Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik

Datenintegration für die „Global-as-local-view-extension“-Technik

Masterarbeit

Erstgutachter: Prof. Dr. Andreas Heuer
Zweitgutachter: Prof. Dr. Kurt Sandkuhl
Betreuer: Dipl.-Inf. Ilvio Bruder
Dipl.-Inf. Dortje Löper
vorgelegt von: Georgi Straube
Matrikelnummer: 7200517
Abgabedatum: 29.10.2013

Inhaltsverzeichnis

1. Einführung

1.1. Motivation

Das viel beschworene Informationszeitalter hat Konzepte und Techniken hervorgebracht, die eine vereinheitlichte Sicht auf Daten anbieten und effiziente Möglichkeiten zum Austausch und zur Auswertung der Daten ermöglichen. Das gespeicherte und ausgetauschte Datenvolumen wächst dabei unaufhaltsam. Die International Data Corporation (IDC) hat am 01. Dezember 2011 einen Pressebericht veröffentlicht, in dem prognostiziert wird, dass im Jahr 2012 das weltweite Datenvolumen 2,7 Zettabyte betragen wird - 48 Prozent mehr als im Vorjahr [IDC]. Cisco Systems schätzt, dass sich der globale IP-Verkehr im Jahr 2012 auf 0,42 Zettabyte (35 Exabyte monatlich) belief [CIS].

Die Daten, die weltweit erzeugt, gespeichert und ausgetauscht werden, zeichnen sich in hohem Maße durch Heterogenität aus, da sie häufig unabhängig voneinander erhoben, auf unterschiedliche Weise modelliert und mit verschiedenen Technologien gespeichert werden. Heterogenität kann auch beabsichtigt werden, wenn beispielsweise eine physische Verteilung der Daten vorliegt und die lokalen Quellen einen gemeinsamen Datenbestand je nach Funktion der zugreifenden Anwendung unterschiedlich repräsentieren.

Es besteht häufig der Wunsch, Daten durch Überwindung der Heterogenität zu integrieren und eine vereinheitlichte Sicht herzustellen. Die Integration von Daten durch eine Föderation heterogener und autonomer Systeme ist dabei in erster Linie durch den Aufkauf eines Unternehmens durch ein anderes Unternehmen oder eine Unternehmensfusion motiviert. Eine übergreifende Sicht auf die einzelnen Datenbestände wird geschaffen, indem eine globale Anwendung eingeführt wird, auf die lokale Anwendungen abgebildet werden. Das so entstehende föderierte System stellt dabei eine kurzfristige Lösung dar. Längerfristig wird ein Gesamtdatenbestand angestrebt, der aus der Vereinheitlichung der lokalen Datenbestände hervorgeht.

Eine weitere Anwendungsgrundlage lässt sich in der Medizin finden. Dort erleichtert der Austausch von Patientendaten zwischen unterschiedlichen Einrichtungen die Arbeit der Ärzte und des Pflegepersonals und trägt damit zu einer besseren Behandlung bei. Auch für den Austausch von Forschungsergebnissen und im Kontext von Web-Anwendungen ist die Datenintegration von Bedeutung [BH08].

1.2. Zielstellung

Das Ziel dieser Arbeit ist es, ein Konzept für die Integration von Daten aus heterogenen, lokalen Datenquellen zu entwickeln. Dabei sollen Abbildungen von den Datenquellen

in ein globales Schema definiert werden. Das globale Schema soll auf Basis des *Entity-Attribute-Value*-Modells (EAV) gestaltet werden, was zu Beginn von Kapitel 4 motiviert wird.

Im Gegensatz zur „traditionellen“ Datenintegration sollen Anfragen im integrierten System jedoch nicht an das globale Schema gestellt werden, sondern weiterhin an die lokalen Schemata, wobei diese um Daten aus anderen Quellen erweitert werden. Dieser Ansatz wird als *Global as local view extension* bezeichnet. Er erfordert, dass die Inversen von Schemaabildungen berechnet werden, weshalb in 3.2 die damit verbundenen theoretischen Grundlagen vorgestellt werden.

Der Schwerpunkt der Arbeit liegt in der Integration strukturierter Daten, die in Schemata verschiedener Datenmodelle vorliegen können. In dieser Arbeit sollen dabei das Relationenmodell und XML als zu integrierende Datenmodelle betrachtet werden. Für diese sollen Abbildungen auf das EAV-Modell sowie die entsprechenden Rückabbildungen festgelegt werden. Als Integrationsmodell zeichnet sich EAV durch eine hohe Flexibilität aus. Es liegt ein generisches Schema vor, mit dem die Struktur unterschiedlicher Schemata erfasst werden kann. Als Nachteil ist zu nennen, dass dieses Schema semantisch wenig aussagekräftig ist. Das EAV-Modell wird näher in 3.3 vorgestellt.

Der Gegensatz zwischen „traditioneller“ Datenintegration und der Global-as-local-view-Technik wird durch die Abbildungen 1.1 und 1.2 veranschaulicht. Diese sind der Einleitung zu Kapitel 4 entnommen, in dem das Konzept dieser Arbeit vorgestellt wird. In Abbildung 1.1 ist angedeutet, dass die Daten der lokalen Schemata auf das globale Schema abgebildet werden und alle Anfragen im integrierten System an dieses globale Schema gestellt werden. Vor der Integration wurden alle Anfragen an die lokalen Schemata gestellt; für Updates ist dies weiterhin der Fall. Dies entspricht dem „konventionellen“ Ansatz zur Datenintegration. Im Gegensatz dazu werden beim Global-as-local-view-extension-Ansatz, wie er in Abbildung 1.2 dargestellt ist, Anfragen weiterhin an die lokalen Datenbanken gestellt, die um Daten jeweils anderer Datenbanken angereichert wurden. Abbildungen gibt es daher nicht nur aus den lokalen Schemata auf das globale Schema, sondern auch umgekehrt.

Die Ermittlung von Korrespondenzen zwischen Schemata ist nicht Bestandteil dieser Arbeit, sondern wird für die Integration als gegeben angenommen. Ebenso wird die Auflösung von Konflikten im Rahmen dieser Arbeit nicht behandelt.

1.3. Aufbau der Arbeit

Die vorliegende Arbeit umfasst einschließlich der Einleitung sieben Kapitel. In Kapitel 2 werden die grundlegenden Begriffe und Schritte der Datenintegration dargestellt.

Der aktuelle Forschungsstand wird in Kapitel 3 geschildert. Hierbei wird auf Schema Mappings und die Ansätze zur Invertierung von Schema Mappings eingegangen. Darüber hinaus wird das Entity-Attribute-Value-Datenmodell (EAV) im Kontext der Datenintegration vorgestellt. Im Ergebnis sollen Konzepte und Techniken für die in dieser Arbeit

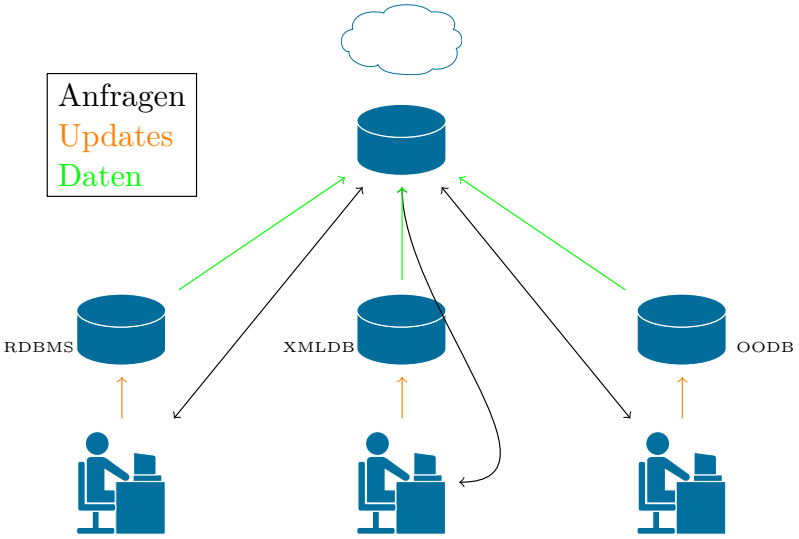

Abbildung 1.1.: Datenintegration mit globalem Schema

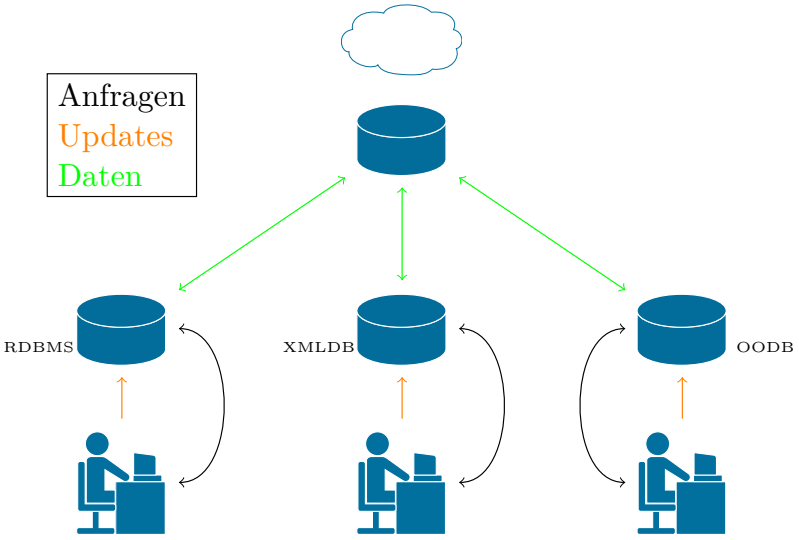

Abbildung 1.2.: Integrierte Daten als Erweiterung des lokalen Schemas

zu entwickelnde Datenintegrationstechnik abgeleitet werden.

Auf Basis der vorangegangen Kapitel wird in Kapitel 4 ein Konzept entwickelt, das die Global-as-local-view-extension-Datenintegrationstechnik auf Grundlage des EAV-Modells als globalem Schema umsetzen soll. Die prototypische Implementierung dieses Konzepts wird in Kapitel 5 beschrieben.

Eine Evaluierung des entwickelten Konzepts und der Implementierung wird in Kapitel 6

vorgenommen. Die Grundlage hierfür bilden das in dieser Arbeit laufende Beispiel zur Bücherverwaltung (siehe unten) sowie ein der Medizin entnommenes Beispiel, das auf dem medizinischen Standard *DICOM*[1] aufbaut.

Das abschließende Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und bietet einen Ausblick auf Themen für mögliche Folgearbeiten.

1.4. Laufendes Beispiel

Das laufende Beispiel, das in der Arbeit zur Veranschaulichung von Konzepten und zur Evaluation genutzt werden soll, befasst sich mit Büchern und der Verwaltung von Büchern. Ein konkretes Beispiel ist eine öffentliche Bibliothek, die z.B. auf folgende Art modelliert werden kann: Es werden Buchexemplare verwaltet, wobei jedem Buchexemplar ein Buchtitel zugeordnet ist und jedes Exemplar sich in einer der Filialen der Bibliothek befindet. Die Exemplare werden dabei über eine eindeutige Inventarnummer und die Buchtitel über die ISBN identifiziert, wobei angenommen wird, dass die ISBN eines Buchtitels für alle Auflagen identisch ist. Jeder Mitarbeiter arbeitet in genau einer Filiale. Leser können Buchexemplare ausleihen und Anschaffungsvorschläge machen, die von einem Mitarbeiter bearbeitet werden. Dieses Beispiel wird in Abbildung 1.3 in Form eines *Entity-Relationship*-Modells (ER-Modell) veranschaulicht.

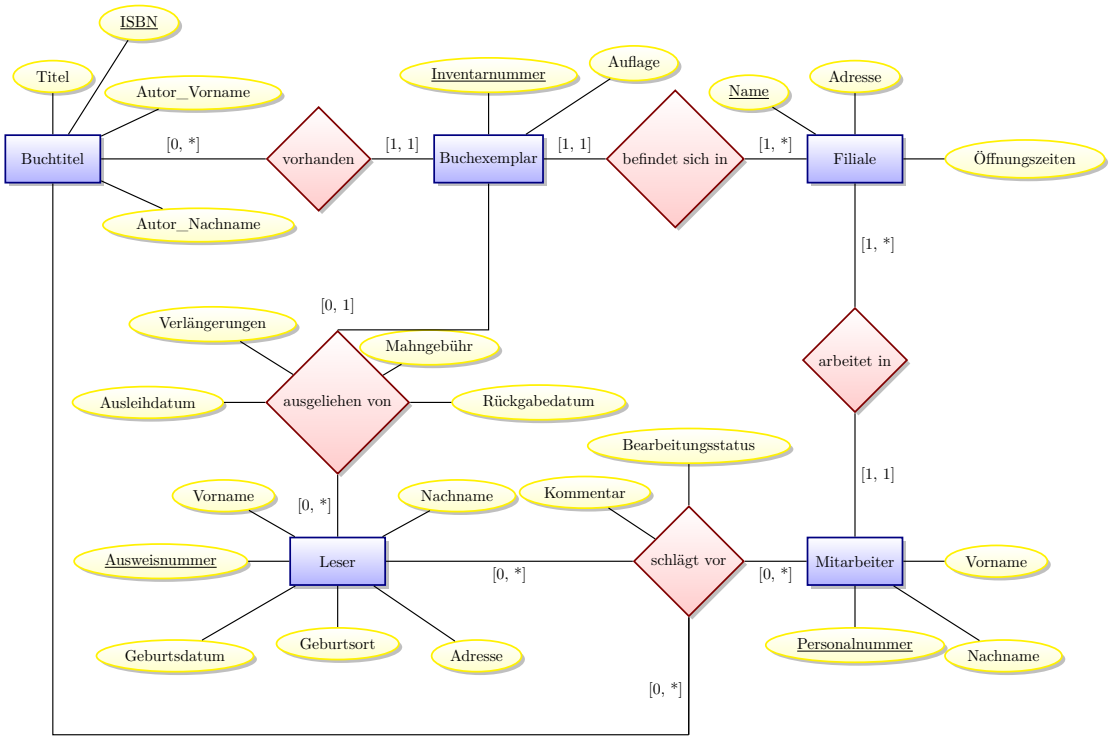

Abbildung 1.3.: Bibliothek

[1] `http://medical.nema.org/standard.html`

2. Grundlagen der Datenintegration

In diesem Kapitel wird ein Überblick über grundlegende Themen der Datenintegration angegeben. Dieser dient als allgemeine Einführung, in der zur Abgrenzung auch Aspekte der Datenintegration betrachtet werden, die nicht Bestandteil dieser Arbeit sind. In Kapitel 3 wird dann auf spezielle Themengebiete eingegangen, die für diese Arbeit von Bedeutung sind.

2.1. Integrationsschritte

Haas charakterisiert Datenintegration in [Haa07] als mehrstufigen Prozess, der die folgenden Schritte umfasst:

1. **Verständnis**

   Diese Phase umfasst die Analyse der zu integrierenden Daten, wobei Eigenschaften wie Integritätsbedingungen oder Datentypen sowie Beziehungen zwischen den Daten (z.B. Fremdschlüsselbedingungen) erfasst werden. Darüber hinaus können die Daten statistisch ausgewertet werden, z.B. können besonders häufig vorkommende Werte ermittelt werden.

2. **Vereinheitlichung**

   Es wird aufbauend auf den Ergebnissen der ersten Phase ein integriertes Zielschema entworfen und eine Festlegung bezüglich der Repräsentation der Daten getroffen. Es werden Abbildungen von den Quellschemata ins das Zielschema definiert. Des Weiteren muss entschieden werden, wie Datenkonflikte aufgelöst werden, z.B. wenn Daten unvollständig oder inkonsistent sind. Weiterhin sollten gleiche Objekte erkannt und zusammengeführt werden, um Duplikate zu vermeiden. Hierbei können wiederum widersprüchliche Angaben auftreten.

3. **Spezifikation**

   In diesem Schritt werden Abbildungen von den lokalen Quellen zum Zielschema definiert, das im zweiten Schritt entworfen wurde. Daraus werden Transformationsvorschriften oder Sichten in einer konkreten Sprache (z.B. SQL oder XQuery) abgeleitet.

4. **Ausführung**

   Die Integration der Daten erfolgt in dieser Phase. Diese kann **materialisiert** erfolgen, was bedeutet, dass die integrierten Daten gespeichert werden. Im Gegensatz

dazu werden bei einer **virtualisierten** Integration keine Datensätze im integrierten Zielschema gespeichert. Stattdessen werden Anfragen an das globale Schema entgegengenommen und in Anfragen an die lokalen Datenquellen übersetzt, wobei die Ergebnisse integriert und wiederum im globalen Schema dargestellt werden. Im relationalen Modell kann eine virtualisierte Integration beispielsweise mit Sichten realisiert werden.

Leser und Naumann stellen in [LN07] konkrete Techniken zur Datenintegration vor, die sich schwerpunktmäßig den Schritten 3 und 4 zuordnen lassen. Eine Ausnahme bildet das **Schema Matching**, womit eine Methode zur semiautomatisierten Ermittlung von Korrespondenzen bezeichnet wird. Korrespondenzen beschreiben „einen semantischen Zusammenhang zwischen Elementen verschiedener Schemata“ [LN07, S. 115f.]. Sie können auch manuell gefunden werden.

Die als Ergebnis der ersten beiden Schritte festgelegten Korrespondenzen bilden die Grundlage für die **Schemaintegration** und für **Schema Mappings**. Bei der Schemaintegration wird auf Basis der Quellschemata ein globales Schema entwickelt, wobei die Abbildungen von den lokalen Schemata in das globale Schema als Ergebnis dieses Prozesses entstehen. Bei der Erstellung von Schema Mappings hingegen wird eine Abbildung von einem Schema in ein anderes Schema spezifiziert, wobei beide Schemata bereits gegeben sind. Im Kontext der Datenintegration werden Schema Mappings somit dann eingesetzt, wenn das Zielschema bereits feststeht und nach Abbildungen von den Quellschemata in das Zielschema gesucht wird.

Die **Anfragebearbeitung** spielt insbesondere bei der virtuellen Integration eine zentrale Rolle. Sie umfasst mehrere Phasen wie z.B. die Übersetzung einer lokalen Anfrage in eine globale Anfrage und die Integration der Ergebnisse.

In diesem Kapitel werden das Schema Matching und die Anfragebearbeitung näher betrachtet. Eine formale Darstellung von Schema Mappings erfolgt in Kapitel 3. In Kapitel 4 wird untersucht, wie mit Hilfe von Schema Mappings die in dieser Arbeit zu definierenden Abbildungen und Rückabbildungen erstellt werden können.

2.2. Schema Matching

Die manuelle Ermittlung von Korrespondenzen zwischen Schemata kann sich als schwierig erweisen, insbesondere wenn Schemata aufgrund ihrer Komplexität unüberschaubar werden. Probleme können sich beispielsweise durch gleiche Attributnamen mit unterschiedlicher Bedeutung ergeben, z.B kann `name` auf den Namen einer Person oder einen Produktnamen hinweisen. Zudem ist der Nutzer, der die Korrespondenzen ermitteln möchte, meist nur mit einem der beiden Schemata vertraut. Zusätzliche Schwierigkeiten ergeben sich durch fremdsprachliche Schemata und Attributnamen, deren Bedeutung nicht verständlich ist [LN07, S. 143f.].

Das Schema Matching schafft hier Abhilfe, indem mögliche Korrespondenzen identifiziert und dem Nutzer als solche vorgeschlagen werden. Dieser kann sie dann akzeptieren oder

verwerfen. Rahm und Bernstein nehmen in [RB01] eine Kategorisierung von verschiedenen Ansätzen zum Schema Matching vor, an der sich die nachfolgenden Ausführungen orientieren (siehe Abbildung 2.1).

Grundsätzlich wird zwischen **individuellen** und **kombinierten** Ansätzen unterschieden. Individuelle Ansätze können wiederum schema- oder instanzbasiert sein. Auf diese Unterscheidung wird im Folgenden eingegangen. Des Weiteren werden kombinierte Ansätze kurz betrachtet.

2.2.1. Schemabasierter Ansatz

Bei diesem Ansatz sind die Schemata der zu integrierenden Datenquellen Grundlage für die Ermittlung von Korrespondenzen, nicht aber die Dateninstanzen. Dabei können Namen, Datentypen, Beschreibungen und Beziehungen von Elementen ausgewertet werden.

Es können Beziehungen zwischen einzelnen, atomaren Elementen (z.B. Elemente in XML-Dokumenten oder Spaltennamen in Relationen) betrachtet werden oder zwischen Strukturen, in denen die Elemente eingebettet sind (siehe Abbildung 2.2).

Ein weiteres Kriterium betrifft die **Kardinalität** der Korrespondenzen. Liegt eine 1:1-Korrespondenz vor, so kann diese als Gleichheitsbedingung zwischen den Elementen der beiden Schemata ausgedrückt werden (z.B. `Vorname` = `First name`). Bei einer n:1-Korrespondenz muss eine Funktion angegeben werden, die angibt, wie aus den n Elementen des Quellschemas ein Element des Zielschemas gebildet wird (z.B. eine Funktion, die die String-Werte der Elemente `Vorname` und `Nachname` konkateniert, um den Wert für das Zielelement `Name` zu erzeugen).

Umgekehrt wird bei einer 1:n-Korrespondenz angegeben, wie das eine Element des Quellschemas auf n Elemente des Zielschemas abgebildet wird (z.B. eine Funktion, die aus dem String-Wert des Quellelements `Name` die Werte für die Zielemente `Vorname` und `Nachname` extrahiert). Die Definition von n:m-Beziehungen erfordert meist ein Matching auf Strukturebene, das z.B. durch einen SQL-Ausdruck, der die Werte mehrerer Tabellen verknüpft, umgesetzt werden kann.

Linguistische Matcher nutzen die Namensgleichheit (ggf. nach einer Normalisierung, z.B. Zurückführung auf den Wortstamm), Synonymie und Hyperonymie (is-a-Beziehungen) von Elementnamen oder -beschreibungen (z.B. in Form von Kommentaren), um ein Ähnlichkeitsmaß zu berechnen. Die Edit- oder Levenshtein-Distanz kann dazu eingesetzt werden, Namensähnlichkeiten zu ermitteln (siehe [LN07, S. 148f.].

Constraint-basierte Ansätze nutzen Informationen wie Datentypen, Wertebereiche, Kardinalitäten und Beziehungen zwischen Elementen und Strukturen. Häufig werden nur n:m-Korrespondenzen gefunden, die aber genutzt werden können, um die Auswahl möglicher Korrespondenzen einzuschränken und in Kombination mit anderen Matching-Ansätzen verwendet werden können. Darüber hinaus können Referenzen innerhalb eines Schemas (z.B. Fremdschlüsselbedingungen) für ein Matching auf Strukturebene verwendet werden.

Weiterführende Ansätze bestehen darin, Matchings wiederzuverwenden. So kann z.B. ein Matching S_1 zu T wiederverwendet werden, wenn ein Matching für S_2 zu T gefunden wird und es Ähnlichkeiten zwischen S_1 und S_2 gibt. Die Ermittlung dieser Ähnlichkeit kann wiederum als Matching-Problem aufgefasst werden.

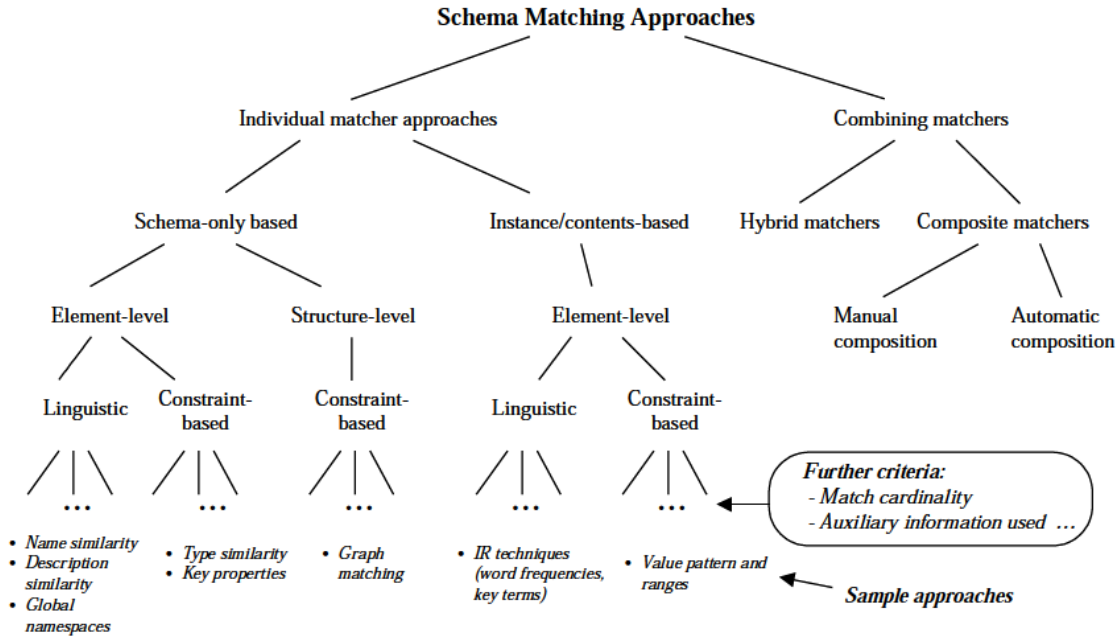

Abbildung 2.1.: Kategorisierung von Schema-Matching-Ansätzen (Quelle: [RB01, S. 338])

2.2.2. Instanzbasierter Ansatz

Bei einem instanzbasierten Ansatz sind die Daten selbst Grundlage für das Schema Matching. **Vertikale Matcher** vergleichen paarweise die Eigenschaften von Attributwerten für ein einzelnes Attribut, während bei **horizontalen Matchern** nach Duplikaten in den beiden Schemata gesucht werden, die der Ermittlung von Korrespondenzen zugrunde gelegt werden [LN07, S. 150].

Bei textuellen Attributwerden können Techniken des Information Retrieval eingesetzt werden, indem z.B. besonders häufig vorkommende Begriffe als Schlüsselwörter für jeweils ein Schema aufgefasst und zwischen den Schemata verglichen werden. Bei numerischen Datentypen bietet es sich an, Wertebereiche oder Muster miteinander zu vergleichen.

Instanzbasierte Ansätze können als alleinige Grundlage für das Schema Matching eingesetzt werden oder aber als Unterstützung für schemabasierte Methoden, indem z.B. die Analyse von Wertebereichen für die constraint-basierte Ermittlung von Korrespondenzen verwendet wird.

Eine Wiederverwendung vorangegangener Matchings kann z.B. durch die Zuordnung von Schlüsselwörtern zu Elementen erreicht werden. Wenn beispielsweise bei einem Matching

der Begriff „IBM“ mit einem Element namens `Firmenname` assoziiert wurde, kann bei einem Mapping zwischen den Schemata S und T eine Korrespondenz zwischen einem Attribut `FName` und `Firmenname` hergestellt werden, wenn unter den Attributwerten von `FName` der Wert „IBM“ besonders häufig auftritt. Es ist hierbei nicht nötig, dass der Wert „IBM“ besonders häufig unter den Attributwerten von `Firmenname` vertreten ist.

Instanzbasiertes Schema Matching wird hauptsächlich für das Finden von Korrespondenzen auf der Ebene von einzelnen Elementen verwendet, da die Anzahl möglicher Kombinationen exponentiell anwächst, wenn man Mengen von Elementen oder Elementstrukturen berücksichtigt.

S1 elements	S2 elements	
Address	CustomerAddress	full structural match of
Street	Street	Address and CustomerAddress
City	City	
State	USState	
ZIP	PostalCode	
AccountOwner	Customer	partial structural match of AccountOwner and
Name	Cname	Customer
Address	CAddress	
Birthdate	CPhone	
TaxExempt		

Abbildung 2.2.: Matching auf Basis von Strukturen (Quelle: [RB01, S. 337])

2.2.3. Kombinierte Ansätze

Ein Matcher wird **hybrid** genannt, wenn er verschiedene Matching-Techniken integriert und ein Ergebnis berechnet. So kann z.B. wie weiter oben erwähnt, das instanzbasierte Matching eine Vorauswahl von möglichen Korrespondenzen für das schemabasierte Matching bereitstellen. Hybride Matcher liefern genauere Ergebnisse und sind effizienter als individuelle Matcher, da falsch-positive Korrespondenzen vermieden werden und die Menge an möglichen Korrespondenzen schrittweise eingeschränkt wird [LN07, S. 154]. Die individuellen Matching-Techniken, die in einem hybriden Matcher eingesetzt werden, sind dabei üblicherweise nicht frei austauschbar.

Kombininerte Matcher gestatten es dem Anwender Matching-Techniken auszuwählen, die z.B. der jeweiligen Anwendungsdomäne entsprechen. Sie sind somit flexibler als hybride Matcher. Die Matching-Techniken werden dabei unabhängig voneinander angewendet und die Resultate kombiniert, wobei eine Gewichtung vorgenommen werden kann.

2.3. Anfragebearbeitung

Wie bereits zu Beginn des Kapitels erwähnt, wird bei der Datenintegration zwischen virtueller und materialisierter Integration unterschieden. Ferner kann man zwischen offline-materialisierenden und online-materialisierenden Systemen differenzieren [LN07, S. 173f.]. Bei online-materialisierenden Systemen werden die Daten aus den lokalen

Quellen unverändert in einem zentralem System gespeichert und die Integration erst dann ausgeführt, wenn Anfragen an das System gestellt werden. Die Bearbeitung von Anfragen steht somit im Zentrum sowohl von virtualisierenden als auch von online-materialisierenden Systemen.

Das Ziel der Anfragebearbeitung ist die Beantwortung globaler Anfragen (also Anfragen an das globale, integrierte Schema) durch Rückgriff auf die lokalen Datenquellen. Hierzu müssen die globalen Anfragen in lokale Anfragen transformiert werden. Die Ergebnisse der Ausführung der lokalen Anfragen werden schließlich in geeigneter Weise aufbereitet und als Ergebnis der globalen Anfrage dargestellt.

Ein grundlegendes Merkmal der in dieser Arbeit zu entwickelnden Datenintegrationstechnik besteht in der Anforderung, dass die Anfragen des Nutzers nicht direkt an das globale Schema gestellt werden, sondern im jeweiligen lokalen Datenmodell formuliert werden, wobei die Daten der lokalen Datenquelle einen Ausschnitt der globalen, integrierten Daten darstellen. Die Anfragebearbeitung unterscheidet sich darin von den in diesem Abschnitt vorgestellten Verfahren, bei denen Anfragen stets an das globale Schema gestellt werden. Diese Verfahren sollen dennoch betrachtet werden, da die grundlegenden Schritte übertragbar sind.

Anhand der Ausführungen von Leser und Naumann in [LN07] sollen in diesem Abschnitt die grundlegenden Schritte der Anfragebearbeitung vorgestellt werden.

2.3.1. Anfrageplanung

Eine globale Anfrage kann auf verschiedene Arten in Teilanfragen zerlegt werden, die wiederum durch unterschiedliche lokale Datenquellen beantwortet werden können. Durch die Kombination von Teilanfragen mit Datenquellen ergeben sich mehrere **Anfragepläne**, die logisch äquivalent zur globalen Anfrage sind.

In den Ergebnissen der Ausführung der Anfragepläne kann es jedoch Abweichungen geben, je nachdem welche Datenquellen in den Anfragepläne verwendet wurden, da die Datenquellen unterschiedliche Daten bereithalten.

Die Grundlage der Anfrageplanung sind Anfragekorrespondenzen, die eine „semantische Beziehung zwischen Elementen des globalen Schemas und Elementen der Exportschemata der Datenquellen“ herstellen [LN07, S. 184f.].

Anfragekorrespondenzen definieren die *Extension*[1] der globalen Relationen auf Grundlage der Extensionen der lokalen Quellen. Es sind folgende Extensionsbeziehungen zwischen einem globalen Schema g und einem lokalen Schema l möglich:

- **Exklusion**: $g \cap l = \emptyset$
- **Inklusion**: $g \supseteq l$ oder $l \supseteq g$

[1] Die Menge von Tupeln einer Relation wird als *Extension* bezeichnet, die Menge der Attribute des entsprechenden Relationenschemas als *Intension* [LN07, S. 20].

- **Überlappung**: $g \cap l \neq \emptyset \wedge g \not\subseteq l \wedge g \not\supseteq l$
- **Äquivalenz**: $g \equiv l$

Für die Definition von Korrespondenzen sind vorrangig die Inklusionsbeziehungen der Form $g \supseteq l$ von Interesse, da hier alle Tupel in der Extension der lokalen Relation l auch zur Extension der globalen Relation g gehören. Bei einer Überlappung ist dies nicht der Fall, d.h. es können bei der Integration Tupel in das Gesamtergebnis aufgenommen werden, die nicht zur Extension von g gehören, was auch für Inklusionsbeziehungen der Form $l \supseteq g$ gilt.

Exklusionsbeziehungen sind für die Datenintegration nicht von Nutzen, da sich kein Tupel aus der Extension von l in der Extension von g wiederfindet. Äquivalenzen können als Sonderfall einer Inklusion aufgefasst werden.

Korrespondenzen können zwischen Relationen, Attributen und Anfragen festgelegt werden, wobei für die Datenintegration Korrespondenzen der Art Relation-Relation, Relation-Anfrage, Anfrage-Relation und Anfrage-Anfrage am wichtigsten sind. Diejenigen Verfahren, deren Basis Relation-Relation oder Relation-Anfrage bilden, tragen die Bezeichnung **Global-as-view** (GaV). Wenn Korrespondenzen der Art Anfrage-Relation verwendet werden, spricht man von **Local-as-view** (LaV). **Global-local-as-view**-Verfahren (GLaV) nutzen Anfrage-Anfrage-Korrespondenzen. Eine formalere Betrachtung von GaV, LaV und GLaV erfolgt in Kapitel 3 im Zusammenhang mit der Definition von Schema Mappings.

Korrespondenzen können in der Form $q_1 \supseteq q_2$ formuliert werden, wobei q_1 eine Anfrage an das globale und q_2 eine Anfrage an ein lokales Schema ist. Daher wird q_1 als globale und q_2 als lokale Anfrage bezeichnet. Bei einer GaV-Korrespondenz besteht die globale Anfrage aus nur einer Relation, während die lokale Anfrage aus konjunktiven Anfragen an ein lokales Schema besteht. Bei einer LaV-Korrespondenz hingegen ist die globale Anfrage eine Konjunktion von Anfragen an das globale Schema, während die lokale Anfrage aus einer einzelnen Relation besteht.

Die Aufgabe der Anfrageplanung besteht darin, die globalen Anfragen der Korrespondenzen so zu kombinieren, dass die Kombination äquivalent zur Benutzeranfrage ist. Werden GaV-Korrespondenzen verwendet, muss in der Kombination nur eine Korrespondenz für jede in der Benutzeranfrage vorkommende globale Relation vertreten sein. Die Anfragen an das globale Schema in der Benutzeranfragen können dann entsprechend der jeweiligen Korrespondenzen durch Anfragen an die lokalen Schemata ersetzt werden. Dies entspricht der Entfaltung einer relationalen Sicht, bei der die Anfrage, durch die die Sicht definiert wird, an Stelle der Sicht eingesetzt wird, womit man eine ausführbare Anfrage erhält.

Bei LaV-Korrespondenzen gestaltet sich die Anfrageplanung weitaus schwieriger, da die globale Anfrage der Korrespondenz nicht aus einer einzelnen Relation über dem globalen Schema besteht, sondern aus einer konjunktiven Anfrage an das globale Schema. Die Ermittlung einer zur Benutzeranfrage äquivalenten Kombination von globalen Anfragen

der Korrespondenzen ist aus diesem Grund eine komplexere Aufgabe als bei GaV.

Ein Algorithmus für die Anfrageplanung mit LaV wird von Alan Y. Halevy in [Hal01] vorgestellt. Der Algorithmus trägt die Bezeichnung **Bucket-Algorithmus**. Für jede Relation r der Benutzeranfrage u wird ein *Bucket* erstellt, der alle Korrespondenzen $q_1 \supseteq q_2$ beinhaltet, deren globale Anfrage q_1 die Relation r enthält. Anschließend wird aus den lokalen Anfragen der Korrespondenzen (wobei aus jedem Bucket eine Korrespondenz gewählt wird) eine konjunktive Anfrage v gebildet. Schließlich wird geprüft, ob die Benutzeranfrage u die Anfrage v enthält, d.h. ob $u \supseteq v$ gilt[2]. Hiermit wird sichergestellt, dass im Ergebnis der Ausführung von v nur Tupel enthalten sind, die korrekt in Bezug auf die Beantwortung von u sind.

Wird das globale Schema zuerst entworfen (Top-Down-Ansatz), liegt die Verwendung von LaV-Korrespondenzen nahe, da es mit GaV-Korrespondenzen nicht möglich ist, das globale Schema allgemeiner als die lokalen Schemata zu gestalten [LN07, S. 231f.]. Wird im umgekehrten Fall das globale Schema aus den Schemata der lokalen Quellen heraus entwickelt (Bottom-up-Ansatz), sind GaV-Korrespondenzen geeigneter, da LaV-Korrespondenzen nicht eingesetzt werden können, wenn die lokalen Schemata allgemeiner als das globale Schema sind.

2.3.2. Anfrageübersetzung

Es muss bereits bei der Spezifikation von Korrespondenzen festgelegt werden, wie die lokalen Anfragen der Korrespondenzen in die Anfragesprachen der Datenquellen (z.B. SQL oder XQuery) übersetzt werden bzw. wie sie in Anfragen an Datenquellen übertragen werden, die über keine dedizierte Anfragesprache verfügen, wie beispielsweise Webseiten.

