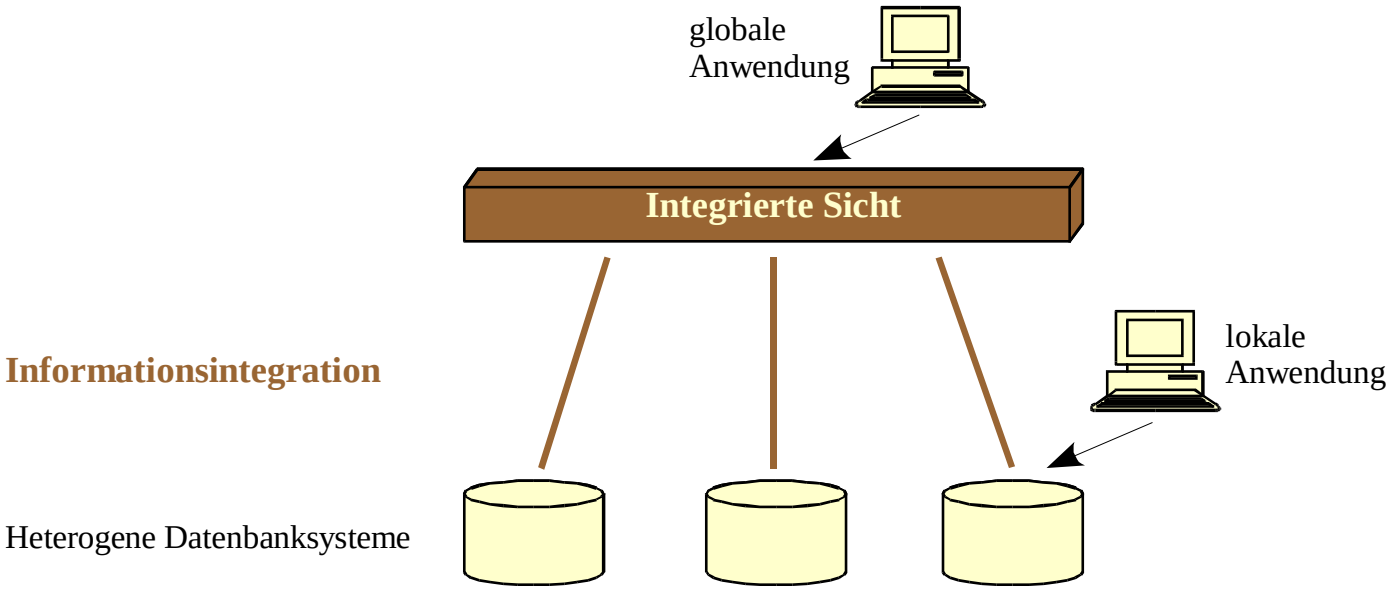
Formen der Heterogenität

Jessica Oksana Schnabel

16. Juli 2004

Integration

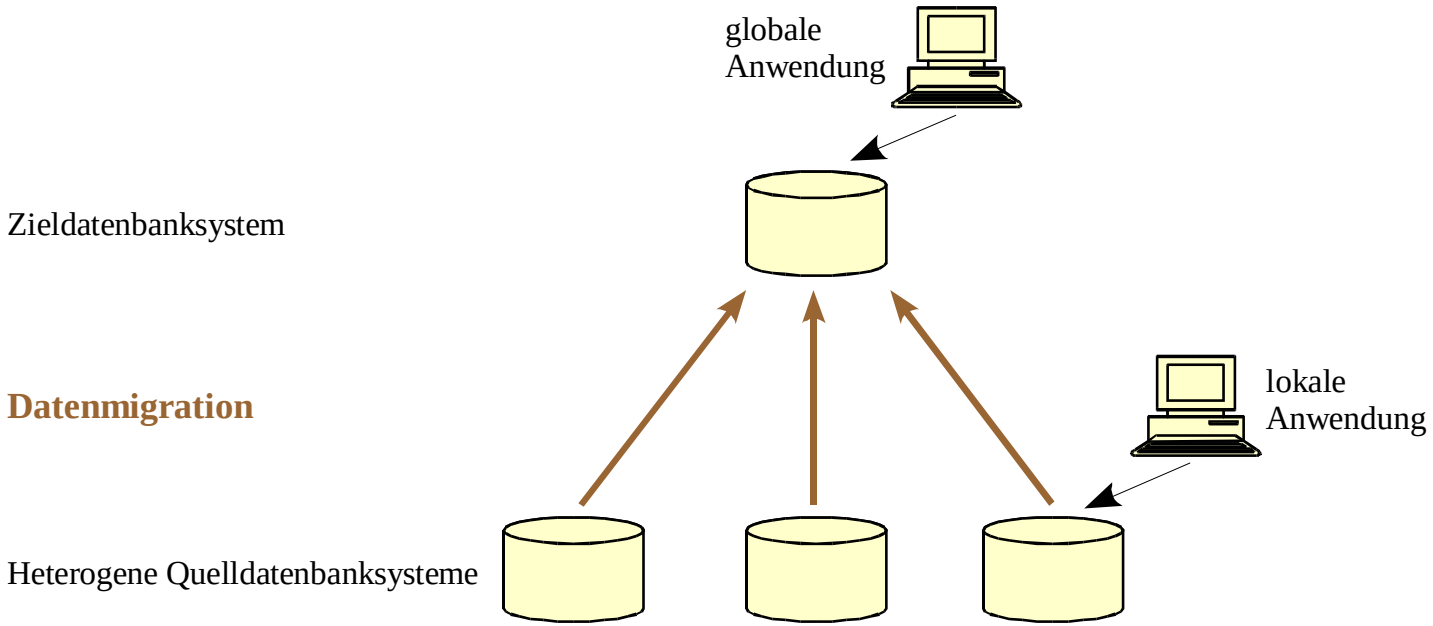
- Daten in heterogenen Datenbanksystemen verteilt
- Einheitlicher zentraler Zugriff über globale integrierte Sicht



