Technische Universität Kaiserslautern
Fachbereich Informatik
Lehrgebiet Datenverwaltungssysteme

Seminar „Grundlagen webbasierter Informationssysteme“ der Arbeitsgruppen Datenbanken und Informationssysteme und Heterogene Informationssysteme im Sommersemester 2004

Formen der Heterogenität

Jessica Oksana Schnabel
j_schnab@informatik.uni-kl.de

Juli 2004
Betreut von Jernej Kovse

Inhaltsverzeichnis

1 Einleitung

Bereits seit vielen Jahren setzen Unternehmen Datenbanksysteme zur effizienten Speicherung und Verwaltung ihrer Daten ein. Durch die weltweite Vernetzung über das Internet oder private Netze ist es heutzutage technisch gesehen möglich, mit Daten aus der ganzen Welt zu arbeiten. Um auf Daten aus verschiedenen Quellen zugreifen zu können, müssen allerdings unterschiedliche Heterogenitätsformen der Datenbanksysteme überbrückt werden. Solche Heterogenitätsformen sind Thema dieser Ausarbeitung. In Abschnitt 2 stellen wir einige beispielhafte Szenarien vor, in denen die Überwindung der Heterogenität von Datenbanksystemen eine entscheidende Rolle spielt. In Abschnitt 3 schauen wir uns die Heterogenitätsformen genauer an, wobei der Schwerpunkt auf Heterogenitätsformen innerhalb der Datenbanksysteme gelegt wird. In Abschnitt 4 stellen wir abschließend einige Lösungsansätze vor und Abschnitt 5 enthält eine Zusammenfassung der wesentlichen Punkte dieser Ausarbeitung und einen Ausblick.

2 Motivation: Interoperabilität von Systemen

Die Kooperationsfähigkeit mehrerer Systeme trotz ihrer Heterogenität wird *Interoperabilität* der Systeme genannt. Im diesem Abschnitt schauen wir uns einige typische Szenarien an, in denen heterogene Informationssysteme miteinander kooperieren. In Abbildung 1 werden die dargestellten Szenarien anschaulich zusammengefasst.

2.1 Integration

Stellen wir uns zur Veranschaulichung einen Zusammenschluss zweier Unternehmen vor. Nehmen wir an, die beiden Unternehmen wollen alle Daten zusammenführen, um einen zentralen Zugriff zu gewährleisten. Die *Integration* der einzelnen heterogenen Datenbanksysteme stellt eine Möglichkeit dar, dieses Vorhaben zu realisieren. Hierbei bleiben die Daten in den bereits bestehenden Datenbanken erhalten und ein zentraler Zugriff auf all diese Daten wird über eine *integrierte Sicht* realisiert. *Globale Anwendungen* können somit über die integrierte Sicht auf die Daten aus den verteilten heterogenen Datenbanksystemen einheitlich zugreifen. In der integrierten Sicht wird ein globales Schema angelegt, welches durch die Integration der einzelnen heterogenen Schemata zu erstellen ist. Im Folgenden werden wir die Schwierigkeiten dieses Prozesses noch kennen lernen. Die einzelnen Quelldatenbanksysteme bleiben autonom, so dass *lokale Anwendungen* nicht eingeschränkt werden. Die Integration der Daten wird in der Literatur auch *Informationsintegration* genannt. Daneben können auch Anwendungssysteme integriert werden, was entsprechend *Anwendungsintegration* genannt wird. In dieser Ausarbeitung beschäftigen wir uns nur mit der Informationsintegration.

2.2 Migration

Eine andere Möglichkeit, Daten eines Systems (Quellsystem) einem zweiten System (Zielsystem) zur Verfügung zu stellen, besteht darin, diese aus dem *Quelldatenbanksystem* in das *Zieldatenbanksystem* zu migrieren. Hier gilt es, die Heterogenität zwischen Quell- und Zielsystem zu überbrücken. Dient die *Datenmigration* der Ablösung des Quellsystems durch das Zielsystem (wie z. B. bei der Ablösung von Legacy-Systemen), so müssen in der darauf folgenden Phase die Anwendungen des Quellsystems auf das Datenbanksystem der neuen Umgebung portiert werden. Diesen Vorgang nennt Sauter *Anwendungsprogramm-Migration* [Sau98]. Obwohl wir uns nicht näher damit beschäftigen, wollen wir allgemein anmerken, dass die Abbildung zwischen den Schemata des Quell- und des Zieldatenbanksystems, die für die Datenmigration erforderlich ist, bei der Anwendungsprogramm-Migration sehr hilfreich sein kann.

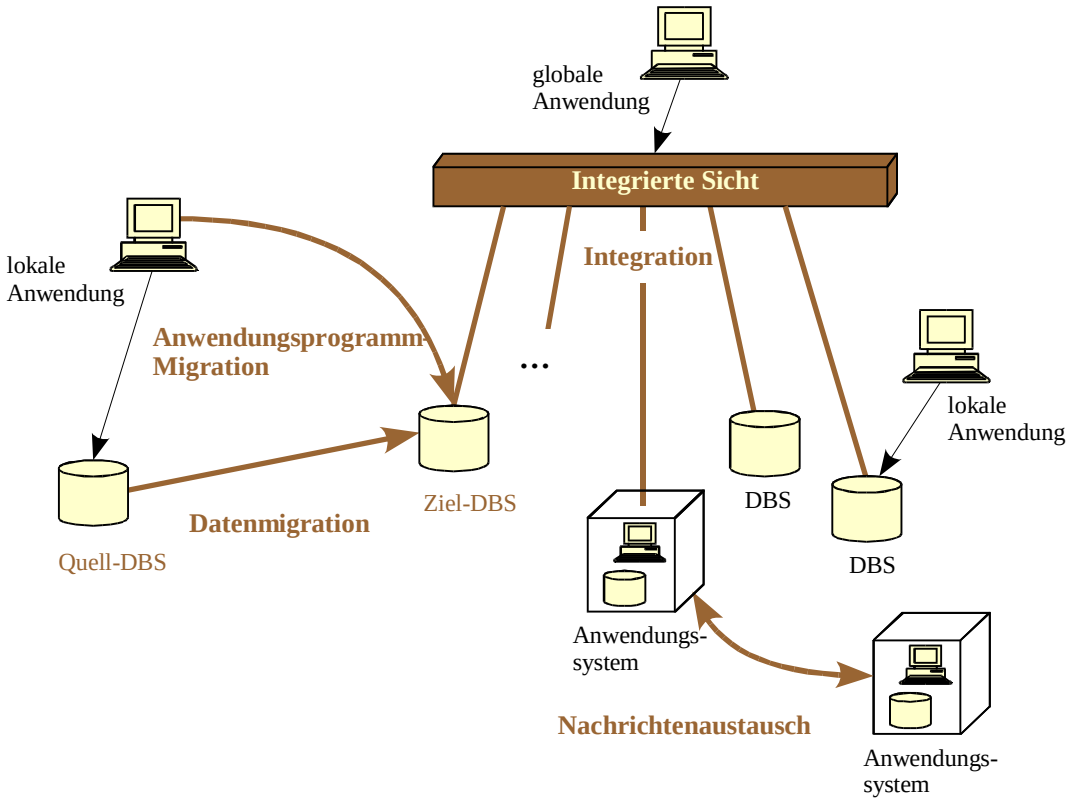

Abbildung 1: Interoperabilität von Systemen

2.3 Nachrichtenaustausch

Viele Handelspartner tauschen Daten über elektronische Nachrichten aus. Hier findet im Gegensatz zu den ersten zwei Szenarien eine direkte Kommunikation zwischen Anwendungen statt. Dabei könnten die Handelspartner ihre Daten in heterogenen Datenbanksystemen speichern und sich für den Austausch auf ein gemeinsames Schema einigen. In diesem Falle müsste die Heterogenität der Schemata, anhand derer die Handelspartner ihre Daten speichern, und dem Austauschschema bzw. dem Nachrichtenformat überwunden werden. Ein Beispielszenario wird in Abschnitt 4.2 in Zusammenhang mit RONDO vorgestellt.

3 Formen der Heterogenität bei der Interoperabilität von Systemen

Für die Unterstützung der verschiedenen Interoperabilitätsformen bildet die Analyse der vorherrschenden Heterogenität zwischen den einzelnen Systemen eine wichtige Grundlage. In der Literatur gibt es unterschiedliche Klassifikationen der Heterogenitätsformen. Hier soll der Leser einen Überblick über wichtige Erscheinungsformen der Heterogenität bekommen, wobei für unsere Betrachtungen vor allem Heterogenitätsformen innerhalb der Datenbanksysteme interessant sind.

