Hochschule Aalen

Projektbericht
Studiengang : Informatik

Erzeugung einer Software zur Modellierung von relationalen Datenbanken

von

Simon Ruttmann

80751

Betreuender Professor: Prof. Dr. Gregor Grambow

Einreichungsdatum : 14. August 2022

Eidesstattliche Erklärung

Hiermit erkläre ich, **Simon Ruttmann**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum Unterschrift (Student)

Kurzfassung

Bei der Verwaltung von Daten nehmen relationale Datenbanksysteme bis heute eine marktdominierende Stellung ein[12, S. 73ff.]. Hierbei ist die Konzeption und Entwicklung von Datenbankmodellen ein essentieller Bestandteil, welcher über einen mehrstufigen Entwurfsprozess realisiert wird.

Die frühe Datenbankentwurfsphase wurde entscheidend mit der Einführung des Entity-Relationship Modells, kurz ER-Modell, geprägt und bildet bis heute den Standard in der Konzeption von relationalen Datenbanksystemen. Dieses Modell ermöglicht die grafische Modellierung von Daten, ihrer Struktur sowie ihrer Zusammenhänge auf abstrakter, konzeptioneller und implementierungsunabhängiger Ebene.

Im Rahmen eines weiterführenden logischen Entwurfs kann auf Basis des ER-Modells eine Überführung in ein relationales Datenmodell erfolgen. Dieses Modell stellt die Grundlage von Schemadefinitionen für relationale Datenbankmanagementsysteme, kurz RDBMS.

Für die Überführung des konzeptionellen Modells in das relationale Modell gibt es eine Reihe von Methoden und Praktiken, welche manuell vom Entwickler des Datenmodells durchgeführt werden. Bei zunehmend größer werdenden Modellen steigt hierbei auch der Aufwand zur Überführung des Modells.

Aufgrund dessen wird im Rahmen dieser Arbeit eine Anwendung entwickelt, welche dem Benutzer bei der Modellierung von relationalen Datenbanken unterstützt. Hierfür wird eine Web-Applikation entwickelt, die dem Entwickler ermöglicht Entity-Relationship Modelle mittels eines Zeichentools zu modellieren.

Darüber hinaus werden Algorithmen entwickelt und in der Anwendung implementiert, welche auf Basis des vom Benutzer erstellten ER-Diagramms ein relationales Modell ermitteln. Die Software stellt dieses Modell daraufhin dem Benutzer grafisch bereit. Zudem ist die Anwendung fähig, mittels des generierten Relationenmodells Schemadefinitionen in Form von SQL für RDBMS zu erzeugen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

In der heutigen Zeit ist das Datenmanagment von Schlagworten wie "Big Data" und "NoSql" geprägt. Dennoch erfreuen sich bewährte relationale Datenbanken auch heute noch hoher Popularität. So besitzen sieben der zehn meist verwendeten Datenbanken im Jahr 2022 ein relationales Modell[6]. Ebenso wie die vier meist genutzten Datenbanken[6].

Die Modellierung von relationalen Datenbanken obliegt hierbei dem Entwickler und wird in der Regel über einen mehrstufigen Entwurfsprozess erstellt. In Abbildung 1.1 sind die wesentlichen Entwurfsphasen und damit verbundenen Artefakte dargestellt.

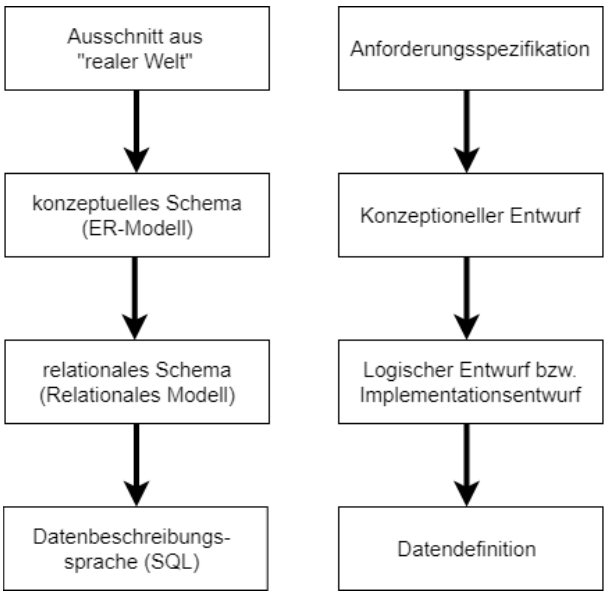

Abbildung 1.1: Artefakte und Phasen des DB-Entwurfs[12, S. 27][20, S. 124]

Wie in der Abbildung 1.1 zu sehen beginnt unmittelbar nach der grundlegenden Anforderunganalyse der konzeptionelle Entwurf. In diesem hat sich die Modellierung von sogenannten Entity-Relationship Modellen als Standard etabliert.

Mittels dieses Modells werden Objekte aus der realen Welt herangezogen und in ein grafisches Modell überführt, welches die Struktur der Objekte und die Beziehungen zwischen diesen auf abstrakter und implementierungsunabhängiger Ebene abbildet.

In relationalen Datenbanksystemen wird aufbauend auf diesem Modell ein relationales Modell erstellt. Dieses Modell bildet eine grafische Repräsentation eines Datenschemas, indem die Daten auf logischer Ebene in Form von Relationen dargestellt werden.

Für die Überführung des konzeptionellen Modells in das relationale Modell gibt es eine Reihe heuristischer Regeln, Methodiken und Praktiken, welche manuell vom Entwickler des Datenmodells durchgeführt werden.[20, S. 144] Bei zunehmend größer werdenden Modellen steigt hierbei auch der Aufwand zur Überführung des Modells.

Darüber hinaus fällt weiterer Aufwand für die Erstellung der Schemadefinitionen in einer Datenbeschreibungssprache, üblicherweise in der Structured Query Language kurz SQL an, um das relationale Modell dem relationalen Zieldatenbanksystem bekannt zu machen.

Der Aufwand ist aufgrund der heuristischen Regeln von wiederkehrender und repetitiver Natur. Dies lässt grundlegend vermuten, dass Teilschritte automatisiert gelöst werden können, um somit den Entwickler zu entlasten.

Der Beweggrund dieser Arbeit besteht daher darin, den Entwickler beim Durchlaufen des Entwurfsprozesses zu unterstützen, in dem repetitive Aufgaben dem Entwickler mittels einer Software abgenommen werden.

1.2 Problemstellung und -abgrenzung

Die Problemstellung geht unmittelbar aus der Motivation hervor. Die vorliegende Arbeit konzentriert sich auf die Konzeption und Entwicklung einer Anwendung zur Modellierung von relationalen Datenbanksystemen.