- Bsp.: Föderierte Datenbanksysteme

Migration

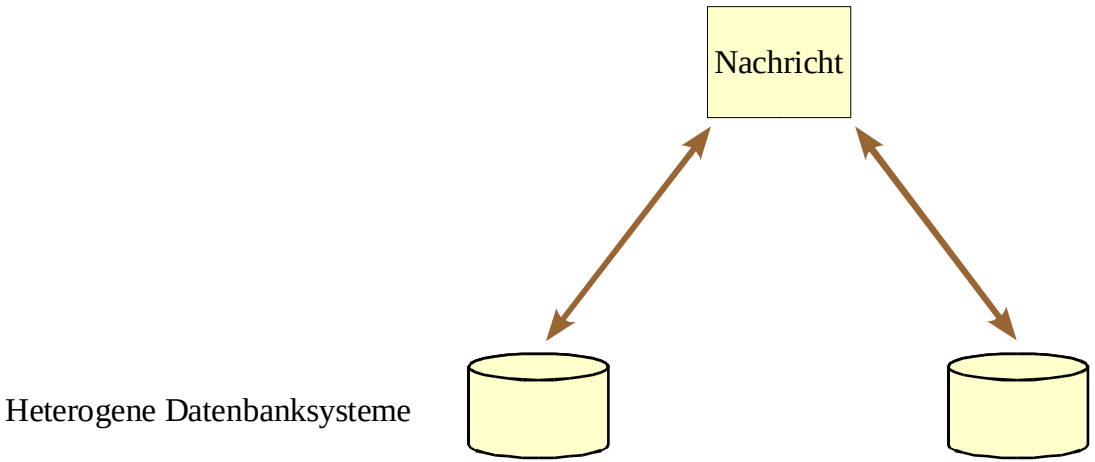
- Daten in heterogenen Quelldatenbanksystemen verteilt
- Migration der Daten aus den Quelldatenbanksystemen in ein Zieldatenbanksystem



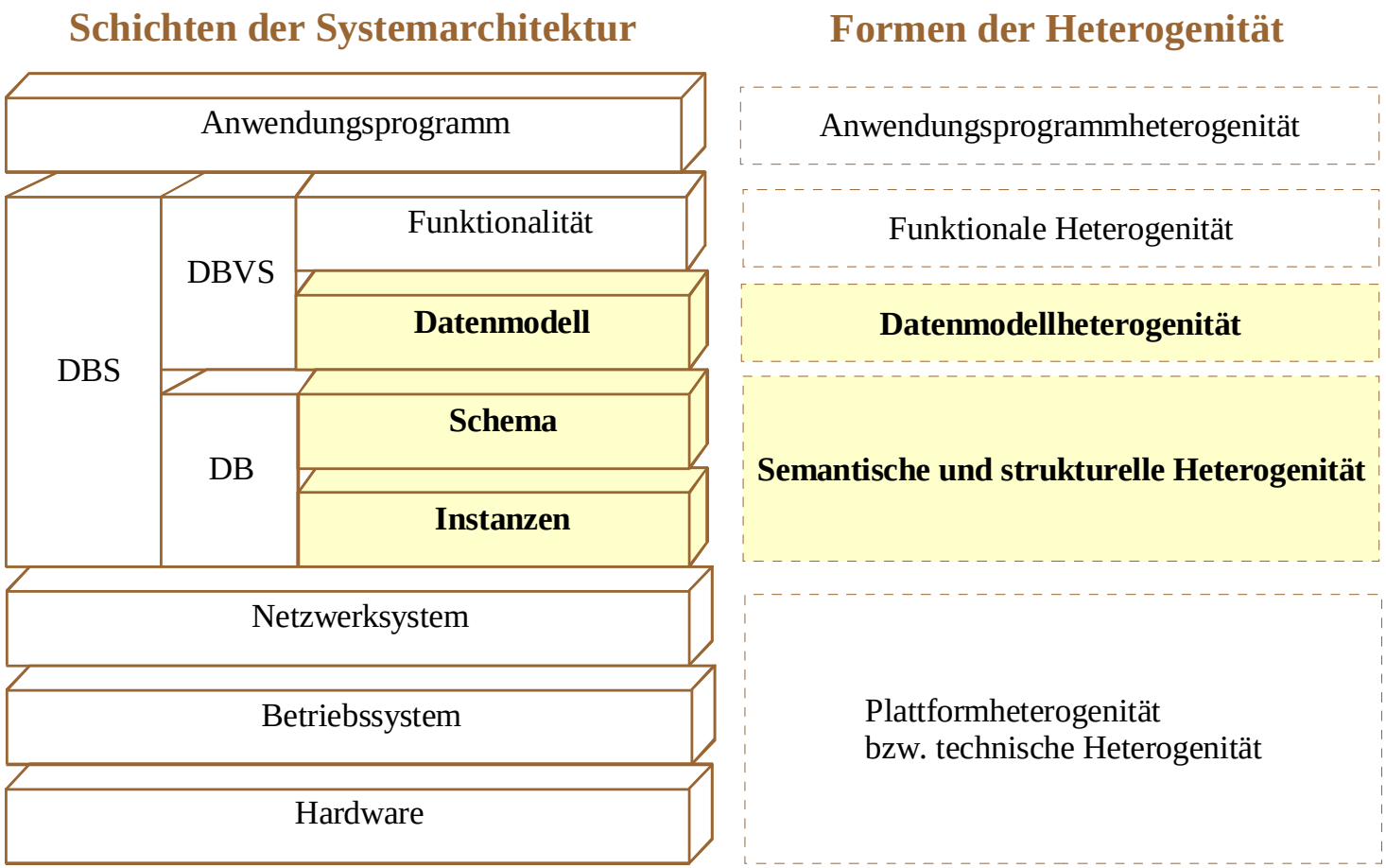
- Bsp.: Data Warehouse, Ablösen veralteter Datenbanksysteme