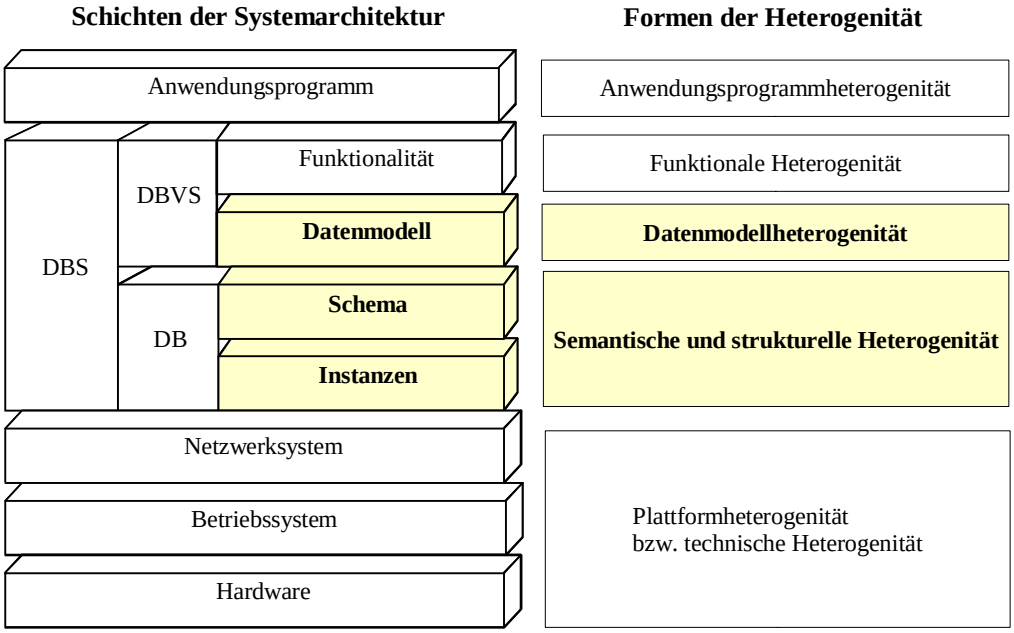

Abbildung 2: Formen der Heterogenität

3.1 Überblick

In Anlehnung an Sauter orientieren wir uns bei der Analyse der Erscheinungsformen der Heterogenität an den Schichten der Systemarchitektur [Sau98]. In Abbildung 2 werden diese anschaulich dargestellt und im Folgenden näher beschrieben.

Informationsverarbeitungsprogramme arbeiten auf Datenbanksystemen, denen Hardware, Betriebssysteme und Netzwerksysteme als Plattform dienen. Unterschiede in den drei unteren Schichten werden unter den Begriffen *Plattformheterogenität* oder *technische Heterogenität* zusammengefasst. Unterschiede in Anwendungsprogrammen (z. B. Verwendung unterschiedlicher Programmiersprachen, Compiler, Bibliotheken, usw.) werden analog als *Anwendungsprogrammheterogenität* bezeichnet. Ein Datenbanksystem (DBS) besteht aus einem Datenbankverwaltungssystem (DBVS) und der eigentlichen Datenbank (DB). Im Datenbankverwaltungssystem werden Funktionalitäten gemäß allgemeinen Konzepten von Datenbanksystemen bereitgestellt (z. B. Transaktionsverwaltung, Fehlerbehandlungsstrategien, Mehrbenutzerbetrieb, und so weiter). Unterschiede auf dieser Schicht werden als *funktionale Heterogenität* bezeichnet. Desweiteren wird im Datenbankverwaltungssystem das Datenmodell festgelegt (z. B. relational, objektorientiert, objekt-relational, hierarchisch, und so weiter). Der Begriff *Datenmodellheterogenität* drückt aus, dass in den Systemen unterschiedliche Datenmodelle verwendet werden. Die Datenbank enthält ein Schema, welches die Strukturierung aller in der Datenbank speicherbaren Datenobjekte des zugehörigen Anwendungsbereichs vorgibt, sowie Instanzen bzw. die Daten selbst. Auf diesen zwei Ebenen wird meist eine Unterscheidung der Heterogenitätsformen in *semantische* und *strukturelle Heterogenität* vorgenommen. Mit semantischer Heterogenität werden Unterschiede in der Bedeutung von Schemaelementen (z. B. Relationen und Attributen im relationalen Datenmodell), sowie der Daten selbst bezeichnet. Strukturelle Heterogenität bezieht sich auf unterschiedliche Strukturierungen der Schemata.

Wenn Daten aus heterogenen Systemen zusammengeführt werden sollen, gilt es vor allem Konflikte, die durch heterogene Datenmodelle, Modellierungen desselben Sachverhaltes in unter-

schiedlichen Schemata oder sich widersprechende bzw. fehlende Daten entstehen, zu überwinden. Da die Überbrückung dieser Heterogenitätsformen eine Grundlage für das Zusammenführen von Daten aus heterogenen Systemen bildet, werden diese im Folgenden genauer beschrieben und im Anschluss darauf Lösungsansätze dazu vorgestellt.

3.2 Datenmodellheterogenität

Unterschiedliche Datenmodelle beinhalten verschiedene Konzepte mit unterschiedlichen Bedeutungen. Diese zeigen sich deutlich in der unterschiedlichen Mächtigkeit der Datenmodelle (z. B. gegebene oder nicht gegebene Unterstützung von Generalisierung, Klassifikation, Assoziation, Aggregation, komplexen Objekten, und so weiter). Je größer die Unterschiede zwischen den verwendeten Datenmodellen sind, desto stärker unterscheidet sich die Struktur der darin modellierten Schemata. Das Konzept der Generalisierung wird z. B. im objektorientierten Datenmodell unterstützt, im relationalen Datenmodell dagegen nicht. Aufgrund dessen können Schemata den gleichen Sachverhalt der realen Welt darstellen, aber aufgrund der Verwendung dieser beiden heterogenen Datenmodelle strukturell sehr unterschiedlich sein. Desweiteren unterscheiden sich die Anfragesprachen unterschiedlicher Datenmodelle.

3.3 Semantische Heterogenität

Semantische Heterogenitätsformen treten in erster Linie auf der Ebene der Schemata auf, da beim Entwurf dieser anwendungsspezifisches Wissen einfließt, von keinem gemeinsamen Verständnis der modellierten Information ausgegangen werden kann und meist keine gemeinsame Begriffsbildung vorliegt (z. B. modellieren Ingenieure häufig anders als DB-Administratoren). Beim Zusammenführen von Daten aus heterogenen Systemen treten vor allem im Bereich der semantischen Heterogenität viele Unklarheiten auf. Stellen wir uns zur Verdeutlichung ein relationales Schema bestehend aus Relationen und Attributen vor. Jede Relation und jedes Attribut hat einen Namen und eine implizite Semantik. Da das Schema nur die Namen beinhaltet, muss durch Interpretation der Namen die zugehörige Semantik rekonstruiert werden. Dabei treten unterschiedliche Unklarheiten auf, die in Abbildung 3 klassifiziert werden.

Unklarheiten können bei einzelnen Begriffen (*Begriffsungenauigkeit*) oder in einem zusammenhängenden Gebilde (*Unschärfe komplexer Gebilde*) auftreten. Einzelne Begriffe, in denen Unklarheiten untersucht werden, können ausdrucksschwache oder ausdrucksstarke Bezeichnungen sein. Im Falle von *ausdrucksstarken Bezeichnungen* handelt es sich entweder um Synonyme oder um Homonyme. *Synonyme* sind unterschiedliche Begriffe mit einer gleichen oder ähnlichen Bedeutung (z. B. 'Quantität', 'Menge', 'Anzahl'). Ein *Homonym* ist ein Begriff, der mehrere Bedeutungen hat. Ursachen für Unklarheiten bei Homonymen können mehrere lexikalische Definitionen für den gleichen Begriff (z. B. 'Artikel' im Sinne von einem Zeitungsartikel oder im Sinne von einem Produkt), eine nicht eindeutige Abkürzung (z. B. 'DB' für eine Datenbank oder die Deutsche Bundesbahn) oder ein nicht eindeutig definierter Umfang des Begriffs (z. B. 'Stückliste' in Bezug auf einzelne Produkte oder Produktgruppen) sein. Bei ausdrucksstarken Bezeichnungen liegt ein gewisses Grundverständnis der Begriffe vor, so dass eine Interpretation prinzipiell möglich ist. Die Bedeutung einer *ausdrucksschwachen Bezeichnung* kann jedoch aufgrund eines undefinierten oder nicht ausreichend definierten Begriffs (z. B. 'Meta-STEP'), einer unbekannten Abkürzung (z. B. 'D') oder einer unbekannten Sprache nicht erfasst werden. Treten Begriffsungenauigkeiten im Kontext ansonsten klarer Bezeichnungen auf, so können diese meist behoben werden. Anders ist es bei *komplexen Gebilden* (wie z. B. einem Schema als Ganzes), welche aufgrund von ungenauem Verständnis einzelner zentraler Begriffe (siehe Begriffsungenauigkeit), fehlendem Zusammenhang zwischen Elementen oder der Undurchschaubarkeit des Gebildes an sich nicht exakt verstanden werden können.