2.3.3. Anfrageoptimierung

Ähnlich wie bei relationalen Datenbankmanagementsystemen (RDBMS) wird auch bei integrierten Systemen eine Anfrageoptimierung durchgeführt. Eines der Hauptziele ist dabei die **Minimierung der Ausführungszeit** für eine Anfrage. Bei einem integrierten System muss zusätzlich berücksichtigt werden, dass die lokalen Datenquellen meist über ein Netzwerk miteinander verbunden sind. Es ergeben sich daher viel größere Latenzen als bei (nicht verteilten) RDMBS. Des Weiteren muss beachtet werden, dass Daten beim Transfer durch das Netzwerk verloren gehen oder fehlerhaft übertragen werden könnten. Somit sind Mechanismen nötig, die eine korrekte und zuverlässige Datenübertragung gewährleisten, wodurch zusätzlicher Overhead entsteht.

Im Gegensatz zu RDBMS, wo eine Anfrage für die gleiche Dateninstanz stets das gleiche Ergebnis erzeugen soll, ist es jedoch akzeptabel, wenn Einschränkungen der Vollständigkeit und Qualität des Anfrageergebnisses vorgenommen werden, um die Ausführungszeit zu verringern.

[2] Dieses Problem ist als *Query Containment* bekannt und wird u.a. in [LN07, S. 213 ff.] behandelt.

Um die Netzwerkübertragungszeit möglichst niedrig zu halten, wird bei einer Integration versucht, möglichst geringe Datenmengen zu übertragen. Insbesondere sollen Zwischenergebnisse, die von den Datenquellen zur Integrationsschicht übertragen werden, möglichst klein sein. Dies kann dadurch erreicht werden, dass Operationen, die das Zwischenergebnis verfeinern und damit verkleinern, auf den Datenquellen ausgeführt werden. So sollten z.B. Projektionen, Selektionen und Joins innerhalb von Teilanfragen immer auf den Datenquellen ausgeführt werden. Joins zwischen Teilanfragen, die unterschiedliche Datenquellen adressieren, können meist nur in der Integrationsschicht ausgeführt werden.

Es kann der Fall auftreten, dass Anfragepläne nur in einer bestimmten Reihenfolge ausführbar sind, z.B. wenn ein Teilplan die Variablenbindung für einen anderen Teilplan bereitstellt. Des Weiteren können Datenquellen beschränkt sein, d.h. bestimmte Prädikate in Anfragen nicht unterstützen. Die Auswertung der Prädikate muss in diesen Fällen in der Integrationsschicht erfolgen.

Eine Optimierung kann auch zwischen Anfrageplänen erfolgen, indem gemeinsame Teilanfragen identifiziert und nur einmal ausgeführt werden, anstatt sie für jeden Anfrageplan neu auszuwerten. Hierbei muss jedoch beachtet werden, dass unterschiedliche Variablenbindungen in den Teilanfragen auch unterschiedliche Ergebnisse erzeugen.

In [LN07, S. 235ff.] werden konkrete Techniken der Anfrageoptimierung betrachtet, auf die hier nicht näher eingegangen werden soll.

2.3.4. Anfrageausführung

Eine Aufgabe der Anfrageausführung besteht in der Verwaltung der für die Integration notwendigen Verbindungen im Netzwerk. Das Integrationssystem muss Verbindungen zu den Datenquellen aufbauen und den Austausch von Teilanfragen und Zwischenergebnissen koordinieren.

Die Integrationsschicht muss „Teilergebnisse gemäß den Konventionen des globalen Schemas“ [LN07, S. 205] transformieren, was u.a. die Konvertierung von Datentypen und die Übersetzung von Begriffen zwischen dem globalen Schema und den lokalen Schemata beinhaltet. Darüber hinaus müssen Variablen vor Ausführung einer Anfrage gebunden werden, entweder an Konstanten aus der Benutzeranfrage oder an Werte anderer Teilanfragen. Das Integrationssystem verwaltet die Variablenbindung.

Das Integrationssystem muss diejenigen Operationen auf den Zwischenergebnissen ausführen, die nicht in den Datenquellen ausgeführt werden können. Dies betrifft insbesondere Joins zwischen Teilanfragen an verschiedene Datenquellen und die Auswertung von Filterprädikaten, die von Datenquellen nicht unterstützt werden.

Zum Zeitpunkt der Anfrageausführung kann eine erneute Optimierung oder Anfrageplanung vorgenommen werden, wenn Datenquellen Zwischenergebnisse nicht in zufriedenstellender Weise liefern oder gänzlich unverfügbar werden.

2.3.5. Ergebnisintegration

Bei der Integration heterogener Daten treten häufig Konflikte auf. Eine **Datenbereinigung** kann durchgeführt werden, um Fehler und Inkonsistenzen zu beseitigen. Ebenso müssen *Duplikate* erkannt werden. Der Duplikaterkennung schließt sich die **Datenfusion** an, bei der Duplikate unter Auflösung möglicher Widersprüche zusammengeführt werden.

2.3.6. Zusammenfassung

In diesem Kapitel wurden einleitend die grundlegenden Schritte der Datenintegration nach Haas dargestellt. Die ersten beiden Schritte, „Verständnis“ und „Vereinheitlichung“, sind nicht Gegenstand dieser Arbeit. Es wird vorausgesetzt, dass die Zusammenhänge zwischen heterogenen Datenbeständen erkannt wurde und eine vereinheitlichte Darstellung unter Auflösung von Konflikten ermittelt wurde. Hierzu können Methoden des Schema Matching eingesetzt werden, die in diesem Kapitel ebenfalls vorgestellt wurden. Im Bereich der Anfragebearbeitung sind vor allem die Anfrageplanung und hier speziell die Anfragekorrespondenzen von Interesse.

3. Stand der Forschung und Technik

In diesem Kapitel werden Themen eingeführt, auf die bei der Vorstellung des Konzepts für die „Global-as-local-view-extension“-Technik in Kapitel 4 Bezug genommen wird. Konkret wird in diesem Unterkapitel eine formale Darstellung von *Schema Mappings*[1] angegeben. Darauf aufbauend werden Definitionen zu *Schema-Mapping-Inversen* und Eigenschaften und Kriterien vorgestellt, die ein Schema Mapping invertierbar machen. Die Darstellung von Inversen orientiert sich an Arbeiten von Fagin [Fag07], [FKPT05], [FKMP05]. Andere Definitionen werden kurz erläutert, wobei dargelegt wird, warum diese für das Konzept dieser Arbeit nicht geeignet sind. Anschließend wird das *Entity-Attribute-Value*-Modell (EAV) vorgestellt. Dabei werden die grundlegenden Komponenten dieses Modells betrachtet als auch Erweiterungen, die in verschiedenen Arbeiten meist vor dem Hintergrund eines bestimmten Anwendungsszenarios entwickelt wurden.

3.1. Schema Mappings

Schema Mappings definieren eine Abbildung zwischen einem Quellschema (das auch als lokales Schema bezeichnet wird) und einem Zielschema (dem globalen Schema). Die folgenden Ausführungen orientieren sich an [CK10]. Eine weitere Darstellung zu Schema Mappings ist in [ABLM10] zu finden.

Schema Mappings werden als deklarative Spezifikation ausgedrückt, d.h. als Ausdruck innerhalb eines logischen Formalismus formuliert, und nicht als prozedurale Vorschrift. Dies hat den Vorteil, dass man bestimmte Eigenschaften formal überprüfen kann, was im Weiteren noch näher ausgeführt wird. Des Weiteren ergibt sich eine Trennung von Spezifikation und Implementation. Die Implementierung kann dabei häufig automatisch aus der Spezifikation abgeleitet werden.

Bevor eine weit verbreitete, konkrete Sprache zur Spezifikation von Schema Mappings betrachtet wird, sollen zunächst einige Begrifflichkeiten und Notationen geklärt werden.

Ein **Datenbankschema** $D = (R_1, \cdots, R_n)$ setzt sich aus Relationensymbolen R_i zusammen, die eine bestimmte feste Stelligkeit besitzen (die Anzahl der Attribute). Eine Instanz $I = (R_1^I, \cdots, R_n^I)$ von D (auch als D-Instanz bezeichnet) besteht aus endlichen **Relationen** R_i^I, die dieselbe Stelligkeit wie das passende Relationensymbol R_i aufweisen. Ein **Fakt** in Bezug auf eine Instanz I ist ein Ausdruck $R_i\underline{\mathbf{a}}$, wobei $\underline{\mathbf{a}}$ ein Tupel von Werten aus der Relation R_i darstellt und $i \leq n$ gilt. Die **aktive Domäne** einer Instanz I ($adom(I)$) ist die Menge aller Werte in den Relationen R_i für $1 \leq i \leq n$.

[1] Schema Mappings wurden in Kapitel 2 kurz erwähnt.

Bei der Beschäftigung mit Schema Mappings verwendet man üblicherweise ein **Quellschema** $S = (S_1, \ldots, S_n)$ und ein **Zielschema** $T = (T_1, \ldots, T_m)$. Diese Schemata sind disjunkt. Eine S-Instanz wird als **Quellinstanz** bezeichnet, eine T-Instanz dementsprechend als **Zielinstanz**. Im Folgenden wird bei einem Instanzenpaar (I, J) I als Quellinstanz und J als Zielinstanz aufgefasst.

Ein **Schema Mapping** $M = (S, T, \Sigma)$ umfasst ein Quellschema S, ein Zielschema T und eine Menge von Sätzen (Formeln ohne freie Variablen) Σ in einem logischen Formalismus. Das ist die **syntaktische Sicht** auf Schema Mappings.

Sei ein Schema Mapping $M = (S, T, \Sigma)$ gegeben. Aus **semantischer Sicht** lässt sich dieses Mapping als Tupel $M = (S, T, W)$ ausdrücken, wobei W die Menge aller Paare (I, J) ist, die alle Sätze in Σ erfüllen (notiert als $(I, J) \models \Sigma$). I ist eine Quell-, J eine Zielinstanz und J wird als **Lösung** für I bezeichnet.

Ten Cate und Kolaitis führen aus, dass die Verwendung von Prädikatenlogik erster Ordnung, wie sie z.B. der Relationenalgebra oder dem Relationenkalkül zugrunde liegt, zur Spezifikation von Schema Mappings die Unentscheidbarkeit von zentralen algorithmischen Problemen (wie der Anfragebearbeitung) bedingen würde, wenn es keine Einschränkungen gibt [CK10, S. 103f.].

Die beiden Autoren betrachten daher eine Menge von Operationen, die eine Mapping-Sprache unterstützen sollte, und formulieren diese in Prädikatenlogik erster Ordnung:

- **Umbenennung**: Eine Quellrelation in eine Zielrelation kopieren und umbenennen.

  $\rightsquigarrow \forall x, y, z \colon S(x, y, z) \rightarrow T(x, y, z)$

- **Projektion** (Spalten löschen): Die Zielrelation entsteht aus der Quellrelation durch das Löschen einer oder mehrerer Spalten.

  $\rightsquigarrow \forall x, y, z \colon S(x, y, z) \rightarrow T(x, y)$

- **Hinzufügung** (Spalten hinzufügen): Die Zielrelation entsteht aus der Quellrelation durch das Hinzufügen einer oder mehrerer Spalten.

  $\rightsquigarrow \forall x, y \colon S(x, y) \rightarrow \exists z T(x, y, z)$

- **Dekomposition**: Eine Quellrelation wird in zwei oder mehr Zielrelationen zerlegt.

  $\rightsquigarrow \forall x, y, z \colon S(x, y, z) \rightarrow T_1(x, y) \wedge T_2(y, z)$

- **Verbund**: Eine Zielrelation wird durch die Verbindung von zwei oder mehr Quellrelationen erstellt.

  $\rightsquigarrow \forall x, y, z \colon S_1(x, y) \wedge S_2(y, z) \rightarrow T(x, y, z)$

- **Kombinationen** der obigen Operationen

Diese logischen Ausdrücke gehören zur Klasse der **source-to-target tupel generating dependencies**. Diese sind wie folgt formal definiert:

- Ein Ausdruck $R_i(x)$ wird als eine *atomare Formel* über D bezeichnet, wenn $D = (R_1, \ldots, R_n)$ ein Datenbankschema und $\underline{\mathbf{x}}$ ein Tupel von Variablen ist, deren Anzahl der Stelligkeit von R_i entspricht sowie $i \leq n$ gilt.
- Eine **tuple generating dependency** (TGD) ist eine Formel der Form $\forall \underline{\mathbf{x}}\colon \phi(\underline{\mathbf{x}}) \rightarrow \exists \underline{\mathbf{y}} \psi(\underline{\mathbf{x}}, \underline{\mathbf{y}})$, wobei $\underline{\mathbf{x}}$ und $\underline{\mathbf{y}}$ Tupel von Variablen, $\phi(\underline{\mathbf{x}})$ eine Konjunktion von atomaren Formeln mit Variablen aus $\underline{\mathbf{x}}$ (jede Variable aus $\underline{\mathbf{x}}$ tritt in mindestens einem Glied von $\phi(\underline{\mathbf{x}})$ auf) und $\psi(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ eine Konjunktion atomarer Formeln mit Variablen aus $\underline{\mathbf{x}}$ und $\underline{\mathbf{y}}$ sind.
- Eine **equality generating dependency** (EGD) ist eine Formel der Form $\forall \underline{\mathbf{x}}\colon \phi(\underline{\mathbf{x}}) \rightarrow x_1 = x_2$, wobei $\underline{\mathbf{x}}$ und $\phi(\underline{\mathbf{x}})$ analog zu TGDs definiert sind sowie x_1 und x_2 Variablen aus $\underline{\mathbf{x}}$ sind.
- Eine TGD wird als **voll** bezeichnet, wenn sie keinen Existenzquantor auf der rechten Seite enthält, also die Form $\forall \underline{\mathbf{x}}\colon \phi(\underline{\mathbf{x}}) \rightarrow \psi(\underline{\mathbf{x}})$ hat.
- Eine **source-to-target tuple generating dependency** (S-T TGD) ist eine TGD, bei der $\phi(\underline{\mathbf{x}})$ eine Konjunktion über einem Quellschema S und $\psi(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ eine Konjunktion über einem Zielschema T ist.

S-T TGDs werden auch als **global-and-local-as-view dependencies** (GLaV) bezeichnet. Hierbei gibt es zwei bedeutende Spezialfälle:

- **Global-as-view dependency** (GaV): Die rechte Seite der Implikation in der S-T TGD besteht aus einer einzelnen atomaren Formel.

  $\rightsquigarrow \forall \underline{\mathbf{x}}\colon \phi(\underline{\mathbf{x}}) \rightarrow R(\underline{\mathbf{x}'})$, wobei $\phi(\underline{\mathbf{x}})$ eine Konjunktion von atomaren Formeln über einem Quellschema ist, $R(\underline{\mathbf{x}'})$ eine atomare Formel über einem Zielschema und die Variablen aus $\underline{\mathbf{x}'}$ in $\underline{\mathbf{x}}$ enthalten sind.
- **Local-as-view dependency** (LaV): Die linke Seite der Implikation in der S-T TGD besteht aus einer einzelnen atomaren Formel.

  $\rightsquigarrow \forall \underline{\mathbf{x}}\colon R(\underline{\mathbf{x}}) \rightarrow \exists \underline{\mathbf{y}} \psi(\underline{\mathbf{x}}, \underline{\mathbf{y}})$, wobei $R(\underline{\mathbf{x}})$ eine atomare Formel über einem Quellschema und $\psi(\underline{\mathbf{x}}, \underline{\mathbf{y}})$ eine Konjunktion von atomaren Formeln über einem Zielschema ist.

Als Beispiel sollen die zwei Quelldatenbankschemata $S_{BIB} = \{\texttt{BUCHTITEL}, \texttt{VORSCHLAG}\}$ und $S_{SHOP} = \{\texttt{SHOP\_BUCH}\}$ sowie das Zielschema $T = \{\texttt{BUCH}\}$ betrachtet werden.

Für die Relationenschemata aus S_{BIB} und S_{SHOP} ergibt sich damit `BUCHTITEL(ISBN, Titel, Autor_Vorname, Autor_Nachname, Verlag, Verlagsort)`, `VORSCHLAG(ISBN, Personalnummer, Leserausweisnummer, Kommentar, Bearbeitungsstatus` und `SHOP_BUCH(ISBN, Titel, Autor, Kurzbeschreibung, Zustand, Preis)`.

Für das im Zielschema T enthaltene Relationenschema `BUCH` soll `BUCH(ISBN, Titel, Autor, Verlag, Verlagsort)` gelten. Es sollen zwei Schema Mappings $M_1 = (S_1, T, \Sigma_1)$ und $M_2 = (S_2, T, \Sigma_2)$ mit

$$\Sigma_1 = \{\forall x, y, z, u, v, w, s, t, r, i \colon \texttt{BUCHTITEL}(x, y, z, u, v, w) \wedge \texttt{VORSCHLAG}(x, s, t, r, i) \rightarrow \texttt{BUCH}(x, y, u, v, i)\} \tag{3.1}$$

$$\Sigma_2 = \{\forall x, y, z, u, v, t \colon \texttt{SHOP\_BUCH}(x, y, z, u, v, t) \rightarrow \exists a, b\, \texttt{BUCH}(x, y, z, a, b)\} \tag{3.2}$$

definiert werden.

Die in Σ_1 enthaltene S-T TGD ist eine GAV-Abhängigkeit, während die in Σ_2 enthaltene S-T TGD eine LAV-Abhängigkeit darstellt.

Das Schema Mapping M_1 wird durch das Instanzenpaar (I_1, J_1) erfüllt. I_1 ist dabei eine Beispielinstanz zum Quellschema S_1 und wird in Tabelle 3.1 dargestellt, während J_1 eine Beispielinstanz zum Zielschema T ist und in Tabelle 3.2 angegeben wird.

In Tabelle 3.3 ist eine Beispielinstanz I_2 zum Quellschema S_2 dargestellt. Eine Beispielinstanz J_2 zum Zielschema T ist in Tabelle 3.4 angegeben. Das Symbol $\perp$ soll hierbei einen Nullwert repräsentieren, d.h. im Sinne von [Mai83, S. 372ff.] einen existierenden, aber unbekannten Wert. Das Instanzenpaar (I_2, J_2) erfüllt das Schema Mapping M_2.

BUCHTITEL					
ISBN	**Titel**	**Autor_Vorname**	**Autor_Nachname**	**Verlag**	**Verlagsort**
3-518-01869-8	Der Steppenwolf	Hermann	Hesse	Suhrkamp	Frankfurt am Main
978-3-942656-29-0	Hundert Jahre Einsamkeit	Gabriel	García-Márquez	A. Springer	Berlin
978-3-86615-523-7	Catch 22	Joseph	Heller	Süddt. Zeitung GmbH	München
978-3-8321-6186-6	Karte und Gebiet	Michel	Houellebecq	DuMont	Köln

VORSCHLAG				
ISBN	**PNr.**	**LNr.**	**Kommentar**	**Status**
978-3-942656-29-0	453	712	wichtigstes Werk des Nobelpreisträgers	angenommen
978-3-942656-29-0	342	592	magischer Realismus	bereits vorgeschlagen
978-3-86615-523-7	897	1064	berühmte Kriegssatire	wurde beschafft
978-3-8321-6186-6	541	1401	mit dem Prix Goncourt ausgezeichnet	in Bearbeitung

Tabelle 3.1.: Beispielinstanz I_1 zum Quellschema S_1

SHOP_BUCH					
ISBN	**Titel**	**Autor**	**Beschreibung**	**Zustand**	**Preis**
3-404-13411-7	The Stand: Das letzte Gefecht	Stephen King	beschreibt ein Weltuntergangsszenario	neu	9,90 Euro
978-3-518-46131-0	Solaris	Stanislaw Lem	thematisiert den Kontakt mit einer außerirdischen Intelligenz	leichte Gebrauchsspuren	8,90 Euro
978-3-86615-523-7	Catch 22	Joseph Heller	Satire über die Absurdität des Krieges	Einband beschändigt	5,90 Euro
978-3-453-40784-8	Per Anhalter durch die Galaxis	Douglas Adams	humorvoller Science-Fiction-Roman	neu	10,00 Euro

Tabelle 3.3.: Beispielinstanz I_2 zum Quellschema S_2

BUCH				
ISBN	**Titel**	**Autor**	**Verlag**	**Verlagsort**
3-404-13411-7	The Stand: Das letzte Gefecht	Stephen King	$\perp$	$\perp$
978-3-518-46131-0	Solaris	Stanislaw Lem	$\perp$	$\perp$
978-3-86615-523-7	Catch 22	Joseph Heller	$\perp$	$\perp$
978-3-453-40784-8	Per Anhalter durch die Galaxis	Douglas Adams	$\perp$	$\perp$

Tabelle 3.4.: Beispielinstanz J_2 zum Zielschema T

BUCH				
ISBN	**Titel**	**Autor**	**Verlag**	**Verlagsort**
978-3-942656-29-0	Hundert Jahre Einsamkeit	García-Márquez	A. Springer	Berlin
978-3-86615-523-7	Catch 22	Heller	Süddt. Zeitung GmbH	München
978-3-8321-6186-6	Karte und Gebiet	Houellebecq	DuMont	Köln

Tabelle 3.2.: Beispielinstanz J_1 zum Zielschema T

3.2. Invertierbarkeit von Schema Mappings

Schema Mappings beschreiben eine gerichtete Abbildung von einem Quellschema in ein Zielschema und „lassen sich im Allgemeinen nicht einfach «umdrehen» bzw. invertieren“ [LN07, S. 125]. Es wäre jedoch wünschenswert, Schema Mappings automatisiert invertieren

lassen zu können, anstatt das zu einem gegebenen Schema Mapping inverse Mapping explizit definieren zu müssen. Melnik et al. beschreiben z.B. in [MRB03] Invertierung als eine primitive auf Mappings[2] ausführbare Operation, die im Rahmen des von ihnen entwickelten *Model-Management*-Frameworks[3] Rondo zur Verfügung steht[4].

Fagin hat in [Fag07] die Invertierbarkeit von Schema Mappings untersucht. Die Definition von Inversen ist dabei schwer fassbar, da Schema Mappings keine Funktionen sind, d.h. mit einer Quellinstanz können mehrere Zielinstanzen assoziiert sein und umgekehrt können einer Zielinstanz mehrere Quellinstanzen zugeordnet sein.

Die Definition einer Inversen nach Fagin soll im Folgenden dargelegt werden [Fag07, S. 8]. Dazu sind zunächst einige Festlegungen nötig.

- **Identitäts-Mapping**: Sei S ein Schema. Dann soll $\hat{S}$ ein Schema sein mit $\{\hat{R} \mid R \in S\}$, wobei $\hat{R}$ ein neues Relationensymbol ist, das dieselbe Stelligkeit wie das Relationensymbol R in S hat. Für jede Instanz I von S gilt, dass es eine entsprechende Instanz $\hat{I}$ für $\hat{S}$ gibt. Ein *Identitäts-Mapping* ist ein Schema Mapping $M_{id} = (S, \hat{S}, \Sigma_{id})$, wobei $\Sigma_{id} = \{R(x_1, \cdots, x_n) \rightarrow \hat{R}(x_1, \cdots, x_n) \mid R \in S\}$ gilt[5]. Eine Zielinstanz J ist eine Lösung für I unter dem Identitäts-Mapping, genau dann wenn $\hat{I} \subseteq J$ gilt[6].
- **Äquivalenz von Schema Mappings**: Zwei Schema Mappings mit demselben Quell- und demselben Zielschema sind *äquivalent* in Bezug auf eine Quellinstanz I, wenn sie dieselbe Menge von Lösungen aufweisen.
- **Komposition von Schema Mappings**: Seien $M_{12} = (R_1, R_2, \Sigma_{12})$ und $M_{23} = (R_2, R_3, \Sigma_{23})$ zwei Schema Mappings, sodass R_1, R_2 und R_3 paarweise keine gemeinsamen Relationssymbole aufweisen. Die Komposition von M_{12} und M_{23} wird durch die Kompositionsformel $\Sigma_{12} \circ \Sigma_{23}$ ausgedrückt, wobei gelten soll: Wenn I eine R_1-Instanz und J eine R_3-Instanz ist, so soll $(I, J) \models \Sigma_{12} \circ \Sigma_{23}$ genau dann gelten, wenn es eine R_2-Instanz J' gibt, sodass $(I, J') \models \Sigma_{12}$ und $(J', J) \models \Sigma_{23}$ gilt [Fag07, S. 7]. Fagin et al. haben in [FKPT05] gezeigt, dass die Kompositionsformel als *second order tuple generating dependency* (SO TGD) gegeben ist, wenn Σ_{12} und Σ_{23} endliche Mengen von S-T TGDS darstellen.

Seien $M_1 = (S, T, \Sigma_1)$ und $M_2 = (T, \hat{S}, \Sigma_2)$ zwei Schema Mappings, bezeichne σ die

[2] Der Begriff Mapping wird hierbei in einem allgemeineren Sinn verwendet, d.h. es ist kein konkreter Formalismus damit verbunden.

[3] Der Begriff *Model Management* wird von Bernstein in [Ber03] eingeführt und bezeichnet einen Ansatz zur Verwaltung von Metadaten, bei dem Modelle (z.B. relationale Schemata oder Interfaces in Programmiersprachen) und Beziehungen zwischen Modellen (Mappings) im Vordergrund stehen.

[4] Die Komposition von Mappings ist ein Beispiel für eine andere primitive Operation.

[5] Da man bei den Schema Mappings, die S-T TGDS als Formalismus nutzen, von einem Quell- in ein Zielschema abbildet und diese per Definition disjunkt sind (siehe 3.1), ist es bei der Definition des Identitäts-Mapping nicht möglich, ein Schema auf sich selbst abbilden.

[6] Es wird nicht nur $\hat{I} = J$ gefordert, da Σ_{id} aus S-T TGDs besteht, für die folgende Eigenschaft gilt: Wenn J eine Zielinstanz ist, die eine Lösung darstellt, so ist auch jede Zielinstanz J' mit $J \subseteq J'$ eine Lösung [Fag07, S. 3].

Kompositionsformel $\Sigma_1 \circ \Sigma_2$ und sei $M_{comp} = (S, \hat{S}, \sigma)$.

M_2 ist eine **Schema-Mapping-Inverse** von M_1, wenn M_{comp} und das Identitäts-Mapping M_{id} äquivalent für I sind. Dies bedeutet, dass M_2 eine Inverse von M_1 für I ist, wenn

$$(I, J) \models \sigma \quad \text{genau dann, wenn} \quad \hat{I} \subseteq J \tag{3.3}$$

gilt [Fag07, S. 9].

Man spricht davon, dass M_2 eine *S-Inverse* von M_1 ist, wenn S eine Menge von Quellinstanzen ist und M_2 nach obiger Definition eine Inverse von M_1 für jede Quellinstanz $I \in S$ darstellt. Beinhaltet die Menge S nur eine einzige Quellinstanz, so wird M_2 als **lokale Inverse** bezeichnet. Wenn S alle Quellinstanzen umfasst, handelt es sich bei M_2 um eine **globale Inverse**. Inversen müssen nicht einzigartig sein, es kann insbesondere mehrere globale Inversen geben [Fag07, S. 10].

Wenn M_2 eine Inverse von M_1 für zwei sich unterscheidende Quellinstanzen I_1 und I_2 ist, so unterscheidet sich die Menge an Lösungen für I_1 unter M_1 von der Lösungsmenge für I_2 unter M_1 [Fag07, S. 11f.]. Dies entspricht der intuitiven Vorstellung, dass M_2 das Schema Mapping M_1 nicht umkehren kann, wenn I_1 und I_2 dieselbe Lösungsmenge aufweisen, da die Lösungsmenge die einzige Information ist, auf deren Grundlage eine Invertierung möglich ist. Diese Feststellung führt zu einer notwendigen Bedingung für die Existenz globaler Inversen, die als **unique solutions property** bezeichnet wird: Für ein Schema Mapping $M = (S, T, \Sigma)$ und zwei unterschiedliche Quellinstanzen I_1 und I_2 muss gelten, dass die Lösungsmengen für I_1 und I_2 nicht identisch sind. Für LAV-Abbildungen (siehe 3.1) ist dies sogar ein hinreichendes Kriterium.

Für ein Schema Mapping $M = (S, T, \Sigma)$, das durch eine endliche Menge von S-T TGDs bestimmt ist, und eine Quellinstanz I lässt sich eine **kanonische lokale Inverse** M_I^{-1} von M für I konstruieren, sofern M invertierbar ist.

Dabei gilt $M_I^{-1} = (T, \hat{S}, \{\beta_{J^*,I}\})$ mit $J^* = chase_\Sigma(I)$, wobei mit $chase_\Sigma(I)$ die Anwendung des Chase-Algorithmus, wie er in [FKMP05, S. 11ff.] definiert wird, auf $(I, \emptyset)$ mit Σ bezeichnet wird. Mit $\beta_{J^*,I}$ wird eine TGD $\phi \rightarrow \psi$ bezeichnet, wobei ϕ eine Konjunktion der Fakten von J^* und ψ eine Konjunktion der Fakten von I darstellt. $\beta_{J^*,I}$ ist eine volle TGD [Fag07, S. 21].

Die kanonische lokale Inverse $M_I^{-1} = (T, \hat{S}, \Sigma_I^{-1})$ ist die allgemeinste Inverse für ein Schema Mapping $M = (S, T, \Sigma)$ und eine Quellinstanz I [Fag07, S. 21]. Dies bedeutet, dass für jede andere Inverse $M' = (T, \hat{S}, \Sigma')$ gilt, dass Σ_I^{-1} durch Σ' impliziert wird.

Die **kanonische S-Inverse** ist eine Inverse für ein durch eine endliche Menge von S-T TGDs definiertes Schema Mapping M und eine Menge von Quellinstanzen S. Die Menge S wird hierbei durch eine Menge von Einschränkungen Γ beschrieben, sodass S genau die Instanzen enthält, die Γ erfüllen. Es gilt dabei, dass Γ eine Menge von TGDs und EGDs

darstellt. Wenn der Fall $\Gamma = \emptyset$ vorliegt, wird die kanonische S-Inverse als **kanonische globale Inverse** bezeichnet.

Sei $M = (S_1, S_2, \Sigma)$ ein Schema Mapping. Die kanonische S-Inverse ist als $M^{-1} = (S_2, \hat{S_1}, \Sigma^{-1})$ definiert, wobei gilt:

- Für eine endliche Quellinstanz I hat die Anwendung des Chase-Algorithmus mit Γ auf I eine endliche Instanz zum Ergebnis oder schlägt dadurch fehl, dass versucht wird, zwei unterschiedliche Werte in I gleichzusetzen.
- S bezeichnet die Menge aller Quellinstanzen, die Γ erfüllen.
- Für jedes Relationensymbol R in S_1 sei I_R eine Quellinstanz, die nur aus dem Fakt $R(\underline{\mathbf{x}})$ besteht, wobei die Variablen in $\underline{\mathbf{x}}$ paarweise verschieden sind.
- Sei I_R^Γ das Ergebnis der Anwendung des Chase mit Γ auf I_R, wobei unterschiedliche Variablen in $\underline{\mathbf{x}}$ durch den Chase gleichgesetzt werden dürfen. Wenn $\Gamma = \emptyset$, so gilt $I_R^\Gamma = I_R$.
- Mit J_R^Γ sei das Resultat der Anwendung des Chase mit Σ auf I_R^Γ bezeichnet.
- Sei $\beta_{J_R^\Gamma, I_R^\Gamma}$ eine TGD $\phi \rightarrow \psi$, wobei ϕ eine Konjunktion der Fakten von J_R^Γ und ψ eine Konjunktion der Fakten von I_R^Γ ist. $\beta_{J_R^\Gamma, I_R^\Gamma}$ ist eine volle TGD [Fag07, S. 23], die als δ_R^Γ bezeichnet werden soll.
- Σ^{-1} beinhaltet für jedes Relationensymbol R in S_1 die entsprechende TGD δ_R^Γ.

In [Fag07, S. 28ff.] wird ein Verfahren vorgestellt, mit dem sich die Menge S von Quellinstanzen I bestimmen lässt, für die ein Schema Mapping $M_2 = (T, \hat{S}, \Sigma_2)$ eine Inverse für ein anderes Schema Mapping $M_1 = (S, T, \Sigma_1)$ darstellt. Die Formel Γ, die S charakterisiert, lässt sich dabei aus der Kompositionsformel $\Sigma_1 \circ \Sigma_2$ durch rein syntaktische Umformungen herleiten. Auf die Details des Verfahrens soll an dieser Stelle nicht weiter eingegangen werden.

Einige einfache Schema Mappings haben nach der hier vorgestellten Definition von Inversen keine globalen Inversen. Dazu gehört z.B. das Schema Mapping $M = (\{R_1\}, \{R_2\}, \{R_1(x, y) \rightarrow R_2(x)\})$, da dieses die unique solutions property verletzt: Die zwei unterschiedlichen Quellinstanzen $I_1 = \{R_1(1, 2)\}$ und $I_2 = \{R_1(1, 3)\}$ haben beide die Lösungsmenge $J = \{R_2(1)\}$. Es wurden daher weniger restriktive Definitionen von Inversen vorgeschlagen. Dazu zählen beispielsweise die *Quasi-Inversen* für Schema Mappings, die in [FKPT08] vorgestellt werden, oder das Konzept des *Maximum Recovery*, das in [APR09] präsentiert wird.

Weniger strikte Definitionen von Inversen eignen sich gut für die Wiederherstellung der Quelldaten bei einer materialisierten Integration, wenn z.B. die Quelldaten nicht mehr verfügbar sind. Das Ziel dabei besteht darin, möglichst viel des ursprünglichen Datenbestandes zu rekonstruieren. Die in dieser Arbeit zu entwickelnde Integrationstechnik soll hingegen eine exakte und verlustfreie Rückabbildung gewährleisten, weswegen weniger strikte Definitionen hier nicht näher betrachtet werden.

3.3. Datenintegration mit dem Entity-Attribute-Value-Modell

Die konventionelle Art der Modellierung in relationalen Datenbanken sieht vor, dass Attribute als Spalten einer Tabelle repräsentiert werden. Diese Art der Modellierung ist ungeeignet, wenn es eine große Anzahl von Attributen in Tabellen gibt, von denen aber nur wenige auf ein Tupel zutreffen [DN07, S. 2f.]. Man kann beispielsweise in einem medizinischen Anwendungsszenario eine große Anzahl von potentiellen Symptomen für verschiedene Krankheitsbilder definieren, von denen aber nur ein Bruchteil bei der Untersuchung eines einzelnen Patienten zutreffen wird. Nachteilig ist die konventionelle Modellierung auch dann, wenn häufig neue Attribute hinzugefügt und/oder alte Attribute entfernt werden, da dann das Schema der Tabellen geändert werden muss.

Das *Entity-Attribute-Value*-Modell kann in diesen Fällen als Alternative zur konventionellen Modellierung eingesetzt werden. Nachfolgend werden die zentralen Elemente dieses Modells aufgeführt.

Entity

Hiermit wird ein Gegenstand, Konzept oder Ereignis bezeichnet. Die Entitys können in einer Tabelle gespeichert werden, wobei sie über einen maschinell erzeugten Primärschlüssel identifiziert werden [DN07, S. 4]. Es können weitere Attribute wie ein Name, eine Beschreibung oder ein Typ, zu dem die Entity gehört, verzeichnet werden.

Attribute

Die Eigenschaften einer Entity werden durch Attribute beschrieben. Diese können ebenfalls in einer einzigen Tabelle verwaltet und über einen maschinell erzeugten Primärschlüssel identifiziert werden. Weiterhin könnte beispielsweise ein Name oder Angaben zur Validierung (Datentyp, Wertebereich, zulässige Wertemenge, regulärer Ausdruck, Default-Wert) erfasst werden. Bei der Speicherung der Attribute muss auf Duplikate und Konsistenz im Allgemeinen geachtet werden.

Value

Trifft ein Attribut auf eine Entity zu, so muss der zugehörige Attributwert erfasst werden. Hierbei wird auf die Entity und das entsprechende Attribut durch einen Fremdschlüssel Bezug genommen, der jeweils die Primärschlüssel der Entity- und Attribute-Tabelle referenziert. Eine einfache Art der Speicherung der Attributwerte besteht darin, alle Werte unabhängig vom tatsächlichen Datentyp des Attributs als String zu speichern. Dinu et al. führen jedoch an, dass dieser Ansatz ineffizient in Bezug auf die Indizierung der Attributwerte sei, wenn numerische oder Datums-Datentypen als String dargestellt werden. So sei keine optimisierte Bereichssuche möglich und für Anfragen, die einen Wertevergleich auf Zahlen durchführen, müsse während der Anfragebearbeitung eine Konvertierung vorgenommen werden [S. 4f.] [DN07].

Stattdessen schlagen sie drei Alternativen vor. Zum einen kann man für jeden Datentyp eine Spalte in der Values-Tabelle definieren sowie eine zusätzliche Indikatorspalte, die anzeigt, in welcher Spalte der Wert für ein gegebenes Tupel zu finden ist. Die restlichen Spalten enthalten den Wert NULL. Hierbei treten viele Nullwerte auf. Nach Ansicht von Dinu et al. sei der Mehraufwand an Speicherplatz für eine solche Tabelle nicht signifikant hoch, jedoch ergäben sich Probleme, wenn das zugrundeliegende Datenbankmanagementsystem (DBMS) Nullwerte indiziert, weil dies zu einer Verschwendung von Index-Speicherplatz führe. Ein besserer Ansatz besteht darin, für jeden Datentyp getrennte Tabellen zu verwenden. Bei der Ermittlung eines Attributwertes wird zunächst in der Attribute-Tabelle geprüft, welchen Datentyp das Attribut hat und anschließend der Wert in der entsprechenden Tabelle nachgeschlagen. Die dritte Variante betrifft nur das DBMS Microsoft SQL Server. Dieses bietet die Möglichkeit, dynamische Datentypen zu verwenden. Hierbei können jedoch durch die fehlende Typsicherheit Laufzeitfehler auftreten, wenn z.B. ein String-Wert versehentlich als Zahl interpretiert wird.

Der Unterschied zwischen der konventionellen relationalen und der EAV-Darstellung ist in Abbildung 3.1 verdeutlicht. Hierbei werden demografische Angaben zu Patienten sowie Laborwerte gespeichert. Bei der konventionellen Modellierung gibt es für demografische Angaben und die Laborwerte jeweils eine Tabelle. Im EAV-Modell werden Informationen zu beiden Bereichen auf die feste, generische Struktur von Entities, Attributen und Werten abgebildet.