Kernsaspekte der Anwendung und dieser Arbeit sind dabei die Modellierung von Entity-Relationship Modellen und der darauf aufbauende Übersetzungsvorgang zwischen dem konzeptuellen Entity-Relationship Modell und dem relationalen Modell. Des Weiteren wird sich auf die Transformation des generierten relationalen Modells in eine Datenbeschreibunssprache fokussiert.

In beiden Überführungsvorgängen wird eine Problemanalyse durchgeführt und auf Basis dieser Lösungskonzepte und Algorithmen zur automatisierten Übersetzung erarbeitet.

1.3 Ziel der Arbeit

Ziel der Arbeit ist es eine Anwendung zu erzeugen, welche den Benutzer unterstützt Entity-Relationship Diagramme zu modellieren. Auf Basis des vom Benutzer gefertigten Diagramms soll, unter der Verwendung der in dieser Arbeit erstellten Algorithmen, eine Überführung in ein relationales Modell, ohne weiteres Eingreifen des Benutzers, erfolgen.

Des Weiteren soll in der Anwendung ein SQL-Generator entwickelt werden, welcher auf Basis des generierten Modells arbeitet und daraus Schemadefinitionen mittels SQL generiert.

1.4 Vorgehen

Die Erreichung des Ziels erfolgt innerhalb mehrerer Kapitel.

In einem ersten Schritt werden die für diese Arbeit notwendigen Grundlagen erarbeitet und vermittelt. Aufbauend darauf erfolgt eine Problem- und Anforderungsanalyse.

Basierend auf der Problem- und Anforderungsanalyse erfolgt innerhalb weiterer Kapitel die Erarbeitung von Lösungskonzepten. Diese beinhaltet konzeptionelle Ansätze zur Umsetzung der Anforderungen, sowie die Entwicklung einer Überführungsstrategie der Modelle.

Darauf folgend findet die Implementierung der Anwendung unter Verwendung der entstandenen Artefakte aus den vorhergehenden Kapiteln statt. Mit Abschluss der Implementierung erfolgt die Inbetriebnahme der Anwendung, sowie die Evaluierung der erreichten Ergebnisse.

2 Grundlagen

In diesem Kapitel wird das für diese Arbeit relevante Grundlagenwissen dargestellt. Dieses lässt sich in drei Bereiche unterteilen.

Im ersten Bereich werden die Grundlagen zur Modellierung von relationalen Datenbanksystemen vermittelt. Darauf folgend werden theoretische Kenntnisse dargestellt, welche innerhalb dieser Arbeit verwendet werden. Zuletzt werden die Technologien, welche zur Erstellung der Anwendung verwendet wurden, in ihren Grundzügen erörtert.

2.1 Grundlagen zur Modellierung von relationalen Datenbanksystemen

In der Einleitung wurden bereits die wesentlichen Entwurfsphasen und Artefakte dieser während der Modellierung von relationalen Systemen vorgestellt.

Im Folgenden wird detaillierter auf die entstehenden Artefakten eingegangen.

2.1.1 Konzeptionelles Datenmodell

Der erste konzeptionelle Entwurf hat das Ziel, aus einem relevanten Ausschnitt der realen Welt ein formales Abbild zu erstellen. Dieses wird mittels eines konzeptuellen Datenmodells beschrieben[24, S. 52].

Im Rahmen dieser Arbeit wird auf das Entity-Relationship Modell als konzeptuelles Modell näher eingegangen, da dieses die Grundlage für die vorliegende Arbeit bildet. Da eine vollständige Darstellung des Modells den Rahmen dieser Arbeit sprengen würde, wird im Folgenden nur auf die für diese Arbeit notwendigen grundlegenden Eigenschaften und Elemente eines ER-Modells eingegangen. Kenntnisse über spezifische ER-Elemente werden in darauf folgenden Kapiteln, sofern nötig, vermittelt.

Die Modellierung von Entity-Relationship Modellen basiert grundlegend auf Entitäten (eng. Entities) und Beziehungen (eng. Relationship)[12, S. 39ff]. Entitäten stellen hierbei Gegenstände der realen Welt dar, welche voneinander physisch oder gedanklich wohl unterscheidbar sind[12, S. 39]. Ähnliche Gegenstände werden zu sogenannten Entitätstypen zusammengefasst.

Um einen Zusammenhang zwischen zwei oder mehreren Entitätstypen auszudrücken, werden Beziehungstypen eingesetzt. Diese drückt eine Gruppe von Beziehungen zwischen Entitäten zweier Entitätstypen aus[12, S. 39]. Die Beziehungstypen können zusätzlich um eine sogenannte Funktionalität ergänzt werden. Diese gibt an, mit vielen Entitäten eine Entität in Zusammenhang steht. [12, S. 41f] Die Angabe der Funktionalität erfolgt unter der Verwendung einer Kardinalitätsangabe zwischen einer Entität und einer Beziehung. Die Angabe der Kardinalität kann mittels verschiedenen Notationen erfolgen[12, S. 46f]. Im Rahmen dieser Arbeit wird sich auf die Min-Max Notation beschränkt.

Die Funktionalität einer Beziehung lässt sich in "Eins zu Eins", "Eins zu Viele" und "Viele zu Viele" Kategorisieren[12, S. 42].

In der Abbildung 2.1 sind die Entitätstypen "Student" und "Vorlesung" dargestellt, welche mittels des Beziehungtyps "hören" in Zusammenhang stehen. Ein Student hört hierbei beliebig viele Vorlesungen. Eine Vorlesung wird von mindestens zehn und maximal 50 Studenten gehört.

Abbildung 2.1: Darstellung eines ER-Diagramms

Für die Darstellung der Merkmale und Eigenschaften von Entitäten werden in ER-Diagrammen Attribute eingesetzt. Diese werden als Ovale dargestellt und mit dem Entitätstyp verbunden. Attribute können ebenfalls an Beziehungstypen angehängt werden, um eine Eigenschaft einer Beziehung zwischen Entitäten auszudrücken[12, S. 42].

Eine Menge von Attributen mittels der eine Entität innerhalb eines Entitätstyp eindeutig identifiziert wird, wird Schlüssel genannt[12, S. 41]. Attribute innerhalb dieser Menge werden zusätzlich unterstrichen dargestellt.

In der Abbildung 2.2 wurde das in Abbildung 2.1 dargestellte ER-Diagramm mit Attributen ergänzt. Ein Student besitzt hierbei eine eindeutige Matrikelnummer. Eine ist durch eine Vorlesungsnummer eindeutig bestimmt.

