Beispiel: DTD “Filmdatenbank“

```
<!ELEMENT Film ( .... ) >
<!ATTLIST   Film FilmNr          ID       #REQUIRED
                 Titel           CDATA    #REQUIRED
                 ErschJahr       CDATA    #REQUIRED
                 Schauspieler    IDREFS   #REQUIRED
                 ....                                 >

<!ELEMENT Person ( .... ) >
<!ATTLIST   Person       PersNr   ID       #REQUIRED
                         Name     CDATA    #REQUIRED
                         Vorname  CDATA    #REQUIRED
                         FilmNr   IDREFS   #REQUIRED
                 ....                                 >
```

Beispiel: Nutzdaten „Filmdatenbank“

```
<Film FilmNr = "F123" Titel = "Der Stadtneurotiker" ....
        Schauspieler = "P16543 P00755 P23098"> ....
</Film>
```

Diskussion

- Nur für Assoziationen ohne Attribute.
- Implementierung der Beziehungen ist gerichtet.
- Kann für eine oder beide "Navigationsrichtungen" vorgesehen werden.
- Nachteil: Verarbeitung der IDREFS-Attributinhalte oft sehr kompliziert.

Implementierung zweistelliger Beziehungen mit Attributen

Implementierung mit eingebetteten Referenzen, wenn eine der Rollen Kardinalität „[0,1]“ hat.

- Vorgehensweise wie zuvor zur Umsetzung von Kardinalität „[0,1]“ beschrieben.
- Für Attribute benötige Kindelemente / Attribute “neben” der Referenz anordnen.

Beispiel: DTD Filmdatenbank

```
<!ELEMENT Film ( Schauspieler*, .... ) >
<!ATTLIST   Film          FilmNr          ID       #REQUIRED
                          Titel           CDATA    #REQUIRED
                          ErschJahr       CDATA    #REQUIRED
                          ....                              >
<!ELEMENT hatSchauspieler EMPTY >
<!ATTLIST   hatSchauspieler PersNr        IDREF    #REQUIRED>
                          ....                              >
```

Beispiel: Nutzdaten „Filmdatenbank“

```
<Film FilmNr = "F123" Titel = "Der Stadtneurotiker" .... >
    <hatSchauspieler PersNr="P16543" />
    <hatSchauspieler PersNr="P00755" />
    <hatSchauspieler PersNr="P23098" /> ....
</Film>
```

Diskussion

- Mehreren Assoziationstypen zwischen den gleichen Klassen problemlos möglich.
- Aber unintuitiv: Attribute sind einer der beiden Navigationsrichtungen zugeordnet.

Implementierung von Kompositionen

Ausgangspunkt: „1:n“-Kompositionsbeziehung zwischen Klassen K1 (Ganzes) und K2 (Komponente).

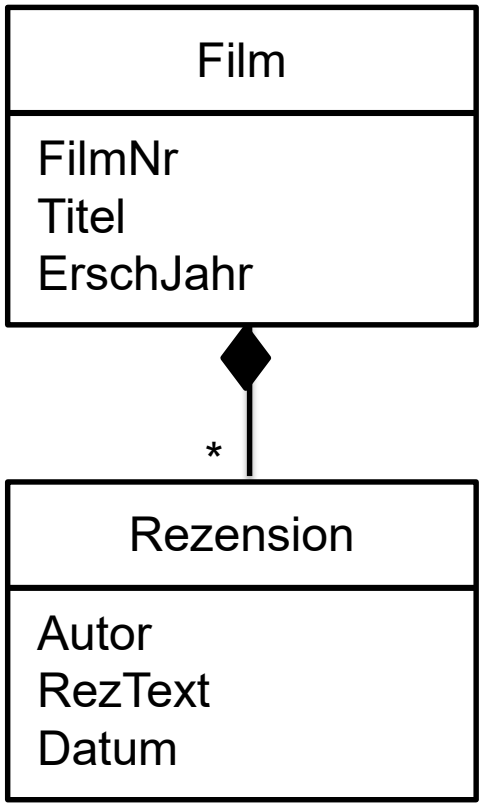
- Jede Komponente exklusiv in einem Ganzen enthalten.
- Behandlung in relationalen DB durch Fremdschlüssel von K2 auf K1.