Nadkarni et al. führen die folgenden **Vorteile** für das EAV-Modell an [NMC+99, S. 479f.]

- Es gibt keine Limitierung bezüglich der Anzahl von Attributen, die definiert werden können. Die meisten DBMS weisen eine Begrenzung für die Anzahl von Spalten pro Tabelle auf, sodass bei konventioneller Modellierung Daten auf mehrere Tabellen verteilt werden müssen, wenn diese Begrenzung erreicht wird. Im EAV-Modell ist dies nicht nötig. Des Weiteren können Attribute hinzugefügt und entfernt werden, ohne dass Änderungen am Schema notwendig wären.
- Es ist im Vergleich zur konventionellen Modellierung sehr speichereffizient, wenn es

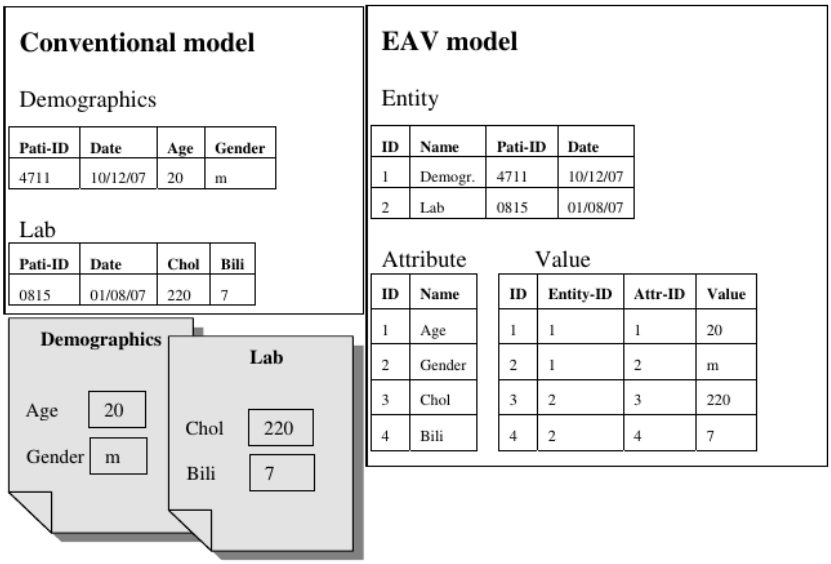

Abbildung 3.1.: Gegenüberstellung des konventionellen und des EAV-Modells (Quelle: [DWR10, S. 8])

viele Attribute gibt, die nur auf wenige Datensätze zutreffen.

- Das zugrundeliegende Datenformat ist einfach und selbst-beschreibend. Damit kann auf die gespeicherten Daten auch in einem neuen Anwendungsszenario zugegriffen werden, das bei der Speicherung der Daten eventuell noch gar nicht vorgesehen wurde.
- Objekt-bezogene Anfragen (Informationen zu einem bestimmten Objekt) sind effizienter umsetzbar als bei konventionell strukturierten Daten: Die Informationen zu einem Objekt/Entity liegen zentral in der Values-Tabelle vor (bzw. den Values-Tabellen, wenn für jeden Datentyp eine einzelne Tabelle eingesetzt wird) vor, sodass man hier nur mit dem Schlüssel der Entity filtern muss. Wurden die Daten hingegen konventionell modelliert, sind die Informationen zu einem Objekt üblicherweise über mehrere Tabellen verteilt. Dieser Vorteil ist jedoch nur dann gegeben, wenn pro Anfrage nur wenige Objekte ermittelt werden sollen.

Dem stehen die folgenden **Nachteile** gegenüber:

- Es ist ein höherer initialer Programmieraufwand erforderlich, um die gleichen Funktionalitäten wie bei einer konventionell modellierten Datenbank zu erhalten. Diese Arbeit muss aber nur einmal durchgeführt werden.
- Die Bearbeitung Attribut-zentrierter Anfragen ist ineffizienter und anspruchsvoller als bei konventioneller Modellierung. Es müssen aus der Value-Tabelle bzw. den Value-Tabellen zunächst alle Attributwertkombinationen ermittelt werden, die der Anfragebedingung entsprechen. Anschließend müssen die dazugehörigen Entities

festgestellt werden. Objekt-bezogene Anfragen, bei denen viele Objekte ermittelt werden sollen, sind ebenfalls nicht so effizient umsetzbar wie in einer konventionell modellierten Datenbank.

- Liegt ein einfaches Schema vor, das wenigen Änderungen unterworfen ist, ist der Mehraufwand, der durch die Einführung des EAV-Modells verursacht wird, nicht gerechtfertigt.

Nadkarni et al. führen in [NMC+99] ein Modell namens *EAV/CR* (EAV with classes and relationships) ein, das das EAV-Modell um Klassen und Beziehungen erweitert. Entities werden hierbei als Objekte verwaltet, die eine Instanz einer Entity-Klasse sind. Objekte, Klassen und Attribute werden über eindeutige Bezeichner (IDs) identifizert. Beziehungen zwischen Klassen (Assoziationen) werden als Klasse abgebildet. Des Weiteren können Klassen Komponentenklassen enthalten. Hierfür wird für die enthaltende Klasse ein Attribut definiert, das eine Objekt-ID zum Wert hat. Das EAV/CR-Modell führt neben den Tabellen für Objekte (Entities), Attribute und Werte noch Tabellen für Klassen und Vererbungshierarchien ein. Attribute werden in Abhängigkeit der Klasse, zu der sie gehören, definiert.

Löper et al. demonstrieren in [LKBH13] die Vorteile der Verwendung des EAV-Modells im Bereich der ambulanten Pflege. Hintergrund ist der Entwurf einer Speicherstruktur, die als Grundlage einer digitalen Pflegeakte eingesetzt werden soll. Da es im Bereich der digitalen Verwaltung von Pflegeinformation viele verschiedene Standards gibt, besteht die Notwendigkeit einer flexiblen Speicherstruktur, die bestehende Standards unterstützt und leicht erweiterbar für neue Standards ist. Darüber hinaus muss berücksichtigt werden, dass zwar viele mögliche Attribute definiert werden können, von denen aber nur wenige auf eine Dateninstanz zutreffen werden. Weitere Anforderungen bestehen bezüglich der Nachvollziehbarkeit von Änderungen (Versionierung), der Wiederherstellung von Originaldokumenten und der Erweiterbarkeit, um neues Wissen und neue Prozesse im Pflegewesen integrieren zu können [LKBH13, S. 16].

Es wird dargelegt, dass das EAV-Modell diesen Anforderungen gerecht wird. Eine Erweiterung wird bezüglich der Abbildung von Hierarchien vorgenommen sowie um Angaben zum Quelldokument, dem ein gespeicherter Datensatz entnommen ist. Auf eine Verbindung zwischen Entities und Attribute wurde verzichtet, da dies zu einer redundanten Darstellung von Attributen führen würde, wenn in den abzubildenden Quelldaten Attribute für eine Vielzahl von Entities definiert werden können [LKBH13, S. 18f.].

In den Abbildungen 3.2 und 3.3 sind jeweils der Import- und Export-Prozess dargestellt, die entwickelt wurden, um Quelldaten in die EAV-Speicherstruktur zu importieren und wieder zu exportieren, indem ein Pflegebericht in einem bestimmten Format erzeugt wird. Der Import-Prozess erfolgt dabei voll automatisiert, ohne dass es einen Eingriff seitens eines Domänenexperten bedarf. Beim Export-Prozess muss von der Pflegekraft eine Auswahl der zu exportierenden Daten getroffen werden. Diese werden schließlich anhand von Mapping-Regeln in das gewünschte Zielformat transformiert. Anschließend wird geprüft, ob die erzeugten Dokumente vollständig und valide sind. Andernfalls wird

der Prozess oder ein Teil des Prozesses wiederholt [LKBH13, S. 20].

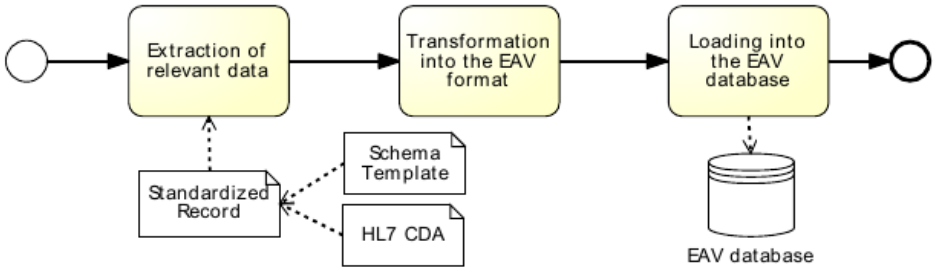

Abbildung 3.2.: Import-Prozess (Quelle: [LKBH13, S. 21])

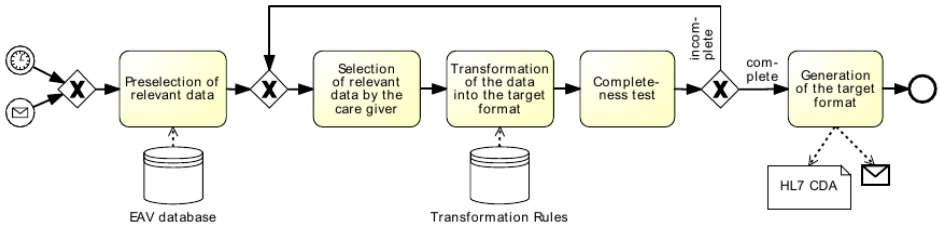

Abbildung 3.3.: Export-Prozess (Quelle: [LKBH13, S. 21])

3.4. Zusammenfassung

In diesem Kapitel wurde der Formalismus der Schema Mappings erläutert. Dazu wurde eine Definition von Schema Mappings vorgestellt und Global-as-view- und Local-as-view-Abhängigkeiten als wichtige Spezialfälle betrachtet. Im Anschluss daran wurde die Idee von Schema-Mapping-Inversen motiviert und die Definitionen von Inversen nach Fagin angegeben (lokale Inversen, S-Inversen und globale Inversen). Es wurden Konstruktionsverfahren für Inversen vorgestellt sowie ein notwendiges Kriterium für globale Invertierbarkeit, das für Local-as-view-Mappings auch hinreichend ist. In 4.1 wird auf der Grundlage der Darstellung aus diesem Kapitel untersucht, inwiefern sich die in dieser Arbeit zu entwickelnde Technik mit Hilfe von Schema Mappings und ihren Inversen umsetzen lässt.

Darüber hinaus wurde das EAV-Modell präsentiert. Hierbei wurden die Komponenten dieses Modells (Entity, Attribute und Value) eingeführt, wobei in diesem Zusammenhang auch auf die Umsetzung dieses Modells mit Relationen eingegangen wurde. Es wurden Vor- und Nachteile angeführt und abschließend ein konkretes Anwendungsbeispiel vorgestellt. Auf der Basis der Ausführungen in diesem Kapitel wird in 4.2 ein erweitertes EAV-Modell für das Konzept der „Global-as-local-view-extension“-Technik vorgestellt.

4. Konzept für die „Global-as-local-view-extension“-Technik

In diesem Kapitel wird auf Grundlage der in den Kapiteln 2 und 3 vorgestellten Grundlagen und aktuellen Forschungsansätze zur Datenintegration ein Konzept für die Integrationstechnik vorgestellt, die in dieser Arbeit entwickelt werden soll.

Der Unterschied zwischen der Global-as-local-view-extension-Technik und dem „herkömmlichen“ Ansatz zur Datenintegration wird durch die Abbildungen 4.1 und 4.2 dargestellt. In Abbildung 4.1 ist der Arbeitsablauf in einer Föderation angegeben, in der das globale Schema im Mittelpunkt steht. Die Daten der lokalen Quellen werden im globalen Schema integriert. Die Anfragen der Nutzer müssen nun an die globale Datenbank gestellt werden, was zur Folge hat, dass diese mit einem für sie ungewohnten neuen Schema arbeiten müssen und eventuell auch mit einem anderen Datenmodell konfrontiert sind, falls das Datenmodell des globalen Schemas nicht dem lokalen Datenmodell entspricht (wenn z.B. lokal eine relationale Datenbank genutzt wird, während global eine Datenbank auf der Basis des objektorientierten Datenmodells verwendet wird). Updates der lokalen Schemata erfolgen weiterhin lokal.

In Abbildung 4.2 wird gezeigt, wie in der in dieser Arbeit behandelten Integrationstechnik die integrierten Daten als Erweiterung der lokalen Schemata dargestellt werden. Alle Anfragen werden ausschließlich an die lokalen Datenbanken gestellt, auch wenn sie Schemabestandteile anderer lokaler Datenquellen betreffen. Updates erfolgen auch hier lokal. Die Integration der lokalen Quellen bringt für den einzelnen Nutzer somit keine Veränderung in Bezug auf Schema und Datenmodell mit sich.

Die Idee der Global-as-local-view-extension-Technik wurde in [FHLM96] vorgestellt und von Conrad in [Con97] als Misch-Architektur bezeichnet, die auf der grundlegenden Annahme basiere, dass es „keine wirklichen globalen Anwendungen gibt, sondern alle Anwendungen lokal auf einem der Komponentensysteme laufen“ [Con97, S. 62].

Der ursprünglichen Idee der Global-as-local-view-extension-Technik, wie sie in [FHLM96] vorgestellt wurde, liegt ein globales konzeptionelles Schema zugrunde, zu dem Abbildungen aus und Rückabbildungen in die lokalen Schemata definiert werden. In dieser Arbeit wurde in Unterkapitel 4.1 zunächst untersucht, wie ein solches globales Schema mit Schema Mappings und ihren Inversen definiert werden kann. Es wird erläutert, warum vor dem Hintergrund der zu erfüllenden Anforderungen die Verwendung von Schema Mappings, wie sie in 3.1 vorgestellt wurden, mit Nachteilen verbunden ist. Die Nutzung des EAV-Modells als Basis eines generischen globalen Schemas wird motiviert. Es wird dargelegt, warum sich Abbildungen und Rückabbildungen in und aus dem EAV-Modell mit den in 3.1 vorgestellten Schema Mappings nur unzureichend beschreiben lassen.

Das erweiterte EAV-Modell, das in dieser Arbeit verwendet wird, wird in 4.2 präsentiert.

Dieses ist datenmodellunabhängig. Die Abbildung zwischen dem generischen Schema und den lokalen Datenquellen erfolgt somit auf Basis des konkreten Datenmodells der Quelle, indem beschrieben wird, wie Modellierungskonstrukte des Datenmodells (Metadaten) auf Elemente des EAV-Modells (Daten) abgebildet werden. Dadurch entfällt die Notwendigkeit, für jedes konkrete Quellschema eine Abbildung in das EAV-Modell festzulegen.

In Unterkapitel 4.3 wird angegeben, wie das relationale und das XML-Datenmodell im erweiterten EAV-Modell dargestellt werden. Die Integration von Daten auf Basis der zusicherungsbasierten Schemaintegration wird in 4.4 dargestellt. Abschließend werden in 4.5 die Rückabbildungen aus dem EAV-Modell in das Relationenmodell und das XML-Datenmodell vorgestellt.

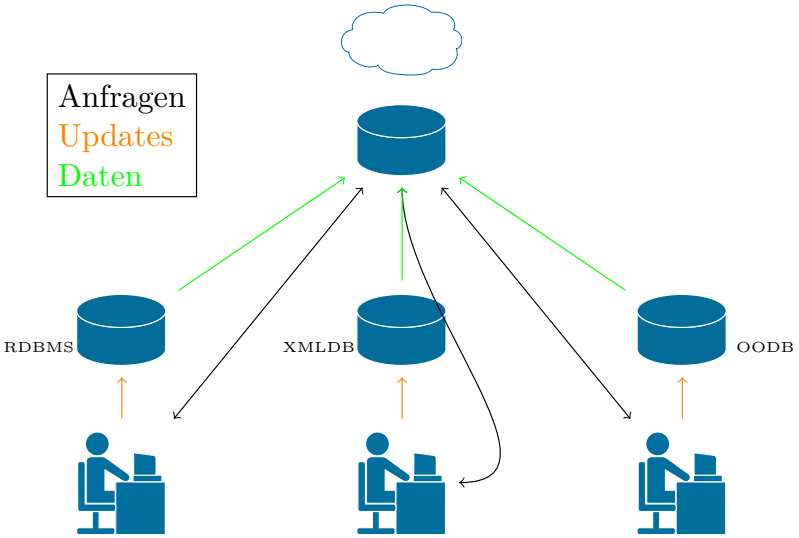

Abbildung 4.1.: Datenintegration mit globalem Schema

4.1. Integration mit Schema Mappings und Schema-Mapping-Inversen

Auf Grundlage der Ausführungen aus 3.1 und 3.2 zu Schema Mappings und ihren Inversen sollen in diesem Unterkapitel Hin- und Rückabbildungen von lokalen Relationenschemata in ein globales relationales Schema untersucht werden. Hierzu seien $S_1, \ldots, S_n$ lokale Quelldatenbankschemata und $T = (R_1^T, \ldots, R_m^T)$ ein globales Zieldatenbankschema.

Für ein Quelldatenbankschema $S_i = (R_1^i, \ldots, R_k^i)$ und dem Zieldatenbankschema T sei ein Schema Mapping $M_i = (S_i, T, \Sigma_i)$ definiert, wobei gilt:

$$\Sigma_i = \{\forall a_1, \ldots, a_s \colon \phi(a_1, \ldots, a_s) \rightarrow \psi(x_1, \ldots, x_{i-1}, \underset{\| \; x_i}{a_1}, \ldots, \underset{\| \; x_{i+s-1}}{a_s}, x_{i+s}, \ldots, x_t)\} \tag{4.1}$$

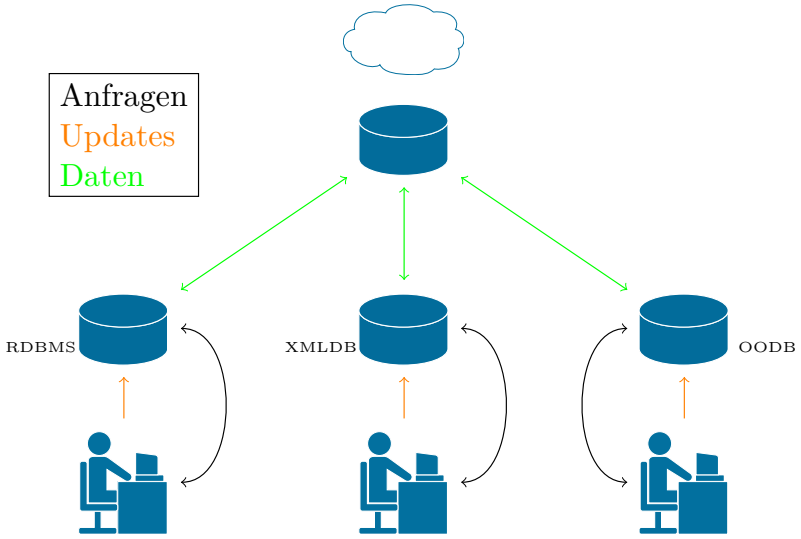

Abbildung 4.2.: Integrierte Daten als Erweiterung des lokalen Schemas

Dabei ist ϕ eine Konjunktion von Relationenschemata aus S_i über einer Menge von Attributen $\{a_1, \ldots, a_s\}$ und ψ eine Konjunktion von Relationenschemata aus T über einer Attributmenge $\{x_1, \ldots, x_t\}$. Es soll gelten, dass die Attributmenge $\{a_1, \ldots, a_s\}$ eine Teilmenge der Attributmenge $\{x_1, \ldots, x_t\}$ ist. Dies ist notwendig, da für die Rückabbildung einer Relation alle ursprünglichen Attribute vorhanden sein müssen, weil andernfalls Informationen verloren gehen und die Relation nicht vollständig rekonstruiert werden kann. Eine Alternative bestünde darin, den Wert mehrerer Attribute in einem Attribut durch eine geeignete Operation zusammenzufassen (z.B. durch Konkatenation von String-Werten), wodurch die geforderte Teilmengenbeziehung nicht mehr notwendig wäre. Bei der Rückabbildung müssen aus dem Wert des kombinierten Attributs die Werte der ursprünglichen Attribute rekonstruiert werden. Diese Art der Codierung und Decodierung von Attributen ist jedoch mit einem deklarativen Ansatz, wie er in dieser Arbeit erarbeitet werden soll, nicht umsetzbar. Die für die Codierung und Decodierung notwendigen Operationen machen einen prozeduralen Ansatz erforderlich.

Die Forderung nach der Teilmengenbeziehung bleibt somit bestehen. Diese führt jedoch dazu, dass bei Tupeln in einer globalen Relation viele Attribute des Relationenschemas nicht zutreffen, d.h. dass dort Nullwerte gespeichert sind.

Diese Problematik soll an einem Beispiel veranschaulicht werden. Seien dazu drei Quelldatenbankschemata $S_1 = (\texttt{SHOP\_BUCH})$, $S_2 = (\texttt{PRIV\_BUCHTITEL}, \texttt{PRIV\_BUCH})$ und $S_3 = (\texttt{BIB\_BUCHTITEL}, \texttt{BIB\_EXEMPLAR})$ gegeben, wobei gilt:

- `SHOP_BUCH(ISBN, Autor, Titel, Kurzbeschreibung, Zustand, Preis, Genre)`
- `PRIV_BUCHTITEL(ISBN, Autor, Titel, Genre, Bewertung)`, `PRIV_BUCH(ISBN, Anschaffungspreis, Anschaffungsdatum, Seitenanzahl, Einband)`

- BIB_BUCHTITEL(ISBN, Autor, Titel, Genre, Verlag, Verlagsort), BIB_EXEMPLAR(Inventarnummer, Auflage, ISBN, Filiale)

Sei $T = (\texttt{BUCH})$ das Zielschema, auf das die Quellschemata abgebildet werden sollen, wobei BUCH(ISBN, Autor, Titel, Genre, Kurzbeschreibung, Zustand, Preis, Bewertung, Anschaffungspreis, Anschaffungsdatum, Seitenanzahl, Einband, Verlag, Verlagsort, Inventarnummer, Auflage, Filiale) gilt. Es seien die Schema Mappings $M_1 = (S_1, T, \Sigma_1)$, $M_2 = (S_2, T, \Sigma_2)$ und $M_3 = (S_3, T, \Sigma_3)$ mit

$$\Sigma_1 = \{\forall a_1, \ldots, a_7 \colon \texttt{SHOP\_BUCH}(a_1, \ldots, a_7) \rightarrow \texttt{BUCH}(a_1, a_2, a_3, a_7, a_4, a_5, a_6, x_8, \ldots, x_{17})\} \tag{4.2}$$

$$\Sigma_2 = \{\forall a_1, \ldots, a_9 \colon \texttt{PRIV\_BUCHTITEL}(a_1, \ldots, a_5) \wedge \texttt{PRIV\_BUCH}(a_1, a_6, \ldots, a_9) \rightarrow \texttt{BUCH}(a_1, \ldots, a_4, x_5, x_6, x_7, a_5, \ldots, a_9)\} \tag{4.3}$$

$$\Sigma_3 = \{\forall a_1, \ldots, a_9 \colon \texttt{BIB\_BUCHTITEL}(a_1, \ldots, a_6) \wedge \texttt{BIB\_EXEMPLAR}(a_7, a_8, a_1, a_9) \rightarrow \texttt{BUCH}(a_1, \ldots, a_4, x_5, \ldots, x_{12}, a_5, \ldots, a_9, x_{13}, \ldots, x_{17})\} \tag{4.4}$$

gegeben.

Das globale Relationenschema Buch weist insgesamt siebzehn Attribute auf, von denen aber nur fünf Attribute (ISBN, Titel, Autor und Genre) in den Schemata aller lokalen Quellen vorzufinden sind. Angenommen, dass jede lokale Datenquelle jeweils 10.000 Datensätze enthält, dann besteht die globale Relation Buch aus 30.000 Tupel und somit $30.000 \times 17 = 510.000$ Attributwerten. Unter den Attributwerten befinden sich dabei 260.000 Nullwerte und 250.000 Nicht-Nullwerte[1].

Unabhängig von dieser Problematik ist die Frage nach Schema-Mapping-Inversen, die eine Rückabbildung aus dem globalen Schema in die lokalen Schemata ermöglichen. In 3.2 wurde die Definition einer Schema-Mapping-Inversen nach Fagin sowie Verfahren zur Konstruktion von kanonischen Inversen vorgestellt, wobei die kanonische globale Inverse (die Inverse für alle Quellinstanzen) von größtem Interesse ist. Diese Verfahren setzen voraus, dass das zu invertierende Schema Mapping eine Inverse besitzt, d.h. invertierbar ist. Es gibt jedoch kein umfassendes Kriterium, um zu entscheiden, ob ein Schema Mapping eine globale Inverse besitzt. Die *unique solutions property*[2] stellt eine notwendige Bedingung für die Existenz einer globalen Inversen dar und ist nur für

[1] Aus dem Quelldatenbankschema S_1 stammen dabei $10.000 \times 7 = 70.000$ Nicht-Null- und $10.000 \times 10 = 100.000$ Nullwerte (sieben Attribute im globalen Schema sind auch im lokalen Schema S_1 enthalten, während dies auf zehn Attribute nicht zutrifft). Aus den Schemata S_2 und S_3 finden sich jeweils $10.000 \times 9 = 90.000$ Nicht-Null- und $10.000 \times 8 = 80.000$ Nullwerte in der globalen Relation Buch wieder (auf S_2 als auch S_3 treffen neun globale Attribute zu).

[2] Zwei unterschiedliche Quellinstanzen I_1 und I_2 dürfen in Bezug auf ein Schema Mapping nicht dieselbe Menge von Zielinstanzen aufweisen. Andernfalls liegt nicht genug Information vor, um aus einer Zielinstanz J mit Hilfe einer Schema-Mapping-Inversen die ursprüngliche Quellinstanz wiederherzustellen, und es kann keine globale Inverse für das Schema Mapping geben.

LAV-Abbildungen auch eine hinreichende Bedingung. Dies steht einer automatisierten Ableitung von globalen Inversen für Schema Mappings im Wege und erfordert menschliches Eingreifen. Weiterhin stellt sich die Frage, wie mit Schema Mappings umgegangen wird, die keine globale Inverse haben.

Schema Mappings und EAV

Die Problematik, dass Tupel in der globalen Relation viele Nullwerte aufweisen, legt die Verwendung des EAV-Modells nahe, das sich für genau dieses Szenario eignet. Ein weiterer Vorteil besteht darin, dass in das EAV-Modell nicht nur das Relationenmodell, sondern auch andere Datenmodelle abgebildet werden können, während Schema Mappings stark auf das Relationenmodell zugeschnitten sind. Wie bereits in 3.3 beschrieben, wird z.B. in [LKBH13] demonstriert, wie XML-Quelldatensätze im EAV-Modell dargestellt werden.

Die Verwendung des EAV-Modells erfordert eine Abbildung von Metadaten auf Daten, während Schema Mappings Abbildungen zwischen Metadaten beschreiben. Bei dem Versuch, die Abbildung zwischen konventioneller Modellierung und EAV-Modellierung für das Relationenmodell mit Schema Mappings zu beschreiben, ergibt sich das Problem, dass der Zusammenhang zwischen den Metadaten der Ursprungsdaten in der EAV-Darstellung verloren geht. Sei ein Schema Mapping $M = (S, T, \Sigma)$ mit $S = \{\mathtt{R_1}, \ldots, \mathtt{R_n}\}$, $T = \{\mathtt{Entity}, \mathtt{Attribute}, \mathtt{Value}\}$ gegeben. Dabei gilt, dass $\mathtt{R_1}, \ldots, \mathtt{R_n}$ Relationenschemata mit einer unterschiedlichen Anzahl von Attributen sind, sowie `Entity(EntityID, EntityName)`, `Attribute(AttributeID, AttributeName)` und `Value(EntityID, AttributeID, Value)`[3]. Das Schema Mapping M soll jedes Tupel einer Relation einer Quellinstanz auf eine EAV-Darstellung abbilden. Es gilt somit:

$$\begin{aligned}\Sigma = \{\forall x_1, \cdots, x_n : \mathtt{R}(x_1, \cdots, x_n) \rightarrow \exists a, b\, \mathtt{Entity}(a, b) \wedge \exists c_1, \cdots, c_n,\\ d_1 \cdots d_n\, \mathtt{Attribute}(c_1, d_1) \wedge \cdots \wedge \mathtt{Attribute}(c_n, d_n) \wedge\\ \mathtt{Value}(a, c_1, x_1) \wedge \cdots \wedge \mathtt{Value}(a, c_n, x_n) \mid \mathtt{R} \in S \wedge b \neq c_1 \neq \cdots \neq c_n\}\end{aligned} \tag{4.5}$$

Jedes Tupel der Ursprungsrelation `R` soll durch den Wert des Attributs `EntityID` der Relation `Entity` eindeutig identifiziert werden. Darüber hinaus werden alle Attribute von `R` in der Zielrelation `Attribute` vermerkt, wobei auch hier jedes Attribut über den Wert des Attributes `AttributeID` eindeutig identifiziert werden soll. Ferner werden der Name der Relation `R` und die Namen der zugehörigen Attribute von `R` als Werte der Attribute `EntityName` bzw. `AttributeName` erfasst. In der Relation `Value` wird schließlich für jedes Tupel und Attribut der entsprechende Wert festgehalten.

Im Folgenden soll an einem Beispiel veranschaulicht werden, dass dieses Schema Mapping

[3] Im Relationenschema `Value` sind `EntityID` und `AttributeID` Fremdschlüssel in Bezug auf die gleichnamigen Attribute der Relationenschemata `Entity` und `Attribute`.

auch Instanzenpaare (I, J) zulässt, die nicht der gewünschten Abbildung entsprechen. Seien dazu

- $S =$ `Buch` mit `Buch(Titel, Autor)` ein Quellschema
- $I = \{\texttt{Buch}(\text{The Stand}, \text{Stephen King})\}$ eine Quellinstanz
- $J = \{\texttt{Entity}(\alpha, \texttt{Buch}), \texttt{Attribute}(\beta_1, \texttt{Titel}), \texttt{Attribute}(\beta_2, \texttt{Autor}), \texttt{Value}(\alpha, \beta_1, \text{The Stand}), \texttt{Value}(\alpha, \beta_2, \text{Stephen King})\}$ eine Zielinstanz

Das Instanzenpaar (I, J) entspricht dem Schema Mapping M und der Intention, mit der dieses Schema Mapping entworfen wurde.

Sei $J' = \{\texttt{Entity}(\alpha, \texttt{Film}), \texttt{Attribute}(\beta_1, \texttt{Titel}), \texttt{Attribute}(\beta_2, \texttt{Regisseur}), \texttt{Value}(\alpha, \beta_1, \text{The Stand}), \texttt{Value}(\alpha, \beta_2, \text{Stephen King})\}$ eine weitere Zielinstanz. Das Instanzenpaar (I, J') ist im Sinne des Schema Mappings M zulässig, entspricht jedoch nicht der informellen Vorstellung einer gültigen Abbildung.

Das Problem liegt darin begründet, dass die Abbildung von Metadaten (in diesem Fall Relationen- und Attributnamen) auf Daten (Attributwerte der Relationen `Entity` und `Attribute`) nicht mit Schema Mappings beschrieben werden kann, weil der den Schema Mappings zugrundeliegende Formalismus der S-T TGDs einen eingeschränkten Ausschnitt aus der Prädikatenlogik erster Ordnung darstellt (siehe 3.1). Um diese Abbildung beschreiben zu können, benötigt man jedoch einen Formalismus, der auf einem Logikkalkül höherer Ordnung basiert. Es ergibt sich jedoch, wie bereits in 3.1 erwähnt, schon bei der uneingeschränkten Verwendung der Prädikatenlogik erster Ordnung die Unentscheidbarkeit zentraler algorithmischer Probleme wie der Anfragebearbeitung [CK10, S. 103f.].

In [LN07] wird in diesem Zusammenhang von *höherstufigen Korrespondenzen* gesprochen, die zur Überwindung schematischer Heterogenität eingesetzt werden können, indem unterschiedliche Datenmodellelemente, die denselben Sachverhalt darstellen, miteinander in Bezug gesetzt werden. Es wird jedoch erwähnt, dass den Autoren kein Ansatz bekannt wäre, der höherstufige Korrespondenzen verarbeiten könnte [LN07, S. 129].

An einem weiteren Beispiel soll gezeigt werden, dass auch die *unique solutions property* für das Schema Mapping M verletzt ist, sodass keine globale Inverse für M existieren kann (siehe 3.2). Seien hierfür

- $S =$ `R` mit `R(X, Y)` ein Quellschema
- $I_1 = \{\texttt{R}(0, 1)\}$ und $I_2 = \{\texttt{R}(1, 0)\}$ zwei unterschiedliche Quellinstanzen
- $Js = \{\{\texttt{Entity}(a, b), \texttt{Attribute}(c_1, d_1), \texttt{Attribute}(c_2, d_2), \texttt{Value}(a, c_1, 0), \texttt{Value}(a, c_2, 1)\} \mid a, b, c_1, d_1, c_2, d_2 \text{ sind Variablen} \wedge a \neq c_1 \neq c_2\}$ eine Menge von Zielinstanzen J

Dann ist I_1 und I_2 in Bezug auf das Schema Mapping M die gleiche Menge Js von Zielinstanzen zugeordnet: Bis auf die abgebildeten Quellattributwerte 0 und 1, die in beiden Quellinstanzen vorkommen, sind die restlichen Zielattributwerte der Relationen

`Entity`, `Attribute` und `Value` beliebig wählbar (solange sie die Ungleichheitsbedingungen erfüllen). Somit kann zwischen den Zielinstanzmengen von I_1 und I_2 nicht unterschieden werden. Hierdurch ist die *unique solutions property* verletzt.

Fazit

Schema Mappings stellen ein theoretisch gut fundiertes Konzept dar und sind somit ein geeignetes Mittel zur Beschreibung von Abbildungen von lokalen Schemata in ein globales Schema. Unabhängig davon, ob eine materialisierte oder virtualisierte Integration angestrebt wird, ist das integrierte System in den meisten Fällen so ausgelegt, dass Anfragen an das globale Schema gestellt werden müssen. Sollen hingegen Anfragen an die lokalen Schemata gestellt werden, müssen Rückabbildungen definiert werden. Um einen Informationsverlust zu verhindern, müssen hierbei sämtliche Elemente der lokalen Schemata im globalen Schema aufgenommen werden. Dies führt dazu, dass das globale Schema sehr groß wird und die globalen Datensätze gleichzeitig viele Nullwerte aufweisen. Auch lässt sich die Ableitung von Inversen nicht automatisieren.

Die Modellierung des globalen Schemas im EAV-Modell bietet sich vor dem Hintergrund des großen Schemas und der vielen Nullwerte an. Die Abbildung zwischen lokalen Datenquellen und dem globalen EAV-Schema lässt sich mit Schema Mappings jedoch nur unzureichend beschreiben.

4.2. Erweitertes EAV-Modell

In diesem Unterkapitel soll das Modell vorgestellt werden, das zur Integration eingesetzt wird. Es handelt sich dabei um eine Erweiterung des EAV-Modells, wie es in 3.3 vorgestellt wurde. Eine Übersicht über die einzelnen Elemente und ihre Beziehungen zueinander ist in den Abbildungen 4.3 und 4.4 in Form eines EER-Diagramms[4] angegeben.

Im Folgenden werden die Elemente und Beziehungen des EAV-Modells beschrieben. Es soll im Relationenmodell realisiert werden, weshalb auch auf die Umsetzung in diesem Modell eingegangen wird. Dabei werden die Relationen und ihre Beziehung zueinander beschrieben. In A.1 (im Anhang) ist ein UML-Diagramm angegeben, das auf der Basis einer Datenbank entstanden ist, mit der das EAV-Modell umgesetzt wurde.

[4] Das EER-Modell erweitert das ER-Modell um Konzepte wie benutzerdefinierte Datentypen, mengenwertige und strukturierte Attribute sowie Spezialisierung, Generalisierung und Partitionierung, die mit Hilfe eines Typkonstruktors ausgedrückt werden. Eine Übersicht findet sich in [SSH10, S. 254ff.]