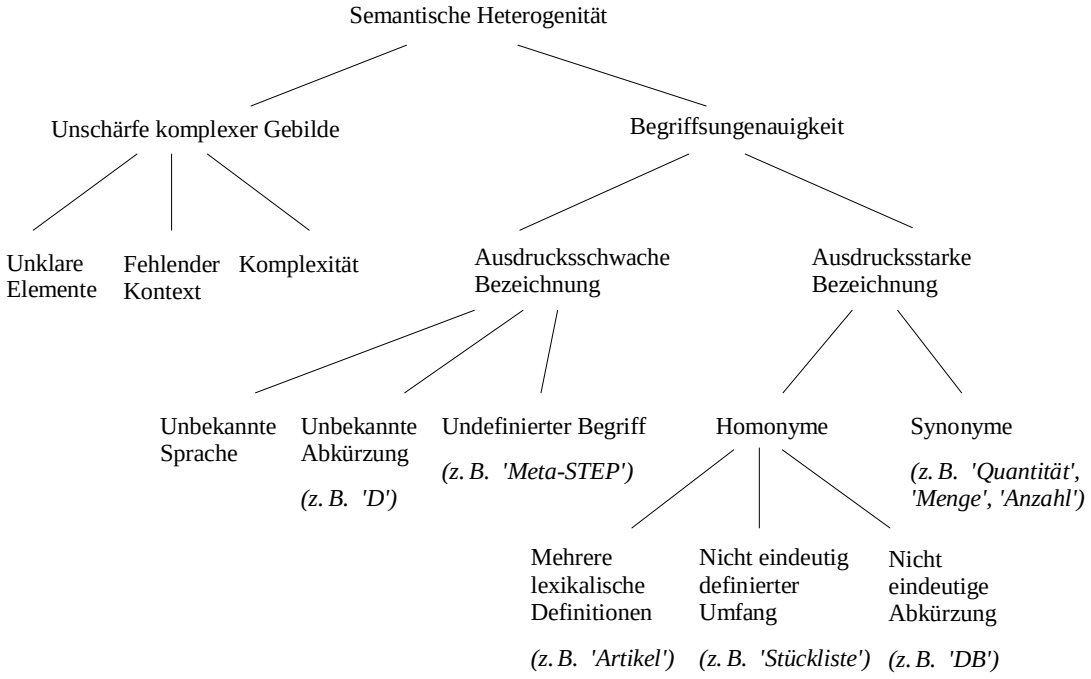

Abbildung 3: Klassifikation semantischer Heterogenität (nach [Sau98])

Mit dem Begriff *Schemaelemente* wollen wir alle Komponenten, die ein beliebiges Schema beinhalten kann (wie z. B. Tabellen, Spalten, Klassen, XML-Elemente usw.) bezeichnen. Die Bedeutung der Daten wird durch die Schemaelemente, denen sie zugewiesen sind, definiert. Das bedeutet, dass die Schemata die Hauptquelle für die Analyse der Semantik der Daten darstellen. Aufgrund der Unklarheiten, die dabei auftreten können, ist diese Analyse allerdings nicht immer erfolgreich. Schlechte oder gar nicht vorhandene Dokumentation fördert diese Problematik. Oftmals müssen deswegen zur Klärung der Semantik von Schemaelementen die zugehörigen Daten analysiert werden. Ein weiterer semantischer Konflikt entsteht, wenn die Daten eines Schemas von unterschiedlichen Definitionsbereichen (z. B. Angabe von Preisen in unterschiedlichen Währungen) ausgehen[1]. Auch hier könnte die Analyse der Daten erforderlich werden.

Die Überbrückung der semantischen Heterogenität kann aufgrund der zahlreichen Unklarheiten nicht komplett automatisiert werden. Um jegliche Missinterpretation der modellierten Information zu vermeiden, ist die Erarbeitung einer gemeinsamen Begriffsdefinition der beteiligten Benutzergruppen meist unumgänglich. Es ist jedoch sehr wünschenswert, automatische Verfahren zur Unterstützung heranziehen zu können, die einen möglichst großen Teil zur Überbrückung der Heterogenität beitragen.

3.4 Strukturelle Heterogenität

In Abschnitt 3.2 wurde bereits beschrieben, dass die Verwendung unterschiedlicher Datenmodelle eine starke Auswirkung bzgl. der strukturellen Heterogenität auf die Schemata hat. Aber auch Schemata des gleichen Datenmodells weisen bei unabhängiger Entwicklung meist eine unterschiedliche Struktur auf. Die folgenden Beispiele sollen einige dieser Unterschiede anhand des relationalen Modells verdeutlichen.

[1] Diese Heterogenitätsform wird in der Literatur auch *Repräsentationsheterogenität* genannt [BKLW99].

Der gleiche Sachverhalt kann in unterschiedlichen Elementen des Datenmodells modelliert sein. Zur Veranschaulichung stellen wir uns die Unterscheidung von Personen nach ihrem Geschlecht als Sachverhalt vor. Die Bespiele dazu sind an [Nau04] angelehnt. Das erste Beispiel soll die Modellierung des Sachverhaltes in einer *Relation* vs. der Modellierung des gleichen Sachverhaltes in einem *Attribut* verdeutlichen.

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

In Schema A findet die Unterscheidung des Geschlechtes der Personen durch Verwendung von zwei Tabellen statt, wobei in einer Tabelle die männlichen und in der anderen Tabelle die weiblichen Personen gespeichert werden. In Schema B hingegen ist für männliche und weibliche Personen nur eine Tabelle vorgesehen, wobei die Unterscheidung zwischen männlich und weiblich in Attributen dieser Tabelle stattfindet. Das nächste Beispiel dient der Verdeutlichung der Modellierung des Sachverhaltes als *Attribut* vs. der Modellierung dessen als *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

In Schema A wird das Geschlecht der Personen durch zwei entsprechende Attribute modelliert. Das jeweils zutreffende Attribut soll für die einzelnen Tupel mit einem X versehen werden. Schema B enthält nur ein Attribut, welches das Geschlecht der Personen modellieren soll. Die Unterscheidung findet anhand der eingetragenen Werte für die einzelnen Tupel statt. Ein weiteres Beispiel zeigt die Modellierung des Sachverhaltes in einer *Relation* vs. der Modellierung des gleichen Sachverhaltes als *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

In Schema A findet (wie schon im ersten Beispiel) die Unterscheidung des Geschlechtes der Personen anhand von zwei Tabellen statt. In Schema B wird das Geschlecht in den Werten, die in der dafür vorgesehenen Spalte einzutragen sind, unterschieden.

Die Verwendung unterschiedlicher Elemente des Datenmodells zur Modellierung des gleichen Sachverhaltes wird in der Literatur unter dem Begriff *schematische Heterogenität* zusammengefasst. Diese wurde in den obigen Beispielen veranschaulicht. In dieser Ausarbeitung wird diese Heterogenitätsform im Rahmen der strukturellen Heterogenität vorgestellt, da sie als eine Spezialform dieser Heterogenität aufgefasst werden kann. Bei der Unterscheidung zwischen schematischer und struktureller Heterogenität bezeichnet *strukturelle Heterogenität* unterschiedliche Strukturierung bei der Modellierung des gleichen Sachverhaltes, trotz Verwendung gleicher Elemente des Datenmodells. Beispielsweise können im relationalen Modell die gleichen Attribute in unterschiedlich vielen Tabellen gruppiert sein. Das kann z. B. aufgrund eines unterschiedlichen Normalisierungsgrades vorkommen. Im Folgenden ist ein anschauliches Beispiel dazu zu sehen.

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 A sieht vor, zu jeder Vorlesung den Professor mit all seinen Attributen abzuspeichern. In Schema B dagegen werden Vorlesungen und Professoren in unterschiedlichen Tabellen repräsentiert und die Beziehung durch einen Fremdschlüssel realisiert. Diese Schemata haben einen unterschiedlichen Normalisierungsgrad.

4 Lösungsansätze zur Überwindung von Heterogenität

In diesem Abschnitt befassen wir uns damit, wie Korrespondenzen zwischen unterschiedlichen Schemata trotz semantischer und struktureller Heterogenität automatisch entdeckt werden können.

4.1 Schema Matching

In Abschnitt 2 haben wir einige Szenarien gesehen, in denen Korrespondenzen zwischen heterogenen Schemata ermittelt werden müssen. Bei der Schema-Integration gilt es, ähnliche Strukturen in unterschiedlichen Schemata zu finden, welche dann als Integrationspunkte dienen. Für die Migration von Daten (wie dies z. B. in einem Data Warehouse geschieht) muss eine Abbildung zwischen den einzelnen Quellschemata und dem Zielschema ermittelt werden. Für den Austausch von elektronischen Nachrichten verwenden Handelspartner ein gemeinsames Austauschschema. Hier ist es erforderlich Korrespondenzen zwischen den Schemata, in denen die Handelspartner ihre Daten speichern, und dem Austauschschema, in dem die Daten versendet werden, zu finden. Automatische Verfahren, die nach Korrespondenzen zwischen heterogenen Schemata suchen, werden *Schema-Matching-Verfahren* genannt. Eine große Herausforderung für solche Verfahren ist das Überwinden der semantischen und der strukturellen Heterogenität der Schemata. Im Folgenden wird Schema-Matching diskutiert und beispielhaft der Algorithmus Cupid vorgestellt.