Nachrichtenaustausch

- Datenaustausch über das Internet
- Speicherung der Daten in heterogenen Datenbanksystemen
- Austausch über ein gemeinsames Schema bzw. Nachrichtenformat



Formen der Heterogenität – Überblick

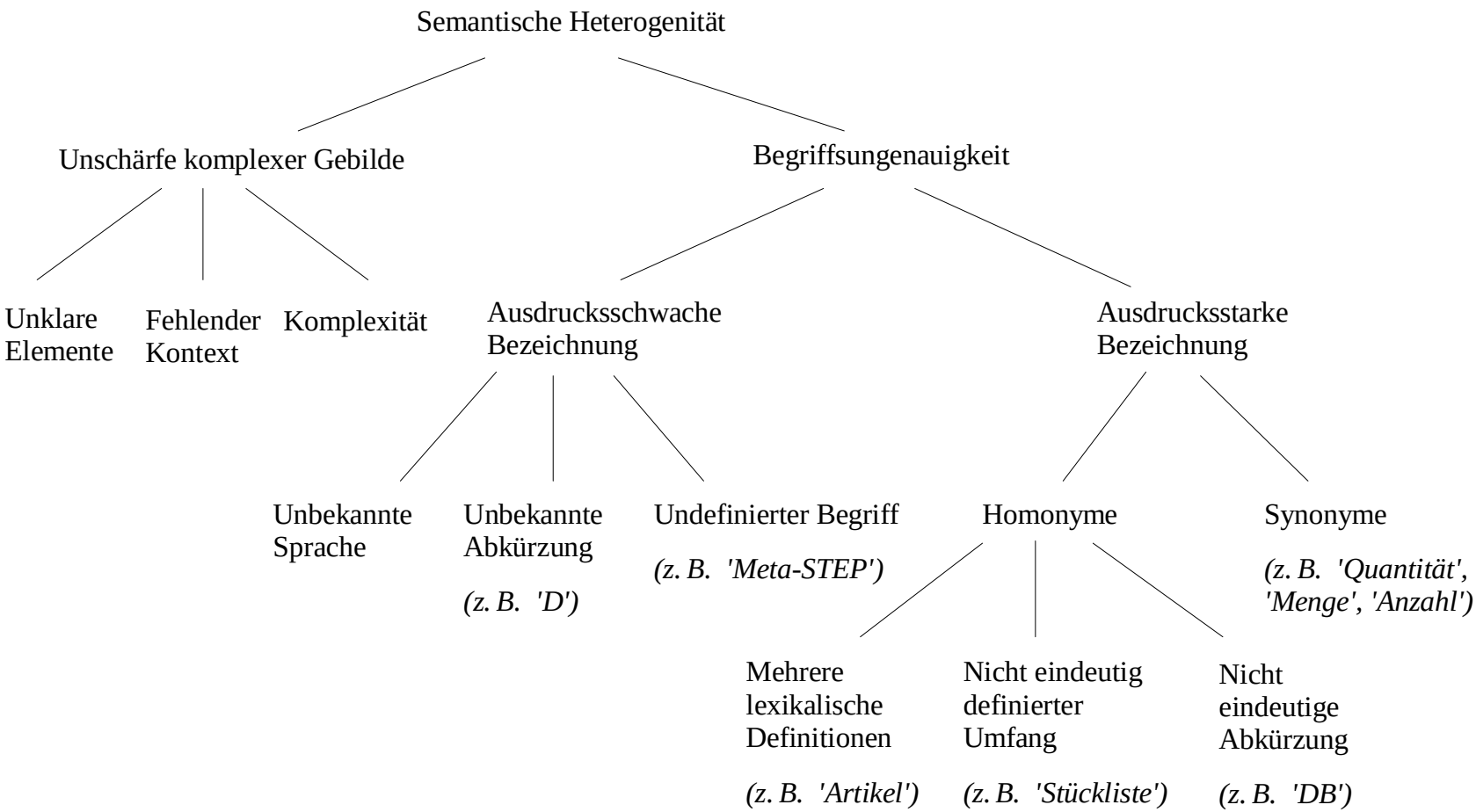


Datenmodellheterogenität

Verschiedene Datenmodelle

- Unterstützung unterschiedlicher Konzepte
  (z. B. Generalisierung – im objektorientierten Datenmodell unterstützt, im relationalen Datenmodell nicht)
- Auswirkungen auf die Struktur der Schemata
- Unterschiedliche Anfragesprachen

Semantische Heterogenität



Strukturelle Heterogenität

- Strukturelle Unterschiede aufgrund von Datenmodellheterogenität
- Modellierung des gleichen Sachverhaltes in unterschiedlichen Konzepten einesDatenmodells
  (auch *schematische Heterogenität* genannt)

  Beispiele im relationalen Datenmodell:

  Relation vs. Attribut:

Tabelle Männer (Schema A)

Vorname	Nachname
Peter	Meier

Tabelle Frauen (Schema A)

Vorname	Nachname
Eva	Klein

Tabelle Personen (Schema B)

Vorname	Nachname	männlich	weiblich
Peter	Meier	X	
Eva	Klein		X

Strukturelle Heterogenität (2)

Attribut vs. Wert eines Attributes:

Tabelle Personen (Schema A)

Vorname	**Nachname**	**männlich**	**weiblich**
Peter	Meier	X	
Eva	Klein		X

Tabelle Personen (Schema B)

Vorname	**Nachname**	**Geschlecht**
Peter	Meier	männlich
Eva	Klein	weiblich

Relation vs. Wert eines Attributes:

Tabelle Männer (Schema A)

Vorname	**Nachname**
Peter	Meier

Tabelle Frauen (Schema A)

Vorname	**Nachname**
Eva	Klein

Tabelle Personen (Schema B)

Vorname	**Nachname**	**Geschlecht**
Peter	Meier	männlich
Eva	Klein	weiblich

Strukturelle Heterogenität (3)

- Strukturelle Unterschiede bei der Modellierung des gleichen Sachverhaltes, trotz Verwendung gleicher Konzepte eines Datenmodells

  Beispiel im relationalen Datenmodell:

  Gruppierung der gleichen Attribute in unterschiedlich vielen Tabellen:

Tabelle Vorlesungen (Schema A)

V_ID	V_Name	V_Zeit	...	Prof_ID	Prof_Name	...