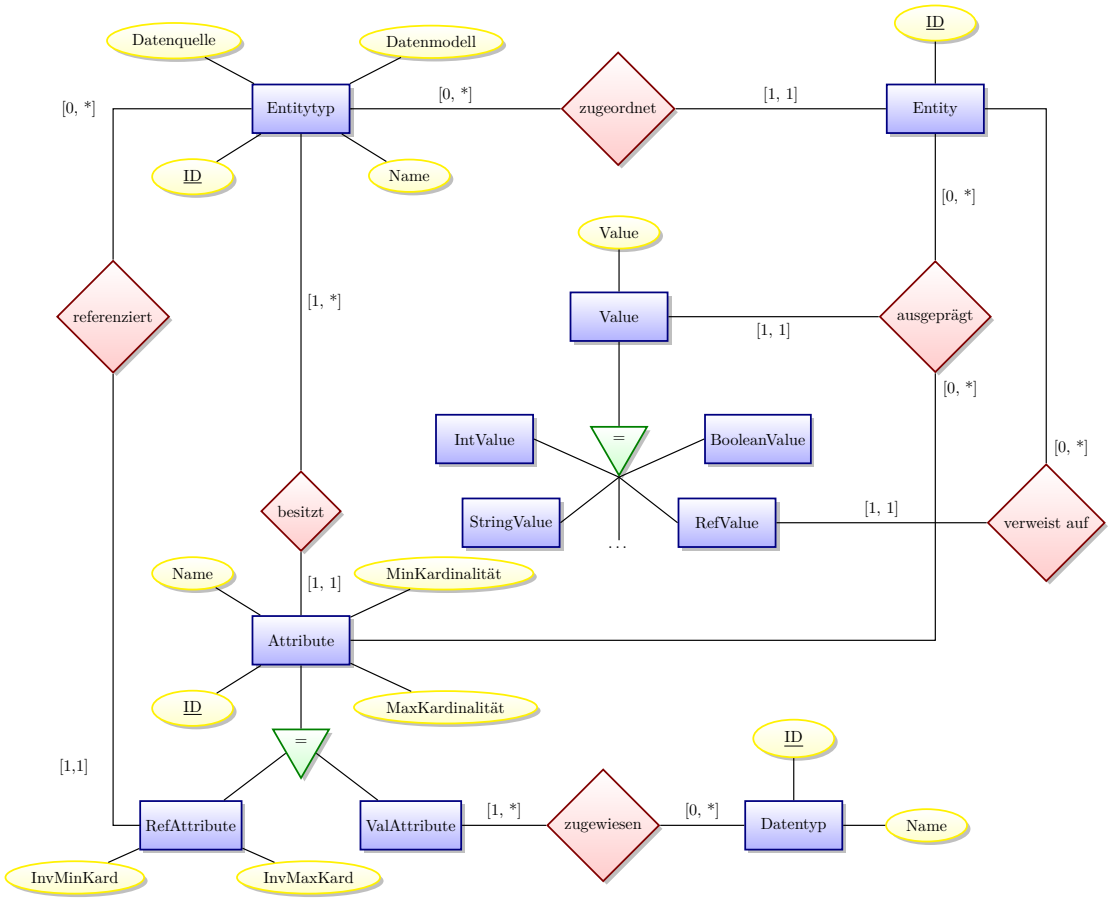

Abbildung 4.3.: EER-Diagramm für das erweiterte EAV-Modell

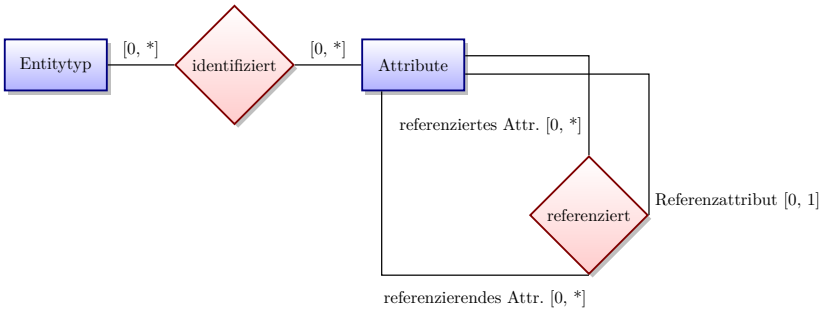

Abbildung 4.4.: Identifizierende Attribute und referenzierende Attribute

Entitytyp

Jeder Entity ist ein Entitytyp zugeordnet. In den meisten Datenmodellen ist eine Trennung von Typ und Werten eines Elements gegeben. Im Relationenmodell wird z.B.

zwischen dem Relationenschema und der dazugehörigen Relation unterschieden, im objektorientierten Modell zwischen Klasse und Objekt. Der Aufbau von Dokumenten in der Auszeichnungssprache XML lässt sich in einem externen Dokument beschreiben (hierauf wird später bei der Abbildung von XML auf EAV eingegangen). Es ist daher sinnvoll, diese Unterscheidung auch im EAV-Modell beizubehalten, insbesondere weil eine verlustlose Rückabbildung in die lokalen Datenmodelle möglich sein muss.

Im Schemaintegrationsprozess werden aus den Schemata der lokalen Quellen neue, integrierte Schemata erzeugt. Es ist auch aus diesem Grund sinnvoll, eine Trennung zwischen Entity und Entitytyp einzuführen, um für die neuen Schemata neue Entitytypen zu erzeugen, die unabhängig von den Daten der lokalen Quellen sind, die als Entitys abgebildet werden.

Das EAV/CR-Modell von Nadkarni et al., auf das in 3.3 eingegangen wurde, ist ein Beispiel für ein EAV-Modell, in dem eine Trennung zwischen Entitytyp und Entity vorhanden ist. Der Entitytyp wird dort als Klasse bezeichnet.

Die Entitytypen werden in einer Relation `EntityType` dargestellt. Jeder Entitytyp hat einen Namen, der sich aus dem Namen des abgebildeten Elements der lokalen Datenquelle ergibt. Die Identifizierung erfolgt über einen maschinell erzeugten Primärschlüssel, da zwischen den lokalen Schemata Überlappungen hinsichtlich der Namen auftreten können. Des Weiteren ist es notwendig, die Datenquelle zu vermerken, da diese Information im Integrationsprozess benötigt wird, um die zu einem Entitytyp aktuell vorhandenen Entitys aus der Datenquelle ermitteln zu können. Die Angabe darüber, in welchem Datenmodell das dem Entitytypen entsprechende Element im lokalen Schema modelliert ist, kann bei der Rückabbildung eingesetzt werden.

Die Relation `EntityType` hat dementsprechend die Attribute `ID` (das Primärschlüsselattribut), `Name`, `DataSource` und `DataModel`.

Entity

Eine Entity kann als Instanz eines Entitytyps aufgefasst werden. Zu einem Entitytyp kann es zu einem Zeitpunkt mehrere Entitys geben, die in Anlehnung an das objektorientierte Datenbankmodell als aktuelle **Entitymenge** des entsprechenden Entitytyps bezeichnet werden soll [Heu97, S. 299]. Jede Entity soll eindeutig identifizierbar sein, um die Identitäten der Elemente im lokalen Schema, die auf Entitys abgebildet werden, zu erhalten. Daher muss die Zuordnung zwischen lokalen und globalen Identitäten verwaltet werden. Diese ist notwendig, damit für ein in einem integrierten Schema erzeugtes Element eine Instanz aus den Daten der lokalen Quellen generiert werden kann.

Die Entitymengen aller Entitytypen, die definiert sind, werden in einer Relation `Entity` vorgehalten, die nur zwei Attribute enthält: Das Primärschlüsselattribut `ID`, das der Identifikation der Entities dient, und das Fremdschlüsselattribut `EntityTypeID`, das auf das Attribut `ID` der Relation `EntityType` verweist.

Attribute

Jeder Entitytyp verfügt über mindestens ein Attribut. Die Zugehörigkeit von Attributen zu Entitytypen wird hierbei explizit verwaltet und nicht nur implizit über die Zuordnung von Attributwerten zu Entities. Dies führt zu einer Redundanz in der Darstellung der Attribute, da gleiche Attribute unterschiedlicher Entitytypen mehrfach aufgeführt werden. Für die virtualisierte Integration ist dies jedoch von Vorteil, da die Integration erst zum Zeitpunkt der Anfrage erfolgt. Zu diesem Zeitpunkt sind keine Entities und damit auch keine Attributwerte zu den Entities vorhanden, aus denen die Zugehörigkeit von Attributen zu Entitytypen abgeleitet werden könnte. Auch werden trotz der expliziten Zuordnung von Attributen zu Entitytypen Nullwerte nicht im EAV-Modell gespeichert, da einer Entity kein Attributwert zugeordnet wird, wenn das entsprechende Attribut nicht auf die Entity zutrifft bzw. für die Instanz des lokalen Schemas, die auf die Entity abgebildet wird, ein Nullwert vermerkt ist.

Für jedes Attribut muss eine minimale und maximale Kardinalität angegeben werden, die festlegen, wie viele Werte einem Attribut minimal bzw. maximal zugeordnet werden. Als minimale Kardinalität kann hierbei 0 oder eine natürliche Zahl größer als 0 angegeben werden. Die maximale Kardinalität ist eine natürliche Zahl größer 0 oder der Wert n, womit eine beliebige Anzahl bezeichnet werden soll. Das erweiterte EAV-Modell unterstützt hierdurch mengenwertige Attribute. Atomare (nicht mengenwertige) Attribute haben eine minimale Kardinalität von 0 oder 1 und eine maximale Kardinalität von 1.

Es wird eine Unterscheidung zwischen wertbasierten Attributen und Referenzattributen getroffen. **Wertbasierte Attribute** haben einen primitiven Datentyp wie beispielsweise String, Integer oder Boolean. **Referenzattribute** verweisen auf andere Entitytypen, d.h. ihr Wert entspricht der Identität einer oder mehrerer Entities (mengenwertige Attribute sind zulässig) dieses Typs. Referenzattribute sind außerdem **bidirektional**, weshalb eine inverse minimale und maximale Kardinalität angegeben werden muss.

Im EER-Diagramm in Abbildung 4.3 sind die wertbasierten Attribute durch die EER-Entity *ValAttribute* und die Referenzattribute durch die EER-Entity *RefAttribute* repräsentiert. Beide zusammen stellen eine totale Partitionierung der EER-Entity *Attribute* dar[5]: Es gibt nur wertbasierte Attribute und Referenzattribute und diese beiden Mengen sind disjunkt.

Die allgemeinen Informationen über Attribute, die auf beide Attributkategorien zutreffen, werden in der Relation `Attribute` mit den Attributen `ID` (Primärschlüssel), `EntityTypeID`, `Name`, `MinCardinality` und `MaxCardinality` gespeichert. Die Identifikation über das Attribut `ID` ist für Zuordnung von Attributwerten zu Entities in den Value-Relationen von Bedeutung. Das Attribut `EntityTypeID` referenziert über eine Fremdschlüsselbeziehung das Attribut `ID` der Relation `EntityType` und drückt die Zugehörigkeit eines Attributs

[5] Eine totale Partitionierung wird in einem EER-Diagramm durch den mit einem Gleichheitszeichen beschrifteten Typkonstruktor dargestellt [SSH10, S. 261f.]. Der Typkonstruktur selbst wird durch ein Dreieck dargestellt.

zu einem Entitytypen aus.

Die Relation `RefAttribute` besitzt die Attribute `AttributeID` (Primärschlüssel und Fremdschlüssel in Bezug auf das Attribut `ID` der Relation `Attribute`), `RefEntityTypeID` (Fremdschlüssel in Bezug auf das Attribut `ID` der Relation `EntityType`) sowie `InvMinCardinality` und `InvMaxCardinality`, deren Werte jeweils die minimale und maximale inverse Kardinalität für das Referenzattribut darstellen. Das Attribut `RefEntityTypeID` legt fest, welcher Entitytyp durch das Referenzattribut referenziert wird.

Alle Datentypen werden in einer Relation `DataType` aufgelistet, jeweils mit einem Attribut `ID` als Primärschlüssel und einem weiteren Attribut `Name`. Ein wertbasiertes Attribut kann über einen Datentyp (z.B. `integer`) oder mehrere Datentypen (z.B. `string` und `integer`) verfügen. Jedem Datentyp ist in einer Ausprägung des Attributs ein Wert zugeordnet. In der Relation `ValAttrDataTypes` wird die Zuordnung von Attributen zu Datentypen verwaltet: Es sind zwei Attribute `AttributeID` und `DataTypeID` gegeben, die Fremdschlüssel in Bezug auf das Attribute `ID` jeweils der Relationen `Attribute` und `DataType` sind. Beide Attribute zusammen bilden den Primärschlüssel der Relation `ValAttrDataTypes`.

Die Angabe über die Datentypen ist notwendig, wenn z.B. alle Attributwerte eines wertbasierten Attributs ermittelt werden sollen, da vom Datentyp abhängt, welche Relationen hierfür angefragt werden müssen.

Die Entscheidung, wertbasierten Attributen mehr als nur einen Datentypen zuweisen zu können, wird durch die Abbildung von XML auf das EAV-Modell (siehe 4.3.2) und die Methode der zusicherungsbasierten Schemaintegration (siehe 4.4) motiviert, wo sich dies als notwendig erweist.

Sollte das Quelldatenmodell ein Schlüsselkonzept unterstützen, so können in der Relation `KeyAttributes(EntityTypeID, AttributeID)` für einen Entitytypen die dazugehörigen Schlüsselattribute vermerkt werden. Es wird dabei von nur einem (ausgezeichneten) Schlüssel pro Entitytyp ausgegangen, wie z.B. dem Primärschlüssel für Relationenschemata. Diese Relation ergibt sich aus der Beziehung *identifiziert* zwischen *Entitytyp* und *Attribute* im EER-Diagramm in Abbildung 4.4). In der Relation `RefAttributeMapping(AttributeID, ReferencingAttributeID, ReferencedAttributeID)` kann für ein Referenzattribut die Zuordnung zwischen referenzierenden und referenzierten Attributen gespeichert werden, sofern diese Information im Quelldatenmodell vorhanden ist. Dies ist beispielsweise für Fremdschlüssel im Relationenmodell der Fall, jedoch nicht für Objektreferenzen im objektorientierten Modell. Diese Relation ergibt sich aus der Beziehung *referenziert* im EER-Diagramm in Abbildung 4.4, an der *Attribute* in der Rolle als Referenzattribut, referenzierendes Attribut und referenziertes Attribut mehrfach beteiligt ist.

Value

Die Zuordnung von Entities zu Attributwerten erfolgt in separaten Relationen: Für

jeden primitiven Datentyp gibt es eine eigene Relation, in der die Werte wertbasierter Attribute aufgenommen werden. Die Werte der Referenzattribute werden in der Relation `RefValue` gespeichert. Im EER-Diagramm in Abbildung 4.3 sind die EER-Entities *Entity*, *Attribute* und *Value* durch die Beziehung *ausgeprägt* miteinander verknüpft[6]. Sowohl Entities als auch Attribute können dabei mehrfach an dieser Beziehung teilhaben: Für den Typ einer Entity können mehrere Attribute definiert sein, deren Werte verzeichnet werden. Auch können Attribute mengenwertig sein, sodass auch für ein einzelnes Attribut mehrere Werte definiert sein können. Beides führt dazu, dass eine Entity mehrfach an der Beziehung beteiligt ist. Auf der anderen Seite kann es zu einem Entitytypen mehrere aktuelle Entities geben, sodass ein einzelnes Attribut mehrfach beteiligt ist. Dies trifft ebenfalls auf mengenwertige Attribute zu.

Die EER-Entities *IntValue*, *StringValue*, *BooleanValue* und *RefValue* bilden eine totale Partitionierung der EER-Entity *Value*, d.h. sie sind disjunkt und enthalten in ihrer Vereinigung alle Zuordnungen von Attributwerten zu Entities. Wie im Diagramm in Abbildung 4.3 angedeutet, können für andere primitive Datentypen weitere EER-Entities definiert werden.

Die Relationen `RefValue`, `IntValue`, `StringValue`, `BooleanValue`, sowie potentielle weitere Relationen für andere primitive Datentypen, besitzen die Attribute `EntityID`, `AttributeID` und `Value`. Die Attribute `EntityID` und `AttributeID` sind dabei Fremdschlüssel in Bezug jeweils auf das Attribut `ID` der Relationen `Entity` und `Attribute`. Für die Relation `RefValue` gilt darüber hinaus, dass das Attribut `Value` einen Fremdschlüssel für das Attribut `ID` der Relation `Entity` darstellt: Der Wert eines Referenzattributes muss auf eine oder mehrere aktuell existierende Entities verweisen. Für die Relationen, mit denen die Werte wertbasierter Attribute gespeichert werden, entstammen die Werte des Attributs `Value` dem Wertebereich des primitiven Datentyps, der mit der Relation assoziiert ist. Für die Relation `IntValue` gilt beispielsweise, dass die Werte des Attributs dem Wertebereich des Datentyps *Integer* entnommen sind.

Die Attribute `EntityID`, `AttributeID` und `Value` bilden den Primärschlüssel einer Value-Relation: Durch die Mengenwertigkeit der Attribute können für eine Entity und ein Attribut mehrere Werte definiert sein, wobei jeder Wert aber nur einmal vorkommen darf.

4.3. Abbildungen in das EAV-Modell

In diesem Unterkapitel wird für zwei konkrete Datenmodelle die Abbildung in das EAV-Modell definiert. Dabei handelt es sich zum einen um das Relationenmodell und zum anderen um die Auszeichnungssprache XML. Diese beiden Datenmodelle wurden ausgewählt, da sie beide sehr weit verbreitet sind, jedoch meist mit einer unterschiedlichen

[6] Die EER-Entity *Value* ist eigentlich überflüssig, da sie durch die EER-Entities *Entity*, *Value*, die Beziehung *ausgeprägt* und die entsprechenden Kardinalitäten bereits impliziert wird (und in der Umsetzung im Relationenmodell dafür ein neues Relationenschema erzeugt wird). Sie ist nur aufgeführt, um den Zusammenhang zwischen der allgemeinen Value-Relation und den speziellen Value-Relationen aufzuzeigen.

Zielstellung genutzt werden. Das Relationenmodell bietet Vorteile für große, strukturierte Datenbestände, die effizient gespeichert und angefragt werden sollen, während XML auch die Darstellung semistrukturierter Daten ermöglicht [KM03, S. 1]. Die Integration semistrukturierter XML-Dokumente soll in dieser Arbeit allerdings nicht betrachtet werden: Dieses Thema wird in parallel ablaufenden Arbeiten behandelt (ohne Rückabbildung). Ein weiterer wichtiger Unterschied besteht darin, dass XML-Dokumente häufig dazu genutzt werden, um Informationen z.B. im Web auszutauschen. Dieser Austausch wird insbesondere durch die Möglichkeit der Vorgabe einer Schemabeschreibung gefördert. Diese Schemabeschreibung kann von einem XML-Dokument referenziert werden, wodurch eine standardisierte Darstellung ermöglicht wird. In vielen Anwendungsbereichen werden Standards auf XML-Basis definiert. Im Gesundheitswesen gibt es beispielsweise den Standard *Health Level 7* (HL7) [DAB+06]. Im Bibliothekswesen ist der Standard *Metadata Object Description Schema* verbreitet [McC04].

4.3.1. Relationenmodell

Die folgenden Definitionen zum Relationenmodel orientieren sich an [Heu13] und [SSH10].

Ein **Relationenschema** $R = \{A_1, \ldots, A_n\}$ ist eine Teilmenge eines Universums U, das eine nicht-leere, endliche Menge von **Attributen** $A \in U$ bezeichnet. Durch eine total definierte Funktion $dom\colon U \to D$ wird jedem Attribut A eine Domäne $d \in D$ zugeordnet. Die Menge $D = \{domain(T) \mid \tau \in T\}$ ist eine Menge von endlichen oder abzählbar unendlichen Mengen, wobei T eine Menge konkreter Datentypen ist wie z.B. Integer, String oder Boolean. $w \in dom(A)$ wird als **Attributwert** bezeichnet.

Eine **Relation** r über einem Relationenschema $R = (A_1, \ldots, A_n)$ $(n \in \mathbb{N}_0)$ ist eine endliche Menge von totaldefinierten Abbildungen, die **Tupel** genannt werden. Ein Tupel t ist dabei folgendermaßen definiert:

$$t\colon R \to \bigcup_{d \in D} d$$

Es gilt $t(A) \in dom(A)$. Eine Relation r über einem Relationenschema R wird auch mit $r(R)$ bezeichnet.

Ein **Datenbankschema** ist eine Menge von Relationenschemata $S = \{R_1, \ldots, R_m\}$ $(m \in \mathbb{N}_0)$. Eine **Datenbankinstanz** über einem Datenbankschema $S = \{R_1, \ldots, R_m\}$ ist dann eine Menge von Relationen $d = \{r_1, \ldots, r_m\}$ mit $r_i(R_i)$ für alle $i \in \{1, \ldots, m\}$, die auch als $d(S)$ abgekürzt wird.

Im Folgenden soll erläutert werden, wie ein Datenbankschema S und eine dazugehörige Datenbankinstanz $d(S)$ auf das EAV-Modell abgebildet werden. Sei $R \in S$ mit $R = \{A_1, \ldots, A_n\}$ ein Relationenschema. Dann wird R auf einen Entitytypen abgebildet, dessen Attribute den relationalen Attributen $A_1, \ldots, A_n$ entsprechen. Für die Kardinalität jedes Attributs A_i gilt dabei, dass die minimale Kardinalität entweder 0 oder 1 und

die maximale Kardinalität 1 beträgt. Relationenschemata in erster Normalform dürfen keine mengenwertigen Attribute enthalten [SSH10, S. 175f.], somit kann die maximale Kardinalität nicht größer als 1 sein. Aus verschiedenen Gründen kann es vorkommen, dass in einer Relation einem Attribut kein Wert zugeordnet werden kann. Es wird dann ein Nullwert mit dem Attribut assoziiert. In relationalen Datenbanksystemen kann durch Integritätsbedingungen für ein Relationenschema und ein Attribut festgelegt werden, dass dieses keinen Nullwert annehmen darf. In diesem Fall beträgt die minimale Kardinalität 1. Dies entspricht auch der Definition einer Relation als einer Menge von *totaldefinierten* Abbildungen (siehe oben).

Die Abbildung eines Datenbankschemas $S = \{R_1, \ldots, R_m\}$ mit $R_i = \{A_{i1}, \ldots, A_{in}\}$ (für $i \in \{1, \ldots, m\}$) auf das EAV-Modell wird in den Tabellen 4.1 und 4.2 veranschaulicht. Dabei wird von einer einzigen Datenquelle ausgegangen, einer fiktiven Instanz des relationalen Datenbankmanagementsystems MySQL. Der Ausdruck $f(dom(A))$ in der Tabelle 4.2 hat für ein Attribut A die ID des Datentyps zum Wert, dessen Wertebereich der Domäne $dom(A)$ entspricht und der in der Relation `DataType` (hier nicht dargestellt) definiert sein muss. Des Weiteren soll der Ausdruck $\{0|1\}$ in der Spalte `MinCard` der Relation `Attribute` bedeuten, dass die Kardinalität des relationalen Attributs im EAV-Modell 0 oder 1 betragen kann, je nachdem ob Nullwerte zugelassen sind oder nicht. Die maximale Kardinalität (in der Spalte `MaxCard` verzeichnet) beträgt hingegen immer 1.

EntityType			
`ID`	`Name`	`DataModel`	`DataSource`
α_1	R_1	relational	MySQL@192.168.10.1:3306
α_2	R_2	relational	MySQL@192.168.10.1:3306
⋮	⋮	⋮	⋮
α_m	R_m	relational	MySQL@192.168.10.1:3306

Tabelle 4.1.: Abbildung von Relationenschemata

Attribute				
ID	EntityTypeID	Name	MinCard	MaxCard
β_1	α_1	A_{11}	{0|1}	1
β_2	α_1	A_{12}	{0|1}	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
β_s	α_m	A_{mn}	{0|1}	1

ValAttrDataType	
AttributeID	DataTypeID
β_1	$f(dom(A_{11}))$
β_2	$f(dom(A_{12}))$
$\vdots$	$\vdots$
β_s	$f(dom(A_{mn}))$

Tabelle 4.2.: Abbildung von relationalen Attributen

Die Abbildung einer Datenbankinstanz $d(S) = \{r_1, \ldots, r_m\}$ zu einem Datenbankschema S soll aus Gründen der Übersichtlichkeit am Beispiel nur einer Relation, $r_1(R_1) \in d(S)$, veranschaulicht werden. Die Abbildung mehrerer Relationen wird an einem Beispiel geschildert, das in Unterkapitel A.2 des Anhangs angegeben ist. Es gelte $R_1 = \{A_1, \ldots, A_n\}$ und $r_1 = \{t_1, \ldots, t_p\}$. Des Weiteren sei $P \subseteq R_1$ der Primärschlüssel des Relationenschemas, d.h. eine ausgezeichnete minimale, identifizierende Attributmenge. Dann wird jedes Tupel $t_i \in r_1$ auf eine Entity abgebildet. Die Wertemenge der Primärschlüsselattribute $t_i(P)$ stellt eine lokale Identifizierbarkeit (innerhalb der Relation) sicher. Der Wertemenge $t_i(P)$ wird dann ein Wert γ_i zugeordnet, der das Tupel bzw. die Entity, auf die das Tupel abgebildet wird, global identifiziert, d.h. innerhalb der Föderation. Schmitt et al. untersuchen in [SS95] die Verwaltung von lokalen und globalen Identitäten sowie die Zuordnung zwischen beiden. Diese Zuordnung muss dynamisch sein, da z.B. Schlüsselattribute und ihre Werte im Relationenmodell nach außen sichtbar und veränderbar sind. Auf die spezifischen Herausforderungen, die von Schmitt et. al für diese Problematik identifiziert wurden (z.B. Migration und Restrukturierung von Objekten), soll hier nicht näher eingegangen werden.

Die Zuordnung von Tupeln zu Entities wird in Tabelle 4.3 veranschaulicht. Dabei gilt $P = \{A_1, \ldots, A_l\}$ für den Primärschlüssel P des Relationenschemas R_1. Darüber hinaus werden für den Entitytypen, der aus der Abbildung des Relationenschemas R_1 entsteht, die Primärschlüsselattribute vermerkt. Dies ist in Tabelle 4.4 dargestellt, wobei gilt, dass α_1 die ID des Entitytyps ist, auf den R_1 abgebildet wurde, und $\beta_1, \ldots, \beta_l$ die IDs der Attribute sind, auf die die Primärschlüsselattribute $A_1, \ldots, A_l$ abgebildet wurden. Diese Information wird für die Rückabbildung in das relationale Datenmodell benötigt (siehe 4.5.1).

Der Inhalt der EAV-Relation `Entity` für die abzubildende Relation r_1 ist in Tabelle 4.5 dargestellt. Diese enthält die globalen Identitäten der Entities, die zu der aktuellen

Entitymenge eines in `EntityType` definierten Entitytyps gehören (siehe Tabelle 4.1). Im relationalen Fall entspricht ein Entitytyp einem Relationenschema und die aktuelle Entitymenge den Tupeln einer Relation über diesem Schema.

Relation $r_1 \mapsto$ Entity-IDs			
A_1	...	A_l	`EntityID`
$t_1(A_1)$	...	$t_1(A_l)$	γ_1
$t_2(A_1)$	...	$t_2(A_l)$	γ_1
⋮	⋮	⋮	⋮
$t_p(A_1)$	...	$t_p(A_l)$	γ_p

Tabelle 4.3.: Zuordnung von Werten der Attribute des Primärschlüssels P zu globalen Identitäten der Entities ($P = \{A_1, \ldots, A_l\}$)

KeyAttributes	
`EntityTypeID`	`AttributeID`
α_1	β_1
α_1	β_2
⋮	⋮
α_1	β_l

Tabelle 4.4.: Erfassung der Attribute, aus denen sich der Primärschlüssel zusammensetzt

Entity	
`ID`	`EntityTypeID`
γ_1	α_1
γ_2	α_1
⋮	⋮
γ_p	α_1

Tabelle 4.5.: Abbildung von Tupel einer Relation als Entitymenge

Schließlich muss für jedes Attribut $A_i \in R_1$ und jedes Tupel $t_j \in r_1(R_1)$ der Attributwert $t_j(A_i)$ in der `Value`-Relation erfasst werden, die dem Datentyp des Attributs A_i entspricht. Der Datentyp kann über den Wert des Attributs `DataTypeID` der EAV-Relation `ValAttrDataType` ermittelt werden (siehe Tabelle 4.2), der auf einen Datentyp in der Relation `DataType` verweist. Hat ein Attribut A z.B. den Datentyp `integer`, so wird der Attributwert $t(A)$ in der EAV-Relation `IntValue` verzeichnet. Die Speicherung der Attributwerte für die Tupel einer Relation und Attribute des entsprechenden Relationenschemas wird in Tabelle 4.6 veranschaulicht. Um die Darstellung zu vereinfachen, werden Datentypen nicht berücksichtigt. Dies wird im Beispiel zur Abbildung des relationalen Modells nachgeholt, das in A.2 angegeben ist.

Value		
`EntityID`	`AttributeID`	`Value`
γ_1	β_1	$t_1(A_1)$
γ_1	β_2	$t_1(A_2)$
$\vdots$	$\vdots$	$\vdots$
γ_1	β_n	$t_1(A_n)$
$\vdots$	$\vdots$	$\vdots$
γ_p	β_1	$t_p(A_1)$
γ_p	β_2	$t_p(A_2)$
$\vdots$	$\vdots$	$\vdots$
γ_p	β_n	$t_p(A_n)$

Tabelle 4.6.: Abbildung von Tupelattributwerten

Fremdschlüsselbedingungen

Eine Fremdschlüsselbedingung für zwei Relationenschemata R_1 und R_2 hat die Form $X(R_1) \rightarrow Y(R_2)$, wobei $X \subseteq R_1$ und $Y \subseteq R_2$ gilt. Die Attributmenge X wird dann als Fremdschlüssel für R_1 bezüglich Y in R_2 bezeichnet.

Eine Datenbankinstanz $d(S)$ genügt einer Fremdschlüsselbedingung $X(R_1) \rightarrow Y(R_2)$ genau dann, wenn es in der Datenbankinstanz eine Relation $r_2(R_2)$ gibt, für die Y Primärschlüssel ist und die Bedingung

$$\{t(X) \mid t \in r_1\} \subseteq \{t(Y) \mid t \in r_2\}$$

erfüllt ist [SSH10, S. 93].

Eine Fremdschlüsselbeziehung $X(R_1) \rightarrow Y(R_2)$ zwischen zwei Relationenschemata R_1 und R_2 wird im erweiterten EAV-Modell durch **Referenzattribute** abgebildet. Dabei werden beide Richtungen der Fremdschlüsselbeziehung berücksichtigt. Seien α_1 und α_2 die IDs (Werte des Attributs `ID` der Relation `EntityType`) der Entitytypen, auf die R_1 und R_2 abgebildet werden. In Tabelle 4.7 ist dann das Referenzattribut dargestellt, das aus einer Fremdschlüsselbeziehung $X(R_1) \rightarrow Y(R_2)$ abgeleitet wird. Jedem Tupel der Relation $r_1(R_1)$ wird über die Fremdschlüsselbedingung genau ein Tupel aus der Relation $r_2(R_2)$ zugeordnet. Im EAV-Modell bedeutet dies, dass jeder Entity genau eine andere Entity zugeordnet wird. Es ergibt sich eine minimale und maximale Kardinalität von 1. Sind für die Attributmenge X jedoch Nullwerte zulässig, so beträgt die minimale Kardinalität 0. Durch den Ausdruck $\{0|1\}$ in der Spalte `MinCard` in Tabelle 4.7 soll daher verdeutlicht werden, dass beide Werte zulässig sind.

Für die Rückrichtung gilt, dass jedem Tupel aus $r_2(R_2)$ über die Fremdschlüsselbedingung kein oder beliebig viele Tupel aus $r_1(R_1)$ zugeordnet werden können. Somit können jeder Entity kein oder beliebig viele Entities des entsprechenden Typs zugeordnet werden. Dies wird durch die Werte 0 und n für die Spalten `InvMinCard` und `InvMaxCard` in Tabelle 4.7 zum Ausdruck gebracht.

Für die Rückabbildung sollte festgehalten werden, welche Fremdschlüsselbeziehungen auf welche Referenzattribute abgebildet wurde. Insbesondere ist dabei von Bedeutung, welche relationalen Attribute aus beiden Relationen an der Fremdschlüsselbedingung beteiligt sind. Ein Beispiel für eine Zuordnung von einer Fremdschlüsselbeziehung zu einem Referenzattribut bzw. zur ID des Referenzattributs ist in Tabelle 4.8 angegeben. Dabei wird für eine Fremdschlüsselbeziehung $X(R_1) \rightarrow Y(R_2)$, die auf ein Referenzattribut β_i abgebildet wurde, die Zuordnung zwischen referenzierendem und referenzierten EAV-Attribut festgehalten. Dabei gilt, dass $\alpha_s, \ldots, \alpha_t$ und $\alpha_u, \ldots, \alpha_v$ die IDs der Attribute sind, die aus der Abbildung der relationalen Attribute aus X bzw. Y entstanden sind. Diese Zuordnung wird bei der Rückabbildung in das Relationenmodell verwendet, die in 4.5.1 vorgestellt wird.

Attribute				
`ID`	`EntityTypeID`	`Name`	`MinCard`	`MaxCard`
β_i	α_1	R_2	$\{0\|1\}$	1

RefAttribute			
`AttributeID`	`RefEntityTypeID`	`InvMinCard`	`InvMaxCard`
β_i	α_2	0	n

Tabelle 4.7.: Abbildung einer Fremdschlüsselbeziehung als Referenzattribut

RefAttributeMapping		
`AttributeID`	`ReferencingAttributeID`	`ReferencedAttributeID`
β_i	α_s	α_u
...	...	...
β_i	α_t	α_v

Tabelle 4.8.: Zuordnung von referenzierenden und referenzierten Attributen für eine Fremdschlüsselbedingung

4.3.2. XML

Die Auszeichnungssprache XML (*Extensible Markup Language*) sieht Möglichkeiten zur Vorgabe von Schemata vor, die den Aufbau eines XML-Dokumentes beschreiben, obwohl XML-Dokumente auch ohne Angabe eines Schemas erstellt werden können. **Datenzentrierte** XML-Dokumente zeichnen sich dadurch aus, dass alle Dokumente gleich oder zumindest ähnlich strukturiert sind, d.h. dass sie regulär sind. Des Weiteren haben alle Angaben im Dokument einen Datentyp, die Dokumente sind also getypt.

Semistrukturierte und dokumentzentrierte Dokumente zeichnen sich hingegen durch Irregularität und eine fehlende oder nicht durchgängig konsistente Typisierung aus [HM13]. Da in dieser Arbeit ausschließlich datenzentrierte XML-Dokumente behandelt werden, wird davon ausgegangen, dass für alle zu integrierenden XML-Quellen ein Schema vorliegt.

DTDs (*Document Type Definitions*) und XML Schema sind zwei verbreitete Methoden für die Spezifikation eines Schemas für XML-Dokumente. DTDs sind Teil der Empfehlung des W3C[7] zu XML [W3Ca]. XML Schema basiert ebenfalls auf einer Empfehlung des W3C [W3Cb]. Es enthält „wesentlich umfangreichere Darstellungsmöglichkeiten“ als DTDs [KM03, S. 115]. Dazu gehören beispielsweise verschiedene primitive Datentypen, aus denen eigene Datentypen abgeleitet werden können. Zudem verfügt XML Schema im Gegensatz zu DTDs über eine XML-Syntax. Die Vorteile von XML Schema gegenüber DTDs werden z.B. in [SOA] beschrieben.

Aufgrund seiner höheren Ausdrucksstärke und der Möglichkeit der Typisierung soll die Abbildung von XML auf EAV anhand von XML Schema beschrieben werden. Die Ausführungen zu XML Schema orientieren sich hierbei an [KM03, S. 115ff.]. Die Beschreibung der Abbildung von XML auf EAV erfolgt dabei weniger formal als beim Relationenmodell und basiert auf einem Beispiel, das wesentliche Konzepte von XML Schema aufzeigen soll. Der Grund hierfür ist, dass die Spezifikation von XML Schema, wie sie in offizieller Form in [W3Cb] angegeben ist, selbst vorwiegend auf einer verbalen Beschreibung aufbaut.

XML Schema

Einfache Typen können aus einer Menge von vordefinierten Datentypen wie z.B. `string`, `integer`, `float`, `boolean` ausgewählt werden. Darüber hinaus können auf Grundlage der vordefinierten Datentypen benutzerdefinierte Typen eingeführt werden. Dies wird durch die Angabe von *Facets* erreicht, die eine Einschränkung eines Basisdatentypen beschreiben. Für den Datentyp `integer` kann beispielsweise ein minimaler und/oder maximaler Wert vorgeschrieben werden. Der Datentyp `string` kann durch eine maximale Längenangabe oder einen regulären Ausdruck eingeschränkt werden.

Die Einschränkungen stellen eine Form lokaler Integritätsbedingungen dar. Diese werden im erweiterten EAV-Modell jedoch nicht abgebildet, sondern bei der Definition von Attributkorrespondenzen im Zuge der zusicherungsbasierten Schemaintegration implizit berücksichtigt (siehe Unterkapitel 4.4). In Listing 4.1 sind Beispiele für Elementdeklarationen angegeben, die sich auf vordefinierte Basisdatentypen beziehen. In Listing 4.2 ist eine Elementdeklaration angegeben, die auf einem benutzerdefinierten Typ basiert: Der Typ des Elements `Gattung` ergibt sich aus einer Aufzählung von String-Werten, stellt also eine Einschränkung des Datentyps `string` auf eine Auswahl von Werten dar.

[7] Das *Word Wide Web Consortium* ist eine Organisation, die sich mit der Standardisierung von Technologien im Web-Umfeld beschäftigt (siehe `http://www.w3.org/`).

```
1 <xs:element name="Urteil" type="xs:string"/>
2 <xs:element name="Schlagwort" type="xs:string"/>
```

Listing 4.1: Elemente mit vordefiniertem Basisdatentyp

```
1 <xs:element name="Gattung">
2   <xs:simpleType>
3     <xs:restriction base="xs:string">
4       <xs:enumeration value="Roman"/>
5       <xs:enumeration value="Erzählung"/>
6       <xs:enumeration value="Novelle"/>
7       <xs:enumeration value="Gedichtband"/>
8       <xs:enumeration value="Fachbuch"/>
9     </xs:restriction>
10  </xs:simpleType>
11 </xs:element
```

Listing 4.2: Element mit benutzerdefiniertem Basisdatentyp

Mit **komplexen Typen** wird die Zuordnung von Attributen zu Elementen und die Schachtelung von Elementen beschrieben. Ein Beispiel hierfür ist in Listing A.1 (im Anhang) dargestellt. Das deklarierte Element `Buch` basiert auf einem komplexen Datentypen, der sich aus einer Sequenz anderer Elemente zusammensetzt. Eine **Sequenz**, eingeleitet und abgeschlossen durch die Tags `<xs:sequence>` und `</xs:sequence>`, legt fest, dass die Subelemente in einer festgelegten Reihenfolge im Dokument auftauchen müssen. Die minimale und maximale **Vorkommenshäufigkeit** wird dabei jeweils durch die Attribute `minOccurs` und `maxOccurs` spezifiziert, die bei der Deklaration der Unterelemente angegeben werden müssen. Für das Attribut `maxOccurs` kann hierbei auch der Wert `unbounded` angegeben werden, was bedeutet, dass es keine Einschränkung des maximalen Vorkommens gibt. Fehlen diese beiden Attribute, so wird davon ausgegangen, dass jedes Subelement genau einmal vorkommen darf, d.h. es gilt `minOccurs=maxOccurs="1"`.