Abbildung 4: Beispiel-Input-Schemata für Match

4.1.1 Grundbegriffe

Match ist eine Operation, die zwei Schemata als Input bekommt und als Output eine Abbildung zurückgibt, welche Korrespondenzen zwischen Elementen der beiden Schemata identifiziert. Diese Abbildung wird in der Literatur *Mapping* genannt. Ein Mapping besteht aus mehreren *Mapping-Elementen*, wobei jedes dieser Mapping-Elemente angibt, *dass* ein oder mehrere Elemente des einen Schemas mit einem oder mehreren Elementen des anderen Schemas korrespondieren. Desweiteren kann jedes Mapping-Element einen *Mapping-Ausdruck*[2] besitzen, welcher spezifiziert, *wie* die entsprechenden Schemaelemente miteinander korrespondieren. Ein Mapping-Ausdruck kann gerichtet (z. B. eine bestimmte Funktion von den Elementen aus dem einen Schema zu den Elementen aus dem anderen Schema) oder ungerichtet (also eine Beziehung zwischen einer Kombination von Elementen der beiden Schemata) sein. In den Mapping-Ausdrücken können Operatoren (wie z. B. $=$, $\leq$), Funktionen (wie z. B. Addition, Konkatenation), Beziehungen (wie z. B. 'is-a', 'part-of', 'overlaps', 'contains') oder beliebige andere Ausdrücke vorkommen.

Die meisten bisher existierenden Match-Algorithmen machen von Mapping-Ausdrücken keinen Gebrauch. Beispielsweise würde ein Match-Algorithmus bei Anwendung auf die Schemata A und B aus Abbildung 4 folgendes Ergebnis liefern: „Student.Matrikelnummer $\cong$ Stud.MatNr“, „{Student.Nachname, Student.Vorname} $\cong$ Stud.Name“ (wobei $\cong$ für Korrespondenz steht). Ein vollständig spezifiziertes Ergebnis eines Match-Aufrufs würde aber für jedes Mapping-Element auch einen Mapping-Ausdruck beinhalten. Die Mapping-Ausdrücke wären in diesem Beispiel “Student.Matrikelnummer = Stud.MatNr“ und „Concat(Student.Vorname, Student.Nachname) = Stud.Name“.

Da Schema Matching für viele unterschiedliche Anwendungen von Bedeutung ist, sollte es als eine generische, selbständige Komponente entwickelt werden.

4.1.2 Klassifikation der Schema-Matching-Verfahren

In Abbildung 5 ist eine Klassifikation der Schema-Matching-Verfahren zu sehen. Diese gibt einen guten Überblick über unterschiedliche Vorgehensweisen zur Entdeckung von Korrespondenzen zwischen Elementen voneinander verschiedener Schemata.

Es wird zwischen einzelnen und kombinierten Matching-Verfahren unterschieden. *Einzelne Matching-Verfahren* können schema- oder instanzenbasiert sein. *Schemabasierte Verfahren* verwenden nur Informationen aus den Schemata, nicht aus den Daten selbst. Als Informationen können hier Namen, Beschreibungen und Datentypen von Schemaelementen, Arten von Beziehungen (z. B. 'part-of', 'is-a', usw.) zwischen Schemaelementen, Integritätsbedingungen und Strukturen der Schemata dienen. Doch auch die Daten selbst können einen wichtigen Beitrag zum Verstehen der Bedeutung von Schemaelementen leisten. Das ist vor allem dann der Fall, wenn nicht genügend Information aus den Schemata abgeleitet werden kann (wie z. B. bei se-

[2]Einige beispielhafte Mapping-Ausdrücke sind in Abbildung 6 auf Seite 12 zu sehen.

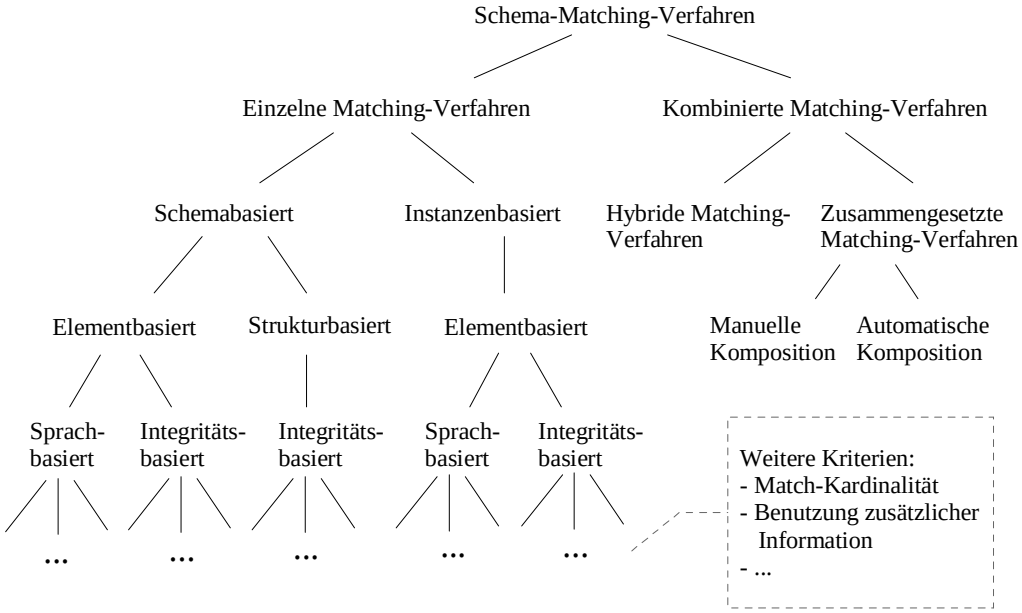

Abbildung 5: Klassifikation der Schema-Matching-Verfahren (nach [RB01])

mistrukturierten Daten). Auch wenn die Schemata genügend Information enthalten, können *instanzenbasierte Verfahren* zum Erkennen falscher Interpretation der Informationen aus den Schemata verwendet werden. Auf der nächsttieferen Ebene findet eine Unterscheidung nach der Granularität der Matching-Verfahren statt. Bei *elementbasiertem Matching* werden einzelne Elemente der Schemata isoliert betrachtet, d. h. der strukturelle Kontext, in den diese eingebettet sind, wird außer acht gelassen. *Struturbasiertes Matching* zielt dagegen auf das Entdecken von Korrespondenzen zwischen Strukturen, d. h. Elemente der Schemata werden in ihrem strukturellen Kontext betrachtet. Die nächsttiefere Ebene unterscheidet zwischen sprach- und integritätsbasierten Verfahren *Sprachbasierte Verfahren* können Namen der Schemaelemente, sowie Beschreibungen dieser in Form von Text benutzen, um semantische Korrespondenzen zwischen Schemaelementen zu finden. Schemata beinhalten oft Integritätsbedingungen zur Definition von Datentypen, Wertebereichen, Optionalität, Typen von Beziehungen, Kardinalitäten, und so weiter. Wenn beide Schemata solche Informationen beinhalten, können diese in *integritätsbasierten Verfahren* zum Entdecken von Ähnlichkeiten verwendet werden. Ähnlichkeiten können hierbei z. B. auf der Äquivalenz von Datentypen und Wertebereichen, Schlüsseleigenschaften (wie z. B. Primärschlüssel, Fremdschlüssel) und Beziehungskardinalitäten (z. B. 1:1-Beziehung) basieren. Werden integritätsbasierte Verfahren für sich alleine ausgeführt, so geben diese oft unpassende n:m- Korrespondenzen zurück, da mehrere Elemente in einem Schema ähnliche bzw. gleiche Integritätsbedingungen haben können. Solche Verfahren sind aber in Kombination mit anderen Marching-Verfahren (wie z. B. sprachbasierten Verfahren) sehr nützlich. Hier können diese z. B. zur Reduktion der Anzahl von weiteren Vergleichen verwendet werden. Integritätsbedingungen wie Referenzen innerhalb eines Schemas (z. B. Fremdschlüssel) und nahe verwandte Information (z. B. Beziehungen wie 'part-of') können als Strukturen interpretiert und deshalb in strukturbasierten Matching-Verfahren verwendet werden. *Sprachbasierte Charakterisierung* der Daten bei instanzenbasierten Matching-Verfahren kann z. B. bei Text angewendet werden. Eine solche Charakterisierung basiert auf Information-Retrieval-Techniken, wie z. B. dem Extrahieren von Schlüsselwörtern und Themen basierend auf der relativen Häufigkeit von Wörtern und Wörter-

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

Abbildung 6: Beispiele für Match-Kardinalitäten

Kombinationen. Auf strukturiertere Daten (z. B. numerische Daten) kann eine *integritätsbasierte Charakterisierung* angewandt werden, die z. B. Muster entdeckt. So können z. B. Telefonnummer und ISBN-Nummern anhand ihres spezifischen Musters erkannt werden.