Tabelle Vorlesungen (Schema B)

ID	Name	...	Professor

Fremdschlüssel →

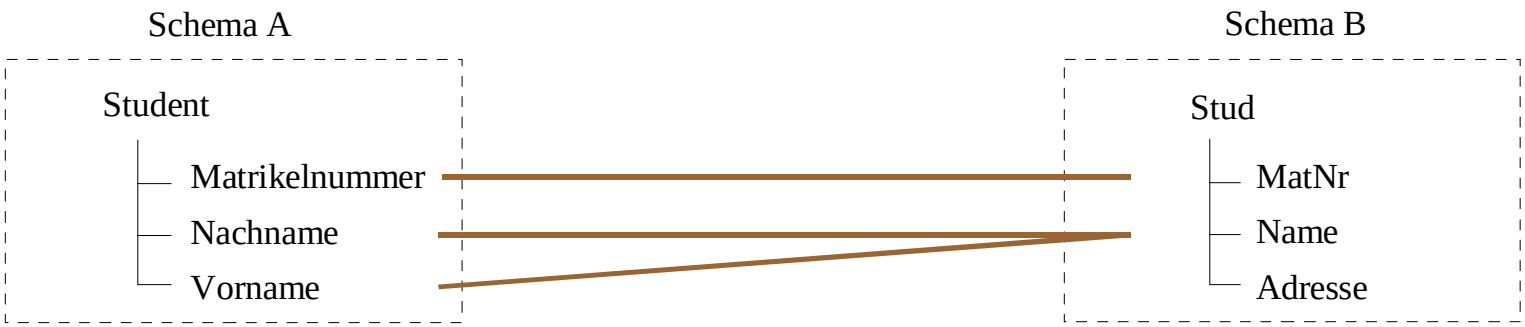
Tabelle Professoren (Schema B)

ID	Name	...

Schema Matching – Grundbegriffe

- *Schema-Matching-Verfahren*: Verfahren, zur Ermittlung von Korrespondenzen zwischen Elementen heterogener Schemata
- Input einer Match-Operation: zwei Schemata
- Output: *Mapping* bzw. Abbildung zwischen den beiden Schemata
- *Mapping-Element*: Korrespondenz zwischen Elementen der beiden Schemata
- *Mapping-Ausdruck*: Semantik eines Mapping-Elementes

Schema Matching – Grundbegriffe (Beispiel)



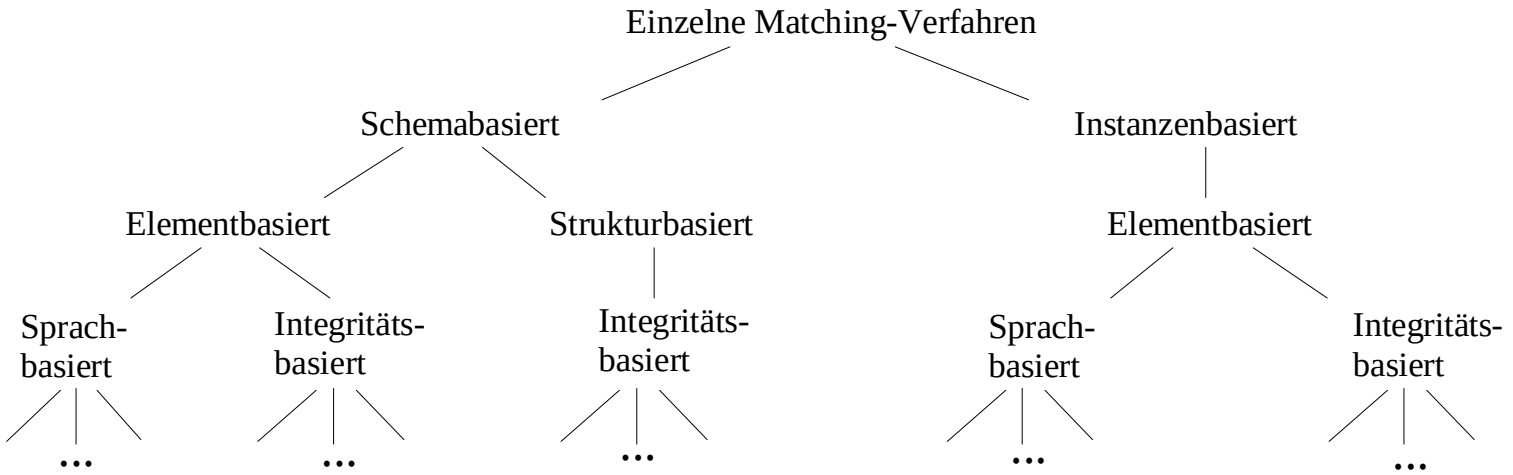
- *Mapping-Elemente*:

  „Student.Matrikelnummer ≅ Stud.MatNr“
  „{Student.Nachname, Student.Vorname} ≅ Stud.Name“
  (≅ *steht für Korrespondenz*)

- *Mapping-Ausdrücke*:

  “Student.Matrikelnummer = Stud.MatNr“
  “Concat(Student.Vorname, Student.Nachname) = Stud.Name“

Schema-Matching-Verfahren



- *Namen der Schemaelemente*
- *Beschreibungen der Schemaelemente*

- *Datentypen*
- *Wertebereiche*

- *Beziehungen zw. Schemaelementen (z. B. Aggregation, Assoziation, Generalisierung)*
- *Referenzen (z. B. Fremdschlüsselbeziehungen)*
- *Beziehungskardinalitäten*

- *Information-Retrieval-Techniken (z. B. Extrahieren von Schlüsselwörtern basierend auf der relativen Häufigkeit von Wörtern und Wörterkombinationen)*

- *Muster in strukturierten Daten (z. B. Telefonnummern, ISBN-Nummern)*
- *Wertebereiche*

Schema-Matching-Verfahren – Weitere Kriterien

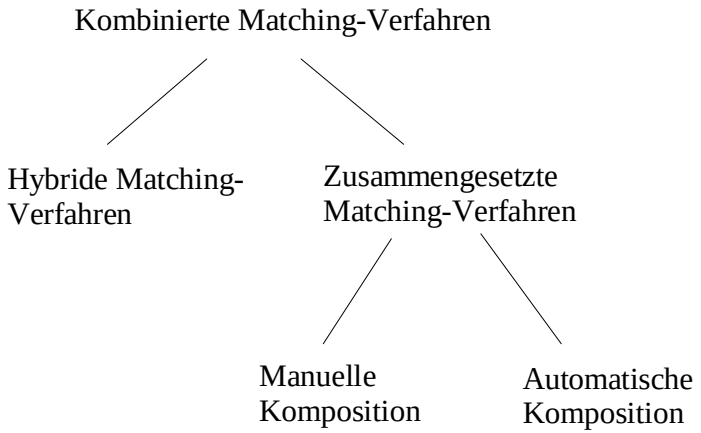
- *Verwendung zusätzlicher Information*

  (z. B. Wörterbücher, Thesauri, Benutzereingaben, bereits ermittelte Mappings)
- *Match-Kardinalität*

  Beispiele:

Kardinalität	**Element(e) aus Schema A**	**Element(e) aus Schema B**	**Mapping-Ausdruck**
1:1	Quantität	Menge	Quantität = Menge
1:n	Name	Vorname Nachname	Name = Concat(Vorname, Nachname)
n:1	Stundenlohn Monatsstunden	Monatslohn	Stundenlohn * Monatsstunden = Monatslohn
n:m	Buch Verlag	Buch.Titel Buch.ISBN Verlag.ISBN Verlag.Name	Buch, Verlag = select b.Titel, v.Name from Buch b, Verlag v where b.ISBN = v.ISBN

Kombinierte Schema-Matching-Verfahren

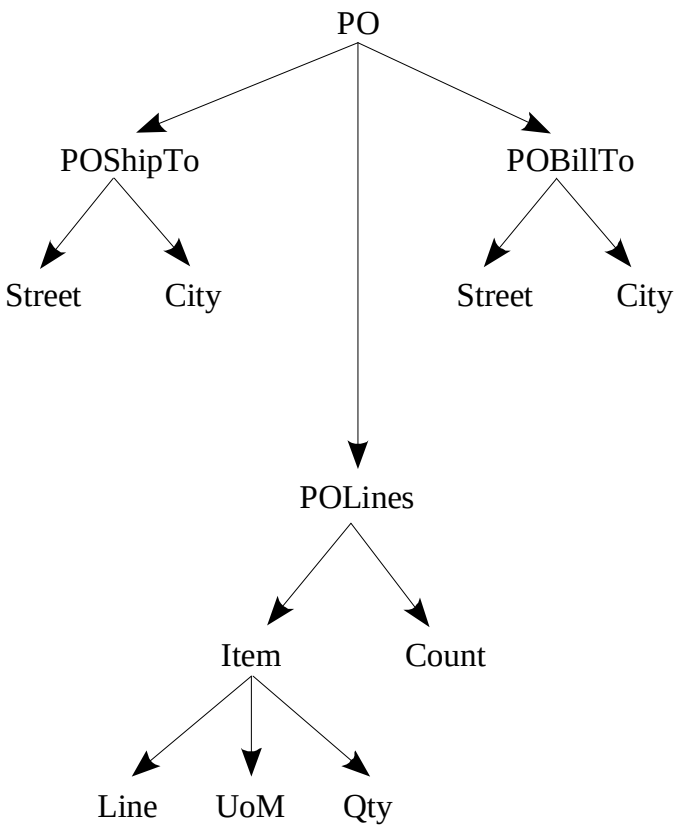


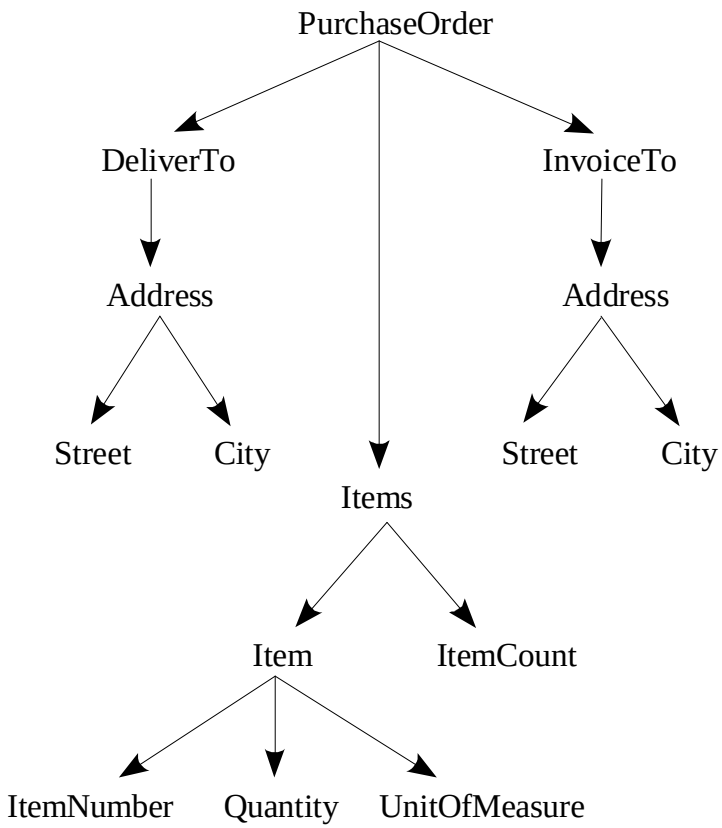
- *Hybride Matching-Verfahren*: Integration mehrerer einzelner Matching-Verfahren
- *Zusammengesetzte Matching-Verfahren*: Kombinierung der Ergebnisse mehrerer unabhängig durchgeführter Match-Algorithmen