Implementierung von Kompositionen

Behandlungsmöglichkeit in XML:

- Einschachteln.
- So kein Fremdschlüsselattribut erforderlich.

Beispiel: Analyse-Klassendiagramm „Filmdatenbank“



Beispiel: DTD Filmdatenbank

```
<!ELEMENT Film ( Rezension*, .... ) >
<!ATTLIST   Film          FilmNr          ID        #REQUIRED
                          Titel           CDATA     #REQUIRED
                          ErschJahr       CDATA     #REQUIRED
                          ....                              >
<!ELEMENT Rezension (Autor, RezText, Datum) />
```

Umsetzung von Typhierarchien

Umsetzung von Typhierarchien

Mögliche Implementierungen von Typhierarchien mit Oberklasse „O“ und Unterklassen „U1, …“:

1. Analog zu Umsetzung in relationale Schemata:
   - Horizontale Partitionierung (trivial).
   - Vertikale Partitionierung.
   - Gemeinsamer Subtyp.
2. Typhierarchie in Kindelement umsetzen:
   - Entspricht vertikaler Partitionierung.

Vertikale Partitionierung

Tabelle mit speziellen Attributen von Unterklassen „Ui“:

- Fremdschlüssel auf Tabelle für Oberklasse „O“.
- Kann als Komposition umgesetzt werden.
- 1:1-Beziehung in beiden Richtungen:
  - 1. Richtung: Element mit den speziellen Attributen als Kindelement.
  - 2. Richtung: Element mit den allgemeinen Attributen als Kindelement.

Vertikale Partitionierung

Beispiel: spezielle Attribute in Kindelement.

```
<!ELEMENT O     ( ... direkte Attribute von O,
                  ( spezAttrU1 | .... | spezAttrUn )? ) >
<!ELEMENT spezAttrU1     (:spezielle Attribute von U1:) >
....
<!ELEMENT spezAttrUn     (:spezielle Attribute von Un:) >
```

- Kindelement entfällt für Instanzen von O.
- Genau eine der Alternativen für Instanz eines Untertyps.

Vertikale Partitionierung

Alternative:

- Kindelementtypen nicht benennen, sondern direkt deren Definition statt „spezAttrUi“ einsetzen.
- Nachteil: Strukturen schlechter erkennbar (z.B. bei einem Element vom Typ „O“ schwer erkennbar, welcher konkrete Subtyp vorliegt).

Vertikale Partitionierung

Beispiel: allgemeine Attribute in Kindelement.

```
<!ELEMENT U1 ( O, :spezielle Attribute von U1: ) >
....
<!ELEMENT U2 ( O, :spezielle Attribute von Un: ) >
<!ELEMENT O ..... >
```

Gemeinsamer Subtyp

- ER-Attribute, die in XML-Attribute umgesetzt werden: alle Attribute beim Elementtyp für die Oberklasse „O“ vereinigen.
- ER-Attribute, die in Kindelemente umgesetzt werden: passend sortierte Sequenz von Kindelementtypen bei Oberklasse „O“.
- Kindelementtypen bei Bedarf als optional angeben.

Beispiel

```
<!ELEMENT O ( ... direkte Attribute von O,
                U1A1? | U1A2? | .... |
                U2A1? | U2A2? | .... |
                UnA1? | UnA2? | .... ) >
<!ELEMENT U1A1 ..... >
```

Prüfungsstoff

- Grundlegende Vorgehensweise beim Entwurf von DTDs erklären.
- Alternativen bei Detailentscheidungen diskutieren und vergleichen.
- Vergleich des DTD-Entwurf mit dem Entwurf relationaler Schemata.