Zusätzlich können Matching-Verfahren nach der *Kardinalität* ihrer Ergebnisse unterschieden werden. Es können 1:1-, 1:n-, n:1- und n:m-Korrespondenzen zwischen Schemaelementen ermittelt werden. Abbildung 6 soll dies an einigen Beispielen verdeutlichen. Die meisten Verfahren bilden jedes Element des einen Schemas auf jenes Element des anderen Schemas ab, für welches die höchste Ähnlichkeit berechnet wurde. Somit werden für einzelne Schemaelement-Paare 1:1-Mappings erstellt, wobei im Ganzen meist mehrere Schemaelemente auf ein Schemaelement abgebildet werden können.

Die meisten Match-Algorithmen verwenden nicht nur die zwei Input-Schemata, sondern auch noch zusätzliche Informationen, wie z. B. Wörterbücher, Thesauri[3] und Benutzereingaben. Auch das Wiederverwenden von gemeinsamen Schemaelementen und bereits ermittelten Mappings ist in Anbetracht der Tatsache, dass Abbildungen zwischen sehr vielen Schemata erstellt werden sollen und dass Schemata oft sehr ähnlich sind, vielversprechend. Beispielsweise wiederholen sich im E-Commerce-Bereich viele Teilstrukturen innerhalb unterschiedlicher Nachrichtenformate (wie z. B. Namens- und Addressfelder).

Wie wir gesehen haben, gibt es unterschiedliche Verfahren, die ein Match-Algorithmus implementieren kann. Jedes dieser Verfahren benutzt unterschiedliche Informationen und ist deshalb je nach Anwendungsgebiet mehr der weniger geeignet. Ein Algorithmus kann aber auch mehrere Verfahren miteinander kombinieren, um so bessere Ergebnisse zu erzielen. Solche *kombinierten Matching-Verfahren* werden in zwei Typen unterteilt: *hybride Matching-Verfahren*, welche mehrere einzelne Matching-Verfahren integrieren, und *zusammengesetzte Matching-Verfahren*, welche die Ergebnisse unabhängig durchgeführter Match-Algorithmen kombinieren. Die Kombination mehrerer Matching-Verfahren führt im Vergleich zu einzelnen Verfahren zu genaueren Ergebnissen, da hier mehr Informationen genutzt werden. Zusätzlich kann in hybriden Verfahren durch frühzeitiges Herausfiltern von Schemaelement-Paaren, die nur in einem von mehreren Kriterien korrespondieren, die Effektivität gesteigert werden. Ein Vorteil, den zusammengesetzte Verfahren mit sich bringen, ist ihre Flexibilität. Sie ermöglichen die Auswahl aus einem Repertoire von einzelnen Matching-Verfahren (z. B. für unterschiedliche Anwendungsbereiche) und eine flexible Anordnung der einzelnen Verfahren (wodurch z. B. die Auswahl zwischen einer parallelen oder einer sequentiellen Ausführung möglich wird). Die Auswahl der einzelnen Verfahren, das Festlegen ihrer Ausführungsreihenfolge und die Kombination der separat bestimmten Ergebnisse kann entweder *automatisch* durch Implementierung im Match-Algorithmus oder von einem Benutzer

[3]Ein *Thesaurus* ist ein Wörterbuch für sinnesverwandte Wörter.

manuell durchgeführt werden. Der manuelle Ansatz ist leichter zu implementieren (da Match generisch gegenüber unterschiedlichen Anwendungsbereichen sein soll) und gibt dem Benutzer mehr Kontrolle. Die Interaktion mit einem Benutzer ist ohnehin unerlässlich, da Match-Algorithmen nur Match-Vorschläge machen können, welche der Benutzer dann akzeptieren, ablehnen oder abändern kann.

Die obige Beschreibung unterschiedlicher Matching-Verfahren soll dem Leser einen Überblick geben. Eine detailliertere Beschreibung, sowie ein Vergleich mehrerer Match-Algorithmen (wie z. B. SemInt, LSD, SKAT, DIKE, Cupid) kann in [RB01] nachgelesen werden. Im Folgenden wollen wir uns einen Match-Algorithmus namens *Cupid* genauer ansehen.

4.1.3 Beispielalgorithmus Cupid

Cupid ist ein *schemabasierter Match-Algorithmus* von Microsoft Research. Er ist in die Kategorie der *hybriden Match-Algorithmen* einzuordnen, da er sowohl *element-* als auch *strukturbasiertes Matching* implementiert. Als erstes findet ein *sprachbasiertes Matching* (basierend auf Schemaelementnamen) mit *integritätsbasierten* Einflüssen (Verwendung von Datentypen und 'part-of'-Beziehungen) auf der Elementebene statt. Daraufhin wird jedes der beiden Schemata in eine Baumstruktur umgewandelt, auf der dann ein Matching auf Strukturebene stattfindet. Cupid verwendet Thesauri zur Bestimmung von Synonymen[4], Abkürzungen, Akronymen[5] und Hyperonymen[6] als *zusätzliche Informationsquellen*. *Mapping-Ausdrücke* werden in diesem Algorithmus nicht verwendet. Es werden 1:1- oder 1:n-Mappings erstellt. Cupid ist darauf eingestellt, Korrespondenzen zwischen atomaren Elementen (z. B. Blätter einer Baumstruktur), in denen viel Schema-Semantik erfasst ist, zu finden. Desweiteren ist Cupid *generisch* in Bezug auf unterschiedliche Datenmodelle und Anwendungsbereiche.

Der Algorithmus wird fortlaufend an den Beispiel-XML-Schemata 'PO' und 'PurchaseOrder' aus Abbildung 7 veranschaulicht. Die Schemata sind als Graphen repräsentiert, deren Knoten Elemente der Schemata darstellen. Obwohl einem Betrachter schnell Ähnlichkeiten zwischen den beiden Schemata auffallen, sind diese Ähnlichkeiten aufgrund der strukturellen und der semantischen Heterogenität der beiden Schemata algorithmisch nur schwer zu entdecken.

Cupid erstellt eine Abbildung zwischen Schemaelementen der zwei Input-Schemata in *drei Phasen*, die im Folgenden in Anlehnung an [MBR01] beschrieben werden.

In der ersten Phase findet das *sprachbasierte Matching* mit integritätsbasierten Einflüssen auf Elementebene statt. Sich entsprechende Elemente unterschiedlicher Schemata haben oft semantisch äquivalente Namen, die sich aber syntaktisch aufgrund von Abkürzungen (z. B. 'Qty' für 'Quantity'), Akronymen (z. B. 'UoM' für 'UnitOfMeasure'), unterschiedlicher Zeichensetzung, Synonymen (z. B. 'Bill' und 'Invoice'), usw. unterscheiden.

Um die Namen der einzelnen Elemente vergleichbar zu machen, findet eine *Normalisierung* dieser folgendermaßen statt: Der Name jedes einzelnen Elementes wird mit Hilfe eines konfigurierbaren Tokenizers in eine Menge von einzelnen Tokens zerlegt (z. B. 'POBillTo' in {PO, Bill, To}). Der Tokenizer verwendet z. B. Großbuchstaben, Punkte, Sonderzeichen, usw. zur Identifikation der einzelnen Tokens. Als nächstes werden Abkürzungen und Akronyme durch ihr ursprüngliches Wort ersetzt (z. B. {PO, Bill, To} durch {Purchase, Order, Bill, To}). Präpositionen, Artikel, usw. werden als Tokens markiert, die während dem Vergleich ignoriert werden sollen. Elemente, die ein Token haben, welches mit einem bekannten *Konzept* in Beziehung steht, werden mit dem

[4] Ein *Synonym* ist ein Wort, das einem anderen Wort in Bezug auf die Bedeutung ähnlich oder gleich ist.

[5] Ein *Akronym* ist ein aus den Anfangsbuchstaben mehrerer Wörter gebildetes Kurzwort (z. B. 'EDV' aus 'elektronische Datenverarbeitung')

[6] Ein *Hyperonym* ist ein übergeordneter Begriff bzw. ein Wort, das in einer übergeordneten Beziehung zu einem oder mehreren anderen Wörtern steht und inhaltlich allgemeiner als diese ist (z. B. 'Medikament' zu 'Pille', 'Tablette', 'Kapsel', und so weiter).