Cupid – Eigenschaften

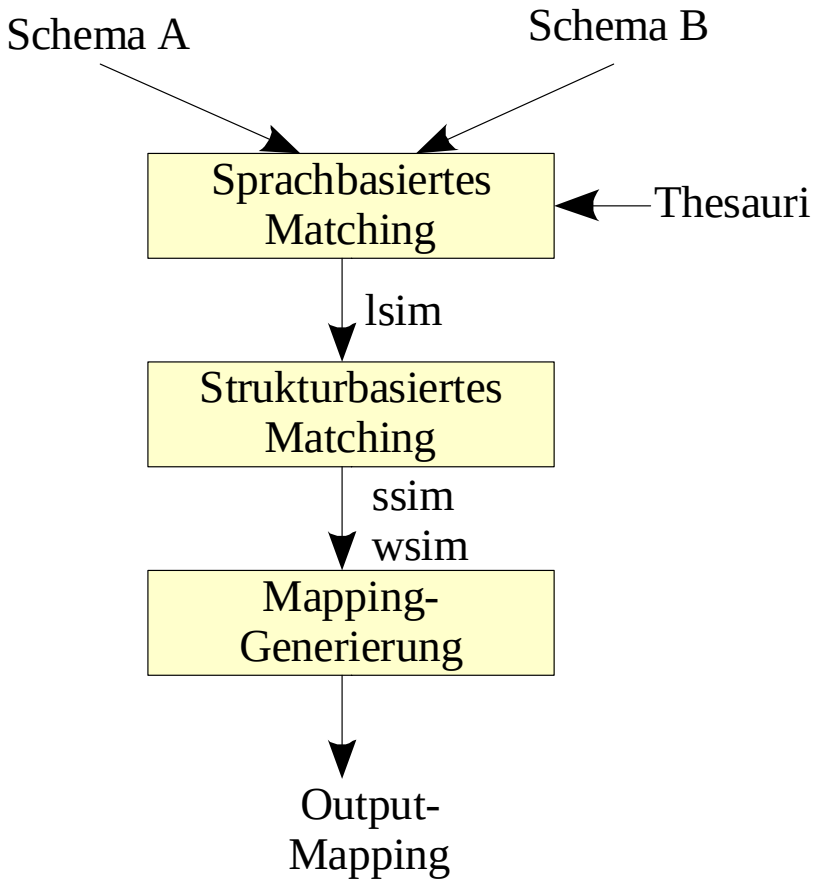
- Match-Algorithmus von Microsoft Research
- Schemabasiert
- Hybrid: element- und strukturbasiertes Matching
- Thesauri als zusätzliche Informationsquellen
- Keine Mapping-Ausdrücke
- 1:1 oder 1:n Mappings
- Generisch in Bezug auf Datenmodelle und Anwendungsbereiche

Cupid – Beispiel-Input-Schemata





Cupid – 3 Phasen



Cupid – Sprachbasiertes Matching

- Basiert auf den Namen der Schemaelemente
- 1. Schritt: Normalisierung
  - Zerlegen der Schemaelementnamen in einzelne Tokens
    (z. B. 'POBillTo' in {PO, Bill, To})
  - Ersetzen von Abkürzungen und Akronymen durch ihr ursprüngliches Wort
    (z. B. {PO, Bill, To} durch {Purchase, Order, Bill, To})
  - Markieren von Präpositionen, Artikeln, usw. als zu ignorierende Tokens
  - Markieren von Elementen, die Tokens enthalten, welche mit bekannten Konzepten in Beziehung stehen, mit den Konzeptnamen
    (z. B. 'Price' und 'Cost' mit 'Money')
  - Akronyme, Abkürzungen, zu ignorierende Wörter und Konzepte werden dabei mit Hilfe eines Thesaurus bestimmt.