Neben einer Sequenz kann auch eine **Auswahl** von Unterelementen angegeben werden (durch die Tags `<xs:choice>` und `</xs:choice>`), von denen im XML-Dokument eins vorkommen darf. Dies führt jedoch zu einer irregulären Struktur von XML-Dokumenten, weswegen diese Modellierungsmöglichkeit nicht berücksichtigt werden soll. Durch die Tags `<xs:all>` und `</xs:all>` wird ausgedrückt, dass alle Unterelemente maximal einmal in einer beliebigen Reihenfolge auftauchen dürfen.

Ein Beispiel für **Attributdeklarationen** sind die Deklarationen der Attribute `ISBN`, `Titel`, `Autor` und `Verlag` für das Element `Buch` in Listing A.1. Dabei werden ein Name und ein Typ (vorgegeben oder benutzerdefiniert) festgelegt sowie angegeben, ob der Wert optional ist (`use="optional"`) oder nicht (`use="required"`). Des Weiteren kann mit `value` ein Defaultwert angegeben werden, der angenommen wird, wenn das Attribut weggelassen wird. In Kombination mit `use="fixed"` kann damit auch ein fester Wert für das Attribut angegeben werden.

Auf deklarierte Elemente kann durch das XML-Attribute `ref` Bezug genommen werden, wovon in den Beispielen in Listing A.1 Gebrauch gemacht wird.

Nicht berücksichtigt werden bei der Abbildung Elemente, die leer sein können (was in XML Schema durch die Angabe von `xs:nillable` ausgedrückt wird), da dies eine irreguläre Struktur von XML-Dokumenten zur Folge haben kann: Elemente des gleichen Namens können im selben Dokument leer sein oder nicht. Auch werden Listentypen nicht betrachtet, die als Attribut- oder Elementinhalt eine Liste von Werten eines elementaren Datentyps zulassen.

Hingegen werden Union-Typen (`xs:union`) berücksichtigt, die eine Vereinigung der Wertebereiche verschiedener Datentypen wie z.B. `string` und `integer` ermöglichen. Im EAV-Modell ist es ebenfalls möglich, einem Attribut mehrere Datentypen zuzuweisen, sodass Attribute und Elemente, die auf einem Union-Typen basieren, verlustfrei in das EAV-Modell überführt werden können. Dies soll an dieser Stelle jedoch nicht weiter ausgeführt werden. Stattdessen wird bei der Betrachtung der Rückabbildung aus EAV in XML in 4.5.2 darauf eingegangen.

In den folgenden zwei Abschnitten wird die Abbildung von XML auf EAV anhand der Abbildung der Komponenten eines XML Schemas beschrieben. Dabei wird zunächst auf Entitytypen und ihre Attribute sowie anschließend auf Entities und Attributwerte eingegangen. Im zweiten Abschnitt werden auch die inversen Kardinalitäten von Referenzattributen diskutiert.

Abbildung von XML auf EAV - Entitytypen und Attribute

Ein deklariertes Element mit komplexem Typ wird auf einen Entitytypen abgebildet. In Tabelle A.10 (im Anhang) ist diese Abbildung für die in Listing A.1 deklarierten Elemente dargestellt. Es wird davon ausgegangen, dass die lokale Quelle eine Instanz des XML-Datenbankmanagementsystems BaseX[8] darstellt.

XML-Attribute werden auf wertbasierte Attribute abgebildet. Die minimale Kardinalität wird durch den Wert von `use` bestimmt. Wenn `use="required"` oder `use="fixed"` gilt, so hat die minimale Kardinalität den Wert 1, andernfalls (`use="optional"`) den Wert 0. Da ein XML-Attribut pro Element nur einmal vorkommen darf, weist die maximale Kardinalität den Wert 1 auf. In Tabelle A.7 (im Anhang) ist diese Abbildung für die XML-Attribute `ISBN`, `Titel`, `Autor`, `Verlag`, `Anschaffungsdatum`, `Lesernummer`, `Name`, `Öffnungszeiten`, `Benachrichtigungsintervall` und `aktiviert` der Elemente `Buch`, `Leser`, `Standort` und `Interessenprofil` angegeben.

Alle Elemente, die innerhalb eines anderen Elements geschachtelt sind und auf einem einfachen Typ basieren, werden ebenfalls als wertbasierte Attribute des übergeordneten Elements abgebildet. Im Beispiel trifft das auf das Element `Urteil` zu, das dem Element `Buch` untergeordnet ist, und auf die Elemente `Schlagwort` und `Gattung`, die dem Ele-

[8] `http://basex.org/`

ment `Interessenprofil` untergeordnet sind. Die Kardinalitäten ergeben sich aus den Werten der XML-Schema-Attribute `minOccurs` und `maxOccurs`, die festlegen, wie oft ein Unterelement jeweils minimal und maximal vorkommen darf. Fehlen diese Angaben, wie es für das Element `Urteil` der Fall ist, so wird von einem Wert von 1 ausgegangen, sodass sich für das wertbasierte Attribut `Urteil` in der EAV-Darstellung die Kardinalität 1:1 ergibt. Für die wertbasierten Attribute `Schlagwort` und `Gattung` ergibt sich aus den Werten von `minOccurs` und `maxOccurs` die Kardinalität 1:n (siehe Tabelle A.7).

Die Schachtelung von Elementen, die einen komplexen Typ aufweisen, wird durch Referenzattribute erfasst. Dies trifft im Beispiel aus Listing A.1 auf das Element `Standort` zu, das dem Element `Buch` untergeordnet ist, auf das Unterelement `Interessenprofil` des Elements `Leser` sowie auf die Elemente `Buch`, `Leser`, `Standort` und `Interessenprofil`, die innerhalb des Elements `Bibliothek` geschachtelt sind. Die minimale und maximale Kardinalität des Referenzattributs ergibt sich aus den Werten von `minOccurs` und `maxOccurs` (siehe oben).

Die inverse minimale Kardinalität für die Referenzattribute β_6 und β_9, mit denen die Schachtelung der Elemente `Standort` und `Interessenprofil` unterhalb jeweils des Elements `Buch` bzw. `Leser` abgebildet wird (siehe Tabelle A.7), hat den Wert 0, während die inverse maximale Kardinalität unbeschränkt ist. Dies liegt daran, dass Elemente in einem XML-Dokument zu einer Entity zusammengefasst werden, wenn Folgendes gilt: Sie werden auf Entities desselben Typs abgebildet, weisen die gleichen Werte für alle wertbasierten Attribute auf und referenzieren die gleichen Entities über Referenzattribute. In der Objektorientierung entspricht dies der **flachen Gleichheit** (shallow equality) [Heu97, S. 185]: Zwei Objekte sind oberflächlich gleich, wenn sie dieselben Werte aufweisen und dieselben Objekte referenzieren. Würde man hingegen jedes in einem XML-Dokument vorkommende Element als einzelne Entity auffassen, so ergäbe sich eine Kardinalität von 1:1, da die Zuordnung zum übergeordneten Element eindeutig wäre. Durch die Zusammenlegung von oberflächlich gleichen Entities kann hingegen ein Element mehreren Elternelementen zugeordnet sein, sodass sich eine unbeschränkte maximale Kardinalität ergibt.

Die Elemente `Standort` und `Interessenprofil` dürfen im XML-Dokument sowohl unterhalb der Elemente `Buch` und `Leser` vorkommen, als auch eine Hierarchieebene höher direkt unter dem Wurzelement `Bibliothek`. Aus diesem Grund hat die inverse minimale Kardinalität für die Referenzattribute, mit denen die Schachtelung von `Standort` und `Interessenprofil` unterhalb von `Buch` bzw. `Leser` abgebildet wird, den Wert 0.

Für die Referenzattribute β_{16} und β_{17}, mit denen die Schachtelung der XML-Elemente `Buch` und `Leser` unterhalb des Elements `Bibliothek` abgebildet wird, gilt eine Kardinalität von 1:1. Da `Bibliothek` das Wurzelelement ist, kann es nur eine Entity diesen Typs geben. Somit sind Entities der Typen `Buch` und `Leser` eindeutig dieser Entity zugeordnet. Für die Referenzattribute β_{18} und β_{19}, mit denen die Schachtelung der Elemente `Standort` und `Interessenprofil` unterhalb des Wurzelements abgebildet wird, ergibt sich eine minimale Kardinalität von 0, da in einem XML-Dokument Elemente diesen Namens auch als Unterelemente der Elemente `Buch` und `Leser` vorkommen können und somit

nicht zwingend unterhalb des Wurzelements geschachtelt sein müssen. Die maximale Kardinalität beträgt ebenfalls 1.

Um bei der Rückabbildung die durch `xs:sequence` festgelegte Reihenfolge der Subelemente wiederherstellen zu können, sollte zusätzlich festgehalten werden, welche Position ein Element innerhalb des übergeordneten Elements einnimmt. Die Speicherung der Position kann dabei in Bezug zum Referenzattribut erfolgen, mit dem die Schachtelung des Elements abgebildet wird. In Tabelle 4.9 ist dies für das Beispiel aus Listing A.1 dargestellt.

AttributeID	Position innerhalb des Elternelements
β_6	1
β_9	1
β_{16}	1
β_{17}	2
β_{18}	3
β_{19}	4

Tabelle 4.9.: Speicherung der Position, die ein Unterelement innerhalb des übergeordneten Elements einnimmt

Abbildung von XML auf EAV - Entities und Attributwerte

Die Abbildung auf Entities wird exemplarisch anhand des XML-Dokuments aus Listing 4.3 veranschaulicht.

Jedes in einem XML-Dokument vorkommende Element wird auf eine neue Entity abgebildet, deren Typ durch die entsprechende Elementdeklaration im XML Schema festgelegt ist. In XML Schema wird die Möglichkeit vorgesehen, Eindeutigkeitsbedingungen sowie Schlüssel und Referenzen auf Schlüssel als Integritätsbedingung festzulegen [KM03, S. 122f.]. Diese Integritätsbedingungen sind jedoch nicht zwingend erforderlich. Im Gegensatz dazu sind Schlüssel im Relationenmodell ein zentrales Modellierungskonzept. So muss z.B. nach der in [Dat03] vorgestellten Definition der ersten Normalform eine Relation einen eindeutigen Schlüssel aufweisen, damit die erste Normalform erfüllt ist. Aus diesem Grund wird die Identität eines Elements durch die Werte von wertbasierten Attributen und Referenzattributen in der Darstellung als Entity bestimmt. Wie bereits weiter oben beschrieben, werden dabei Entities desselben Entitytyps, die gleiche Werte für wertbasierte Attribute aufweisen und dieselben Entities referenzieren, zu einer Entity zusammengefasst. In Tabelle A.8 (im Anhang) wird die Zuordnung von Attributwerten und Referenzen zu Entity-IDs für das Beispieldokument aus Listing 4.3 angegeben. Dabei werden die beiden `Standort`-Elemente in Zeile 17 und 21 zu einer Entity zusammengefasst, da sie die gleichen Werte für die Attribute `Name` und `Öffnungszeiten` besitzen. Im Unterschied dazu werden die `Buch`-Elemente in Zeile 15 und 19 auf zwei unterschiedliche Entities abgebildet: Zwar referenzieren sie dieselbe `Standort`-Entity, doch sind die Werte

aller wertbasierten Attribute unterschiedlich.

In Tabelle A.9 (im Anhang) sind alle Entities, die sich aus der Abbildung des XML-Dokuments aus Listing 4.3 ergeben, mit den zugehörigen Entitytypen aufgeführt, die sich wiederum aus der Abbildung des XML Schemas ergibt, das in den Listings 4.1, 4.2 und A.1 angegeben ist. In Tabelle A.11 (im Anhang) ist die Zuordnung von Werten für wertbasierte Attribute und Entity-IDs für Referenzattribute zu Entities aufgeführt.

```
<Bibliothek>
  <Leser Lesernummer="1">
    <Interessenprofil Benachrichtigungsintervall="10"
        aktiviert="true">
      <Schlagwort>Programmierung</Schlagwort>
      <Schlagwort>Java</Schlagwort>
      <Gattung>Fachbuch</Gattung>
    </Interessenprofil>
  </Leser>
  <Leser Lesernummer="2">
    <Interessenprofil Benachrichtigungsintervall="20"
        aktiviert="false">
      <Schlagwort>Computernetzwerke</Schlagwort>
      <Gattung>Fachbuch</Gattung>
    </Interessenprofil>
  </Leser>
  <Buch ISBN="978-3-8362-1506-0" Titel="Java ist auch eine Insel"
      Autor="Christian Ullenboom" Verlag="Galileo Press">
    <Urteil>umfangreiche Einführung</Urteil>
    <Standort Name="Südstadtbibliothek" Öffnungszeiten="Mo.-Fr.,
        08:00-20:00" />
  </Buch>
  <Buch ISBN="3-86894-137-1" Titel="Computernetzwerke" Autor="Andrew
      Tanenbaum" Anschaffungsdatum="15.08.2012">
    <Urteil>verständliches Grundlagenwerk</Urteil>
    <Standort Name="Südstadtbibliothek" Öffnungszeiten="Mo.-Fr.,
        08:00-20:00" />
  </Buch>
</Bibliothek>
```

Listing 4.3: Beispiel für ein XML-Dokument

4.4. Zusicherungsbasierte Schemaintegration

In diesem Unterkapitel wird beschrieben, wie die Elemente der lokalen Quellen nach der Translation ins EAV-Modell integriert werden. Der hier vorgestellte Ansatz orientiert sich an der Methode der zusicherungsbasierten Schemaintegration, die von Spaccapietra et al. in [SPD92] und [SP94] vorgestellt wurde und in [Con97, S. 85ff.] ausführlich erläutert wird. Diese Methode wurde als Grundlage ausgewählt, da sie nicht auf ein spezifisches Datenmodell zugeschnitten ist und darüber hinaus nicht nur die Integration von Objekttypen (Entitytypen im EAV-Modell), sondern auch von Beziehungen zwischen den Objekttypen (im EAV-Modell durch Referenzattribute ausgedrückt) ermöglicht.

Im Folgenden werden die Begrifflichkeiten, die in [SPD92] und [SP94] verwendet werden, mit Bezug auf die korrespondierenden Komponenten im EAV-Modell definiert. Darauf aufbauend wird erläutert, welche Zusicherungen zwischen Elementen und Beziehungen ausgedrückt werden können. Im Anschluss daran werden die Integrationsregeln vorgestellt, anhand derer die Schemaintegration durchgeführt wird.

Im Integrationsprozess in dieser Arbeit wird von einer lokalen Datenquelle bzw. einem lokalen Schema ausgegangen. Für bestimmte Elemente des lokalen Schemas können Korrespondenzen und darauf basierend Zusicherungen zu Elementen anderer lokaler Schemata festgelegt werden. Anhand dieser Angaben wird dann eine Schemaintegration durchgeführt. Der Anwender, der mit der lokalen Quelle arbeitet, kann dann die lokalen Schemaelemente anfragen, die unverändert bleiben, als auch die durch die Integration entstandenen neuen Elemente, die als Erweiterung der lokalen Schemaelemente angesehen werden können. Bei der Integration eines lokalen Schemaelements mit einem Element aus einem anderen lokalen Schema entsteht im integrierten Schema ein integriertes Element. Dieses kann wiederum mit einem weiteren Element aus einem dritten Schema integriert wird. Dieser Prozess kann solange fortgesetzt werden, bis alle Elemente aus anderen Schemata, die zu einem Element im lokalen Schema äquivalent sind, integriert wurden. Dies entspricht einer binären Integrationsstrategie (siehe [Con97, S. 75]).

Bei der Integration von Entitytypen werden nur wertbasierte Attribute betrachtet. Referenzattribute werden getrennt behandelt. Die Grundidee besteht darin, die Verkettung von Entitytypen über Referenzattribute in den zu integrierenden Schemata zu vergleichen und zu integrieren. Dafür werden *Links* und *Pfade* definiert.

Link

Ein *Link* $E_1 \leftrightarrow^r E_2$ ist eine bidirektionale Beziehung zwischen zwei Entitytypen, die einem Referenzattribut r eines Entitytypen E_1 entspricht, das auf den Entitytypen E_2 verweist. Dabei werden dem Link die Kardinalitäten des Referenzattributes r in beide Richtungen zugeordnet, d.h. ein Link hat eine minimale und maximale Kardinalität sowie eine **inverse** minimale und maximale Kardinalität. Wenn es nur ein Referenzattribut zwischen E_1 und E_2 gibt, so kann auch einfach nur $E_1 \leftrightarrow E_2$ geschrieben werden. Der Übersicht wegen wird im Folgenden nur $E_1 \leftrightarrow E_2$ geschrieben, da der Name des Referenzattributs bei der Integration keine Rolle spielt.

Pfad

Seien $E_1, \ldots, E_n$ Entitytypen, sodass $\forall i \in \{1, \ldots, n-1\}$ der Entitytyp E_i mit dem Entitytypen E_{i+1} durch einen Link $E_i \leftrightarrow E_{i+1}$ verbunden ist. Dann wird $E_1 \leftrightarrow E_2 \leftrightarrow \cdots \leftrightarrow E_{n-1} \leftrightarrow E_n$ als *Pfad* bezeichnet. Die Kardinalitäten für E_1 und E_n werden folgendermaßen bestimmt [SPD92, S. 98]

- Die minimale Kardinalität von E_1 ergibt sich aus dem Produkt der minimalen

Kardinalitäten aller Links $E_i \leftrightarrow E_{i+1}$ mit $i \in \{1, \ldots, n-1\}$

- Die maximale Kardinalität von E_1 ergibt sich aus dem Produkt der maximalen Kardinalitäten aller Links $E_i \leftrightarrow E_{i+1}$ mit $i \in \{1, \ldots, n-1\}$
- Die minimale Kardinalität von E_n ergibt sich aus dem Produkt der **inversen** minimalen Kardinalitäten aller Links $E_i \leftrightarrow E_{i+1}$ mit $i \in \{1, \ldots, n-1\}$
- Die maximale Kardinalität von E_n ergibt sich aus dem Produkt der **inversen** maximalen Kardinalitäten aller Links $E_i \leftrightarrow E_{i+1}$ mit $i \in \{1, \ldots, n-1\}$

Dabei gilt, dass jede Multiplikation mit n ebenfalls n, d.h. eine unbeschränkte Kardinalität, zum Ergebnis hat.

Zusicherungen für Entitytyp-Entitytyp- und Entitytyp-Attribut-Korrespondenzen

In diesem Abschnitt werden zum einen Zusicherungen für Korrespondenzen zwischen Entitytypen betrachtet, für die zusätzlich Korrespondenzen zwischen Attributen festlegt werden können[9]. Zum anderen können Korrespondenzen auch zwischen Entitytypen und wertbasierten Attributen aufgestellt werden.

Zwischen zwei Entitytypen oder einem Entitytyp und einem Attribut können vier Arten von Zusicherungen formuliert werden. Diese entsprechen den Extensionsbeziehungen zwischen einem globalen und einem lokalen Schema, die in Unterkapitel 2.3.1 im Zusammenhang mit der Anfrageplanung vorgestellt wurden:

- **Äquivalenz**: $X_1 \equiv X_2$

  Damit wird ausgedrückt, dass X_1 und X_2 in jedem Zustand dieselbe Menge von Objekten der realen Wert repräsentieren.

- **Inklusion**: $X_1 \supseteq X_2$

  Die Menge der durch X_1 repräsentierten Objekte stellt eine Obermenge der Menge von Objekten dar, die durch X_2 repräsentiert werden.

- **Überlappung**: $X_1 \cap X_2$

  Es kann eine nicht-leere Schnittmenge zwischen den Mengen von Objekten geben, die durch X_1 und X_2 repräsentiert werden, die jedoch nicht in jedem Zustand gegeben sein muss.

[9] In der in [SPD92] und [SP94] vorgestellten Originalversion der Methode der Schemaintegration werden auch komplexe Attribute berücksichtigt, sodass die zusätzlichen Attributkorrespondenzen auch für Zusicherungen von Korrespondenzen zwischen einem Entitytyp und einem komplexen Attribut eingeführt werden können. In dem in dieser Arbeit verwendeten EAV-Modell sind komplexe Attribute nicht vorgesehen, sodass diese auch bei der Schemaintegration nicht behandelt werden.

- **Semantischer Zusammenhang bei Disjunktheit**: $X_1 \sim X_2$[10]

  Die Schnittmenge zwischen den Objektmengen von X_1 und X_2 ist in jedem Zustand leer. Es gibt jedoch einen semantischen Zusammenhang zwischen X_1 und X_2, der nahelegt, im integrieren Schema ein Element (Entitytyp oder wertbasiertes Attribut) X einzuführen, das X_1 und X_2 in sich vereint.

Wenn eine Zusicherung für eine Korrespondenz zwischen zwei Entitytypen E_1 und E_2 aufgestellt wird, so müssen auch Zusicherungen für die Korrespondenzen zwischen den Attributen der beiden Typen angegeben werden. Daraus werden die Attribute des Entitytypen E bestimmt, der sich aus der Integration von E_1 und E_2 ergibt. Es können dieselben Arten von Zusicherungen wie oben formuliert werden, wobei diese hier in Bezug auf die möglichen Beziehungen zwischen den Attributen A_1 und A_2 interpretiert werden:

- **Gleichheit**: $A_1 = A_2$

  Sei ein Objekt der realen Welt gegeben, das durch die Entities e_1 und e_2 jeweils des Typs E_1 bzw. E_2 modelliert wird. Dann gilt, dass die Attribute A_1 und A_2 für e_1 und e_2 die gleichen Werte aufweisen.

- **Inklusion**: $A_1 \supseteq A_2$

  Sind A_1 und A_2 mengenwertige Attribute, so ist die Wertemenge von A_1 stets eine Obermenge der Wertemenge von A_2. Wenn A_1 und A_2 keine mengenwertigen Attribute sind, so sind die Werte der beiden Attribute entweder gleich oder für A_2 ist ein Nullwert gegeben.

- **Überlappung**: $A_1 \cap A_2$

  A_1 und A_2 sind mengenwertige Attribute, deren Wertemengen sich schneiden können.

- **Semantischer Zusammenhang bei ungleichen Werten**: $A_1 \sim A_2$[11]

  Die Werte der beiden Attribute sind nie gleich bzw. die Schnittmenge zwischen ihren Wertenmengen ist immer leer (bei mengenwertigen Attributen), jedoch besteht ein Zusammenhang zwischen ihnen.

Die Zusicherungen für Attributkorrespondenzen werden im Zusammenhang mit der Zusicherung für Korrespondenzen zwischen Entitytypen notiert, z.B. als „$E_1 \equiv E_2$ with corresponding attributes: $A_{11} \supseteq A_{21}, A_{12} = A_{22}, A_{13} \cap A_{23}$“. Die Zusicherungen für Attributkorrespondenzen legen die Beschreibungskonflikte fest [SPD92, S. 101f.].

[10] In der ursprünglichen Vorstellung der zusicherungsbasieren Schemaintegration in [SPD92] wird an Stelle von $X_1 \sim X_2$ die Notation $X_1 \neq X_2$ verwendet. Diese ist etwas missverständlich, da sie den Eindruck erweckt, dass *gar kein* Zusammenhang zwischen X_1 und X_2 besteht.

[11] Auch hier wird die Notation $A_1 \sim A_2$ an Stelle der in [SPD92] ursprünglich eingeführten Notation $A_1 \neq A_2$ verwendet (siehe Fußnote 10).

Über die Zusicherungen hinaus kann für jede Attributkorrespondenz eine Funktion angegeben werden, die z.B. eine Übersetzung von Werten aus einer Domäne in die andere spezifiziert oder festlegt, wie aus den Werten eines mengenwertigen Attributs ein Wert für ein korrespondierendes Attribut berechnet wird, dem nur ein einzelner Wert zugewiesen werden kann.

Zusicherungen für Pfadkorrespondenzen

Analog zu den Zusicherungen zwischen zwei Entitytypen bzw. zwischen einem Entitytyp und einem Attribut, können auch für korrespondierende Pfade Zusicherungen angegeben werden, die eine Äquivalenz, Inklusion, Überlappung oder einen semantischen Zusammenhang bei Disjunktheit ausdrücken können. Jeder Pfad $E_1 \leftrightarrow \cdots \leftrightarrow E_n$ repräsentiert in einem gegebenen Zustand eine Menge von Paaren von Entities (e_1, e_2), deren Typ E_1 und E_n entspricht. Eine Zusicherung für eine Korrespondenz zwischen zwei Pfaden beschreibt demnach in welcher Beziehung die Mengen von Paaren stehen, die durch die Pfade repräsentiert werden. Da die Bedeutung der vier Korrespondenzarten bereits weiter oben erläutert wurde, wird an dieser Stelle darauf verzichtet.

Grundlage der Schemaintegration

In diesem Abschnitt wird beschrieben, wie zwei Entitytypen E_1 und E_2 zu einem neuen Entitytypen E integriert werden. Da die Behandlung von Inklusions-, Überlappungs- und Disjunktheits-Zusicherungen „noch nicht näher untersucht worden“ ist [Con97, S. 99] und um den Rahmen dieser Arbeit nicht zu sprengen, werden hier und bei der Vorstellung der Integrationsregeln in den folgenden Abschnitten nur Äquivalenzzusicherungen betrachtet.

Seien E_1 und E_2 zwei Entitytypen, für die die Äquivalenzzusicherung $E_1 \equiv E_2$ gilt. Darüber hinaus sei E_1 die Menge von wertbasierten Attributen $\{A_1, \ldots, A_n, B_1, \ldots, B_s\}$ und E_2 die Menge von wertbasierten Attributen $\{A'_1, \ldots, A'_n, C_1, \ldots, C_t\}$ zugeordnet. Es gelten die Attributkorrespondenzzusicherungen $attcor_1(A_1, A'_1), \ldots, attcor_n(A_n, A'_n)$ mit $attcor_i(A_i, A'_i) = A_i \odot A'_i$ ($\odot \in \{\equiv, \supseteq, \cap, \neq\}$). Dann wird mit der Operation `integrate-join` ein neuer Entitytyp

$$E = \texttt{integrate-join}(E_1, E_2, attcor_1(A_1, A'_1), \ldots, attcor_n(A_n, A'_n))$$

erzeugt [Con97, S. 89f.]. Dabei wird nach den folgenden Punkten verfahren [SPD92, S. 109ff.]

1. Jedes Attribut B_i, das zum Entitytypen E_1 gehört, aber nicht zu E_2, wird mit demselben Wertebereich und derselben Kardinalität als Attribut von E aufgenommen. Analog wird mit jedem Attribut C_j verfahren, das für E_2, aber nicht für E_1

definiert ist.

2. Für zwei Attribute A_i und A'_i, für die eine Korrespondenzzusicherung $attcor_i(A_i, A'_i)$ gegeben ist, wird ein neues Attribut A für E in Abhängigkeit von der Zusicherungsart bestimmt:

   - $A_i = A'_i$

     Entweder A_i oder A'_i kann als neues Attribut übergenommen. Es wird das Attribut des lokalen Schemas bevorzugt, dessen Elemente mit Schemaelementen anderer Quellen integriert werden (siehe die Einleitung zu Beginn dieses Unterkapitels).

   - $A_i \supseteq A'_i$

     A_i wird als weniger eingeschränktes Attribut mit seinem Wertebereich und den entsprechenden Kardinalitäten übernommen.

   - $A_i \cap A'_i$

     Das neue Attribut A hat als Wertebereich die Vereinigung der Wertebereiche von A_1 und A_2. Im erweiterten EAV-Modell bedeutet dies, dass dem neuen Attribut die Datentypen der Ursprungsattribute zugewiesen werden, was einer der Gründe ist, warum bei der Definition des EAV-Modells die Möglichkeit vorgesehen wurde, einem Attribut mehr als einen Datentypen zuordnen zu können.

     Die minimale Kardinalität ergibt sich aus dem Maximum der minimalen Kardinalitäten von A_1 und A_2, die maximale Kardinalität aus der Summe der maximalen Kardinalitäten der beiden Ursprungsattribute. Dass die minimale Kardinalität aus dem Maximum und nicht der Summe der minimalen Kardinalitäten berechnet wird, erklärt sich dadurch, dass wenn die Werte der Attribute A_1 und A_2 übereinstimmen, diese nur einmal übernommen werden. Weichen die Werte hingegen ab, müssen sie für jedes Attribut erfasst werden.

   - $A_i \sim A'_i$

     Die Vereinigung der Wertebereiche von A_i und A'_i ergibt den Wertebereich des neuen Attributs. Wie oben bereits erwähnt wird dies im EAV-Modell durch die Zuweisung der Datentypen der beiden Quellattribute als Datentypen des neuen Attributs realisiert.

     Die minimale und maximale Kardinalität resultiert aus der Summe jeweils der minimalen und maximalen Kardinalitäten von A_1 und A_2. Die Werte der beiden Ursprungsattribute können nie übereinstimmen, daher ergeben sich die Werte des neuen Attributs immer als Vereinigung der Werte der beiden Quellattribute.

Darüber hinaus muss festgelegt werden, wie die Entities für den durch die Integration entstandenen neuen Entitytypen E aus den Entities der Ausgangstypen E_1 und E_2

generiert werden. Da die Zusicherung besteht, dass E_1 äquivalent zu E_2 ist, gibt es eine bijektive Zuordnung, die diejenigen Entities der beiden Typen in Beziehung setzt, die das gleiche Objekt der realen Welt abbilden. Die Methode der zusicherungsbasierten Integration verlangt, dass die Korrespondenzzusicherung $attcor_1(A_1, A_1')$ die Identität von E_1 bijektiv auf die Identität von E_2 abbildet [SPD92, S. 111], [Con97, S. 89]. Diese Forderung kann auf mehrere identifizierende Attribute erweitert werden, die durch Korrespondenzzusicherungen bijektiv aufeinander abgebildet werden.

Für das Relationenmodell werden durch den Primärschlüssel Attribute ausgezeichnet, deren Werte jedes Tupel einer Relation und damit auch jede Entity in der EAV-Darstellung der Relation identifizieren. Wie bereits erwähnt sind für die Auszeichnungssprache XML bzw. XML Schema als Schemabeschreibung Schlüssel und Eindeutigkeitsbedingungen jedoch von nicht so großer Bedeutung. In Unterkapitel 4.3.2 wurde bei der Abbildung von XML festgelegt, dass Elemente als identisch angesehen werden und somit als eine Entity zusammengefasst werden, wenn sie als Entities dieselben Werte für wertbasierte Attribute und Referenzattribute (bei Elementen mit komplexem Typ) aufweisen. Die Identität einer Entity ergibt sich somit aus den Werten aller Attribute. Für die Festlegung von Korrespondenzen zwischen identifizierenden Merkmalen zweier Entitytypen erweist sich dies als problematisch, da nicht erwartet werden kann, dass beide Entitytypen dieselben Attribute aufweisen bzw. eine Integration in diesem Fall trivial wäre. Aus diesem Grund müssen für Entitytypen, die aus der Abbildung einer XML-Quelle stammen, identifizierende Attribute explizit ausgezeichnet werden, bevor eine Schemaintegration vorgenommen werden kann.

Seien e_1 und e_2 zwei Entities der Typen E_1 und E_2, die dasselbe Objekt der realen Welt repräsentieren, was mit Hilfe der Korrespondenzzusicherungen erkannt wurde, die identifizierende Attribute beider Entitytypen miteinander in Bezug setzen. Dann setzt sich die Entity e des Typs E, der aus der Integration von E_1 und E_2 entstanden ist, folgendermaßen zusammen [SPD92, S. 111]

- $e.B_i = e_1.B_i$, wenn B_i ein Attribut ist, das für E_1 definiert ist, nicht aber für E_2
- $e.C_i = e_2.C_i$, wenn C_i ein Attribut ist, das für E_2 definiert ist, nicht aber für E_1
- $e.A_i = e_1.A_1$, wenn $attcor_i(A_i, A_i')$ entweder $A_i = A_i'$ (und E_1 das lokale Schema ist, das mit Elementen anderer lokaler Quellen erweitert wird) oder $A_i \supseteq A_i'$ ist
- $e.A_i = e_1.A_i \cup e_2.A_i'$, wenn $attcor_i(A_i, A_i')$ entweder $A_i \cap A_i'$ oder $A_i \sim A_i'$ ist

Erste Integrationsregel

Im Folgenden werdend die Regeln vorgestellt, anhand derer Entitytypen und Pfade integriert werden. Eine lokale Quelle mit einer Menge von Entitytypen und Pfaden, die sich aus den Referenzattributen der Entitytypen ergeben, soll dabei als (lokales) Schema bezeichnet werden.

Die erste Integrationsregel beschreibt, wie bei der Integration zweier Schemata Schemaelemente integriert werden, die mit keinem Element im anderen Schema über eine Zusicherung in Bezug gesetzt werden können. Sei E_1 ein Entitytyp des Schemas S_1, für den es im Schema S_2 keine Entsprechung gibt. Dann wird E_1 mit all seinen wertbasierten Attributen in das integrierte Schema S übernommen. Jeder Link $E_1 \leftrightarrow E_2$ in S_1, der sich aus einem Referenzattribut von E_1 ergibt, das auf E_2 verweist, wird als $E_1' \leftrightarrow E_2'$ in das Schema S übernommen. Hierbei sind E_1' und E_2' die Entitytypen in S, die sich aus der Integration von E_1 und E_2 ergeben.

Zweite Integrationsregel

Sei E_1 ein Entitytyp aus Schema S_1 und E_2 ein Entitytyp aus Schema S_2, für die

$$E_1 \equiv E_2 \text{ with corresponding attributes:}$$
$$attcor_1(A_1, A_1'), \ldots, attcor_n(A_n, A_n')$$

gilt. E_1 verfüge über die wertbasierten Attribute $\{A_1, \ldots, A_n, B_1, \ldots, B_s\}$ und E_2 über die ebenfalls wertbasierten Attribute $\{A_1', \ldots, A_n', C_1, \ldots, C_t\}$. Der integrierte Entitytyp E ergibt sich dann aus $E = \texttt{integrate-join}(E_1, E_2, attcor_1(A_1, A_1'), \ldots, attcor_n(A_n, A_n'))$ (siehe oben).

Dies soll an einem Beispiel verdeutlicht werden. Seien $\texttt{Buch}_1$ mit den Attributen ($\texttt{ISBN}_1$, $\texttt{Titel}_1$, $\texttt{Autor}_1$, $\texttt{Auflage}_1$, $\texttt{Schlagwörter}_1$, Urteil, Verlag) und $\texttt{Buch}_2$ mit den Attributen ($\texttt{ISBN}_2$, $\texttt{Titel}_2$, $\texttt{Autor}_2$, $\texttt{Auflage}_1$, $\texttt{Schlagwörter}_2$, Note, Preis) zwei Entitytypen. Die Attribute $\texttt{Auflage}_1$, $\texttt{Auflage}_2$ und Note sind vom Datentyp integer, Preis vom Datentyp decimal und die restlichen Attribute vom Datentyp string. Das Attribut $\texttt{Auflage}_2$ soll eine minimale Kardinalität von 0 und eine maximale Kardinalität von 1 haben, die Attribute $\texttt{Schlagwörter}_1$ und $\texttt{Schlagwörter}_2$ sollen mengenwertig sein und haben eine minimale Kardinalität von 0 und eine unbegrenzte maximale Kardinalität (ausgedrückt durch den symbolischen Wert n). Alle anderen Attribute haben eine minimale und maximale Kardinalität von 1. Es soll $\texttt{Buch}_1 \equiv \texttt{Buch}_2$ mit den folgenden Attributkorrespondenzzusicherungen gelten:

- $\texttt{ISBN}_1 = \texttt{ISBN}_2$
- $\texttt{Titel}_1 = \texttt{Titel}_2$
- $\texttt{Autor}_1 \cap \texttt{Autor}_2$

  Die Werte des Attributs $\texttt{Autor}_1$ folgen keinem bestimmten Format, d.h. zulässige Angaben können beispielsweise „Douglas Adams“, „D. Adams“ oder „Adams, Douglas“sein. Die Werte von $\texttt{Titel}_2$ werden hingegen auf das Format *voller Nachname, voller Vorname* eingeschränkt (z.B. „Adams, Douglas“). In XML Schema kann dies erreicht werden, indem der einfache Datentyp xs:string durch einen regulären

Ausdruck eingeschränkt wird[12]. Hiermit werden die Typeinschränkungen, die in XML Schema zulässig sind und bei der Abbildung in das EAV-Modell zunächst nicht erfasst wurden (siehe 4.3.2), bei der Integration berücksichtigt.

- $\texttt{Auflage}_1 \supseteq \texttt{Auflage}_2$

  Das Attribut $\texttt{Auflage}_2$ ist für $\texttt{Buch}_2$ optional, d.h. als Wert ist ein Nullwert zulässig, während dies für $\texttt{Auflage}_1$ und $\texttt{Buch}_1$ nicht der Fall ist.

- $\texttt{Schlagwörter}_1 \cap \texttt{Schlagwörter}_2$

  Die vergebenen Schlagwörter für Entities des Typs $\texttt{Buch}_1$ und $\texttt{Buch}_2$ können sich in manchen Fällen überschneiden.

- $\texttt{Urteil} \sim \texttt{Note}$

  Zulässige Bewertungen für Bücher, die als Entities des Typs $\texttt{Buch}_1$ repräsentiert werden, sind „Nobelpreis verdächtig“, „hervorragend“, „lesenswert“ und „muss man nicht gelesen haben“. Im Falle von $\texttt{Buch}_2$ werden Schulnoten von 1-6 vergeben. Zwischen den Attributen `Urteil` und `Note` besteht somit ein semantischer Zusammenhang, auch wenn die Domänen unterschiedlich ist (`string` vs. `integer`) und dadurch keine gemeinsamen Werte vorhanden sein können.