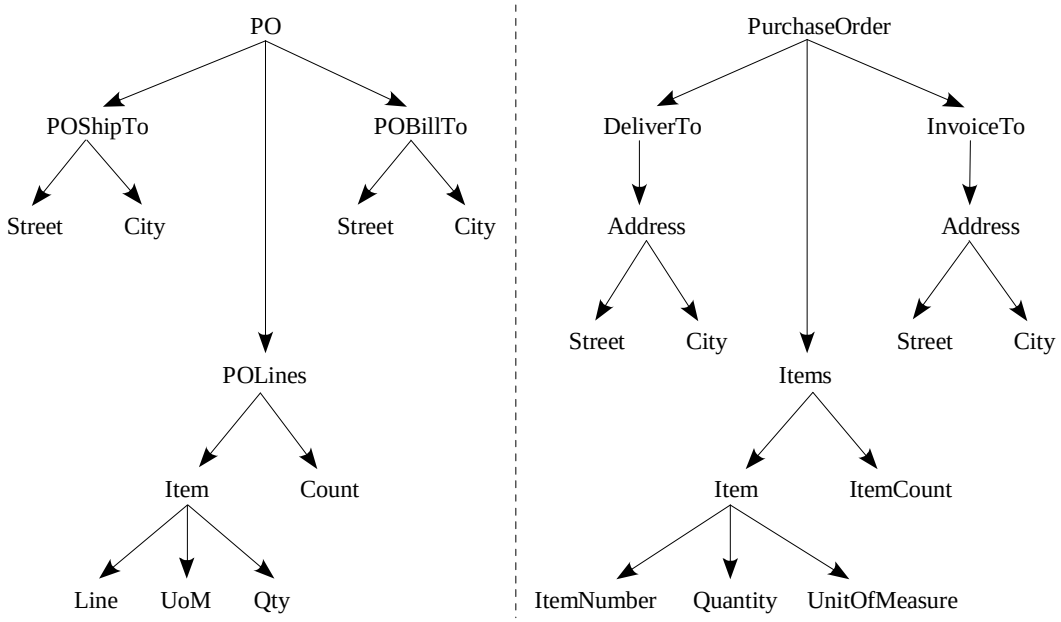

Abbildung 7: Beispielschemata für Cupid (nach [MBR01])

entsprechenden Konzeptnamen verbunden (z. B. Elemente mit den Tokens 'Price', 'Cost' und 'Value' mit dem Konzept 'Money'). Zur Bestimmung von Abkürzungen, Akronymen, zu ignorierenden Wörtern und Konzepten wird ein Thesaurus benutzt, der sowohl Umgangssprache, als auch anwendungsspezifische Ausdrücke, wie z. B. spezielle Begriffe aus dem Anwendungsbereich 'Bestellungen' (im Englischen 'Purchase Orders'), beinhalten kann. Jedes Token wird als einer der folgenden *Tokentypen* markiert: Zahl, Sonderzeichen (z. B. #), allgemeines Wort (Präpositionen und Konjunktionen), Konzept (siehe obige Beschreibung) oder Inhalt (der ganze Rest).

Als nächstes findet in jedem der beiden Schemata eine *Kategorisierung* statt. Eine Kategorie ist eine Gruppe von Schemalementen, die über eine Menge von Schlüsselwörtern identifiziert werden kann. Kategorien und Schlüsselwörter werden folgendermaßen bestimmt: Für jedes Konzept im jeweiligen Schema wird eine Kategorie angelegt. Beispielsweise werden alle Schemaelemente, die mit dem Konzept 'Money' verbunden sind, in eine Kategorie mit dem Schlüsselwort 'Money'eingeordnet. Desweiteren wird für jeden Datentyp im jeweiligen Schema eine Kategorie angelegt. Alle numerischen Datentypen werden z. B. in eine Kategorie mit dem Schlüsselwort 'Number' eingeordnet. Jedes Element, welches andere Elemente im Schema „enthält", definiert ebenfalls eine Kategorie. Beispielsweise sind 'Street' und 'City' in 'Address' „enthalten" und werden somit in einer Kategorie mit dem Schlüsselwort 'Address' gruppiert. Die Kategorien werden für jedes Schema getrennt bestimmt. Jedes Element aus dem jeweiligen Schema wird bereits existierenden oder neuen Kategorien zugeordnet. Dabei kann ein Element in mehrere Kategorien eingeordnet werden. Durch die Kategorisierung wird die Anzahl der anschließenden Element-zu-Element Vergleiche reduziert, indem nur die Elemente miteinander verglichen werden, die kompatiblen Kategorien angehören. Zwei Kategorien sind kompatibel, wenn die *Namensähnlichkeit* ihrer Schlüsselwortmengen einen gegebenen Schwellwert th_{ns}[7] überschreitet. Die Namensähnlichkeit wird sowohl *für Schlüsselwortmengen der Kategorien*, als auch *für Mengen*

[7]Eine Beschreibung der Kriterien für das Setzen der unterschiedlichen Schwellwerte und Konstanten, die in Cupid verwendet werden, ist in [MBR01] nachzulesen.

der Tokens der Elementnamen gleichermaßen errechnet. Folgende Formel gibt die Berechnung der Namensähnlichkeit (*name similarity*, *ns*) für zwei Mengen T_1 und T_2 an:

$$ns\left(T_1, T_2\right) = \frac{\sum_{t_1 \in T_1}\left[\max_{t_2 \in T_2} sim\left(t_1, t_2\right)\right] + \sum_{t_2 \in T_2}\left[\max_{t_1 \in T_1} sim\left(t_1, t_2\right)\right]}{|T_1| + |T_2|}$$

Die Namensähnlichkeit zweier Mengen ergibt sich, wie der Formel zu entnehmen ist, aus dem Durchschnitt der maximalen Ähnlichkeit jedes Tokens mit einem Token der anderen Menge. Dabei wird für zwei Tokens t_1 und t_2 die Ähnlichkeit $sim\left(t1, t2\right)$ einem Thesaurus für Synonyme und Hyperonyme entnommen. Jeder Eintrag des Thesaurus ist mit einem Koeffizienten im Intervall $[0, 1]$ versehen, welcher die Stärke der Ähnlichkeit zwischen den beiden Wörtern angibt. Ist für ein Wortpaar kein Eintrag vorhanden, so werden Teilstrings der beiden Wörter verglichen, um gemeinsame Vor- bzw. Nachsilben zu entdecken.

Als letzter Schritt dieser Phase findet der *Vergleich* von Element-Paaren statt. Für jedes Element-Paar aus kompatiblen Kategorien wird ein *sprachlicher Ähnlichkeitskoeffizient* (*linguistic similarity coefficient*, *lsim*) berechnet. Sprachliche Ähnlichkeit basiert dabei auf der *Namensähnlichkeit der Elemente*. Die Namensähnlichkeit zweier Elemente m_1 und m_2 wird als gewichteter Durchschnitt der Namensähnlichkeit pro Tokentyp anhand der folgenden Formel berechnet:

$$ns\left(m_1, m_2\right) = \frac{\sum_{i \in TokenType} w_i \times ns\left(T_{1i}, T_{2i}\right)}{\sum_{i \in TokenType} w_i \times \left(|T_{1i}| + |T_{2i}|\right)}, \text{wobei} \sum w_i = 1$$

Wie bereits weiter oben erklärt, gehört jedes Token zu einem von 5 Tokentypen (Zahl, Sonderzeichen, allgemeines Wort, Konzept, Inhalt). T_{1i} und T_{2i} sind Tokens der Elemente m_1 und m_2 des Typs i und w_i ist ein Gewichtungsfaktor für die einzelnen Tokentypen. Der Wert des Gewichtungsfaktors w_i basiert auf der Relevanz der jeweiligen Tokentypen, z. B. ist er für Inhalt und Konzept höher gewählt, als für Zahl, Sonderzeichen und allgemeines Wort. Der sprachliche Ähnlichkeitskoeffizient *lsim* wird schließlich berechnet, indem die Namensähnlichkeit der Elemente mit der maximalen Ähnlichkeit der Kategorien zu denen die Elemente gehören multipliziert wird:

$$lsim(m_1, m_2) = ns(m_1, m_2) \times \max_{c_1 \in C_1, c_2 \in C_2} ns(c_1, c_2)$$

Das Ergebnis dieser Phase ist eine Tabelle mit *lsim*-Koeffizienten zwischen Elementen der beiden Schemata. Diese Koeffizienten liegen in einem $[0, 1]$ Intervall, wobei 1 für eine vollständige „sprachliche Übereinstimmung“ steht. Elementen, die keiner kompatiblen Kategorie zugehören, wird 0 zugewiesen.

In der zweiten Phase wird das ursprüngliche Schema in einen Baum umgewandelt, auf dem dann ein *strukturbasiertes Matching* von den Blättern zur Wurzel stattfindet. Dies geschieht mit Hilfe des TreeMatch-Algorithmus, der in Abbildung 8 zu sehen ist und im Folgenden genauer erklärt wird.