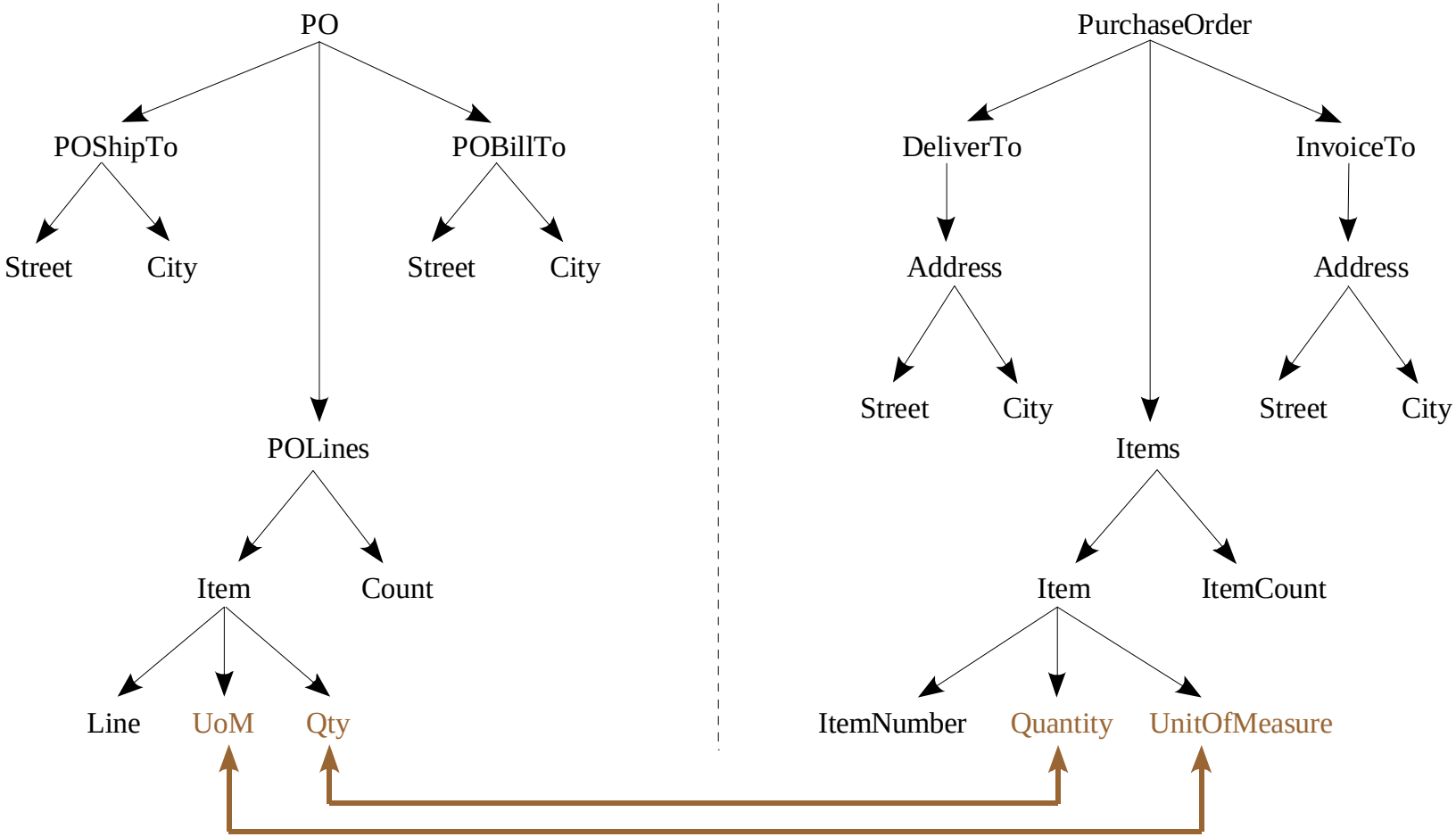
Cupid – Sprachbasiertes Matching (2)

- 2. Schritt: Kategorisierung
  - Kategorie: Gruppe von Schemaelementen, die durch eine Schlüsselwortmenge identifiziert wird
  - Anlegen einer Kategorie:
    Für jedes Konzept (z. B. 'Money')
    Für jeden Datentyp (z. B. 'Number')
    Für jedes Schemaelement, das andere Schemaelemente „enthält“
  - Kategorien für beide Schemata getrennt bestimmt
  - Elemente können mehreren Kategorien zugeordnet werden
  - Kategorisierung reduziert anschließende Element-zu-Element-Vergleiche

Cupid – Sprachbasiertes Matching (3)

- 3. Schritt: Vergleich
  - Vergleich der aus den Schemaelementnamen extrahierten Tokens mit Hilfe eines Thesaurus, der Synonyme und Hyperonyme für diesen Zweck enthält
  - Berechnung eines sprachlichen Ähnlichkeitskoeffizienten (*lsim*) für jedes Elementpaar aus kompatiblen Kategorien der zwei Schemata
- Ergebnis des sprachbasierten Matching:
  Tabelle mit *lsim*-Koeffizienten für Elementpaare der zwei Schemata ($lsim \in [0,1]$)

Cupid – Beispiel-Input-Schemata



Cupid – Strukturbasiertes Matching

- Umwandeln der ursprünglichen Schemata in Baumstrukturen
- Algorithmus:

```
1   TreeMatch(SourceTree S, TargetTree T)
2       for each s∈S, t∈T where s,t are leaves
3           set ssim (s,t) = datatype-compatibility(s,t)
4       S' = post-order(S), T' = post-order(T)
5       for each s in S'
6           for each t in T'
7               compute ssim(s,t) = structural-similarity(s,t)
8               wsim(s,t) = w_struct.ssim(s,t) + (1-w_struct).lsim (s,t)
9               if wsim(s,t) > th_high
10                  increase-struct-similarity(leaves(s),leaves(t),c_inc)
11              if wsim(s,t) < th_low
12                  decrease-struct-similarity(leaves(s),leaves(t),c_dec)
```