Als Ergebnis der Anwendung der `integrate-join`-Operation auf $\texttt{Buch}_1$ und $\texttt{Buch}_2$ ergibt sich dann der Entitytyp `Buch` mit den Attributen $\texttt{ISBN}_1$, $\texttt{Titel}_1$, `Autor`, $\texttt{Auflage}_1$, `Schlagwörter`, `Bewertung`, `Verlag` und `Preis`. Die Attribute, die sich durch die Korrespondenzzusicherungen $\equiv$ und $\supseteq$ aus den Quellattributen der beiden Entitytypen ergeben, entstehen durch die Übernahme eines der beiden Quellattribute, die durch die Korrespondenz miteinander in Bezug gesetzt werden. Die Attribute `Verlag` und `Preis`, für die es keine Korrespondenz im jeweils anderen Entitytypen gibt, werden von $\texttt{Buch}_1$ bzw. $\texttt{Buch}_2$ übernommen. Die Attribute, die als Ergebnis einer Korrespondenzzusicherung der Art $\cap$ oder $\sim$ für `Buch` definiert werden, sollen im Folgenden näher betrachtet werden:

- `Autor`

  Der einzige Unterschied zu den beiden Ausgangsattributen $\texttt{Autor}_1$ und $\texttt{Autor}_2$ besteht darin, dass die maximale Kardinalität 2 statt 1 beträgt. Das Attribut kann also zwei Werte aufnehmen: Die Werte der Attribute $\texttt{Autor}_1$ und $\texttt{Autor}_2$ für zwei Entities e_1 und e_2, die dasselbe Buch modellieren, können voneinander abweichen, da für den Entitytyp $\texttt{Buch}_1$ im Unterschied zu $\texttt{Buch}_2$ nicht einheitlich festgelegt wurde, in welchem Format die Autorennamen abgespeichert werden (siehe oben). Dem Attribut `Autor` könnten somit z.B. die Werte „D. Adams“ und „Adams,

[12] Streng genommen kann durch den regulären Ausdruck nur festgelegt werden, dass der Wert für das Attribut aus zwei Strings besteht, die durch ein Komma separiert werden. Des Weiteren kann eine bestimmte Mindestlänge der Strings spezifiziert werden und/oder das Punktuationszeichen „.“ als Bestandteil der Strings ausgeschlossen werden, um zu unterbinden, dass Abkürzungen angegeben werden. Die Idee, dass zuerst der Nachname und dann der Vorname angegeben werden, kann hingegen nicht erfasst werden.

Douglas“ zugewiesen werden. Stimmen hingegen die Werte von $\texttt{Autor}_1$ und $\texttt{Autor}_2$ überein, so wird nur ein Wert gespeichert.

- `Schlagwörter`

  Dieses Attribut entspricht in seinem Datentyp und Kardinalitäten den beiden Attributen $\texttt{Schlagwörter}_1$ und $\texttt{Schlagwörter}_2$.

- `Bewertung`

  Das Attribut `Bewertung` ergibt sich aus der Integration der beiden Quellattribute `Urteil` und `Note`. Sowohl die minimale als auch die maximale Kardinalität beträgt 2. Der Wertebereich des Attributs ergibt sich aus der Vereinigung der Wertebereiche der Datentypen `string` und `integer`. Mögliche Werte für dieses Attribut sind z.B. („Nobelpreis verdächtig“, 1), („hervorragend“, 2) oder („lesenswert“, 2).

Dritte Integrationsregel

Diese Regel beschreibt, wie einzelne Links integriert werden. Seien E_{11} und E_{12} zwei Entitytypen des Schemas S_1 und E_{21} und E_{22} zwei Entitytypen des Schemas S_2 sowie $E_{11} \leftrightarrow E_{12}$ und $E_{21} \leftrightarrow E_{22}$ zwei Links. Wenn die Zusicherungen $E_{11} \equiv E_{21}$, $E_{12} \equiv E_{22}$ und $E_{11} \leftrightarrow E_{12} \equiv E_{21} \leftrightarrow E_{22}$ gelten, dann entsteht der Link $E_1 \leftrightarrow E_2$ als Ergebnis der Integration der beiden Ausgangslinks. Dabei gilt, dass E_1 aus der Integration von E_{11} und E_{21} sowie E_2 aus der Integration von E_{12} und E_{22} resultiert.

Als Beispiel seien die Entitytypen $\texttt{Buch}_1$, `Filiale`, $\texttt{Buch}_2$, `Niederlassung` und die Links $\texttt{Buch}_1 \leftrightarrow$ `Filiale` sowie $\texttt{Buch}_2 \leftrightarrow$ `Niederlassung` gegeben, wobei $\texttt{Buch}_1 \equiv \texttt{Buch}_2$, `Filiale` $\equiv$ `Niederlassung` und $\texttt{Buch}_1 \leftrightarrow$ `Filiale` $\equiv \texttt{Buch}_2 \leftrightarrow$ `Niederlassung` gilt. Aus der Integration der beiden Links ergibt sich der Link `Buch` $\leftrightarrow$ `Standort`, wenn `Buch` und `Standort` die Entitytypen sind, die sich jeweils aus der Integration von $\texttt{Buch}_1$ und $\texttt{Buch}_2$ bzw. `Filiale` und `Niederlassung` ergeben.

Vierte Integrationsregel

Mit Hilfe dieser Regel können Pfade integriert werden. Seien $E_{11}, \ldots, E_{1n}$ Entitytypen im Schema S_1 und $E_{21}, \ldots, E_{2m}$ Entitytypen im Schema S_2, für die $E_{11} \equiv E_{21}$ und $E_{1n} \equiv E_{2m}$ gilt, wobei E_1 und E_2 die Entitytypen seien, die aus der Integration von E_{11} und E_{21} bzw. E_{1n} und E_{2m} entstehen. Dann gilt folgendes:

- Besteht die Zusicherung $E_{11} \leftrightarrow E_{1n} \equiv E_{21} \leftrightarrow E_{22} \leftrightarrow \cdots \leftrightarrow E_{2m-1} \leftrightarrow E_{2m}$, so entsteht als Ergebnis der Integration der Pfad $E_1 \leftrightarrow E'_{22} \leftrightarrow \cdots \leftrightarrow E'_{2m-1} \leftrightarrow E_2$, wobei E'_{22} und E'_{2m-1} die integrierten Entitytypen zu E_{22} und E_{2m-1} sind (analog gilt dies für alle Entitytypen auf dem Pfad zwischen E_1 und E_2). Es ist nicht nötig, den direkten Link $E_1 \leftrightarrow E_2$ in das integrierte Schema einzubinden, da dieser aus dem Pfad bereits abgeleitet werden kann [SPD92, S. 117]. Hierdurch wird Redundanz vermieden.

- Besteht die Zusicherung $E_{11} \leftrightarrow E_{12} \leftrightarrow \cdots \leftrightarrow E_{1n-1} \leftrightarrow E_{1n} \equiv E_{21} \leftrightarrow E_{22} \leftrightarrow \cdots \leftrightarrow E_{2m-1} \leftrightarrow E_{2m}$, so entstehen als Ergebnis der Integration zwei Pfade $E_1 \leftrightarrow E'_{12} \leftrightarrow \cdots \leftrightarrow E'_{1n-1} \leftrightarrow E_2$ und $E_1 \leftrightarrow E'_{22} \leftrightarrow \cdots \leftrightarrow E'_{2m-1} \leftrightarrow E_2$, wobei E'_{12}, E'_{1n-1}, E'_{22} und E'_{2m-1} die integrierten Entitytypen zu E_{12}, E_{1n-1}, E_{22} und E_{2m-1} sind (analog gilt dies für alle Entitytypen auf beiden Pfaden zwischen E_1 und E_2). Es müssen beide Pfade in das integrierte Schema übertragen werden, da sie Teilpfade enthalten, die nicht äquivalent zueinander sind und die verloren gehen würden, wenn einer der beiden Pfade nicht übertragen wird [SPD92, S. 117]. Durch eine Integritätsbedingung wird die Äquivalenz beider Pfade im integrierten Schema ausgedrückt.

Als Beispiel seien die Entitytypen `Standort`, `Filiale`, `Gattung`, `Kategorie`, $\texttt{Buch}_1$ und $\texttt{Buch}_2$ gegeben, für die folgende Äquivalenzen gelten:

- $\texttt{Standort} \equiv \texttt{Filiale}$
- $\texttt{Gattung} \equiv \texttt{Kategorie}$
- $\texttt{Buch}_1 \equiv \texttt{Buch}_2$

Seien $\texttt{Standort} \leftrightarrow \texttt{Gattung}$ und $\texttt{Filiale} \leftrightarrow \texttt{Buch}_2 \leftrightarrow \texttt{Kategorie}$ zwei Pfade, die äquivalent sind. Der erste Pfad besteht aus einem direkten Link und gibt für einen Bibliotheksstandort an, welche literarischen Gattungen dort vertreten sind. Der zweite Pfad vermittelt dieselbe Information, jedoch wird diese indirekt durch die Zuordnung von Filialen zu Büchern und Büchern zu Kategorien ausgedrückt. Im integrierten Schema braucht daher nur der zweite Pfad übernommen werden.

Fünfte Integrationsregel

Die fünfte Integrationsregel beschreibt, wie strukturelle Heterogenität im EAV-Modell überwunden werden kann, d.h. wie ein Entitytyp und ein wertbasiertes Attribut integriert werden, wenn sie als äquivalent erkannt wurden. Seien dazu E_1 und E_2 zwei Entitytypen sowie A ein wertbasiertes Attribut von E_1, für das $A \equiv E_2$ gilt. Sei E_1' der Entitytyp, der sich aus der Integration von E_1 im integrierten Schema ergibt. Dann wird das wertbasierte Attribut A von E_1 in ein Referenzattribut A' von E_1' umgewandelt, das auf den Entitytypen E_2' verweist, der sich aus der Integration von E_2 ergibt. Die Kardinalitäten des Attributs bleiben dabei erhalten. Der Name des Attributs A' soll dem Namen von E_2' entsprechen.

In den Listings 4.4 und 4.5 ist beispielhaft angegeben, wie in XML derselbe Sachverhalt mit unterschiedlichen Modellierungselementen dargestellt werden kann: In Listing 4.4 ist die Angabe zum Autor als XML-Attribut gegeben, während in Listing 4.5 dieselbe Angabe als Unterelement angegeben ist. In der EAV-Darstellung entspräche dies jeweils den Entitytypen `Fachbuch` mit den wertbasierten Attributen $\texttt{Titel}_1$, `Autor` und `Auflage` und `Gedicht` mit den Referenzattributen `Verfasser` und `Inhalt`, die auf die gleichnamigen Entitytypen verweisen. Es gelte `Autor` $\equiv$ `Verfasser`. Seien `Verfasser`$'$ und `Fachbuch`$'$ die integrierten Entitytypen zu `Verfasser` und `Fachbuch`. Der integrierte Entitytyp `Fachbuch`$'$ hat dann das Referenzattribut `Verfasser`$'$, das den gleichnamigen Entitytypen referenziert. In Listing 4.4 ist für dieses Beispiel das Resultat der Anwendung der fünften Integrationsregel dargestellt.

```
1 <Fachbuch Titel="Java ist auch eine Insel" Autor="Christian
      Ullenboom" Verlag="Galileo Press"/>
```

Listing 4.4: Angabe zum Autor als XML-Attribut

```
1 <Gedicht>
2   <Verfasser>Ezra Pound</Verfasser>
3   <Inhalt>
4       In einer Metrostation
5
6       Das Erscheinen dieser Gesichter in der Menge:
7       Blütenblätter auf einem nassen, schwarzen Zweig.
8   </Inhalt>
9 </Gedicht>
```

Listing 4.5: Angabe zum Autor/Verfasser als Unterelement

```
1 <Fachbuch' Titel="Java ist auch eine Insel" Verlag="Galileo Press">
2   <Verfasser'>Christian Ullenboom</Verfasser'>
3 </Fachbuch'>
```

Listing 4.6: Ergebnis der Anwendung der fünften Integrationsregel

4.5. Rückabbildungen aus dem EAV-Modell

In diesem Unterkapitel wird vorgestellt, wie die Rückabbildung von Entitytypen und Entities in das Relationenmodell und das XML-Datenmodell erfolgt. Bei der Betrachtung von Entitytypen werden wertbasierte Attribute und Referenzattribute betrachtet. Bei den wertbasierten Attributen wird zwischen atomaren und mengenwertigen Attributen unterschieden sowie danach, ob den Attributen ein Datentyp oder mehr als ein Datentyp zugewiesen ist.

Beispiel für dieses Unterkapitel

Die folgenden Abschnitte sollen mit Hilfe eines Beispiels veranschaulicht werden. Sei dazu `Buch` ein Entitytyp mit den atomaren Attributen

- ISBN: `string`
- Titel: `string`
- Autor: `string`
- Erscheinungsdatum: `date`
- Preis: `decimal`
- gebraucht: `boolean`

und den mengenwertigen Attributen

- Rezensenten: `string` mit einer minimalen Kardinalität von 5 und einer maximalen Kardinalität von 10
- Bewertungen: `string` $\cup$ `integer` mit einer minimalen Kardinalität von 2 und einer unbeschränkten maximalen Kardinalität (Werte dieses Attributs können entweder vom Typ `string` oder vom Typ `integer` sein)

Des Weiteren soll für den Entitytyp `Buch` das Referenzattribut `Filiale` gegeben sein, das einen gleichnamigen Entitytyp referenziert und eine minimale und maximale Kardinalität von 1 aufweist. Der Entitytyp `Filiale` soll die Attribute `Name` und `Adresse` besitzen.

4.5.1. Relationenmodell

Entitytypen

Ein Entitytyp wird auf ein gleichnamiges Relationenschema abgebildet. Wertbasierte Attribute des Entitytyps werden entweder auf Attribute dieses Relationenschemas abgebildet oder auf andere Relationenschemata, die mittels Fremdschlüsselbedingungen mit dem Relationenschema verknüpft werden, auf das der Entitytyp abgebildet wurde. Referenzattribute werden als Beziehungen zwischen Relationenenschemata erfasst, die ebenfalls durch Fremdschlüsselbedingungen ausgedrückt werden. Dies wird in den folgenden Abschnitten näher ausgeführt.

Es ergibt sich die Frage, aus welchen Attributen der Primärschlüssel des neu erzeugten Relationenschemas gebildet wird. Wenn der Entitytyp aus der Abbildung eines Relationenschemas entstand, so kann der ursprüngliche Primärschlüssel verwendet werden. Im Allgemeinen gilt dies für Entitytypen, die aus der Abbildung von Schemakonstrukten eines Datenmodells entstanden sind, das identifizierende Attribute unterstützt.

Für die zusicherungsbasierte Schemaintegration wurde in 4.4 gefordert, dass die Entitytypen identifizierende (wertbasierte) Attribute aufweisen. Diese können als Primärschlüsselattribute übernommen werden, sofern die identifizierenden Attribute atomar und nicht mengenwertig sind, da sie nur dann auf Attribute des Relationenschemas abgebildet werden können (siehe unten). Eine weitere Möglichkeit besteht darin, *künstliche Schlüssel* einzuführen (siehe [SSH10, S. 326]), über die jedes Tupel einer Relation identifiziert wird.

Wertbasierte Attribute

Ein **atomares** wertbasiertes Attribut hat eine minimale Kardinalität von 0 oder 1 und eine maximale Kardinalität von 1. Besitzt ein solches Attribut nur einen Datentypen, so kann das Attribut des Entitytyps direkt auf ein relationales Attribut abgebildet werden, das denselben Datentyp aufweist. Beträgt die minimale Kardinalität 0, so ist für das relationale Attribut ein Nullwert zulässig. Sind dem Ausgangsattribut hingegen mehrere Datentypen zugeordnet, so wird das Attribut auch auf mehrere Attribute abgebildet, je eins pro Datentyp. Bei der Erfassung eines Werts für das Ausgangsattribut weisen alle relationalen Attribute, die nicht dem Datentyp des Werts entsprechen, einen Nullwert auf. Das relationale Attribut, dessen Datentyp mit dem des tatsächlichen Werts übereinstimmt, enthält dann diesen Wert.

Ein wertbasiertes Attribut wird als **mengenwertig** bezeichnet, wenn seine minimale und/oder maximale Kardinalität größer als 1 ist (die maximale Kardinalität kann auch unbeschränkt sein, was durch den symbolischen Wert n ausgedrückt wird). Ein solches Attribut wird auf ein eigenes Relationenschema abgebildet. Das ursprüngliche Attribut wird dann ein Attribut dieses Relationenschemas. Weist das Attribut hingegen mehrere

Datentypen auf, muss für jeden Datentyp ein eigenes relationales Attribut eingeführt werden.

Der Zusammenhang zwischen dem Relationenschema, auf das der Entitytyp abgebildet wurde, und dem Relationenschema, auf das das mengenwertige Attribut des Entitytyps abgebildet wurde, wird durch ein drittes Relationenschema hergestellt. Dieses enthält die Primärschlüsselattribute der beiden Relationenschemata, die jeweils als Fremdschlüssel auf das Entitytyp-Relationenschema und das Attribut-Relationenschema verweisen. Dies entspricht einer minimalen Kardinalität von 0 und einer unbeschränkten maximalen Kardinalität. Andere Kardinalitäten, die einen konkreten numerischen Wert größer als 1 (aber nicht n) für die minimale oder maximale Kardinalität vorgeben, können mit Fremdschlüsselbedingungen nicht ausgedrückt werden. Im Relationenmodell wäre es zwar möglich, globale Integritätsbedingungen zu formulieren, die andere Kardinalitäten umsetzen könnten, jedoch sind in der Anfragesprache SQL als Form der referentiellen Integrität nur Fremdschlüsselbeziehungen vorgesehen [SSH10, S. 392ff.], sodass die globalen Integritätsbedingungen in der Praxis nicht umsetzbar wären.

Referenzattribute

Ein Referenzattribut wird durch eine Fremdschlüsselbedingung erfasst. Diese setzt das Relationenschema R_1, auf das der Entitytyp abgebildet wurde, dem das Attribut zugeordnet ist, mit dem Relationenschema R_2 in Bezug, auf das der Entitytyp abgebildet wurde, der durch das Referenzattribut referenziert wird. Diese Fremdschlüsselbeziehung hat dann die Form $X(R_1) \rightarrow Y(R_2)$. Wenn das Referenzattribut dabei aus der Abbildung einer relationalen Fremdschlüsselbeziehung entstand, so ist bekannt, über welche Attribute X und Y die Fremdschlüsselbedingung umgesetzt werden kann (siehe 4.3.1). Andernfalls müssen die Primärschlüsselattribute des Relationenschemas, auf das der Entitytyp abgebildet wurde, der das Referenzattribut besitzt, in das Relationenschema des Entitytyps aufgenommen werden, der referenziert wird. Die Primärschlüsselattribute leiten sich dabei wie bereits erwähnt entweder aus identifizierenden Attributen des Entitytyps ab oder stellen künstliche Schlüssel dar (siehe oben).

Wenn für das Referenzattribut eine andere Kardinalität als 0:1 oder 1:1 vorliegt, so muss wie bei mengenwertigen wertbasierten Attributen ein drittes Relationenschema erzeugt werden, das die Primärschlüsselattribute der beiden Relationenschemata aufnimmt und durch eine Fremdschlüsselbeziehung auf diese verweist. Hierdurch entsteht eine minimale Kardinalität von 0 und eine maximale Kardinalität von n. Auch an dieser Stelle ergibt sich das Problem, dass Kardinalitäten größer als 1 (aber nicht n) mit Fremdschlüsselbedingungen nicht abgebildet werden können (siehe den Abschnitt zu wertbasierten Attributen oben).

Beispiel

Der Entitytyp `Buch` wird auf ein Relationenschema `Buch(ISBN, Titel, Autor, Erscheinungsdatum, Preis, gebraucht)` abgebildet. Dabei wird davon ausgegangen, dass das Attribut `ISBN` den Entitytypen eindeutig identifiziert, weshalb es zum Primärschlüssel ausgezeichnet wird.

Das mengenwertige Attribut `Rezensenten` wird auf das Relationenschema `Rezensenten(ID, Rezensent)` abgebildet, wobei das Attribut `ID` einen künstlichen Schlüssel darstellt. Über das Relationenschema `Buchrezensenten(ISBN, RezensentID)` wird der Zusammenhang zwischen `Buch` und `Rezensent` hergestellt. Dabei gelten die Fremdschlüsselbedingungen

- `ISBN`(`Buchrezensenten`) → `ISBN`(`Buch`) und
- `RezensentID`(`Buchrezensenten`) → `ID`(`Rezensent`)

Das Attribut `Bewertungen`, das ebenfalls mengenwertig ist, wird auf das Relationenschema `Bewertung(ID, Bewertung`$_1$`, Bewertung`$_2$`)` abgebildet. Dabei soll gelten, dass das relationale Attribut `Bewertung`$_1$ vom Typ `string` ist, während das Attribut `Bewertung`$_2$ vom Typ `integer` ist. Dies entspricht dem Umstand, dass die Werte des ursprünglichen mengenwertigen Attributs `Bewertungen` sowohl String- als auch Integerwerte sein können. Der Zusammenhang zwischen Büchern und Bewertungen wird über das Relationenschema `Buchbewertung(ISBN, BewertungID)` hergestellt, wobei folgende Fremdschlüsselbeziehungen gelten:

- `ISBN`(`Buchbewertung`) → `ISBN`(`Buch`) und
- `BewertungID`(`Buchbewertung`) → `ID`(`Bewertung`)

Das Referenzattribut `Filiale` bezieht sich auf einen Entitytypen des gleichen Namens. Dieser sei auf ein Relationenschema `Filiale(Name, Adresse)` abgebildet. Da das Referenzattribut eine minimale und maximale Kardinalität von 1 besitzt, kann es durch eine Fremdschlüsselbedingung abgebildet werden und ein zusätzliches Relationenschema muss nicht eingeführt werden. Dazu muss das Attribut `Name` des Relationenschemas `Filiale` als Attribut `Filialenname` des gleichen Typs (`string`) in das Relationenschema `Buch` aufgenommen werden. Dann ergibt sich die Fremdschlüsselbeziehung `Filialenname`(`Buch`) → `Name`(`Filiale`).

4.5.2. XML

Entitytypen

Ein Entitytyp wird auf ein XML-Element abgebildet. Wertbasierte Attribute des Entitytyps werden auf Attribute dieses Elements abgebildet, sofern es sich um atomare Attribute handelt. Liegt hingegen ein mengenwertiges Attribut vor, so wird dieses auf ein Unterelement abgebildet. Referenzattribute werden durch Schlüsselreferenzen erfasst.

Wertbasierte Attribute

Jedes atomare Attribut eines Entitytypen wird als XML-Attribut abgebildet. Beträgt die minimale Kardinalität 0, so wird dies in XML Schema durch die Angabe von `use="optional"` in der Attributdeklaration ausgedrückt (siehe die Vorstellung von XML Schema in Unterkapitel 4.3.2), andernfalls gilt `use="required"`. Ein mengenwertiges Attribut wird als Unterelement abgebildet. Dabei können die minimalen und maximalen Kardinalitäten des mengenwertigen Attributs bei der Deklaration des Unterelements direkt als Werte von `minOccurs` und `maxOccurs` übernommen werden.

Liegt ein Attribut vor, das mehr als einen Datentypen hat, so kann dies durch das `union`-Konstrukt bei der Typdefinition in XML Schema umgesetzt werden.

Referenzattribute

Ein Referenzattribut wird auf eine Schlüsselreferenz (`xs:keyref`) abgebildet, die dem referenzierenden Element zugeordnet wird und sich auf den Schlüssel `xs:key` des referenzierten Elements beziehen muss[13]. Wie bei der Rückabbildung in das relationale Modell ergibt sich auch hier die Frage, wie der Schlüssel für das referenzierte Element gebildet wird. Ist für den Entitytypen vermerkt, welche Attribute eine Entity dieses Typs eindeutig identifizieren, so kann diese Information, die aus der Abbildung aus dem ursprünglichen Datenmodell in das EAV-Modell entstanden ist, für die Abbildung auf XML verwendet werden. Ist dies nicht der Fall, kann ein künstliches identifizierendes Attribut eingeführt werden.

Für jedes Attribut des referenzierten Schlüssels wird ein Unterelement mit einfachem Typ im referenzierenden Element eingefügt, dessen Elementinhalt den Attributwert enthalten soll. Die minimale und maximale Kardinalität des Referenzattributes werden als Werte der Attribute `minOccurs` und `maxOccurs` der Unterelemente übernommen. Die Schlüsselreferenz setzt sich dann aus den Unterelementen zusammen. Der Schlüssel selbst setzt sich aus den identifizierenden Attributen des referenzierten Elements zusammen.

[13] Schlüssel und Referenzen auf Schlüssel in XML Schema werden in [KM03, S. 122f.] beschrieben.

Beispiel

In Listing 4.7 sind die Elementdeklarationen für die mengenwertigen Attribute `Rezensenten` und `Bewertungen` angegeben. Auf diese wird in Listing 4.8 bei der Deklaration des Elements `Buch` mit Angaben der entsprechenden Werte für `minOccurs` und `maxOccurs` verwiesen, mit denen die jeweiligen Kardinalitäten abgebildet werden. Dort ist auch die Abbildung der atomaren Attribute auf XML-Attribute angegeben. Des Weiteren wird dargestellt, wie das Referenzattribut `Filiale` abgebildet wird: Die Schlüsselreferenz `Filiale_Keyref` setzt das Element `Filiale_Name` mit dem Attribut `Name` des Elements `Filiale` (siehe Listing 4.9) in Bezug. Das Attribut `Name` ist durch die Schlüsseldeklaration `Filiale_Key` zum Schlüssel ausgezeichnet worden.

```
1 <xs:element name="Rezensenten" type="xs:string"/>
2 <xs:element name="Bewertungen">
3   <xs:simpleType>
4     <xs:union memberTypes="xs:string xs:integer"/>
5   </xs:simpleType>
6 </xs:element>
```

Listing 4.7: Abbildung der mengenwertigen Attribute

```
1 <xs:element name="Buch">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element ref="Rezensenten" minOccurs="5" maxOccurs="10"/>
5       <xs:element ref="Bewertungen" minOccurs="2"
            maxOccurs="unbounded"/>
6       <xs:element name="Filiale_Name" minOccurs="1" maxOccurs="1"/>
7     </xs:sequence>
8     <xs:attribute name="ISBN" type="xs:string" use="required"/>
9     <xs:attribute name="Titel" type="xs:string" use="required"/>
10    <xs:attribute name="Autor" type="xs:string" use="required"/>
11    <xs:attribute name="Erscheinungsdatum" type="xs:date"
          use="required"/>
12    <xs:attribute name="Preis" type="xs:decimal" use="required"/>
13    <xs:attribute name="gebraucht" type="xs:boolean" use="required"/>
14  </xs:complexType>
15  <xs:keyref name="Filiale_Keyref" refer="Filiale_Key">
16    <xs:selector xpath="//Buch"/>
17    <xs:field xpath="Filiale_Name"/>
18  </xs:keyref>
19 </xs:element>
```

Listing 4.8: Abbildungen der atomaren Attribute und Referenzattribute

```
 1 <xs:element name="Filiale">
 2   <xs:complexType>
 3     <xs:attribute name="Name" type="xs:string" use="required"/>
 4     <xs:attribute name="Adresse" type="xs:string" use="required"/>
 5   </xs:complexType>
 6   <xs:key name="Filiale_Key">
 7     <xs:selector xpath="//Filiale"/>
 8     <xs:field xpath="Name"/>
 9   </xs:key>
10 </xs:element>
```

Listing 4.9: Deklaration des Elements `Filiale`

4.6. Fazit

In diesem Kapitel wurde das Konzept für die „Global-as-local-view-extension“-Technik vorgestellt. Im Folgenden wird zusammenfassend ein Überblick über den Inhalt dieses Kapitels angegeben.

Integration mit Schema Mappings und Schema-Mapping-Inversen

Zu Beginn des Kapitels wurde untersucht, inwiefern Schema Mappings und ihre Inversen für das zu entwickelnde Konzept einsetzbar sind. Für die Rückabbildung in die lokalen Schemata aus dem globalen Schema müssen im globalen Schema alle Attribute der lokalen Schemata enthalten sein. Aus dieser Forderung ergeben sich jedoch für Datensätze im globalen Schema viele Nullwerte, da für Attribute, die einem lokalen Schema entstammen und nur dort vorhanden sind, kein Wert für einen konkreten Datensatz aus einem anderen lokalen Schema angegeben werden kann. Dieses Problem motivierte die Verwendung des EAV-Modells als globales Datenmodell. Es zeigte sich jedoch, dass die Abbildung auf das EAV-Modell mit Schema Mappings nicht zufriedenstellend erfassbar ist. Der Ansatz, die Abbildung auf das EAV-Modell mit Schema Mappings zu beschreiben, wurde daher nicht weiter verfolgt.

Erweitertes EAV-Modell

Im Anschluss daran wurde ein erweitertes EAV-Modell vorgestellt. Dieses führt eine Unterscheidung zwischen **wertbasierten Attributen** und **Referenzattributen** ein. Mit Hilfe von Referenzattributen sollen Beziehungen zwischen Schemaelementen erfasst werden, die es in vielen Datenmodellen gibt. Ebenso werden **Kardinalitäten** für Attribute beider Arten unterstützt. Des Weiteren kann wertbasierten Attributen mehr als nur ein Wertebereich bzw. ein Datentyp zugeordnet werden. Somit können Datenmodellkonstrukte abgebildet werden, die ebenfalls mehr als einen Datentyp aufweisen (z.B. das `union`-Konstrukt bei XML Schema). Darüber hinaus spielt diese Eigenschaft

bei der zusicherungsbasierten Schemaintegration eine Rolle, wo Attribute verschiedener Wertebereiche zu einem Attribut vereinigt werden.

Abbildungen in das erweiterte EAV-Modell

Es wurde vorgestellt, wie Schemaelemente und Dateninstanzen des **Relationenmodells** und der Auszeichnungssprache **XML** (in Verbindung mit der Schemabeschreibungssprache XML Schema) auf das EAV-Modell abgebildet werden.

Im Relationenmodell werden Relationenschemata auf Entitytypen abgebildet. Die Attribute des Relationenschemas werden als wertbasierte Attribute des entsprechenden Entitytyps und Fremdschlüsselbeziehungen zwischen Relationenschemata als Referenzattribute erfasst. Primärschlüssel- und Fremdschlüsselattribute werden vermerkt, um diese Information bei der Rückabbildung verwenden zu können. Tupel einer Relation werden als Entities repräsentiert.

In einer XML-Schema-Datei deklarierte Elemente werden auf Entitytypen abgebildet, sofern sie einen komplexen Typ aufweisen, d.h. XML-Attribute und/oder Kindelemente besitzen. Elemente mit einfachem Typ werden auf wertbasierte Attribute des übergeordneten Elements abgebildet. Dies trifft auch auf XML-Attribute eines Elements zu. Die Schachtelung von Elementen wird durch Referenzattribute abgebildet.

Zusicherungsbasierte Schemaintegration

Auf Grundlage der zusicherungsbasierten Schemaintegration wurde beschrieben, wie Entitytypen integriert werden können, die dieselbe Menge von Objekten der realen Welt repräsentieren. Die Integration basiert auf der Beschreibung der Beziehungen zwischen den wertbasierten Attributen der Entitytypen. Referenzattribute werden gesondert bei der Integration von Pfaden berücksichtigt. Ein Pfad besteht aus mehreren Referenzattributen und setzt Entitytypen in Bezug zueinander, die durch die Referenzattribute verbunden sind. Als Ergebnis der Integration entstehen integrierte Entitytypen und Pfade. Diese werden anschließend auf die lokalen Datenmodelle rückabgebildet.

Durch die in diesem Kapitel beschriebene Form der Integration werden neue Schemata erzeugt, die vom lokalen Quellschema getrennt sind. Ein damit verbundener Nachteil ist, dass die Nutzer dieses neue Schema kennenlernen müssen, um Anfragen stellen zu können. Dem könnte entgegengewirkt werden, indem das Integrationsverfahren so modifiziert wird, dass das lokale Schema stärker berücksichtigt wird.

Rückabbildung in die lokalen Datenmodelle

Abschließend wurde die Rückabbildung aus dem EAV-Modell auf das Relationenmodell und XML betrachtet.

Entitytypen werden auf Relationenschemata im relationalen Datenmodell abgebildet. Wertbasierte Attribute des Entitytyps werden als relationale Attribute des entsprechenden Relationenschemas repräsentiert, wenn sie eine Kardinalität von 0:1 oder 1:1 haben. In beiden Fällen muss das wertbasierte Attribut auf mehrere relationale Attribute aufgeteilt werden, falls dem wertbasierten Attribut mehr als nur ein Datentyp zugeordnet ist (ein relationales Attribut pro Datentyp). Referenzattribute können direkt mit Fremdschlüsselbeziehungen umgesetzt werden, wenn eine Kardinalität von 0:1 oder 1:1 vorliegt. Andernfalls muss ein Relationenschema eingeführt, in dem die Zuordnung zwischen den Entities der Entitytypen bzw. Tupel der Relationenschemata, die als Ergebnis der Abbildung der Entitytypen vorliegen, festgehalten wird.

In der Rückabbildung auf XML werden Entitytypen auf Elemente mit komplexem Typ abgebildet. Wertbasierte Attribute mit einer Kardinalität von 0:1 oder 1:1 können als XML-Attribute dargestellt werden. Liegt eine andere Kardinalität vor, so wird das wertbasierte Attribut als Element mit einfachem Typ repräsentiert, das dem Element untergeordnet ist, auf das der Entitytyp abgebildet wird. Referenzattribute werden durch Schlüssel und Schlüsselreferenzen in XML Schema umgesetzt.

5. Implementierung

In diesem Kapitel wird die Implementierung des Demonstrators beschrieben, mit dem das in Kapitel 4 beschriebene Konzept umgesetzt werden soll. Da die Umsetzung nur prototypisch erfolgt, wird im Demonstrator eine materialisierte Integration vorgenommen. In einem realen Anwendungsszenario wäre die in Kapitel 4 vorgestellte Technik Bestandteil einer auf virtueller Integration basierenden Föderation, wozu zusätzlich geeignete Anfragetechniken entwickelt werden müssen.

Im Folgenden wird zunächst beschrieben, welche Technologien bei der Implementierung eingesetzt wurden. Anschließend wird die Client-Server-Architektur der Anwendung vorgestellt.

5.1. Verwendete Technologien

5.1.1. Java

Der Demonstrator wurde in der objektorientierten Programmiersprache Java[1] realisiert. Die Plattformunabhängigkeit hat diese Wahl maßgeblich beeinflusst: In Java geschriebene Anwendungen sind in jedem Betriebssystem, für das eine Implementierung der *Java Virtual Machine* vorliegt, lauffähig. Im Kontext der Datenintegration ist dies ein wichtiger Faktor, da sich die Heterogenität der lokalen Datenquellen u.a. durch verschiedene Betriebssysteme ergibt, die nicht binärkompatibel zueinander sind. Für die in dieser Arbeit entwickelte Anwendung ist dies vor allem für die Client-Komponente (siehe 5.2.2) von Vorteil, die auf den Plattformen der lokalen Datenquellen lauffähig sein muss. Aber auch für die Server-Komponente (siehe 5.2.3) ist dies vorteilhaft, da eine flexible Auswahl der Plattform ermöglicht wird, auf der die Server-Komponente betrieben werden soll.

Des Weiteren kann bei der Entwicklung mit Java auf ein vielfältiges Angebot von Bibliotheken zurückgegriffen werden, was sich in der Implementierung des Demonstrators vor allem bei der Anbindung der Datenbankmanagementsysteme (DBMS) und der Realisierung der Netzwerkkommunikation als nützlich erwies. Hervorgehoben sei das Framework *Netty*[2], das die Umsetzung einer asynchronen, Ereignis-basierten Client-Server-Kommunikation erleichtert.

[1] `http://www.java.com`
[2] `http://www.netty.io/`

5.1.2. MySQL

Das relationale Datenbankmanagementsystem MySQL[3] wurde ausgewählt, da es weit verbreitet ist und den im ANSI/ISO Standard SQL:2003 beschriebenen Aufbau von Tabellen zur Verwaltung der relationalen Metadaten (`INFORMATION_SCHEMA`) umsetzt[4]. Diese Metadaten spielen eine zentrale Rolle bei der Abbildung relationaler Quellen auf das EAV-Modell (siehe 5.2.2). MySQL wird nicht nur auf Client-Seite als relationale Datenquelle eingesetzt, sondern auch auf dem Server, da das in 4.2 vorgestellte EAV-Modell in einer relationalen Datenbank umgesetzt wird.

5.1.3. BaseX

Bei BaseX[5] handelt es sich um ein XML-DBMS. Es zeichnet sich durch eine vollständige Unterstützung von XQuery sowie eine Client-Server-Architektur aus, sodass nicht nur lokale, sondern auch entfernte Anfragen an das DBMS gestellt werden können. Aus diesem Grund wurde es zur Demonstration der Abbildung von XML auf das EAV-Modell und für die umgekehrte Rückabbildung ausgewählt. Einige relationale DBMS unterstützen ebenfalls die Speicherung und Verarbeitung von XML-Dokumenten. Diese wurden bei der Auswahl eines XML-DBMS bewusst ausgeklammert, um Überlappungen zwischen relationalen Datenquellen und XML-Datenquellen zu vermeiden und somit die Unterschiede zwischen den beiden Datenmodellen deutlicher hervorzuheben.

5.2. Architektur

In diesem Unterkapitel wird die Client-Server-Architektur des Demonstrators erläutert, wobei zunächst eine Übersicht über den Kommunikationsfluss zwischen Client und Server angegeben wird und im Anschluss die Client- und Server-Komponenten beschrieben werden.

5.2.1. Übersicht

Der Kommunikationsablauf zwischen einem Client und dem Server wird in Abbildung 5.1 in Form eines UML-Sequenzdiagramms veranschaulicht. `Client1` stellt eine Client-Instanz dar, die sich bei der Server-Instanz `Server` neu registriert, während `Client2` bereits registriert ist. Die Registrierung erfolgt dabei durch eine `join`-Nachricht (Nachricht zum Beitritt), die als Inhalt das lokale Entityschema von `Client1` enthält, d.h. alle Entitytypen, die auf Basis des lokalen Schemas, das `Client1` zugeordnet ist, konstruiert werden. Der `Server` vermerkt diese Information und benachrichtigt `Client1` über den Erfolg dieses Vorgangs. `Client1` fragt durch die `getSchemas()`-Nachricht beim `Server` nach, welche anderen Entityschemata bekannt sind. Diese Anfrage wird vom `Server` auf Basis der bei ihm registrierten Clients beantwortet.