Der *strukturelle Ähnlichkeitskoeffizient* (*structural similarity coefficient*, *ssim*) wird für jedes Blätter-Paar mit einem Wert im Intervall $[0, 0.5]$ initialisiert, der die Kompatibilität der Datentypen dieser beiden Elemente angibt (Zeile 2 bis 3). Identische Datentypen erhalten den Initialwert 0.5, damit dieser im weiteren Verlauf des Algorithmus erhöht werden kann. Die jeweiligen Werte werden einer Kompatibilitätstabelle entnommen. Die Elemente werden je Baum in Nachordnung (post-order)[8] sortiert (Zeile 4). Somit werden die nachfolgenden Schritte, die in einer Schleife

[8] Duchlaufen eines Baums in Nachordnung erfolgt folgendermaßen: Erst wird der linke Unterbaum in Nachordnung durchlaufen, dann wird der rechte Unterbaum in Nachordnung durchlaufen und schließlich wird die Wurzel verarbeitet (kurz: links, rechts, Wurzel).

```
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)
```

Abbildung 8: TreeMatch-Algorithmus (nach [MBR01])

ausgeführt werden (Zeile 5 bis 12) immer zuerst für die Söhne und erst danach für ihre Väter abgearbeitet. Für jedes Element-Paar s, t der zwei Schemata wird der strukturelle Ähnlichkeitskoeffizient *ssim* neu bestimmt (Zeile 7). Sind beide Elemente Blätter, so behält das jeweilige Element-Paar seinen Initialwert. Falls jedoch eines der Elemente kein Blatt ist, wird *ssim* folgendermaßen berechnet:

$$ssim\,(s,t) = \frac{|\{x|x \in l\,(s) \wedge \exists y \in l\,(t)\,, sk\,(x,y)\} \cup \{x|x \in l\,(t) \wedge \exists y \in l\,(s)\,, sk\,(y,x)\}|}{|l\,(s) \cup l\,(t)|}$$

Hierbei ist $l(s)$ die Menge der Blätter im Unterbaum des Elementes s. Bei der Berechnung wird die Anzahl der Blätter, die mindestens einen *strong link* (*sk*) zu einem Blatt des jeweils anderen Unterbaums haben, durch die gesamte Anzahl der Blätter der zwei Unterbäume geteilt. Dabei besitzt ein Blatt des einen Schemas einen strong link zu einem Blatt des anderen Schemas, wenn der *gewichtete Ähnlichkeitskoeffizient* (*weighted similarity*, *wsim*) dieser beiden Blätter eine gegebene Schwelle th_{accept} überschreitet, was auf ein potentiell akzeptables Mapping zwischen diesen Elementen hinweist. Im nächsten Schritt wird für das jeweilige Element-Paar die gewichtete Ähnlichkeit *wsim* errechnet, indem die bereits ermittelten Ähnlichkeitskoeffizienten *lsim* und *ssim* gewichtet und addiert werden (Zeile 8). Der Faktor w_{struct} hat einen Wert zwischen 0 und 1 und dient der Gewichtung der beiden Ähnlichkeitskoeffizienten. Überschreitet der gewichtete Ähnlichkeitskoeffizient *wsim* eines Element-Paares die Schwelle th_{high}, bedeutet dies, dass diese beiden Elemente sehr ähnlich sind. In diesem Falle wird für jedes Blätter-Paar der beiden Unterbäume dieses Element-Paares der strukturelle Ähnlichkeitskoeffizient *ssim* um den Faktor c_{inc} erhöht (Zeile 9 bis 10). Die Begründung hierfür ist, dass Blätter mit sehr ähnlichen Vätern einen ähnlichen Kontext haben und deshalb eine höhere strukturelle Ähnlichkeit besitzen als solche, die in einem unterschiedlichen Kontext auftreten. Im Beispiel sind 'POBillTo' und 'InvoiceTo' sehr ähnlich. Diese starke Ähnlichkeit der beiden Elemente würde das Erhöhen der strukturellen Ähnlichkeitkoeffizienten für Paare ihrer Söhne 'City' und 'Street' bewirken, um sie enger aneinander zu binden als andere 'City'-und-'Street'-Paare. Analog wird für jedes Blätter-Paar der beiden Unterbäume eines Element-Paares der strukturelle Ähnlichkeitskoeffizient *ssim* um einen Faktor c_{dec} verringert, wenn der gewichtete Ähnlichkeitskoeffizient *wsim* dieses Element-Paares eine gegebene Schwelle th_{low} unterschreitet (Zeile 11 bis 12).

In der dritten Phase wird das Mapping zwischen den beiden Schemata generiert. Im einfachsten Fall müssen die einzelnen Mapping-Elemente nur auf der Ebene der Blätter erstellt werden. Dabei wird für jedes Blattelement t im Zielschema überprüft, ob jenes Blattelement s aus dem Quellschema, welches den höchsten gewichteten Ähnlichkeitskoeffizienten *wsim* zu t besitzt, den

Schwellwert th_{accept} erreicht bzw. überschreitet ($wsim\,(s,t) \geq th_{accept}$). Ist dies der Fall, so wird ein Mapping-Element von s nach t generiert. Das resultierende Mapping zwischen den beiden Schemata kann die Kardinalität 1:n haben, da ein Element aus dem Quellschema auf mehrere Elemente aus dem Zielschema abgebildet werden kann. Falls die jeweilige Anwendung ein 1:1-Mapping fordert, wird ein zusätzlicher anwendungsspezifischer Abbildungsgenerator benötigt, der die berechneten Ähnlichkeitskoeffizienten als Input erhält und daraus ein 1:1-Mapping erstellt. Um Mapping-Elemente für Schemaelemente, die keine Blätter sind, zu generieren, wird ein zweiter Durchlauf in Nachordnung benötigt, um die Ähnlichkeitskoeffizienten dieser Elemente neu zu berechnen. Dies ist nötig, da das Aktualisieren der Ähnlichkeitskoeffizienten der Blätter-Paare die strukturellen Ähnlichkeitskoeffizienten von Elementen, die keine Blätter sind, beeinflusst haben könnte. Daraufhin kann wie für Elemente auf Blattebene vorgegangen werden.

In diesem Abschnitt wurde die grundlegende Vorgehensweise von Cupid vorgestellt, für eine tiefergehende Lektüre verweisen wir den Leser auf [MBR01].

4.2 Model Management mit RONDO

Rondo ist ein Prototyp einer Programmierplattform für das Management von Modellen. Schema-Matching ist in einem der Operatoren, die Rondo zur Verfügung stellt, implementiert. Eine Programmierplattform wie Rondo ermöglicht es, mit relativ wenig Aufwand Programme zu schreiben, die z. B. Korrespondenzen zwischen heterogenen Schemata mit Hilfe des Match-Operators entdecken und diese Information weiter nutzen, um die beiden Schemata zu einem Schema zu vereinen. In dieser Ausarbeitung soll Rondo nicht genauer behandelt werden, deshalb verweisen wir den Leser auf [MRB03].

Zur Veranschaulichung der Verwendung von Rondo schauen wir uns ein Beispiel aus [MRB03] an. Stellen wir uns dazu ein Unternehmen vor, welches elektronischen Handel betreibt und Daten über Bestellungen an einen Geschäftspartner senden muss. Die Daten sind in einer relationalen Datenbank, also in einem relationalen Schema *s1* gespeichert. Für den Datenaustausch haben sich beide Geschäftspartner auf ein gemeinsames XML-Schema *d1* geeinigt. Das Schema *d1* unterscheidet sich semantisch und strukturell von *s1*. Das relationale Schema *s1* unterliegt periodischen Änderungen. Angenommen zwei Attribute werden gelöscht und drei neue Attribute kommen hinzu. Diese Änderungen müssen in das Austauschschema *d1* übernommen werden. Das neue relationale Schema, welches durch Änderungen in *s1* entstanden ist, nennen wir s2 und das neue XML-Schema, das entstehen soll, nennen wir *d2*. Diese Änderungsübernahme kann in Rondo mit dem Skript (PropagateChanges-Operator), welches in Abbildung 9 zu sehen ist, realisiert werden. Abbildung 10 dient der Veranschaulichung der Funktionsweise des PropagateChanges-Operators.

```
operator PropagateChanges(s1, d1, s1_d1, s2, c, s2_c)
1  s1_s2 = Match(s1, s2);
2  ⟨d1′, d1′_d1⟩ = Delete(d1, Traverse(All(s1) − Domain(s1_s2), s1_d1));
3  ⟨c′, c′_c⟩ = Extract(c, Traverse(All(s2) − Range(s1_s2), s2_c));
4  c′_d1′ = c′_c ∗ Invert(s2_c) ∗ Invert(s1_s2) ∗ s1_d1 ∗ Invert(d1′_d1);
5  ⟨d2, c′_d2, d1′_d2⟩ = Merge(c′, d1′, c′_d1′);
6  s2_d2 = s2_c ∗ Invert(c′_c) ∗ c′_d2 + Invert(s1_s2) ∗ s1_d1 ∗ Invert(d1′_d1) ∗ d1′_d2;
7  return ⟨d2, s2_d2⟩;
```