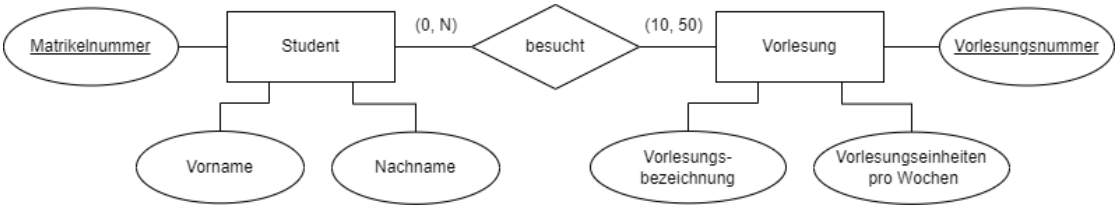

Abbildung 2.2: Erweiterung des ER-Diagramms mit Attributen

Im Rahmen dieser Arbeit wird zwischen Entitäten und Entitättypen, sowie Beziehungen und Beziehungstypen nicht weiter unterschieden. Diese Begriffe werden folgend Synonym eingesetzt und beziehen sich auf den Typ.

2.1.2 Relationales Datenmodell

Das relationale Datenmodell entsteht innerhalb eines logischen Entwurfs. Dieser verwendet in der Regel als Ausgangssituation das im konzeptionellen Entwurf entstandene ER-Modell[12, S. 75][12, S. 27]. Die Übersetzung vom ER-Modell zum relationalen Modell ist umfangreich und wird in Kapitel 6.1 adressiert. In diesem Kapitel wird auf die grundlegenden Eigenschaften des relationalen Modells eingegangen.

Das relationale Modell besteht aus einer Menge von Relationen beziehungsweise Tabellen[12, S. 75]. Die Spalten der Tabelle werden als Attribute oder Felder bezeichnet[12, S. 74]. Über diese werden Eigenschaften der darin enthaltenen Datensätze festgelegt. Jede Zeile in der Tabelle entspricht dabei einem Datensatz.

Jeder Datensatz benötigt darüber hinaus einen Primärschlüssel, der einen einzigartigen Wert in der Tabelle besitzt. Der Schlüssel kann sich hierbei auch über mehrere Spalten hinweg bilden. Um Datensätze miteinander zu verknüpfen, werden sogenannte Fremdschlüssel verwendet[12, S. 77]. Ein Fremdschlüssel wird hierbei in einer Relation hinterlegt und referenziert den Primärschlüssel einer anderen Relation.

In der Abbildung 2.3. ist eine konkrete Ausprägung des Sachverhalts aus Abbildung 2.2 im relationalen Modell abgebildet. Die Matrikelnummer bildet hierbei den Primärschlüssel der Relation Student. Die Vorlesungsnummer entspricht dem Primärschlüssel der Relation Vorlesung. Verknüpft werden die Datensätze mittels der Relation Besucht, welche jeweils einen Fremdschlüssel auf die anderen Relationen hält. Der Primärschlüssel der Besucht Relation ist hierbei die Matrikel- und Vorlesungsnummer.

Student		
Matrikelnummer	Vorname	Nachname
1234567	Hans	Müller
7654321	Georg	Schmidt
.....	.....	.....

Besucht	
Matrikelnummer	Vorlesungsnummer
1234567	0000001
7654321	0000001
.....	.....

Vorlesung		
Vorlesungs-nummer	Vorlesungs-bezeichnung	Vorlesungs-einheiten / Woche
0000001	Analysis	4
0000002	Rechnernetze	2
.....	.....	.....

Abbildung 2.3: Ausprägung einer Tabelle

Ist die konkrete Ausprägung der Relation nicht relevant, so kann dieses Modell wie in Abbildung 2.4 abgebildet werden.

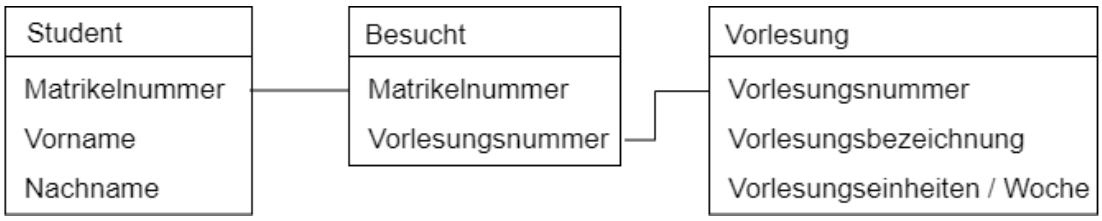

Abbildung 2.4: Relationenmodell

2.1.3 Structured Query Language

Bei der Structured Query Language handelt es sich um eine Datenbeschreibungs- und Datenmanipulationssprache. Mit dieser kann das Relationenmodell einem relationalen Zieldatenbanksystem bekannt gemacht werden. Darüber hinaus kann mittels SQL Datensätze vom Datenbanksystem abgefragt, neue Datensätze hinzugefügt oder bestehende Daten manipuliert werden.

In dieser Arbeit wird sich auf die Beschreibung des Relationenmodells beschränkt.

Hierfür wird für jede Relation ein sogenanntes Create-Table Statement erstellt. In jedem dieser Statements werden der Name der Tabelle, sowie dessen Spalten mit den dazugehörigen Datentypen angegeben. Zusätzlich wird angegeben, welche Attribute Teil des Primärschlüssels sind und welche Attribute als Fremdschlüssel auf eine weitere Tabelle verweisen.

Im Quelltext 2.1 ist der SQL-Code zur Erstellung der Tabellen Student und Besucht aus der Abbildung 2.4 dargestellt.

```
1  CREATE TABLE IF NOT EXISTS Student (
2    Matrikelnummer Integer,
3    Vorname        varchar(255),
4    Nachname       varchar(255),
5
6    PRIMARY KEY (Matrikelnummer)
7  );
8
9  CREATE TABLE IF NOT EXISTS Besucht (
10   Matrikelnummer    Integer,
11   Vorlesungsnummer  Integer,
12
13   PRIMARY KEY (Matrikelnummer, Vorlesungsnummer),
14
15   FOREIGN KEY (Matrikelnummer)
16     REFERENCES Student(Matrikelnummer),
17
18   FOREIGN KEY (Vorlesungsnummer)
19     REFERENCES Vorlesung(Vorlesungsnummer)
20 );
```

Quelltext 2.1: SQL-Statements zur Erstellung von Tabellen

2.2 Theoretische Grundlagen

Im Laufe der Arbeit wird gezeigt werden, dass ER-Diagramme in Graphen- und Baumstrukturen überführbar sind. Im Folgenden werden zwei Algorithmen vorgestellt, welche im späteren Verlauf Anwendung finden.

2.2.1 Traversieren von Bäumen

Um einen Baum zu durchlaufen gibt es eine Reihe von Algorithmen. Diese unterscheiden sich hierbei im Wesentlichen darin, zu welchem Zeitpunkt ein Knoten im Baum bearbeitet wird[19, S. 377ff].

Im Rahmen dieser Arbeit finden der sogenannte Preorder- und Postorder-Durchlauf Anwendung.

Der Preorder-Durchlauf beginnt an der Wurzel des Baumes. Dieser verarbeitet den aktuellen Knoten, traversiert den linken Knoten und im Anschluss den rechten Knoten ausgehend vom aktuellen Knoten[19, S. 378].