[3] `http://www.mysql.com/`
[4] `http://dev.mysql.com/doc/refman/5.0/en/information-schema.html`
[5] `http://basex.org/`

Schließlich wird lokal bei `Client1` eine Integration des lokalen Schemas mit einem der anderen Schemata vorgenommen (`integrateSchemas(localSchema, remoteSchema)`). Als Ergebnis entsteht ein integriertes Schema (`integratedSchema`). Dieses wird materialisiert (`populateSchema(integratedSchema)`), indem zunächst beim `Server` alle Entities, d.h. Instanzen zu Entitytypen im entfernten Schema (`remoteSchema`) angefragt werden. Der `Server` leitet diese Anfrage an `Client2` weiter, da diesem die lokale Datenquelle zugeordnet ist, der das Entitieschema `remoteSchema` entstammt. Anschließend werden die lokalen Entities und die Entities des entfernten Schemas gemäß der definierten Korrespondenzen zwischen den Entitytypen der beiden Schemata integriert.

Das so entstandene integrierte Schema kann wiederum mit einem anderen der entfernten Schemata integriert werden, sodass nacheinander alle anderen lokalen Schemata mit dem eigenen Schema integriert werden können.

Alle ausgetauschten Nachrichten werden asynchron verarbeitet. Dies ist insbesondere für den Server von Vorteil. Einkommende Nachrichten eines Clients werden an einen neu erzeugten Thread delegiert, der die Nachricht verarbeitet. Wenn die Verarbeitung abgeschlossen ist, kann das Ergebnis an den Client gesendet werden, sofern eine Rücknachricht erwartet wird. Der Server ist somit in der Lage, mit einer Vielzahl von Clients parallel zu kommunizieren. Die asynchrone Client-Server-Kommunikation wurde auf Basis des Frameworks *Netty* entwickelt (siehe 5.1.1).

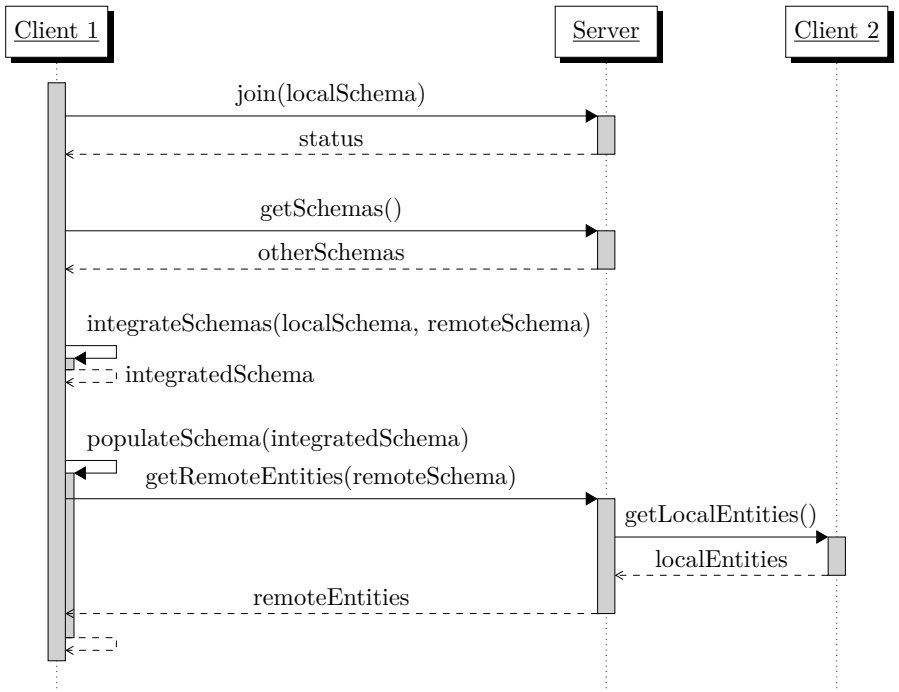

Abbildung 5.1.: UML-Sequenzdiagramm zum Kommunikationsablauf zwischen Client und Server

5.2.2. Client

Die Client-Komponente der Anwendung hat mehrere Aufgaben und damit verbundene Funktionalitäten. Einige Funktionalitäten sind datenmodellspezifisch. Diese wurden durch ein Interface modelliert, das jeweils für das Relationenmodell und XML implementiert werden muss. Andere Funktionalitäten hingegen sind unabhängig vom verwendeten Datenmodell, sodass es genügt, diese in nur einer gemeinsamen Klasse zu implementieren. Im Folgenden wird eine Auswahl der wichtigsten Interfaces und Klassen vorgestellt.

Das Interface `IJoinHandler` wird von den Klassen `RelJoinHandler` und `XMLJoinHandler` implementiert. Deren Aufgabe ist die Erzeugung einer `JoinRequestMessage`, die alle Entitytypen der lokalen Datenquelle enthält und zur Registrierung des Clients an den Server geschickt wird. Im Falle des `RelJoinHandler` werden die dafür erforderlichen Metadaten durch eine Reihe von SQL-Anfragen an die DBMS-interne Datenbank `INFORMATION_SCHEMA` ermittelt. Die Vorgehensweise beim `XMLJoinHandler` ist anders: Statt mehrere XQuery-Anfragen an die BaseX-Instanz zu stellen, wird als Anfrage ein XQuery-Skript ausgeführt, das ein XML-Dokument generiert, welches die gewünschte Abbildung von XML-Elementen auf Entitytypen beschreibt. Dieses Dokument wird anschließend geparst.

Die Klasse `SchemaIntegrator` führt die Integration des lokalen Schemas mit dem Schema einer anderen Datenquelle durch. Die Korrespondenzen zwischen den Entitytypen werden in einer XML-Datei spezifiziert, die als Eingabe eingelesen wird (ein Beispiel ist in Listing B.2 im Anhang angegeben)). Die Entities für das so entstandene integrierte Schema werden aus den Entities der lokalen und der entfernten Datenquelle durch Methoden der Klasse `EntityIntegrator` erzeugt. Der entfernte Client exportiert dabei seine Entities. Dieser Exportprozess ist in den Klassen `RelDataExportHandler` und `XMLDataExportHandler` umgesetzt, die das Interface `IDataExportHandler` implementieren. In beiden Klassen werden für jeden Entitytyp, der für das Schema definiert ist, alle Entities durch SQL- bzw. XQuery-Anfragen ermittelt.

Die Klassen `RelSchemaBuilder` und `XMLSchemaBuilder`, die das Interface `ISchemaBuilder` implementieren, setzen das integrierte Schema schließlich im jeweiligen Datenmodell um (als relationale Datenbank bzw. als XML Schema) und instanziieren dieses auf Grundlage der integrierten Entities. Die Materialisierung und die Instanziierung werden in einem Schritt ausgeführt, weshalb der Export und die Integration von Daten (Entities) vorher erfolgt. Alternativ könnte die Materialisierung auch von der Instanziierung entkoppelt werden, sodass die Daten erst zu einem späteren Zeitpunkt ermittelt werden müssten.

5.2.3. Server

Der Server hat weniger Funktionalitäten als der Client. Dies liegt u.a. daran, dass eine materialisierte Integration vorgenommen wird. Bei einer virtualisierten Integration würde der Server zusätzliche Aufgaben in Bezug auf die Anfragebearbeitung übernehmen. In

der gegenwärtigen Implementierung zeichnet er sich dafür verantwortlich, die lokalen Schemata aller registrierten Clients in der EAV-Darstellung zu verwalten. Beim Eingang einer `JoinRequestMessage` werden alle Entitytypen extrahiert und in einer Datenbank gespeichert, die das in 4.2 beschriebene erweiterte EAV-Modell realisiert. Die zentralen Funktionalitäten des Servers sind in der Klasse `ServerMessageHandler` gebündelt.

Basierend auf der Zuordnung von Schemata zu Clients leitet der Server Nachrichten zwischen den Clients weiter. Fordert ein Client beispielsweise alle Entities eines bestimmten Schemas an, so schickt er zunächst eine `ExportServerRequestMessage` an den Server. Dieser versendet daraufhin eine `ExportClientRequestMessage` an den Client, dem das betreffende Schema zugeordnet ist. Der befragte Client antwortet mit einer `ExportClientResponseMessage`, welche die gewünschten Entities enthält und unverändert an den anfragenden Client weitergeleitet wird.

5.3. Zusammenfassung und Vergleich mit dem Konzept

In diesem Kapitel wurde die Implementierung des Demonstrators behandelt, mit dem die in Kapitel 4 beschriebene Technik umgesetzt wurde. Der Demonstrator wird anschließend in Kapitel 6 zur Evaluierung eingesetzt. Die Implementierung beruht auf einem Client-Server-Modell und einer asynchronen Kommunikation durch Austausch von Nachrichten. Es wurde eine materialisierte Integration realisiert, in der der Client einen Großteil der Gesamtfunktionalität in sich vereinigt: Die Integration von Schemata und die Materialisierung eines integrierten Schemas erfolgt auf Client-Seite, während der Server im Wesentlichen die Entityschemata aller Clients verwaltet und den Nachrichtenaustausch organisiert. Alternativ dazu kann die Integration von Schemata auch im Server erfolgen, insbesondere wenn es viele Clients gibt, für die gemeinsame integrierte Schemata vorliegen können.

In dem in Kapitel 4 vorgestellten Konzept für die Global-as-local-view-extension-Technik sind für das erweiterte EAV-Modell Relationenschemata vorgesehen, in denen die Datensätze der Quellen als Entities dargestellt werden können. Dies sind die Relationenschemata, mit denen die Werte von wertbasierten Attributen (z.B. die Schemata `StringValue`, `BooleanValue`, `IntValue`) und Referenzattributen (das Schema `RefValue`) erfasst werden. Davon wurde bei der Implementierung des Demonstrators kein Gebrauch gemacht, da die (materialisierte) Integration hier lokal erfolgt. Diese wären in einer zentralisierten online-materialisierenden Integration (siehe Kapitel 2) von Nutzen, wo die Quelldaten in der EAV-Darstellung vorlägen und erst zum Zeitpunkt der Anfrage integriert werden. Die online-materialisierte Integration ähnelt der virtuellen Integration darin, dass die Anfragebearbeitung in beiden Varianten von zentraler Bedeutung ist.

6. Evaluierung

In diesem Kapitel wird die in Kapitel 4 vorgestellte Technik evaluiert. Die Evaluation soll an zwei Beispielen erfolgen. Zum einen wird das laufende Beispiel dieser Arbeit verwendet. Anhand dessen wird die Integration einer relationalen Datenbank und eines XML-Dokuments demonstriert, wobei das integrierte Schema im Relationenmodell dargestellt werden soll. Im zweiten Beispiel liegt eine relationale Datenbank vor, die medizinische Daten gemäß des DICOM-Standards enthält. Diese sollen in ein XML-Dokument transformiert werden, wobei keine Integration vorgenommen wird.

Die Evaluierung bezieht sich auf Funktionalitäten und nicht auf Performance-Messungen, da der Zweck dieser Evaluierung nicht der Vergleich mit bestehenden, ähnlichen Ansätzen ist, sondern die Beurteilung eines neuen Ansatzes.

6.1. Laufendes Beispiel

Dem Beispiel, das in diesem Unterkapitel vorgestellt wird, liegt folgendes Szenario zugrunde: Eine Bibliothek verwaltet Informationen über Bücher, Leser, Mitarbeiter, Filialen und Ausleihen in einer relationalen Datenbank namens `rel_bib`. In einer XML-Datei namens `xml_bib.xml` werden zusätzliche Informationen gespeichert. Dazu gehört, dass jedem Leser ein Interessenprofil zugeordnet wird. Des Weiteren wird für alle Bücher ein schriftliches Urteil festgehalten und das Anschaffungsdatum vermerkt. Manche Angaben, wie die zu Titel, Autor und Standort, überschneiden sich hierbei mit den Angaben in der Datenbank `rel_bib`.

Die ursprünglichen lokalen Schemata werden aus Übersichtlichkeitsgründen nicht hier angegeben, sondern sind im Anhang zu finden: Das Schema der relationalen Datenbank `rel_bib` ist in Form eines UML-Diagramms in Abbildung B.1 (im Anhang) angegeben, das XML Schema zu `xml_bib` kann dem Listing B.1 entnommen werden (ebenfalls im Anhang).

Die Datenbank `rel_bib` kann durch den SQL-Dump `rel_bib.sql` rekonstruiert werden, der im Verzeichnis `/Evaluation/Laufendes_Beispiel/` auf der CD zu finden ist, die dieser Arbeit beiliegt. Dieser Dump enthält auch Beispieldaten. Im gleichen Verzeichnis ist auch die XML-Schema-Datei `xml_bib.xsd` zu finden sowie das XML-Dokument `xml_bib_beispiel.xml`, das Beispieldaten enthält.

Die Abbildung der Schemata auf das EAV-Modell wird aus Platz- und Übersichtlichkeitsgründen nicht in dieser Arbeit angegeben, sondern kann durch den SQL-Dump `eav.sql` (ebenfalls im gleichen Verzeichnis) rekonstruiert werden. Der Inhalt der rekonstruierten Datenbank wurde durch die Anwendung des Demonstrators auf die beiden Quellschemata

erzeugt.

Im Folgenden wird dargestellt, wie die **Entitytypen** der beiden Quellschemata integriert werden. In Listing B.2 (im Anhang) ist das XML-Dokument angegeben, das die Zusicherungen zwischen Entitytypen und Pfaden beschreibt und dem Demonstrator als Eingabe dient (siehe 5.2.2). Um die Übersicht zu erhöhen, werden die aus der Abbildung der relationalen Datenbank `rel_bib` entstandenen Entitytypen und Attribute orange eingefärbt, während Entitytypen und Attribute, die aus der Abbildung des XML Schemas `xml_bib` entstanden sind, blau eingefärbt werden. Die Entitytypen und Attribute, die sich aus der Integration ergeben, sind schwarz dargestellt.

Der Entitytyp `Buch` wird mit dem Entitytyp `Buch` integriert. Die Attribute `ISBN` und `ISBN` sowie `Titel` und `Titel` sind als äquivalent gekennzeichnet, sodass jeweils eins davon in den integrierten Entitytyp übernommen wird. Das Attribut `Verlag` „überdeckt“ das Attribut `Verlag`, da es eine minimale Kardinalität von 1 besitzt, während `Verlag` eine minimale Kardinalität von 0 aufweist, sodass `Verlag` übernommen wird. Die Attribute `Autor` und `Autor` sind als überlappend gekennzeichnet, d.h. ihre Werte können übereinstimmen, dies muss jedoch nicht immer der Fall sein. Es wird ein Attribut `Autor` mit einer Kardinalität von 1:2 erzeugt. Zwischen den Attributen Note und Urteil besteht ein semantischer Zusammenhang, jedoch haben sie unterschiedliche Datentypen (`integer` vs. `string`). Sie werden zu dem Attribut `Bewertung` integriert, dessen Datentyp der Vereinigung der beiden Datentypen entspricht ({`integer`, `string`}). Die Attribute `Seitenanzahl` und `Anschaffungsdatum` werden unverändert übernommen, da es keine Korrespondenz für sie gibt.

Die Entitytypen `Filiale` und `Standort` werden zu dem Entitytypen `Niederlassung` integriert. Die Attribute `Name` und `Name` sind als äquivalent gekennzeichnet und werden zu dem Attribut `Name` integriert. Die Attribute Adresse und Öffnungszeiten werden unverändert übernommen.

Die Entitytypen `Leser` und `Leser` werden zu `Leser` integriert, wobei die Attribute `Leserausweisnummer` und `Lesernummer` zum Attribut `Leserausweisnummer` integriert werden. Die restlichen Attribute von `Leser` entstammen dem Entitytyp `Leser` (`Vorname`, `Nachname`, `Geburtsdatum`, `Geburtsort`, `Adresse`).

Die Entitytypen `Mitarbeiter`, `Ausleihe` und `Interessenprofil` werden mit all ihren wertbasierten Attributen in das integrierte Schema übernommen.

Schließlich müssen **Pfade** (Verkettungen von Referenzattributen) integriert werden:

- Die Pfade Buch↔Filiale und Buch↔`Standort` werden zu dem Pfad `Buch`↔`Niederlassung` integriert.
- Der Pfad `Mitarbeiter`↔Filiale wird als `Mitarbeiter`↔`Niederlassung` übernommen.
- Die Pfade `Ausleihe`↔`Buch` und `Ausleihe`↔`Leser` werden jeweils als `Ausleihe`↔`Buch` und `Ausleihe`↔`Leser` übernommen.

- Der Pfad Leser↔Interessenprofil wird als Leser↔Interessenprofil übernommen.

In Abbildung B.2 (im Anhang) ist das relationale Datenbankschema als UML-Diagramm angegeben, das als Ergebnis der Integration der Schemata `rel_bib` und `xml_bib` und der anschließenden **Rückabbildung** durch den Demonstrator erzeugt wird. Die Relationenschemata und Fremdschlüsselbeziehungen sollen an dieser Stelle näher betrachtet werden.

Das Relationenschema `Buch` hat die Attribute `ISBN: varchar(50)`, `Titel: varchar(50)`, `Verlag: varchar(50)`, `Seitenanzahl: int` und `Filialenname: varchar(50)`.

Da das Attribut `Autor` des integrierten Entitytyps `Buch` eine Kardinalität von 1:2 hat, wurde ein zusätzliches Relationenschema `Autor_Buch` erzeugt, das die Werte dieses Attributs aufnimmt. Es hat die Attribute `Autor: varchar(50)`, mit dem der Attributwert erfasst wird, und `ID: int`, das einen künstlichen Primärschlüssel darstellt. Die Zuordnung zwischen einem Tupel des Relationenschemas `Buch` und dem entsprechenden Tupel in `Autor_Buch` wird über ein drittes Relationenschema namens `Buch_to_Autor` vorgenommen. Dieses dritte Relationenschema enthält die Primärschlüssel der Schemata `Buch` und `Autor_Buch` und referenziert diese durch eine Fremdschlüsselbedingung.

Analog dazu werden die Werte des Attributs `Bewertung` im Relationenschema `Bewertung_Buch` gespeichert und die Zuordnung über das Relationenschema `Buch_to_Bewertung` vorgenommen. Eine Besonderheit besteht darin, dass das Attribut `Bewertung` zwei Datentypen hat (`integer` und `string`), sodass im Relationenschema `Bewertung_Buch` auch zwei Attribute (`Bewertung0: varchar(50)` und `Bewertung1: int`) vorhanden sind, um beide Werte des Attributs aufnehmen zu können.

Es besteht die Fremdschlüsselbeziehung `Filialenname`(`Buch`) → `Niederlassung_Name` (`Niederlassung`).

Das Relationenschema `Interessenprofil` entstammt der XML-Quelle `xml_bib`. Da kein Schlüsselattribut bekannt ist, wird ein künstlicher Primärschlüssel in Form des Attributs `ID: int` eingeführt. Darüber hinaus hat dieses Schema die Attribute `aktiviert: tinyint` (drückt Wahrheitswerte aus) und `Benachrichtigungsintervall: int`. Für die Attribute `Gattung` und `Schlagwort`, die eine Kardinalität von 0:n aufweisen, werden die Relationenschemata `Gattung_Interessenprofil` und `Interessenprofil_to_Gattung` sowie `Schlagwort_Interessenprofil` und `Interessenprofil_to_Schlagwort` erzeugt (siehe oben).

Das Relationenschema `Leser` entspricht dem gleichnamigen Relationenschema des relationalen Quelldatenbankschemas `rel_bib`. Der einzige Unterschied besteht in dem zusätzlichen Attribut `InteressenprofilID`, mit dem durch die Fremdschlüsselbedingung `InteressenprofilID`(`Leser`) → `ID`(`Interessenprofil`) auf das Relationenschema `Interessenprofil` verwiesen wird.

Die Relationenschemata `Mitarbeiter` und `Ausleihe` sind gegenüber dem Datenbankschema `rel_bib` unverändert.

Das relationale Schema `rel_bib` und das XML Schema `xml_bib` wurden damit erfolgreich integriert und in das Relationenmodell rückabgebildet. Ein Nutzer, der bislang nur mit der Datenbank `rel_bib` gearbeitet hat, kann nun Anfragen an die integrierte Datenbank stellen, mit der nun auch diejenigen Daten ermittelt werden können, die bisher nur in der XML-Quelle enthalten waren. Denkbar wären z.B. die Anfragen, die in den Listings 6.1, 6.2 und 6.3 angegeben sind.

Die Anfrage in Listing 6.1 ermittelt alle Büchergattungen, für die sich ein gegebener Leser laut seinem Interessenprofil interessiert. Mit der Anfrage in Listing 6.2 werden die Namen, Öffnungszeiten und Adressen aller Bibliotheksniederlassungen ausgegeben, die in den letzten zehn Tagen neue Bücher angeschafft haben (`DATEDIFF` ist eine MySQL-spezifische Funktion, die die Differenz zwischen zwei Datumsangaben in Tagen ermittelt). In Listing 6.3 ist eine Anfrage angegeben, mit der für alle aktiven Profile die ISBN derjenigen Bücher ermittelt wird, deren Genre mit einem der im Interessenprofil angegebenen Genres übereinstimmt und die angeschafft wurden, nachdem der Leser das letzte Mal über neue Anschaffungen benachrichtigt wurde. Diese Anfrage ist etwas umständlich, da die Werte des ursprünglich mengenwertigen Attributs `Genre` des Entitytyps `Interessenprofil` in einem eigenen Relationenschema ausgelagert werden mussten, das mit dem Relationenschema `Interessenprofil` durch ein weiteres Relationenschema verbunden ist. Somit ist ein Verbund über drei Relationenschemata notwendig, um die Werte für dieses Attribut zu ermitteln.

```
1 SELECT
2 Gattung
3 FROM
4 Gattung_Interessenprofil, Leser
5 WHERE
6 ID = InteressenprofilID AND
7 Leserausweisnummer=337
```

Listing 6.1: 1. SQL-Beispielanfrage

```
1 SELECT DISTINCT
2 Niederlassung_Name, Öffnungszeiten, Adresse
3 FROM
4 Niederlassung, Buch
5 WHERE
6 DATEDIFF(NOW(), Anschaffungsdatum) < 10 AND
7 Filialenname=Niederlassung_Name
```

Listing 6.2: 2. SQL-Beispielanfrage

```
 1 SELECT
 2 Interessenprofil.ID, ISBN
 3 FROM
 4 Interessenprofil, Interessenprofil_to_Genre,
 5 Genre_Interessenprofil, Buch
 6 WHERE
 7 Interessenprofil.ID=Interessenprofil_to_Genre.InteressenprofilID AND
 8 Genre_Interessenprofil.ID=Interessenprofil_to_Genre.GenreID AND
 9 Genre_Interessenprofil.Genre=Buch.Genre AND
10 aktiviert=1 AND
11 Buch.Anschaffungsdatum > Interessenprofil.Letzte_Benachrichtigung
```

Listing 6.3: 3. SQL-Beispielanfrage

6.2. Beispiel aus der Medizin

Dieses Beispiel nutzt als Quelle das Schema einer relationalen Datenbank, die von einem DICOM-Server namens *Conquest*[1] genutzt wird. Die Abkürzung DICOM steht für *Digital Imaging and Communications in Medicine* und bezeichnet einen Standard, der die Speicherung und den Austausch von medizinischen Bilddaten beschreibt[2]. Ein DICOM-Server stellt eine Implementierung dieses Standards dar.

Das Schema der Conquest-Datenbank ist als UML-Diagramm in Abbildung B.3 (im Anhang) angegeben. Auffällig ist, dass keine Fremdschlüsselbeziehungen definiert werden, obwohl solche bestehen. So bezieht sich beispielsweise das Attribut `PatientID` des Relationenschemas `DICOMStudies` auf den gleichnamigen Primärschlüssel des Relationenschemas `DICOMPatients`[3] Im Folgenden soll kurz auf die wichtigsten Relationen eingegangen werden.

In der Relation `DICOMPatients` werden Angaben zu Patienten verwaltet. Dazu gehören der Name des Patienten, das Geburtsdatum und das Geschlecht. Einem Patienten können mehrere Studien zugeordnet sein. Eine Studie bezieht sich dabei auf eine Untersuchung des Patienten. Informationen zu Studien werden in der Relation `DICOMStudies` gespeichert, wobei u.a. der Zeitpunkt der Studie, ein Verweis auf den behandelnden Arzt und eine Beschreibung vermerkt werden. Mit einer Studie können wiederum mehrere Serien assoziiert sein. Eine Serie umfasst mehrere Bilder, die dasselbe Körperteil in der gleichen Orientierung zeigen und dieselbe Modalität aufweisen, d.h. mit demselben bildgebenden Verfahren aufgenommen worden sind (z.B. Röntgen, Computertomographie oder Ultraschall). Serien werden unter Angabe u.a. der Modalität, des untersuchten Körperteils und der Orientierung in der Relation `DICOMSeries` gespeichert, während Bilder in der Relation `DICOMImages` gespeichert werden, wo u.a. der Bildaufnahmezeitpunkt und der Pfad zur Bilddatei erfasst werden.

[1] `http://ingenium.home.xs4all.nl/dicom.html`
[2] `http://medical.nema.org/standard.html`
[3] Diese Beobachtung ergibt sich aus den Beispieldatensätzen, die vom Conquest DICOM Server bei der Initialisierung des Datenbankschemas angelegt werden, und aus Daten, die angelegt werden, wenn dem Server eine neue Bilddatei zur Speicherung übergeben wird.

Die Conquest-Datenbank (einschließlich Beispieldatensätze) kann aus dem SQL-Dump `conquest.sql` rekonstruiert werden, der im Verzeichnis `/Evaluation/Medizinisches_Beispiel/` enthalten ist. Im selben Verzeichnis ist die Datei `conquest.xsd` enthalten, die das XML Schema enthält, das als Ergebnis der Transformation der relationalen Conquest-Datenbank entsteht, sowie die Datei `conquest_beispiel.xml`, die Beispieldatensätze enthält. Aus Platz- und Übersichtlichkeitsgründen werden die beiden Schemata nicht in ihrer Gesamtheit betrachtet. Stattdessen wird exemplarisch das Relationenschema `DICOMPatients` betrachtet. Dieses hat die Attribute `PatientID: varchar(64)`, `PatientNam: varchar(64)`, `PatientBir: char(8)`, `PatientSex: varchar(16)`, `AccessTime: int`, `qTimeStamp: int`, `qFlags: int`, `qSpare: varchar(64)`. In Listing 6.4 ist die Abbildung dieses Relationenschemas auf einen komplexen XML-Typ namens `DICOMPatients_Type` angegeben. Die Deklaration des dazugehörigen XML-Elements `DICOMPatients` erfolgt lokal innerhalb der Deklaration des Wurzelements `conquest` (siehe Listing 6.5). Auf diese Weise werden auch alle anderen XML-Elemente deklariert, auf die die Schemata der Datenbank `conquest` abgebildet werden, um nur genau ein Wurzelement zuzulassen. Innerhalb dieses Wurzelelements werden im XML-Dokument die Tupel aller Relationen der Datenbank `conquest` geschachtelt.

Das Attribut `PatientID` stellte in der relationalen Quelle den Primärschlüssel für das Relationenschema `DICOMPatients` dar. Die Information, dass dieses Attribut eindeutig identifizierend ist, wurde in der EAV-Darstellung vermerkt (in der Relation `KeyAttributes`, siehe 4.2). Bei der Rückabbildung wurde dieses Attribut daher mit dem `xs:key`-Element als Schlüssel ausgezeichnet (siehe Listing 6.4). Wären in der relationalen Quelle Fremdschlüsselbeziehungen definiert, so würden diese mit dem `xs:keyref`-Element unter Bezugnahme auf den Schlüssel eines Elements umgesetzt.

In Listing 6.6 ist ein einfacher XPath-Ausdruck angegeben, mit dem alle Bilderserien ermittelt werden, die Computertomographie als Modalität aufweisen. Listing 6.7 enthält eine XQuery-Anfrage, mit der für alle Bilder die Angaben zu Datum, Uhrzeit und Gerät erfragt werden. Eine geschachtelte XQuery-Anfrage ist in Listing 6.8 dargestellt. Mit dieser Anfrage werden zu jedem Patienten alle Studien unter Angabe des Datums, des behandelnden Arztes und der Beschreibung ermittelt.

```
<xs:complexType name="DICOMPatients_Type">
 <xs:attribute name="qFlags" use="optional" type="xs:integer"/>
 <xs:attribute name="AccessTime" use="optional" type="xs:integer"/>
 <xs:attribute name="PatientSex" use="optional" type="xs:string"/>
 <xs:attribute name="PatientBir" use="optional" type="xs:string"/>
 <xs:attribute name="qTimeStamp" use="optional" type="xs:integer"/>
 <xs:attribute name="qSpare" use="optional" type="xs:string"/>
 <xs:attribute name="PatientNam" use="optional" type="xs:string"/>
 <xs:attribute name="PatientID" use="required" type="xs:string"/>
 <xs:key name="DICOMPatients_Key">
  <xs:selector xpath="/conquest/DICOMPatients"/>
  <xs:field xpath="PatientID"/>
 </xs:key>
</xs:complexType>
```

Listing 6.4: Das Relationenschema `DICOMPatients` als komplexer XML-Typ

```
<xs:element name="conquest">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="DICOMImages" type="DICOMImages_Type"
       minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="DICOMWorkList" type="DICOMWorkList_Type"
       minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="UIDMODS" type="UIDMODS_Type" minOccurs="0"
       maxOccurs="unbounded"/>
   <xs:element name="DICOMPatients" type="DICOMPatients_Type"
       minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="DICOMStudies" type="DICOMStudies_Type"
       minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="DICOMSeries" type="DICOMSeries_Type"
       minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 6.5: Deklaration des Wurzelements

```
/conquest/DICOMStudies[@Modality="CT"]
```

Listing 6.6: XPath-Beispiel

```
for $image in /conquest/DICOMImages
return
 <image>
  <date>{$image/string(@ImageDate)}</date>
  <time>{$image/string(@ImageTime)}</time>
  <device_name>{$image/string(@DeviceName)}</device_name>
 </image>
```

Listing 6.7: 1. XQuery-Beispielanfrage