Abbildung 9: PropagateChanges-Operator (nach [MRB03])

Als erstes müssen die Änderungen, durch die aus *s1* das neue Schema *s2* entstanden ist, festgestellt werden. Dies geschieht durch Anwendung des Match-Operators auf die beiden Schemata

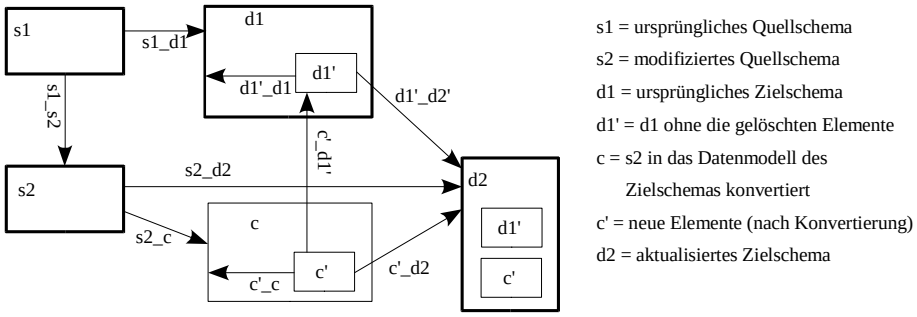

Abbildung 10: Funktionsweise des PropagateChanges-Operators (nach [MRB03])

(Zeile 1). Schemaelemente, für die keine „Partner“ gefunden werden, sind also entweder neu hinzugekommen oder gelöscht worden. Im nächsten Schritt (Zeile 2) wird das Mapping zwischen *s1* (ursprüngliches relationales Schema) und *d1* (ursprüngliches XML-Schema) nach den Elementen durchsucht, die aus *s1* gelöscht worden sind (dies sind diejenigen Elemente in *s1*, die im Mapping zwischen *s1* und *s2* keinen Partner zugewiesen bekommen haben, d. h. in *s2* sind diese nicht mehr vorhanden). Diese Elemente werden aus *d1* gelöscht. Das Ergebnis ist ein neues XML-Schema *d1′* (Teilschema von *d1*), welches alle Elemente aus *d1* enthält, ohne diejenigen Elemente, die den aus *s1* gelöschten Elementen entsprechen. Desweiteren enthält das Ergebnis ein Mapping zwischen *d1′* und *d1*. Der nächste Schritt ist sehr ähnlich. Um ihn zu verstehen, muss allerdings zunächst geklärt werden, wozu das Schema *c* da ist. Dazu erinnern wir uns daran, dass *s1* und *d1* in unterschiedlichen Datenmodellen ausgedrückt sind. Die zu *s1* neu hinzugekommenen Elemente, die in *s2* enthalten sind, haben keine Gegenstücke in *d1*. Das bedeutet, dass die neuen Elemente aus dem Datenmodell von *s1* in das Datenmodell von *s2* konvertiert werden müssen. Die Konvertierung findet mit Hilfe eines Operators SQL2XSD statt, welcher aus einem relationalen Schema ein XML-Schema und ein Mapping zwischen den beiden Schemata erstellt. So entsteht das Schema *c* und das Mapping zwischen *s2* und *c* durch $\langle c, s2\_c\rangle = SQL2XSD\,(s2)$. Es ist zu beachten, dass *c* nicht das gesuchte Schema *d2* ist, da es sich z. B. strukturell von *d2* unterscheidet. Im dritten Schritt (Zeile 3) werden die neu hinzugekommenen Elemente (diejenigen aus *s2*, die im Mapping zwischen *s1* und *s2* keine Partner haben) aus *c* extrahiert. Das Ergebnis ist ein neues XML-Schema c' (Teilschema von *c*), welches die neuen Elemente enthält, und ein Mapping zwischen c' und *c*. Nach den ersten drei Schritten liegen die Schemata *d1′* (Teilschema von *d1* ohne die gelöschten Elemente) und c' (Schema mit den neuen Elementen) vor. Diese müssen nun zusammengesetzt werden, um das gesuchte Schema *d2* zu erhalten. Dies geschieht mit dem Merge-Operator, der allerdings zusätzlich ein Mapping zwischen den beiden Schemata als Eingabe benötigt, welches vorher durch Komposition der Mappings zwischen den Schemata c', *c* , *s2*, *s1*, *d1* und *d1′* zu ermitteln ist (Zeile 4). Die Anwendung des Merge-Operators (Zeile 5) liefert das gesuchte XML-Schema *d2* und ein Mapping zwischen *d2* und c', sowie ein Mapping zwischen *d2* und *d1′*. Zum Schluss (Zeile 6) wird ein Mapping zwischen *s2* und *d2* ermittelt, damit der PropagateChanges-Operator wieder eingesetzt werden kann, wenn sich das relationale Quellschema erneut ändert. Dies geschieht durch Vereinigung der Mappings zwischen *s2* und dem *d1′*-Teil aus *d2* und zwischen *s2* und dem c'-Teil aus *d2*. Die Rückgabe des PropagateChanges-Operators (Zeile 7) ist das gesuchte XML-Schema *d2* und ein Mapping zwischen dem neuen relationalen Schema *s2* und *d2*.

Der vorgestellte PropagateChanges-Operator ist nicht darauf begrenzt, Änderungen von einem relationalen Schema in ein XML-Schema zu propagieren. Der gleiche Operator kann z. B. auch dafür verwendet werden, Änderungen von einem XML-Schema in ein relationales Schema

zu propagieren. In diesem Fall wäre *s1* das ursprüngliche XML-Schema, *s2* das modifizierte XML-Schema, *d1* das relationale Schema und das Schema *c* müsste mit Hilfe eines anderen Operators XSD2SQL ermittelt werden. Der PropagateChanges-Operator könnte dann unverändert angewendet werden.

Rondo stellt eine graphische Oberfläche zur Verfügung, so dass der Benutzer die einzelnen Teilergebnisse falls nötig abändern kann. Besonders das Nachbearbeiten der Ergebnisse des Match-Operators muss möglich sein, um z. B. falsche Zuordnungen entfernen und fehlende hinzufügen zu können.

5 Zusammenfassung und Ausblick

In dieser Ausarbeitung haben wir uns mit verschiedenen Formen der Heterogenität von Systemen beschäftigt. Wir haben gesehen, dass für den Zugriff auf Daten aus heterogenen Datenbanksystemen die Überwindung der semantischen und der strukturellen Heterogenität der Schemata eine große Schwierigkeit darstellt. Schema-Matching-Verfahren gehen dieses Problem auf unterschiedliche Weisen an. Dazu haben wir uns eine Klassifikation und einen Beispielalgorithmus namens Cupid angesehen. Programmierplattformen wie Rondo sollen in Zukunft den Match-Operator, sowie weitere Operatoren zur Verfügung stellen, die dem Management von Modellen dienen. Unseres Wissens nach ist Rondo der erste und zur Zeit noch einzige Prototyp einer solchen Programmierplattform. Doch nicht nur im Bereich des Model Management, sondern auch im Bereich des Schema Matching, gibt es noch sehr viel Forschungsbedarf. Beispielsweise sind Ziele wie die Verwendung textueller Beschreibungen von Schemaelementen, die Wiederverwendung von gemeinsamen Schemaelementen und bereits ermittelten Mappings, sowie die Ausnutzung von Mapping-Ausdrücken vielversprechend.

Literaturverzeichnis

[BKLW99] Busse, Susanne ; Kutsche, Ralf-Detlef ; Leser, Ulf ; Weber, Herbert: *Federated Information Systems: Concepts, Terminology and Architectures.* Forschungsberichte des Fachbereichs Informatik Nr. 99-9. Technische Universität Berlin, 1999

[MBR01] Madhavan, Jayant ; Bernstein, Philip A. ; Rahm, Erhard: *Generic Schema Matching with Cupid.* Technical Report MSR-TR-2001-58. Microsoft Research, 2001

[MRB03] Melnik, Sergey ; Rahm, Erhard ; Bernstein, Philip A.: *Rondo: A Programming Platform for Generic Model Management (Extended Version).* Technical Report. University of Leipzig, 2003

[Nau04] Naumann, Felix: *Informationsintegration I.* Vorlesung. Humboldt-Universität Berlin, WS 03/04

[RB01] Rahm, Erhard ; Bernstein, Philip A.: *A survey of approaches to automatic schema matching.* VLDB Journal 10 (4): 334-350, 2001

[Sau98] Sauter, Günter: *Interoperabilität von Datenbanksystemen bei struktureller Heterogenität.* Dissertation zu Datenbanken und Informationssystemen Bd. 47. infix, 1998