Der Preorder-Durchlauf beginnt ebenfalls an der Wurzel des Baumes. Dieser traversiert jedoch zuerst den linken Knoten, dann den rechten Knoten und danach verarbeitet dieser den aktuellen Knoten[19, S. 379].

In der Abbildung 2.5 sind der Ablauf des Preorder- und Postorder-Durchlaufs dargestellt.

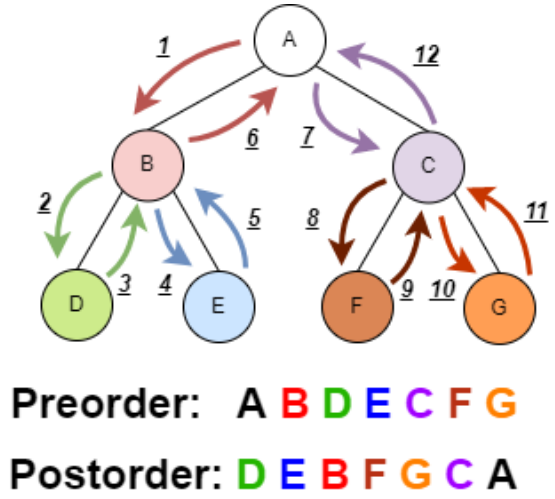

Abbildung 2.5: Traversal eines Baumes im Pre- und Postorder

2.2.2 Topologische Sortierung

Die topologische Sortierung sortiert die Knoten eines gerichteten Graphen so, dass jeder Knoten nach all seinen Vorgängern kommt[19, S. 490]. In anderen Worten, existieren keine Kanten von rechts nach links, wenn die Knoten nach der Sortierung von links nach rechts ausgegeben werden.

Um eine topologische Sortierung auszuführen, wird auf einen gerichteten Graphen zunächst eine Tiefensuche (eng. Depth-First-Search) durchgeführt[19, S. 486].

Hierbei wird jeder Knoten des Graphen zu Beginn als noch nicht besucht markiert. Anschließend wird für jeden noch nicht besuchten Knoten des Graphen ein Subalgorithmus ausgeführt.

Der Subalgorithmus markiert den aktuellen Knoten als aktiv und ermittelt jeden noch nicht besuchten Knoten, zu dem der aktuelle Knoten eine ausgehende Kante besitzt. Der Subalgorithmus wird für jeden dieser Knoten rekursiv ausgeführt. Nach einem Abschluss des Subalgorithmuses, wird der darin aktive Knoten als besucht markiert.

Wird bei dem Tiefensuche-Algorithmus dem aktiven Knoten im Subalgorithmus eine inkrementierende Abschlusszeit hinzugefügt, wenn der Knoten als bearbeitet markiert wird, so ergibt eine Sortierung der Knoten nach der Abschlusszeit eine topologische Sortierung[19, S. 490].

Ein minimaler Algorithmus zur Ermittlung der Abschlusszeiten ist im Quelltext 2.2 dargestellt.

```
1  Function Depth-First-Search(Graph)
2
3      ForEach Node in Graph
4          Node.status <- notVisisted
5      End For
6
7      For Each Node in Graph
8          If Node.status = notVisisted Then
9              DFS-Visit(Node)
10         End If
11     End For
12 End Function
13
14 Function DFS-Visit(Node)
15     Node.status = active
16
17     For Each OutgoingEdge in Node
18         DestinationNode <- OutgoingEdge.Node
19         If DesinationNode.status = notVisisted Then
20             DestinationNode.Predecessor <- Node
21             DFS-Visit(DestinationNode)
22         End If
23     End For
24
25     Node.status <- visited
26     Time++; Node.EndTime <- Time
27 End Function
```

Quelltext 2.2: Algorithmus zur Ermittlung der topologischen Sortierung

2.3 Technologien

Im Folgenden werden die wesentlichen im Projekt verwendeten Technologien und Frameworks kurz erläutert.

2.3.1 Spring und Springboot

Bei dem Spring[22] Framework handelt es sich um ein Programmier- und Konfigurationsmodell für Java-basierte Applikationen. Dieses bietet eine Reihe von Features wie zum Beispiel Dependency Injection, Data Binding, Testtools, Datenbankanbindungen und Typkonvertierung. Spring erleichtert zudem das Erstellen von Webanwendungen durch das Entfernen eines Großteils an Boilerplate-Code.

Spring Boot ermöglicht es, Spring-basierte Anwendungen mit minimalen Vorabkonfigurationen in einen lauffähigen Zustand zu bringen. Spring Boot ermöglicht zudem eine automatische Konfiguration von Drittanbierter-Bibliotheken.

2.3.2 React

React[16] ist eine deklarative JavaScript-Bibliothek zum Erstellen interaktiver Benutzeroberflächen. Die Entwicklung von Ansichten ist hierbei komponentenbasiert. Dies bedeutet, dass einzelne UI-Bereiche als gekapselte und wiederverwendbare Komponente realisiert werden, welche ihren eigenen Zustand verwalten. Für die Erstellung komplexer Benutzerschnittstellen können React-Komponenten beliebig komponiert werden.

React aktualisiert und rendert bei Zustandsänderungen alle betroffenen Komponenten. Es ist zudem möglich umfangreiche Datensätze durch die Komponenten zu leiten, wodurch der Zustand aus dem DOM herausgehalten werden kann.

Die Bibliothek kann in Browsern, Servern und mittels ReactNative in Mobile Apps verwendet werden.

2.3.3 Redux

Redux[18] ist ein vorhersagbarer Zustandscontainer für Javascript-Anwendungen.

Dieser Container ermöglicht es, den Zustand und die Logik zur Zustandsänderung einer Anwendung zu zentralisieren. Der zentralisierte Zustand wird hierbei Redux-Store genannt.

Änderung des Zustandes werden über sogenannte Aktionen beschrieben. Diese können pragmatisch als Events betrachtet werden. Die Logik zur Änderung des Zustandes für eine Aktion wird über sogenannte Reducer implementiert.

Redux verfügt zudem über ein großes Ökosystem an Add-Ons.

In dieser Arbeit wird Redux mit dem Redux-Toolkit verwendet. Dieses stellt eine Reihe von Hilfsfunktionen zur Verfügung, welche einen Großteil an Boiler-Code entfernen.

Zudem ermöglicht es das Toolkit sogenannte Slicer zu erstellen. Diese ermöglichen es den Gesamtzustand von Redux in mehrere kleinere Teile zu zerlegen. Ein Slice ist hierbei eine Sammlung von Redux-Reducern und den dazugehörigen Aktionen.

3 Problem- und Anforderungsanalyse

In diesem Kapitel werden Anforderungen an die zu entwickelnde Software definiert. Weitgehend werden Probleme aufgedeckt, welche zur Erstellung der Software behandelt werden müssen.