```
1 for $patient in /conquest/DICOMPatients
2 return
3  <patientStudies>
4   <patient name="{$patient/string(@PatientNam)}">
5
6   for $patient_study in /conquest/DICOMStudies[@PatientID =
        $patient/string(@PatientID)]
7   return
8    <study date="{$patient_study/string(@StudyDate)}"
         physician="{$patient_study/string(@ReferPhysi)}">
9     {$patient_study/string(@StudyDescr)}
10  </study>
11
12  </patient>
13 </patientStudies>
```

Listing 6.8: 2. XQuery-Beispielanfrage

6.3. Zusammenfassung

In diesem Kapitel wurde die in Kapitel 4 beschriebene Technik auf zwei Beispiele angewendet. Mit dem ersten Beispiel wurde die Integration eines relationalen und eines XML Schemas sowie die anschließende Rückabbildung des integrierten Schemas auf eine relationale Datenbank geschildert. Es wurden Beispiele für SQL-Anfragen vorgestellt, mit denen Informationen abgerufen werden, die der XML-Quelle entstammen und dem Nutzer der relationalen Quelldatenbank vor der Integration nicht zur Verfügung standen. Das zweite Beispiel diente dazu, die Transformation einer relationalen Datenbank in XML-Konstrukte zu veranschaulichen (ohne Integration). Die relationale Quelle ist ein Datenbankschema, das von einem DICOM-Server verwendet wird, um Daten gemäß des DICOM-Standards zu verwalten. Es wurden ein XPath-Ausdruck und zwei XQuery-Anfragen angegeben, mit denen das aus der Transformation entstandene XML-Dokument angefragt werden kann.

7. Zusammenfassung und Ausblick

In diesem Kapitel wird die Arbeit zusammengefasst und ein Ausblick auf Themen für mögliche Folgearbeiten gegeben.

Am Anfang dieser Arbeit wurden in Kapitel 2 die Grundlagen der Datenintegration vorgestellt, wobei die zentralen Schritte der Datenintegration und die Anfrageverarbeitung behandelt wurden. Dabei wird von einem globalen Schema ausgegangen, auf das die Schemata der lokalen Quellen abgebildet werden und an das die Anfragen im integrierten System gestellt werden. Im selben Kapitel wurden Methoden des Schema Matching vorgestellt, die eingesetzt werden, um Korrespondenzen zwischen Schemata zu finden. Damit wurde zur Abgrenzung ein Thema betrachtet, das nicht zu den Themenfeldern dieser Arbeit gehört. Die für die Arbeit relevanten Themen wurden in Kapitel 3 behandelt, wozu Schema Mappings und ihre Inversen sowie das Entity-Attribute-Value-Modell (EAV) gehören.

Das Konzept der Technik, die in dieser Arbeit entwickelt werden sollte, wurde in Kapitel 4 vorgestellt. Es wurde zunächst der Unterschied zwischen der Datenintegration, wie sie in Kapitel 2 dargelegt wurde, und dem Integrationskonzept dieser Arbeit diskutiert. Das Integrationskonzept dieser Arbeit basiert auf der Annahme, dass es für die Nutzer eines integrierten Systems von Vorteil ist, wenn sie ihre Anfragen weiterhin gegen das lokale Schema stellen können, mit dem sie bisher gearbeitet haben, und nicht gegen das globale Schema. Diese Art der Datenintegration muss dann nicht nur Abbildungen von den lokalen Schemata in das globale Schema definieren, sondern auch Rückabbildungen für die umgekehrte Richtung. Zunächst wurde betrachtet, wie Abbildungen und Rückabbildungen mit Schema Mappings und ihren Inversen spezifiziert werden können. Dabei wurde ein Problem mit Nullwerten im globalen Schema festgestellt. Dieses Problem motivierte die Verwendung des EAV-Modells. Es wurde versucht, die Abbildung auf das EAV-Modell mit Schema Mappings zu erfassen, was jedoch nicht zufriedenstellend gelang. Im nächsten Schritt wurde ein erweitertes EAV-Modell vorgestellt und Abbildungen aus dem Relationenmodell und XML in dieses Modell entwickelt. Darauf aufbauend wurde auf Grundlage der zusicherungsbasierten Schemaintegration geschildert, wie Elemente aus beiden Datenmodellen nach der Abbildung auf das EAV-Modell integriert werden können. Anschließend wurden die Rückabbildungen aus dem EAV-Modell in die lokalen Datenmodelle vorgestellt.

In Kapitel 5 wurde die Implementierung des Demonstrators beschrieben, der zur Umsetzung des Konzepts aus Kapitel 4 entwickelt wurde. Es wurde eine Client-Server-Anwendung umgesetzt, die eine lokale Integration von Schemata und Materialisierung von integrierten Schemata gestattet. Diese Anwendung wurde in Kapitel 6 anhand von zwei Beispielen evaluiert. Das erste Beispiel orientiert sich am laufenden Beispiel dieser

Arbeit und demonstriert, wie ein relationales und ein XML Schema integriert werden, wie das integrierte Schema auf eine relationale Datenbank rückabgebildet wird und wie Anfragen an diese Datenbank gestellt werden können. Das zweite Beispiel hat eine relationale Datenbank als Quelle, die medizinische Daten im DICOM-Standard verwaltet. Es wird geschildert, wie diese Datenbank in ein XML-Dokument transformiert wird, dessen Inhalt mit XPath-Ausdrücken und XQuery-Anfragen ausgewertet werden kann.

Eine an diese Arbeit anknüpfende Folgearbeit könnte sich damit beschäftigen, das in Kapitel 4 vorgestellte Konzept mit virtueller Integration umzusetzen. Hierzu müssten insbesondere Techniken der Anfrageverarbeitung untersucht wurden. Bei der „konventionellen“ Datenintegration werden alle Anfragen an das globale Schema gestellt. Dort besteht die Aufgabe darin, diese Anfragen in Anfragen an die lokalen Schemata umzuschreiben. Für die Global-as-local-view-extension-Technik müssten hingegen Anfragen an das erweiterte lokale Schema in Anfragen an das generische EAV-Schema umgeschrieben werden, die wiederum in Anfragen an die lokalen Quellschemata umgeschrieben werden müssten. Die Anfrageverarbeitung wäre in diese Fall komplexer als bei der „konventionellen“ Datenintegration, da bei beiden Schritten sichergestellt werden muss, dass die ursprüngliche Anfrage und die umgeschriebene Anfrage äquivalent sind.

Weitere Arbeiten könnten sich mit einem Integrationsverfahren befassen, das das lokale Schema bei der Integration stärker in den Vordergrund rückt, sodass das integrierte Schema dem lokalen Schema ähnelt. In diesem Zusammenhang spielt auch die Auflösung von Konflikten eine Rolle: Diese sollten zugunsten des lokalen Schemas entschieden werden.

Es könnte untersucht werden, wie sich Korrespondenzen zwischen Schemata in der EAV-Darstellung automatisiert bzw. semi-automatisiert finden lassen. Da die Datenmodellheterogenität im EAV-Modell überwunden ist, ließen sich somit auch Zusammenhänge zwischen Schemata finden, die in unterschiedlichen Datenmodellen vorliegen.

Des Weiteren können Abbildungen und Rückabbildungen für weitere Datenmodelle definiert werden, wie z.B. das objektorientierte Datenmodell. Dazu könnte es ggf. erforderlich sein, das EAV-Modell zu erweitern. Das EAV-Modell könnte auch um weitere Konzepte wie Integritätsbedingungen erweitert werden.

A. Anhang zum Konzept

A.1. ER-Diagramm der EAV-Datenbank

In Abbildung A.1 ist ein UML-Diagramm dargestellt, das aus der relationalen Umsetzung des erweiterten EAV-Modells entstand, das in 4.2 vorgestellt wurde. Es soll vor allem der Übersicht über die Relationen dienen, die an der Umsetzung des EAV-Modells beteiligt sind.

Primärschlüsselattribute sind über dem Strich angegeben. Nicht-identifizierende Beziehungen sind durch eine gestrichelte Linie und identifizierende Beziehungen durch eine durchgängig gezeichnete Linie dargestellt.

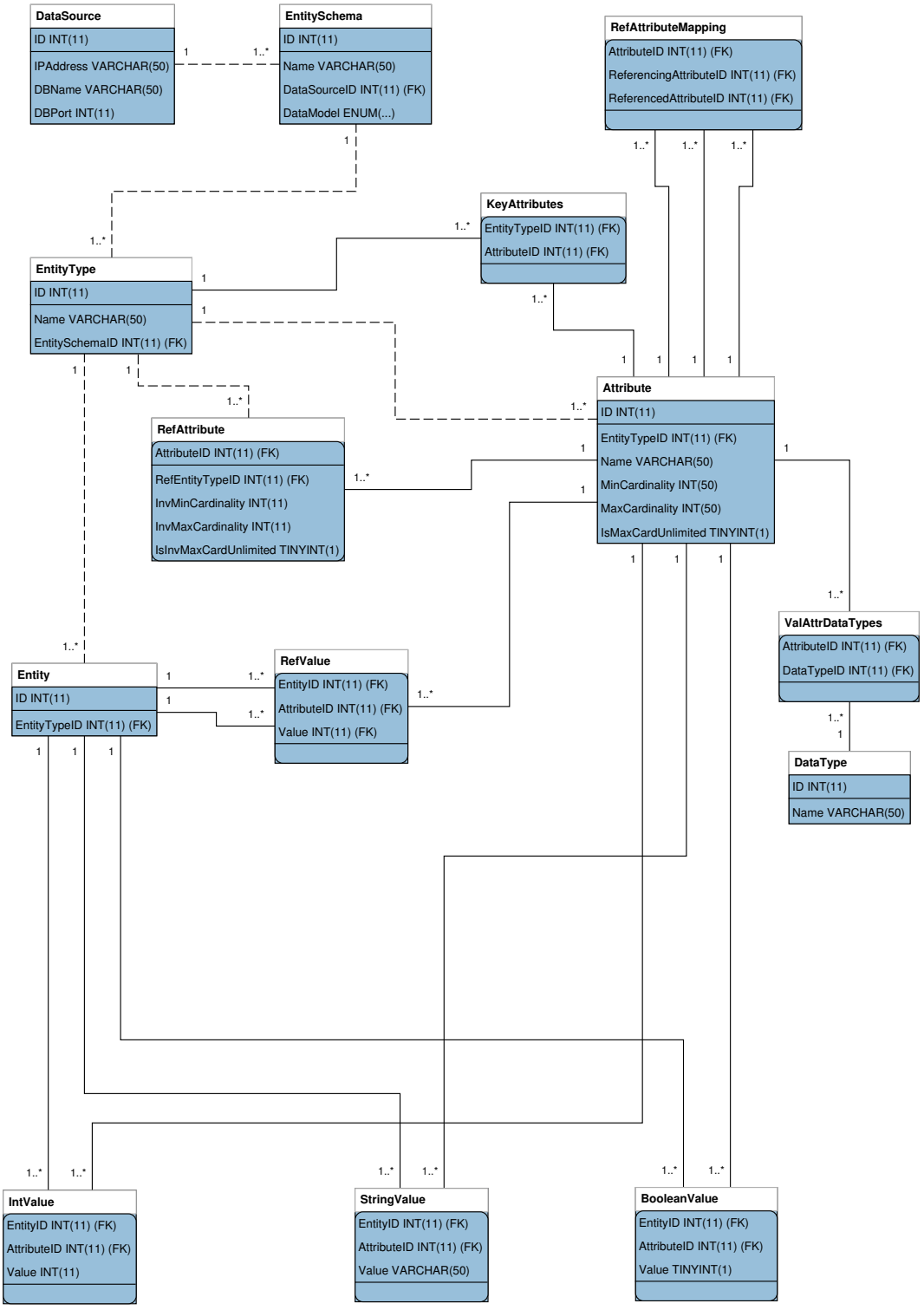

Abbildung A.1.: UML-Diagramm der EAV-Datenbank

A.2. Beispiel für Abbildung des Relationenmodells auf das EAV-Modell

Im Folgenden soll anhand eines Beispiels die Abbildung des Relationenmodells auf das EAV-Modell veranschaulicht werden. Sei dazu $\texttt{SHOP} = \{\texttt{Buch}, \texttt{Kunde}, \texttt{Bestellung}\}$ ein Datenbankschema, wobei für die darin enthaltenen Relationen gilt:

- `Buch(ISBN, Titel, Autor, Preis, gebraucht)`
- `Kunde(Kundennummer, Name, Adresse)`
- `Bestellung(Kundennummer, ISBN, Versandart, Bestelldatum)`

Zusätzlich sollen die Fremdschlüsselbedingungen

- $\texttt{Kundennummer}(\texttt{Bestellung}) \rightarrow \texttt{Kundennummer}(\texttt{Kunde})$
- $\texttt{ISBN}(\texttt{Bestellung}) \rightarrow \texttt{ISBN}(\texttt{Buch})$

gelten. Eine Beispielinstanz zu diesem Datenbankschema ist in Tabelle A.1 dargestellt. Die Abbildung des Datenbankschemas auf Entitytypen und Attribute im EAV-Modell ist in Tabelle A.5 angegeben. Daraus wird ersichtlich, dass nur für die Attribute `gebraucht` und `Versandart` jeweils der Relationen `Buch` und `Bestellung` Nullwerte zulässig sind (gekennzeichnet durch den Wert 0 für das Attribut `MinCard` der EAV-Relation `Attribute`).

In Tabelle A.4 sind die Entities aufgeführt, die sich aus den Tupeln der Beispielrelationen aus Tabelle A.1 ergeben. Die Attributwerte dieser Relationen sind in den EAV-Relationen in Tabelle A.6 gespeichert. Dabei werden Nullwerte (in den Beispielrelationen durch das Symbol $\perp$ gekennzeichnet) nicht erfasst. Im konkreten Beispiel wird für die Entity γ_6 und das Attribut β_{11} kein Wert in der EAV-Relation `StringValue` gespeichert, da für das Tupel t in der Relation `Bestellung` mit den Werten $t(\texttt{Kundennummer}) = 294$ und $t(\texttt{ISBN}) = 978 - 3 - 942656 - 27 - 6$ ein Nullwert für das Attribut `Versandart` gegeben ist. Für die Entity γ_2 und das Attribut β_5 ist in der EAV-Relation `BooleanValue` ebenfalls kein Wert verzeichnet, da das Tupel t' in der Relation `Buch` mit $t'(ISBN) = 978 - 3 - 942656 - 27 - 6$ für das Attribut `gebraucht` einen Nullwert aufweist.

In den Tabellen A.2 und A.3 ist angegeben, wie jeweils Primärschlüsselattribute und Fremdschlüsselbeziehungen erfasst werden.

Buch				
ISBN	Titel	Autor	Preis	gebraucht
3-426-60765-4	Contact	Carl Sagan	14,90	false
978-3-942656-27-6	Der Fremde	Albert Camus	7,95	$\perp$

Kunde		
Kundennummer	Name	Adresse
532	Erika Mustermann	Heidestraße 17, 51147 Köln
294	Max Mustermann	Borkweg 7, 28219 Bremen

Bestellung			
Kundennummer	ISBN	Versandart	Bestelldatum
532	3-426-60765-4	Express	14.05.2012
294	978-3-942656-27-6	$\perp$	25.07.2012

Tabelle A.1.: Beispielinstanz für das Datenbankschema SHOP

KeyAttributes	
EntityTypeID	AttributeID
α_1	β_1
α_2	β_6
α_3	β_9
α_3	β_{10}

Tabelle A.2.: Erfassung der Primärschlüsselattribute

RefAttributeMapping		
AttributeID	ReferencingAttributeID	ReferencedAttributeID
β_{13}	β_9	β_6
β_{14}	β_{10}	β_1

Tabelle A.3.: Erfassung von Fremdschlüsselbedingungen

Entity	
ID	EntityTypeID
γ_1	α_1
γ_2	α_1
γ_3	α_2
γ_4	α_2
γ_5	α_3
γ_6	α_3

Tabelle A.4.: Abbildung von Tupel der Beispielrelationen als Entitymengen

EntityType			
ID	Name	DataModel	DataSource
α_1	Buch	relational	MySQL@192.168.10.1:3306
α_2	Kunde	relational	MySQL@192.168.10.1:3306
α_3	Bestellung	relational	MySQL@192.168.10.1:3306

Attribute				
ID	EntityTypeID	Name	MinCard	MaxCard
β_1	α_1	ISBN	1	1
β_2	α_1	Titel	1	1
β_3	α_1	Autor	1	1
β_4	α_1	Preis	1	1
β_5	α_1	gebraucht	0	1
β_6	α_2	Kundennummer	1	1
β_7	α_2	Name	1	1
β_8	α_2	Adresse	1	1
β_9	α_3	Kundennummer	1	1
β_{10}	α_3	ISBN	1	1
β_{11}	α_3	Versandart	0	1
β_{12}	α_3	Bestelldatum	1	1
β_{13}	α_3	Kunde	1	1
β_{14}	α_3	Buch	1	1

ValAttrDataType	
AttributeID	DataTypeID
β_1	δ_2
β_2	δ_2
β_3	δ_2
β_4	δ_5
β_5	δ_4
β_6	δ_3
β_7	δ_2
β_8	δ_2
β_9	δ_3
β_{10}	δ_2
β_{11}	δ_2
β_{12}	δ_1

DataType	
ID	Name
δ_1	date
δ_2	string
δ_3	integer
δ_4	boolean
δ_5	decimal

RefAttribute			
AttributeID	RefEntityTypeID	InvMinCard	InvMaxCard
β_{13}	α_2	0	n
β_{14}	α_1	0	n

Tabelle A.5.: Abbildung der Relationenschemata aus SHOP

StringValue		
EntityID	AttributeID	Value
γ_1	β_1	3-426-60765-4
γ_1	β_2	Contact
γ_1	β_3	Carl Sagan
γ_2	β_1	978-3-942656-27-6
γ_2	β_2	Der Fremde
γ_2	β_3	Albert Camus
γ_3	β_7	Erika Mustermann
γ_3	β_8	Heidestraße 17, 51147 Köln
γ_4	β_7	Max Mustermann
γ_4	β_8	Borkweg 7, 28219 Bremen
γ_5	β_{10}	3-426-60765-4
γ_5	β_{11}	Express
γ_6	β_{10}	978-3-942656-27-6

IntValue		
EntityID	AttributeID	Value
γ_3	β_6	532
γ_5	β_9	532
γ_4	β_6	294
γ_6	β_9	294

DecValue		
EntityID	AttributeID	Value
γ_1	β_4	14,90
γ_2	β_4	7,95

BooleanValue		
EntityID	AttributeID	Value
γ_1	β_5	false

DateValue		
EntityID	AttributeID	Value
γ_5	β_{12}	14.05.2012
γ_6	β_{12}	25.07.2012

RefValue		
EntityID	AttributeID	Value
γ_5	β_{13}	γ_3
γ_5	β_{14}	γ_1
γ_6	β_{13}	γ_4
γ_6	β_{14}	γ_2

Tabelle A.6.: Abbildung der Relationen der Beispielinstanz zum Datenbankschema SHOP

A.3. Abbildung von XML auf das EAV-Modell

Attribute				
ID	EntityTypeID	Name	MinCard	MaxCard
β_1	α_1	ISBN	1	1
β_2	α_1	Titel	1	1
β_3	α_1	Autor	1	1
β_4	α_1	Verlag	0	1
β_5	α_1	Anschaffungsdatum	0	1
β_6	α_1	Standort	1	1
β_7	α_1	Urteil	1	1
β_8	α_2	Lesernummer	1	1
β_9	α_2	Interessenprofil	1	1
β_{10}	α_3	Name	1	1
β_{11}	α_3	Öffnungszeiten	1	1
β_{12}	α_4	Schlagwort	1	n
β_{13}	α_4	Gattung	1	n
β_{14}	α_4	Benachrichtigungsintervall	1	1
β_{15}	α_4	deaktiviert	1	1
β_{16}	α_5	Buch	0	n
β_{17}	α_5	Leser	0	n
β_{18}	α_5	Standort	0	n
β_{19}	α_5	Interessenprofil	0	n

ValAttrDataType	
AttributeID	DataTypeID
β_1	δ_2
β_2	δ_2
β_3	δ_2
β_4	δ_2
β_5	δ_1
β_7	δ_2
β_8	δ_3
β_{10}	δ_2
β_{11}	δ_2
β_{12}	δ_2
β_{13}	δ_2
β_{14}	δ_3
β_{15}	δ_4

DataType	
ID	Name
δ_1	date
δ_2	string
δ_3	integer
δ_4	boolean

RefAttribute			
AttributeID	RefEntityTypeID	InvMinCard	InvMaxCard
β_5	α_4	0	n
β_8	α_5	0	n
β_{16}	α_1	1	1
β_{17}	α_2	1	1
β_{18}	α_3	0	1
β_{19}	α_4	0	1

Tabelle A.7.: Abbildung von XML-Attributen und Schachtelung auf EAV-Attribute

Standort ↦ Entity-IDs		
Name	Öffnungszeiten	EntityID
Südstadtbibliothek	Mo.-Fr., 08:00-20:00	γ_1

Interessenprofil ↦ Entity-IDs				
Schlagwort	Gattung	Benachrichtigungsintervall	aktiviert	EntityID
Programmierung, Java	Fachbuch	10	true	γ_2
Computernetzwerke	Fachbuch	20	false	γ_3

Buch ↦ Entity-IDs						
ISBN	Titel	...	Verlag	Anschaffungsdatum	Standort	EntityID
978-3-8362-1506-0	Java ist auch eine Insel	...	Galileo Press	$\perp$	γ_1	γ_4
3-86894-137-1	Computernetzwerke	...	$\perp$	15.08.2012	γ_1	γ_5

Leser ↦ Entity-IDs		
Lesernummer	Interessenprofil	EntityID
1	γ_2	γ_6
2	γ_3	γ_7

Bibliothek ↦ Entity-IDs				
Buch	Leser	Standort	Interessenprofil	EntityID
γ_4, γ_5	γ_6, γ_7	$\perp$	$\perp$	γ_8

Tabelle A.8.: Zuordnung von Werten von wertbasierten Attributen und Referenzattributen zu Entity-IDs

Entity	
ID	EntityTypeID
γ_1	α_3
γ_2	α_4
γ_3	α_4
γ_4	α_1
γ_5	α_1
γ_6	α_2
γ_7	α_2
γ_8	α_5

Tabelle A.9.: Abbildung eines XML-Dokumentes als Entitymenge

EntityType			
ID	Name	DataModel	DataSource
α_1	Buch	XML	BaseX@192.168.10.2:1984
α_2	Leser	XML	BaseX@192.168.10.2:1984
α_3	Standort	XML	BaseX@192.168.10.2:1984
α_4	Interessenprofil	XML	BaseX@192.168.10.2:1984
α_5	Bibliothek	XML	BaseX@192.168.10.2:1984

Tabelle A.10.: Abbildung von XML-Elementen auf Entitytypen

StringValue		
`EntityID`	`AttributeID`	`Value`
γ_1	β_{10}	Südstadtbibliothek
γ_1	β_{11}	Mo.-Fr., 08:00-20:00
γ_2	β_{12}	Fachbuch
γ_2	β_{13}	Programmierung
γ_2	β_{13}	Java
γ_3	β_{12}	Fachbuch
γ_3	β_{13}	Computernetzwerke
γ_4	β_1	978-3-8362-1506-0
γ_4	β_2	Java ist auch eine Insel
γ_4	β_3	Christian Ullenboom
γ_4	β_4	Galileo Press
γ_4	β_7	umfangreiche Einführung
γ_5	β_1	3-86894-137-1
γ_5	β_2	Computernetzwerke
γ_5	β_3	Andrew Tanenbaum
γ_5	β_7	verständliches Grundlagenwerk

IntValue		
`EntityID`	`AttributeID`	`Value`
γ_2	β_{14}	10
γ_3	β_{14}	20
γ_6	β_8	1
γ_7	β_8	2

BooleanValue		
`EntityID`	`AttributeID`	`Value`
γ_2	β_{15}	true
γ_3	β_{15}	false

DateValue		
`EntityID`	`AttributeID`	`Value`
γ_5	β_5	15.08.2012

RefValue		
`EntityID`	`AttributeID`	`Value`
γ_4	β_6	γ_1
γ_5	β_6	γ_1
γ_6	β_9	γ_2
γ_7	β_9	γ_3
γ_8	β_{16}	γ_4
γ_8	β_{16}	γ_5
γ_8	β_{17}	γ_6
γ_8	β_{17}	γ_7

Tabelle A.11.: Speicherung von Attributwerten und Elementinhalten

```
<xs:element name="Buch">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Standort"/>
      <xs:element ref="Urteil"/>
    </xs:sequence>
    <xs:attribute name="ISBN" type="xs:string" use="required"/>
    <xs:attribute name="Titel" type="xs:string" use="required"/>
    <xs:attribute name="Autor" type="xs:string" use="required"/>
    <xs:attribute name="Verlag" type="xs:string" use="optional"/>
    <xs:attribute name="Anschaffungsdatum" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="Leser">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Interessenprofil"/>
    </xs:sequence>
    <xs:attribute name="Lesernummer" type="xs:integer"
        use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="Standort">
  <xs:complexType>
    <xs:attribute name="Name" type="xs:string" use="required"/>
    <xs:attribute name="Öffnungszeiten" type="xs:string"
        use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="Interessenprofil">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Schlagwort" minOccurs="1"
          maxOccurs="unbounded"/>
      <xs:element ref="Gattung" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Benachrichtigungsintervall" type="xs:integer"
        use="required"/>
    <xs:attribute name="aktiviert" type="xs:boolean" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="Bibliothek">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Buch" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Leser" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Standort" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Interessenprofil" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing A.1: Elemente mit komplexem Typ

B. Anhang zur Evaluierung

B.1. Laufendes Beispiel

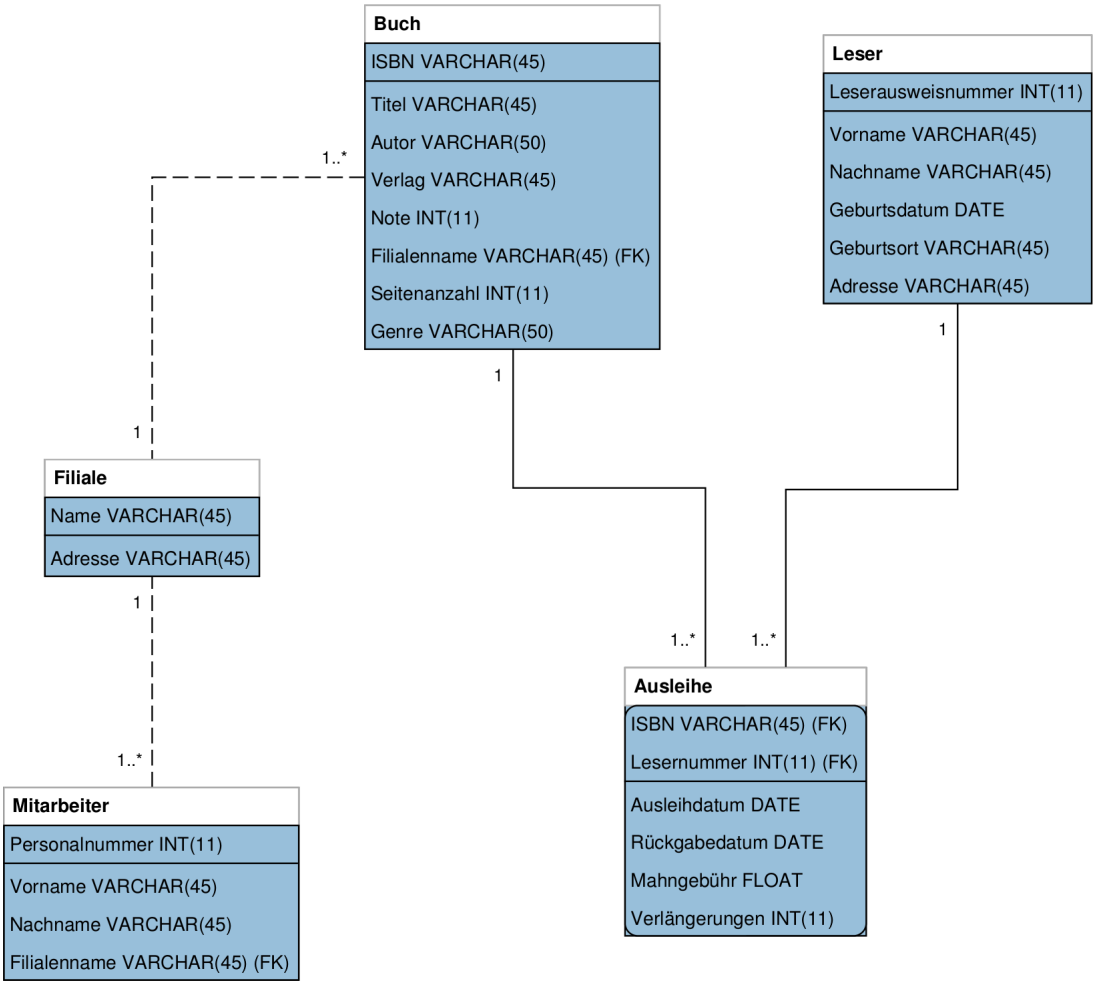

Abbildung B.1.: UML-Diagramm des relationalen Datenbankschemas `rel_bib` (Primärschlüsselattribute sind über dem Strich angegeben. Nicht-identifizierende Beziehungen sind durch eine gestrichelte Linie und identifizierende Beziehungen durch eine durchgängig gezeichnete Linie dargestellt.)

```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
2   <xs:element name="Buch">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element ref="Standort"/>
6         <xs:element name="Urteil" type="xs:string"/>
7       </xs:sequence>
8       <xs:attribute name="ISBN" type="xs:string" use="required"/>
9       <xs:attribute name="Titel" type="xs:string" use="required"/>
10      <xs:attribute name="Autor" type="xs:string" use="required"/>
11      <xs:attribute name="Verlag" type="xs:string" use="optional"/>
12      <xs:attribute name="Anschaffungsdatum" use="required"/>
13    </xs:complexType>
14  </xs:element>
15  <xs:element name="Leser">
16    <xs:complexType>
17      <xs:sequence><xs:element ref="Interessenprofil"/></xs:sequence>
18      <xs:attribute name="Lesernummer" type="xs:integer"
            use="required"/>
19    </xs:complexType>
20  </xs:element>
21  <xs:element name="Standort">
22    <xs:complexType>
23      <xs:attribute name="Name" type="xs:string" use="required"/>
24      <xs:attribute name="Oeffnungszeiten" type="xs:string"
            use="required"/>
25    </xs:complexType>
26  </xs:element>
27  <xs:element name="Interessenprofil">
28    <xs:complexType>
29      <xs:sequence>
30        <xs:element name="Schlagwort" type="xs:string" minOccurs="1"
              maxOccurs="unbounded"/>
31        <xs:element name="Genre" type="xs:string" minOccurs="1"
              maxOccurs="unbounded"/>
32      </xs:sequence>
33      <xs:attribute name="Letzte_Benachrichtigung" type="xs:date"
            use="required"/>
34      <xs:attribute name="aktiviert" type="xs:boolean"
            use="required"/>
35    </xs:complexType>
36  </xs:element>
37 </xs:schema>
```

Listing B.1: XML Schema `xml_bib`

```
1 <schemaCorrespondence remoteSchemaName="bibliothek_xml"
      remoteIP="127.0.0.1" remoteDBPort="1984"
      integratedSchemaName="Bibliothek_Ext">
2     <entityTypeCorrespondence localEntityType="Buch"
          remoteEntityType="Buch" integratedEntityType="Buch">
3         <attrCorr localAttr="ISBN" remoteAttr="ISBN"
              type="EQUIVALENCE" key="true" integratedAttr="ISBN"/>
4         <attrCorr localAttr="Autor" remoteAttr="Autor" type="OVERLAP"
              integratedAttr="Autor"/>
5         <attrCorr localAttr="Titel" remoteAttr="Titel"
              type="EQUIVALENCE" integratedAttr="Titel"/>
6         <attrCorr localAttr="Note" remoteAttr="Urteil"
              type="INTERRELATEDNESS" integratedAttr="Bewertung"/>
7         <attrCorr localAttr="Verlag" remoteAttr="Verlag"
              type="INCLUSION" superSetAttr="localAttr"
              integratedAttr="Verlag"/>
8     </entityTypeCorrespondence>
9
10    <entityTypeCorrespondence localEntityType="Leser"
          remoteEntityType="Leser" integratedEntityType="Leser">
11        <attrCorr localAttr="Leserausweisnummer"
              remoteAttr="Lesernummer" type="EQUIVALENCE" key="true"
              integratedAttr="Leserausweisnummer"/>
12    </entityTypeCorrespondence>
13
14    <entityTypeCorrespondence localEntityType="Filiale"
          remoteEntityType="Standort"
          integratedEntityType="Niederlassung">
15        <attrCorr localAttr="Name" remoteAttr="Name"
              type="EQUIVALENCE" key="true"
              integratedAttr="Niederlassung_Name"/>
16    </entityTypeCorrespondence>
17
18  <pathCorrespondence>
19    <localPath>
20      <link leftHandEntityType="Buch" rightHandEntityType="Filiale"
            refAttribute="Filiale"/>
21    </localPath>
22    <remotePath>
23      <link leftHandEntityType="Buch" rightHandEntityType="Standort"
            refAttribute="Standort"/>
24    </remotePath>
25  </pathCorrespondence>
26
27 </schemaCorrespondence>
```

Listing B.2: Spezifikation von Zusicherungen

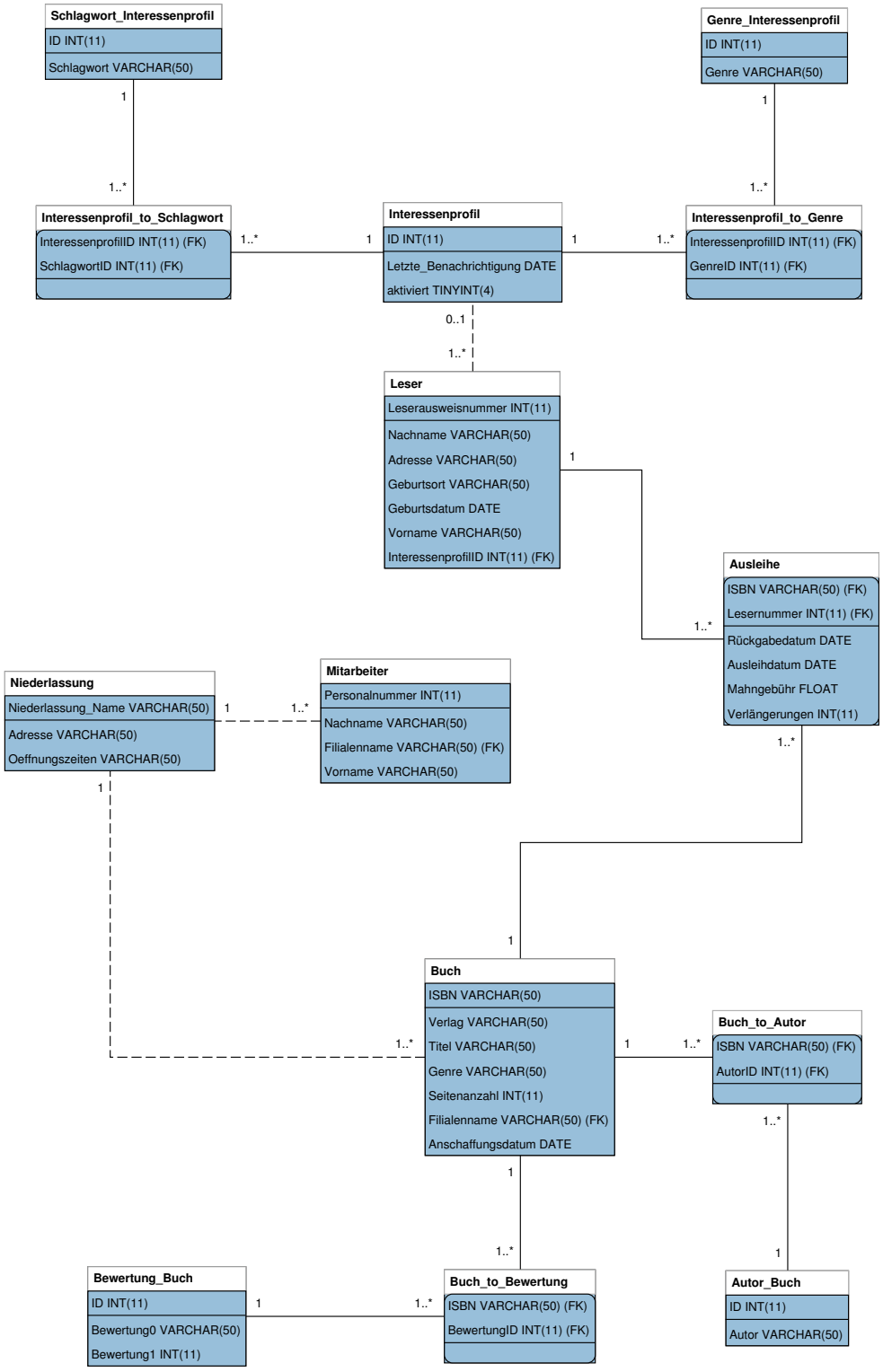

Abbildung B.2.: UML-Diagramm des Datenbankschemas, das aus der Integration der Schemata `rel_bib` und `xml_bib` und der anschließenden Rückabbildung entsteht

B.2. Beispiel aus der Medizin

Abbildung B.3.: UML-Diagramm des Datenbankschemas, das vom DICOM-Server Conquest zur Speicherung der medizinischen Bilddaten genutzt wird

Literaturverzeichnis

[ABLM10] Arenas, Marcelo ; Barcelo, Pablo ; Libkin, Leonid ; Murlak, Filip: Relational and XML Data Exchange. In: *Synthesis Lectures on Data Management* 2 (2010), Nr. 1, S. 1–112

[APR09] Arenas, Marcelo ; Pérez, Jorge ; Riveros, Cristian: The recovery of a schema mapping: bringing exchanged data back. In: *ACM Transactions on Database Systems (TODS)* 34 (2009), Nr. 4, S. 22

[Ber03] Bernstein, Philip A.: Applying model management to classical meta data problems CIDR, 2003

[BH08] Bernstein, Philip A. ; Haas, Laura M.: Information integration in the enterprise. In: *Commun. ACM* 51 (2008), September, Nr. 9, 72–79. `http://dx.doi.org/10.1145/1378727.1378745`. – DOI 10.1145/1378727.1378745. – ISSN 0001–0782

[CIS] *Cisco Visual Networking Index: Forecast and Methodology, 2012–2017*. `www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`, Abruf: 11.07.2013

[CK10] Cate, Balder ten ; Kolaitis, Phokion G.: Structural characterizations of schema-mapping languages. In: *Commun. ACM* 53 (2010), Januar, Nr. 1, 101–110. `http://dx.doi.org/10.1145/1629175.1629201`. – DOI 10.1145/1629175.1629201. – ISSN 0001–0782

[Con97] Conrad, Stefan: *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer DE, 1997

[DAB+06] Dolin, Robert H. ; Alschuler, Liora ; Boyer, Sandy ; Beebe, Calvin ; Behlen, Fred M. ; Biron, Paul V. ; Shvo, Amnon S.: HL7 clinical document architecture, release 2. In: *Journal of the American Medical Informatics Association* 13 (2006), Nr. 1, S. 30–39

[Dat03] Date, CJ: What First Normal Form Really Means. In: *Date on Database: Writings 2000-2006* (2003), S. 127–128

[DN07] Dinu, Valentin ; Nadkarni, Prakash: Guidelines for the effective use of entity–attribute–value modeling for biomedical databases. In: *International journal of medical informatics* 76 (2007), Nr. 11, S. 769–779

[DWR10] Duftschmid, Georg ; Wrba, Thomas ; Rinner, Christoph: Extraction of standardized archetyped data from Electronic Health Record systems based

on the Entity-Attribute-Value Model. In: *International journal of medical informatics* 79 (2010), Nr. 8, S. 585–597

[Fag07] Fagin, Ronald: Inverting schema mappings. In: *ACM Transactions on Database Systems (TODS)* 32 (2007), Nr. 4, S. 25

[FHLM96] Flach, Guntram ; Heuer, Andreas ; Langer, Uwe ; Meyer, Holger: Transparente Anfragen in föderativen Datenbanksystemen. In: *Workshop „Föderierte Datenbanken“: Kurzfassungen der Beiträge, Institutsbericht, Nummer ITI-96-01.* Universität Magdeburg, 1996, S. 45–49

[FKMP05] Fagin, Ronald ; Kolaitis, Phokion G. ; Miller, Renée J ; Popa, Lucian: Data exchange: semantics and query answering. In: *Theoretical Computer Science* 336 (2005), Nr. 1, S. 89–124

[FKPT05] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang-Chiew: Composing schema mappings: Second-order dependencies to the rescue. In: *ACM Transactions on Database Systems (TODS)* 30 (2005), Nr. 4, S. 994–1055

[FKPT08] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang-Chiew: Quasi-inverses of schema mappings. In: *ACM Transactions on Database Systems (TODS)* 33 (2008), Nr. 2, S. 11

[Haa07] Haas, Laura: Beauty and the beast: The theory and practice of information integration. In: *In ICDT*, 2007

[Hal01] Halevy, Alon Y.: Answering queries using views: A survey. In: *The VLDB Journal* 10 (2001), Dezember, Nr. 4, 270–294. `http://dx.doi.org/10.1007/s007780100054`. – DOI 10.1007/s007780100054. – ISSN 1066–8888

[Heu97] Heuer, Andreas: *Objektorientierte Datenbanken.* Bd. 2. Addison-Wesley, 1997

[Heu13] Heuer, Andreas: *Theorie relationaler Datenbanken.* Vorlesung an der Universität Rostock, Sommersemester 2013

[HM13] Heuer, Andreas ; Meyer, Holger: *Objektorientierte Datenbanken und XML-Datenbanken.* Vorlesung an der Universität Rostock, Wintersemester 2012/2013

[IDC] *IDC Predicts 2012 Will Be the Year of Mobile and Cloud Platform Wars as IT Vendors Vie for Leadership While the Industry Redefines Itself.* `http://www.businesswire.com/news/home/20111201005201/en/IDC-Predicts-2012-Year-Mobile-Cloud-Platform`, Abruf: 11.07.2013

[KM03] Klettke, Meike ; Meyer, Holger: *XML & Datenbanken.* dpunkt, 2003

[LKBH13] Löper, Dortje ; Klettke, Meike ; Bruder, Ilvio ; Heuer, Andreas: Enabling flexible integration of healthcare information using the entity-attribute-

value storage model. In: *Health Information Science and Systems* 1 (2013), Nr. 1, S. 9

[LN07] Leser, Ulf ; Naumann, Felix: *Informationsintegration - Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. dpunkt.verlag, 2007. – I–XIII, 1–464 S.

[Mai83] Maier, David: *The theory of relational databases*. Bd. 11. Computer science press Rockville, 1983

[McC04] McCallum, Sally H.: An introduction to the metadata object description schema (MODS). In: *Library Hi Tech* 22 (2004), Nr. 1, S. 82–88

[MRB03] Melnik, Sergey ; Rahm, Erhard ; Bernstein, Philip A.: Developing metadata-intensive applications with Rondo. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 1 (2003), Nr. 1, S. 47–74

[NMC+99] Nadkarni, Prakash M. ; Marenco, Luis ; Chen, Roland ; Skoufos, Emmanouil ; Shepherd, Gordon ; Miller, Perry: Organization of heterogeneous scientific data using the EAV/CR representation. In: *Journal of the American Medical Informatics Association* 6 (1999), Nr. 6, S. 478–493

[RB01] Rahm, Erhard ; Bernstein, Philip A.: A survey of approaches to automatic schema matching. In: *The VLDB Journal* 10 (2001), Dezember, Nr. 4, 334–350. http://dx.doi.org/10.1007/s007780100057. – DOI 10.1007/s007780100057. – ISSN 1066–8888

[SOA] *Soapbox: Why XML Schema beats DTDs hands-down for data*. http://www.ibm.com/developerworks/library/x-sbsch/index.html, Abruf: 29.08.2013

[SP94] Spaccapietra, Stefano ; Parent, Christine: View integration: A step forward in solving structural conflicts. In: *Knowledge and Data Engineering, IEEE Transactions on* 6 (1994), Nr. 2, S. 258–274

[SPD92] Spaccapietra, Stefano ; Parent, Christine ; Dupont, Yann: Model independent assertions for integration of heterogeneous schemas. In: *The VLDB Journal* 1 (1992), Nr. 1, S. 81–126

[SS95] Schmitt, Ingo ; Saake, Gunter: Managing object identity in federated database systems. In: *OOER'95: Object-Oriented and Entity-Relationship Modeling*. Springer, 1995, S. 400–411

[SSH10] Saake, Gunter ; Sattler, Kai-Uwe ; Heuer, Andreas: *Datenbanken: Konzepte und Sprachen*. Hüthig Jehle Rehm, 2010

[W3Ca] *Extensible Markup Language (XML) 1.1 - Document Type Definition*. http://www.w3.org/TR/2004/REC-xml11-20040204/#NT-doctypedecl, Abruf: 29.08.2013

[W3Cb] *XML Schema Recommendation*. `http://www.w3.org/standards/techs/xmlschema#w3c_all`, Abruf: 29.08.2013

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel verfasst habe.

Rostock, den 29.10.2013

Georgi Straube