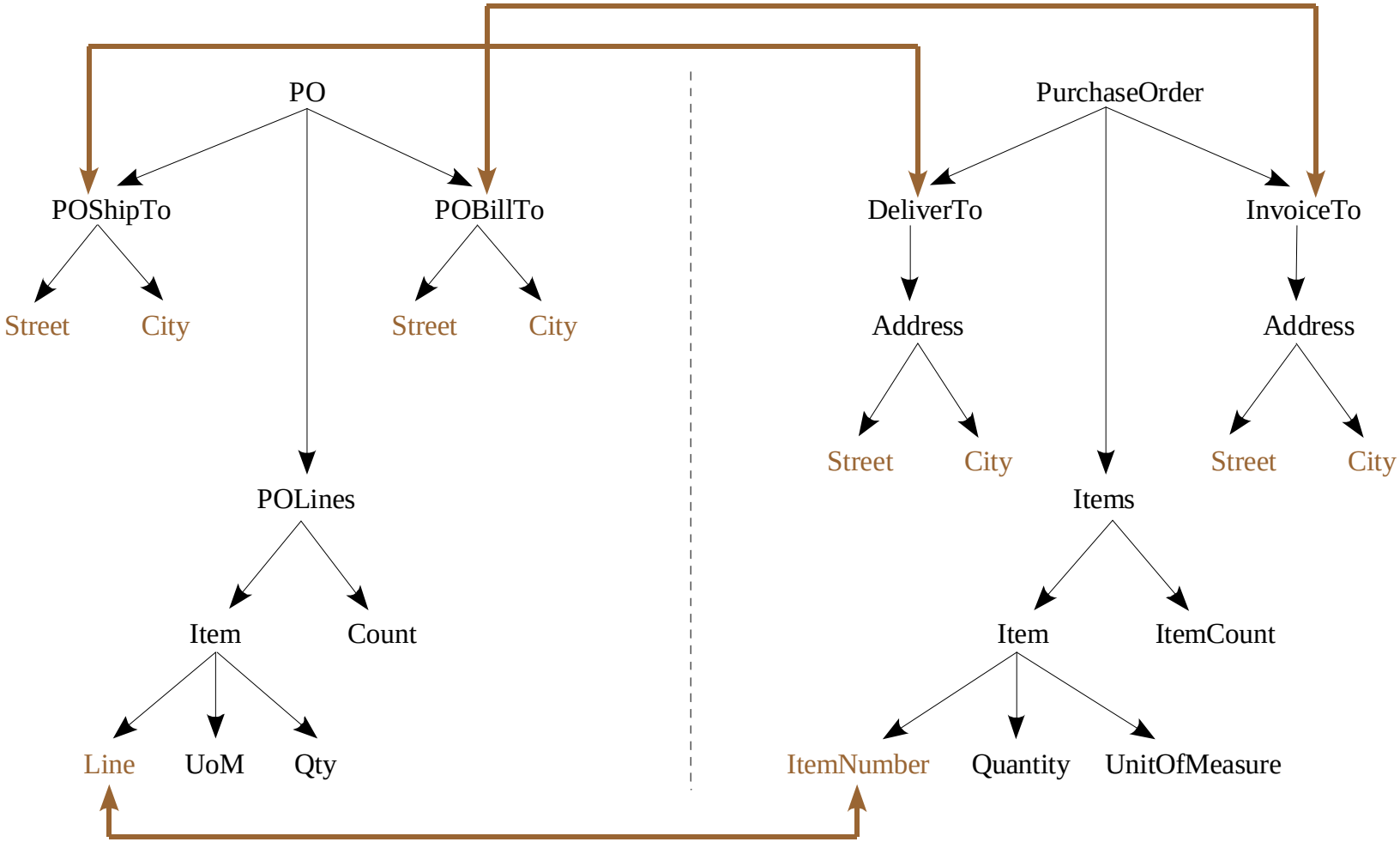
- Zeile 2 bis 3: Initialisieren des strukturellen Ähnlichkeitskoeffizienten *ssim* für die Blätterpaare mit Werten, die die Kompatibilität der Datentypen widerspiegeln (Kompatibilitätstabelle)
- Zeile 4: Sortieren der Elemente je Baum in Nachordnung
- Zeile 7: Neuberechnung des *ssim*-Koeffizienten für jedes Schemaelementpaar, das nicht nur aus Blättern besteht, über die Ähnlichkeit der Blätterpaare in den Unterbäumen der zwei Schemaelemente

Cupid – Strukturbasiertes Matching (2)

```
1   TreeMatch(SourceTree S, TargetTree T)
2       for each s∈S, t∈T where s,t are leaves
3           set ssim (s,t) = datatype-compatibility(s,t)
4       S' = post-order(S), T' = post-order(T)
5       for each s in S'
6           for each t in T'
7               compute ssim(s,t) = structural-similarity(s,t)
8               wsim(s,t) = w_struct.ssim(s,t) + (1-w_struct).lsim (s,t)
9               if wsim(s,t) > th_high
10                  increase-struct-similarity(leaves(s),leaves(t),c_inc)
11              if wsim(s,t) < th_low
12                  decrease-struct-similarity(leaves(s),leaves(t),c_dec)
```

- Zeile 8: Berechnung der gewichteten Ähnlichkeit *wsim*, durch Gewichtung und anschließende Addition der *ssim*- und *wsim*- Koeffizienten
- Zeile 9 bis 10: Falls *wsim*-Wert des Elementpaares eine gewisse Schwelle überschreitet folgt das Erhöhen des *ssim*-Wertes für jedes Blätter-Paar der beiden Unterbäume dieses Elementpaares (Berücksichtigung des strukturellen Kontextes)
- Zeile 11 bis 12: Analog

Cupid – Beispiel-Input-Schemata



Cupid – Mapping-Generierung

- Berücksichtigung nur der Schemaelementpaare, deren *wsim*-Wert einen bestimmten Schwellwert überschreitet
- Erstellen von Mapping-Elementen für Schemaelementpaare mit maximalen *wsim*-Werten

Fazit

- Automatische Auflösung der semantischen und strukturellen Heterogenität von Schemata ist schwierig
- Bereits semiautomatische Schema-Matching-Ansätze wie z. B. Cupid vorhanden
- Noch viel Forschungsbedarf
  (z. B. automatisches Anpassen von Schwellwerten, Wiederverwendung von gemeinsamen Schemaelementen und bereits ermittelten Mappings, Verwendung von Mapping-Ausdrücken)
- Model-Management-Systeme