Das Gesamtproblem lässt sich hierbei in aufeinander aufbauende Teilprobleme zerlegen. Diese werden in den folgenden Unterkapiteln erläutert. Gesammelte Anforderungen werden nummeriert und in einer Übersicht am Ende des Kapitels stichpunktartig dargestellt.

3.1 Generelle Anforderungen an die Software

Da die Hauptmotivation dieser Arbeit darin besteht, einem Benutzer beim Durchlaufen des Entwurfsprozesses von relationalen Datenbanksystemen zu unterstützen und repetitive Aufgaben dem Entwickler abzunehmen, ergeben sich generelle Anforderungen an die Software.

Bei der Entwicklung der Software muss auf die einfache und intuitive Bedienbarkeit der Anwendung geachtet werden. Der Benutzer soll dabei interaktiv die Anwendung bedienen können, ohne weitere Kenntnisse über die Software, z.B. in Form einer Betriebsanleitung, besitzen zu müssen. (Nr. 1)

Die Anwendung richtet sich dabei grundsätzlich an jede Person, welche direkt am Entwurfsprozess beteiligt ist. Da die Anwendung daher möglichst einfach und ohne größeren Aufwand zugänglich gemacht werden soll, entsteht die Anforderung diese als Webanwendung zu realisieren. (Nr. 2)

Neben den oberen Anforderungen muss die entwickelte Software zudem möglichst generisch und modular aufgebaut werden, da diese potenziell im Rahmen weiterer aufbauender Arbeiten weiterentwickelt werden soll. (Nr. 3)

3.2 Anforderungen an das Zeichentool

Die im Rahmen dieser Arbeit entwickelte Anwendung soll den Benutzer unterstützen Entity-Relationship Modelle zu modellieren. Hierfür ist es notwendig ein Eingabesystem zu implementieren, mittels dem der Benutzer die ER-Modelle dem System bekannt machen kann.

Da es sich bei dem Entity-Relationship Modell um ein grafisches Modell handelt, liegt es daher nahe, die Modellierung ebenfalls über ein grafisches Zeichentool zu realisieren. (Nr. 4)

Das Zeichentool muss hierbei die Erstellung, das Löschen und die Benennung von ER-Elementen erlauben.(Nr. 5) Zusätzlich muss ermöglicht werden die Elemente zu verbinden. Je nach Art der Verbindung muss es ebenso ermöglicht werden, den Verbindungen Kardinalitäten zuzuordnen. (Nr. 6)

Da der Benutzer bereits während der Modellierung unterstützt werden soll, muss das Zeichentool interaktiv auf die Benutzereingaben reagieren. Des Weiteren ist darauf zu achten, das Tool möglichst benutzerfreundlich zu entwickeln. (Nr. 7)

Darüber hinaus muss es ermöglicht werden modellierte ER-Diagramme abzuspeichern, sodass diese zu einem späteren Zeitpunkt weiter bearbeitet werden können. (Nr. 8)

3.3 Notwendigkeit zur Festlegung von Regeln des Entity-Relationship Modells

Das ER-Modell selbst hat seine Wurzeln in der Veröffentlichung von Peter Chen "The Entity-Relationship Model - Toward a Unified View of Data"[14] im Jahre 1976. Seither erhielt dieses Modell zahlreiche Ergänzungen, Erweiterungen und Veränderungen. Dabei bildete sich eine Reihe von verschiedenen Varianten des ER-Modells. Stark erweiterte und ausgeprägte Varianten bilden hierbei zum Beispiel das ECR Modell [4] oder das EER Modell [23]. Doch auch innerhalb des traditionellen ER-Modells treten verschiedene Ausprägungen auf[24, S. 88ff.].

Um Algorithmen zur Überführung des ER-Modells auf das relationale Modell zu entwickeln, ist es allerdings zwingend notwendig, dass Randbedingungen und Rahmenbedingungen bekannt sind. Aufgrund dessen ist das in dieser Arbeit zu verwendende ER-Modell eindeutig zu definieren.

Hierfür muss innerhalb der Entwicklung eines Lösungskonzepts eine Auswahl von zu unterstützenden ER-Konstrukten festgelegt werden und darauf aufbauend eindeutige Regeln für alle ER-Elemente definiert werden.

Dabei ist zu beachten, dass die unterstützende Software dem Entwickler größtmögliche Freiheiten bei der Modellierung einräumt. Die Festlegung der Regeln soll somit die Modellierung nur soweit einschränken, wie dies zwingend für eine eindeutige Zuordnung von Elementen notwendig ist. (Nr. 9)

3.4 Validierungs- und Feedbackanforderungen auf Basis der benutzergenerierten Entity-Relationship Modelle

In der vorhergehenden Sektion wurde festgestellt, dass eine Festlegung von Regeln für das Entity-Relationship Modell erfolgen muss.

Da die Anwendung auf Basis von benutzergenerierten Modellen arbeitet, ist eine Einhaltung der Regeln nicht sichergestellt. Die Applikation muss daher fähig sein, benutzergenerierte Modelle auf die Einhaltung der Regeln zu evaluieren. (Nr. 10)

In den generellen Anforderungen an die Software wurde zudem festgelegt, dass die Software benutzerfreundlich und ohne weitere Vorkenntnisse bedienbar sein soll. Aufgrund dessen kann nicht davon ausgegangen werden, dass dem Benutzer die in dieser Arbeit festgelegten Regeln bekannt sind.

Hierdurch entsteht die Anforderung, dass die Software dem Benutzer bei der Modellierung des Entity-Relationship Diagramms Feedback gegeben werden muss. Das Feedback soll hierbei dem Benutzer präzise auf Falscheingaben hinweisen um ihn aktiv bei der Modellierung unterstützen. Dies impliziert, dass die Software resistent gegenüber Falscheingaben des Benutzers ist. (Nr. 11)

3.5 Überführung des Entity-Relationship Modells in das relationale Modell

Ziel der Software ist eine möglichst hohe Entlastung des Entwicklers. Daher ist eine Überführung eines validen ER-Modells ohne weiteres Einwirken des Benutzers zu erstreben. (Nr. 12)

Darüber hinaus muss die Überführung für alle valide ER-Modelle deterministisch sein. Dies bedeutet, dass für ein ER-Modell immer das gleiche relationale Modell ermittelt wird. Mit dieser Anforderung soll, in gewissen Maßen, die Nachvollziehbarkeit gewährleistet werden. (Nr. 13)

Des Weiteren muss das relationale Modell dem Benutzer in geeigneter Form dargestellt werden. Eine intuitive Darstellung ist hierbei grafisch. (Nr. 14)

3.6 Generierung von SQL-Code auf Basis des generierten relationalen Modells

Die Erzeugung von SQL-Code soll ebenfalls ohne weiteres Einwirken des Benutzers geschehen. (Nr. 15) Hiervon ist die Eingabe der Datentypen in das relationale Modell ausgenommen. (Nr. 16)

Um den Benutzer der Software bestmöglich zu unterstützen, soll dem Benutzer nach der Überführung ein Block an SQL-Code übergeben werden, welcher sofern möglich, direkt in ein Datenbanksystem eingegeben werden kann. Das Zieldatenbanksystem ist hierbei eine PostgreSQL Datenbank. (Nr. 16)

3.7 Übersicht über die ermittelten Anforderungen

In der Abbildung 3.1 sind die in diesem Kapitel gesammelten Anforderungen einer Kategorie zugeordnet und tabellarisch dargestellt.

Nr.	Anforderung	Kategorie
1	Intuitive Bedienung der Anwendung	Nicht funktionale Anforderung
2	Implementierung als Webanwendung	Generelle Anforderung
3	Modularität der Software	Generelle Anfoderung
4	Modellierung mittels eines Zeichentools	Benutzerschnittstelle
5	Erstellen, Löschen und Bennenen von ER-Elementen	Benutzerschnittstelle
6	Erstellen, Löschen und Kardinalitätsvergabe von Verbindungen	Benutzerschnittstelle
7	Interaktive Reaktion auf Benutzereingaben	Benutzerschnittstelle
8	Speichern und Laden des modellierten Modells	Benutzerschnittstelle
9	Definition von ER-Regeln	Validierung
10	Durchsetzung von ER-Regeln	Validierung
11	Feedback bei der Modellierung in Bezug auf ER-Regeln	Feedback
12	Übersetzung des ER-Modells ohne Benutzereinwirkung	Übersetzungsalgorithmen
13	Deterministische Übersetzung des ER-Modells	Übersetzungsalgorithmen
14	Graphische Darstellung des relationalen Modells	Benutzerschnittstelle
15	Generierung von SQL-Code ohne Benutzereinwirkung	Übersetzungsalgorithmen
16	Eingabe von Datentypen in das relationale Modell	Benutzerschnittstelle
17	Anzeige des SQL-Codes als ein Block an SQL-Statements	Benutzerschnittstelle

Abbildung 3.1: Übersicht der ermittelten Anforderungen

4 Definition des unterstützten Entity-Relationship Modells

Aus der vorhergehenden Problemanalyse geht hervor, dass Vorraussetzung und Grundlage dieser Arbeit eine klar spezifizierte Definition eines ER-Modells ist, welches eindeutigen Regeln unterliegt.

Im Folgenden werden daher eindeutige Regeln für alle unterstützten ER-Konstrukte definiert.

Bei der Festlegung der Regeln wird hierbei auf die Anforderung geachtet, dem Modellierer des ER-Modells möglichst viele Freiheiten einzuräumen, indem die Regeln nur so streng definiert werden, dass eine eindeutige logische Zuordnung der Elemente ermöglicht wird.

Das in dieser Arbeit definierte ER-Modell lehnt sich dabei an das nach Kemper und Eickler vorgestellte Modell[12] an. Hierbei wird die grafische Notation von ISA-Strukturen nach Vossen übernommen. Das Modell wird zudem um die Attributtypen "Mehrwertiges Attribut"und "Zusammengesetztes Attribut"nach Vossen[24, S. 82] erweitert.

Innerhalb des Modells werden für eine präzisere Angabe der Funktionalitäten die Kardinalitäten mittels der Min-Max Notation[12, S. 46] ausgedrückt.

4.1 Entitäten

Die Definition der Regeln für Entitäten ergibt sich direkt aus den, im Grundlagenkapitel vorgestellten, Grundprinzipien des Entity-Relationship Modells.

1. Eine Entität darf keine unmittelbare Verbindung mit einer weiteren Entität besitzen.
2. Entitäten müssen mindestens ein identifizierendes Attribut besitzen.
3. Entitäten dürfen von beliebig vielen Attributen referenziert werden.

4.2 Relationen

Die Relationsregeln ergeben sich analog zu den Entitäten aus den Grundprinzipien des ER-Modells.

1. Eine Relation muss mindestens zwei Entitätstypen verbinden.
2. Eine Relation kann genau zwei mal einen gleichen starke Entitätentyp verbinden, wodurch reflexive Relationen erlaubt werden.
3. Relationen dürfen von beliebig vielen Attributen referenziert werden.

4.3 Attribute

Im Rahmen dieser Arbeit wird zwischen vier Typen an Attributen unterschieden.

- Reguläre, einwertige Attribute beschreiben ein Merkmal einer Entität.
- Mehrwertige Attribute beschreiben eine mehrfache Ausprägung eines Merkmals einer Entität.
- Identifizierende Attribute, sind einwertige Attribute, welche durch ihren Wert eine Ausprägung eines speziellen Entities identifizieren.
- Schwache Attribute, sind einwertige Attribute, welche durch ihren Wert und unter Miteinbeziehung des identifizierenden Entities, eine Ausprägung eines schwachen speziellen Entities identifizieren.

Reguläre und mehrwertige Attribute erhalten zudem die Fähigkeit als zusammengesetztes Attribut zu agieren. Sie sind neben ihrer Ursprungsform zusätzlich als zusammengesetztes Attribut anzusehen, sobald sie von einem regulären oder mehrwertigen Attribut referenziert werden.

Die hierbei referenzierenden Kindattribute erhalten ebenfalls implizit die Möglichkeit, als zusammengesetztes Attribut zu agieren. Hierbei wird sich an die von Vossen mehrmalige Anwendung der Zusammensetzung und Mehrwertigkeit[24, S. 63] orientiert.

In der Abbildung 4.1 wird eine beispielhafte Ausprägung einer Attributsstruktur dargestellt. Rot hervorgehobene Attribute besitzen hierbei die zusätzliche Rolle als zusammengesetztes Attribut.

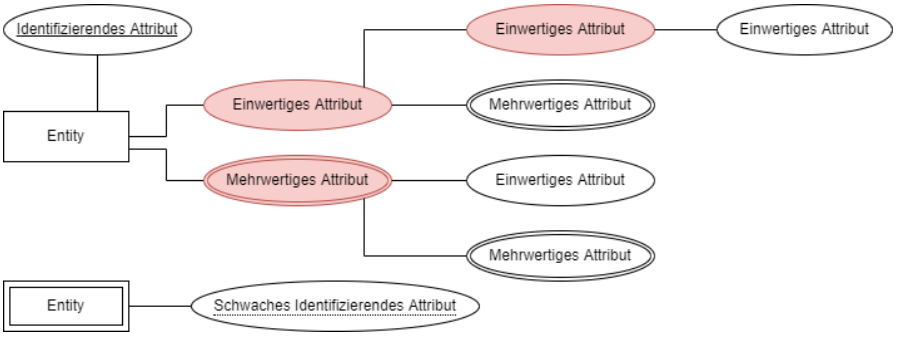

Abbildung 4.1: Beispielhafte Ausprägung einer Attributsstruktur

Durch die mehrfache Anwendung der Zusammensetzung und Mehrwertigkeit muss sichergestellt werden, dass jedes Attribut genau einer Entität oder Relation eindeutig zugeordnet werden kann. Mehrdeutigkeit ist genau dann gegeben, wenn sich durch Verwendung der Zusammensetzungsfunktion zyklischen Abhängigkeiten unter den Attributen ausprägen.

Jedes Attribut lässt sich hingegen eindeutig zuordnen, wenn eine direkte Verbindung oder genau einen Pfad über weitere Attribute zu einer Entität oder Relation besitzt. Durch die notwendige Eindeutigkeit folgt die Einschränkung, dass Attributsstrukturen nur in Form von Bäumen auftreten können.

Es sei angemerkt, dass es sich bei (schwachen) identifizierenden Attributen streng genommen, um ein reguläres Attribut mit einer zusätzlichen Identifikationsfunktion[24, S. 63] handelt. Im Rahmen dieser Arbeit werden reguläre und (schwache) identifizierende Attribute wohl unterschieden, da für diese ebenfalls unterschiedliche Regeln aufgrund der Anwendung der genannten Zusammensetzung getroffen werden müssen.

Attributsregeln

1. Ein reguläres oder mehrwertiges Attribut besitzt genau eine Verbindung zu einem weiteren regulären oder mehrwertigen Attribut, einer Entität oder einer Relation.

2. Bei einer Referenz auf ein Attribut ist diese nur zulässig, wenn die Attribute eindeutig einer Entität oder Relation zuordbar sind

3. Ein identifizierendes Attribut besitzt ausschließlich genau eine Referenz auf eine starke Entität.

4. Ein schwaches identifizierendes Attribut besitzt ausschließlich genau eine Referenz auf eine schwache Entität.

4.4 Existenzabhängige Typen

Grundlegend wird bei existenzabhängigen Typen, folgend auch schwache Typen genannt, zwischen drei Typen unterschieden.

- Schwache Entitäten. Diese sind in ihrer Existenz von übergeordneten Entitäten abhängig.
- Schwache Relationen. Diese beschreiben die Beziehung zwischen der übergeordneten Entität und der existenzabhängigen Entität.
- Schwache identifizierende Attribute. Diese wurden bereits im vorhergehenden Abschnitt behandelt.

Aufgrund ihrer fehlenden Identifizierbarkeit sind schwache Entitäten von einer übergeordneten Entität abhängig. Es ist hierbei anzumerken, dass es sich bei dieser übergeordneten Entität nicht um eine starke Entität handeln muss.

So ist es durchaus möglich, dass es sich bei einer übergeordneten Entität ebenfalls um eine schwache Entität handelt, welche wiederum von einer weiteren übergeordneten Entität abhängt.

Fallbeispiel Vorlesungsskript

Folgend wird dieser Aufbau anhand des Datenmodells eines Vorlesungsskripts plastisch dargestellt.

Ein Vorlesungsskript beinhaltet verschiedene Kapitel, welche nur mittels des Kapitelnames und dem Vorlesungsnamen identifiziert werden können, da verschiedene Vorlesungsskripte gleiche Kapitelnamen, z.B. "Einführung" oder "Zusammenfassung" besitzen können.

Jedes Kapitel besitzt hierbei eine größere Anzahl an Folien, welche jeweils bei eins beginnen. Die Folie ist daher existenzabhängig vom Typ Kapitel.

Des Weiteren können Folien Abbildungen besitzen, welche nur mittels einer Abbildungsbeschriftung innerhalb einer Folie identifiziert werden können. Zudem können an Folien Lesezeichen angebracht werden, um den Inhalt schneller finden zu können. Diese sind nur in einer begrenzten Anzahl an Farben verfügbar, weshalb diese in verschiedenen Folien mehrfach auftreten können.

Die Abbildung 4.2 zeigt hierbei ein vereinfachtes ER-Diagramm, welches ein mögliches Datenmodell eines Vorlesungsskripts widerspiegelt. Zur Übersichtlichkeit werden Kardinalitäten nicht dargestellt.

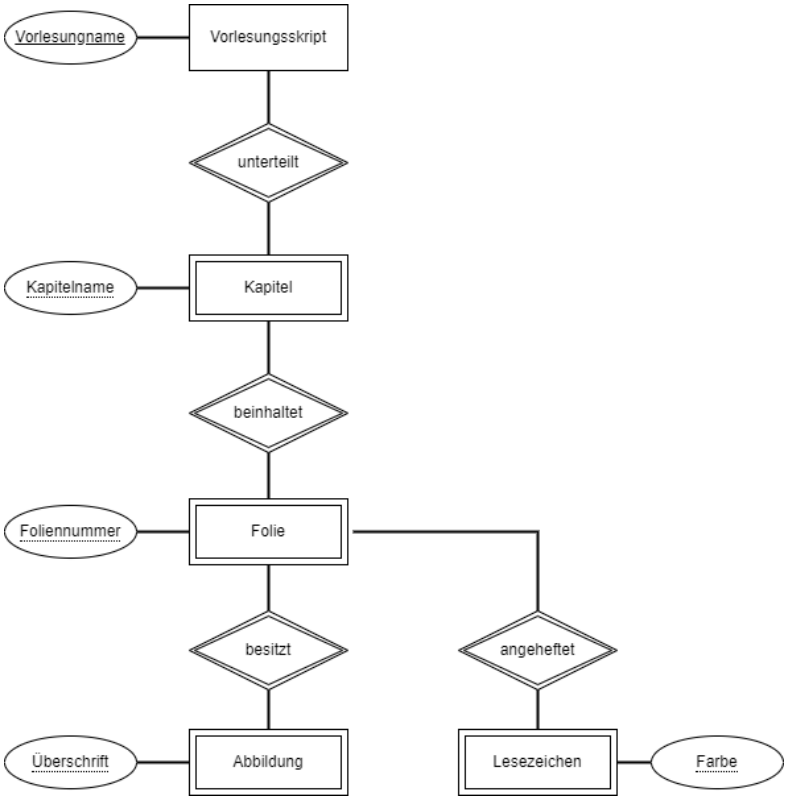

Abbildung 4.2: Fallbeispiel Vorlesungsskript

Als weiterführendes und umfangreicheres Fallbeispiel kann hierfür zum Beispiel die Modellierung einer Airline dem Buch Grundlagen von Datenbanksysteme von Esmasri und Navathe [5, S. 86] herangenommen werden.

Bei existenzabhängigen Typen muss ebenfalls sichergestellt werden, dass einem existenzabhängigen Typ der identifizierende Typ eindeutig zugeordnet werden kann.

Strukturell ist daher der Aufbau von existenzabhängigen Strukturen analog zum Aufbau von Attributsstrukturen eingeschränkt, wobei die Referenz zwischen Attributen in existenzabhängigen Konstrukten mittels schwachen Relationen ausgedrückt wird.

Die Abbildung 4.3 verdeutlicht den oben genannten Zusammenhang.

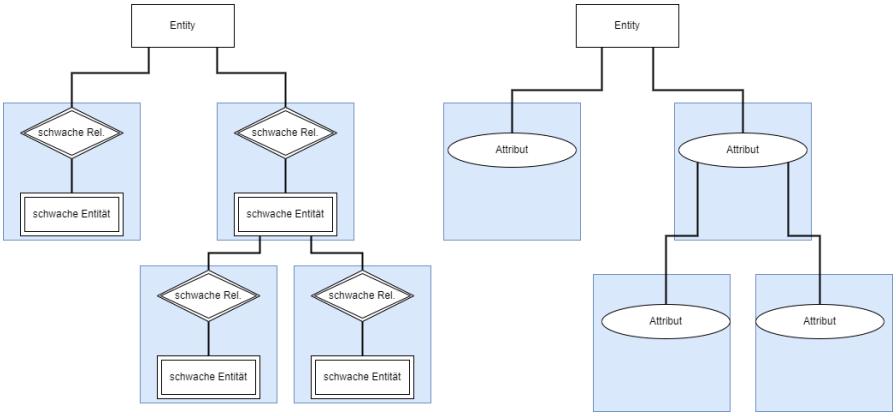

Abbildung 4.3: Vergleich der Struktur von Attributskonstrukten und existenzabhängigen Konstrukten

Es sei zudem angemerkt, dass die Funktionalität von schwachen Relationen nur 1:1 oder 1:N sein darf, da ein schwaches Entity in seiner Existenz immer genau von einem übergeordneten Entity abhängt.

Regeln für existenzabhängige Typen

1. Eine schwache Entität besitzt über eine schwache Relation genau eine Verbindung zu einer weiteren schwachen Entität oder einer starken Entität.
2. Bei einer Referenz auf eine schwache Entität ist diese nur dann zulässig, wenn diese eindeutig zuordbar sind.
3. Schwache Entitäten müssen mindestens ein schwaches identifizierendes Attribut besitzen und dürfen von beliebig vielen Attributen referenziert werden.
4. Eine schwache Entität ist im Bezug auf reguläre Relationen wie eine starke Entität zu behandeln.
5. Ein schwache Relation muss entweder genau eine starke Entität mit genau einer schwachen Entität verbinden oder genau zwei unterschiedliche schwache Entitäten miteinander verbinden.
6. Schwache Relationen dürfen von beliebig vielen Attributen referenziert werden.
7. Die Funktionalität von schwachen Relationen muss 1:1 oder 1:N sein.

4.5 IsA-Struktur

IsA-Strukturen ermöglichen den Einsatz von Generalisierung im konzeptuellen Entwurf. Die Untertypen einer IsA-Struktur erben hierbei alle Attribute des Obertyps. Bei Ober- und Untertypen handelt es sich um Entitäten. Grundlegend gibt es zwei Ausprägungen. Eine partielle, nicht disjunkte Spezialisierung sowie eine partielle, disjunkte Spezialisierung.[24, S.79]

In beiden Fällen muss bei der Festlegung der Regeln für IsA-Strukturen eine mehrfache Vererbung der Attribute eines Obertyps an einen Untertyp ausgeschlossen werden.

Bei der Mehrfachvererbung sind hierbei zwei Fälle abzudecken. Diese sind in Abbildung 4.4 abgebildet. Grün hervorgehobene Elemente stellen hierbei die Vererbungshierarchie der Elemente vor dem Hinzufügen der rot markierten Verbindung dar. Blaue Elemente stellen die Vererbungshierarchie dar, welche durch die Verbindung hinzugefügt wird. Rot gefärbte Elemente zeigen die durch die rote Verbindung entstandene Mehrfachvererbung.

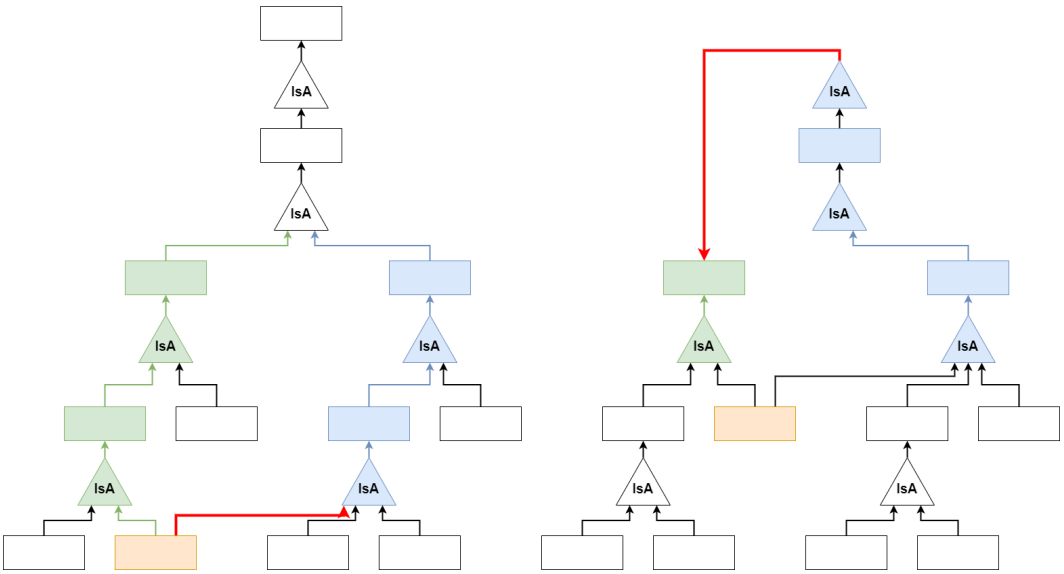

Abbildung 4.4: Darstellung unzulässiger Verbindungen in IsA-Strukturen

Im ersten Fall führt eine Verbindung einer Entität als Untertyp zu einer IsA-Struktur dazu, dass an diese Entität Attribute eines Elements einer höheren Ebene mehrfach vererbt werden. Im zweiten Fall führt eine Verbindung einer Entität als Obertyp zu einer IsA-Struktur dazu, dass an Elemente einer niedrigeren Ebene mehrfach vererbt wird. In der Grafik sind hiervon die gelb hervorgehobenen Entitäten betroffen.

Um die Regeln für IsA-Strukturen festzulegen, werden hierfür zunächst drei Mengen definiert.

Die "Subtypenmenge" einer Entität beinhaltet die Entität selbst, sowie alle weiteren Entitäten an die Attribute vererbt werden.

Die "Supertypenmenge" einer Entität beinhaltet die Entität selbst, sowie alle weiteren Entitäten, von denen Attribute geerbt werden.

Die "beeinflusste Typenmenge" einer Entität beinhaltet die Subtypenmenge und zusätzlich für jede Entität der Subtypenmenge die Supertypenmenge.

Die Abbildung 4.5 stellt die Mengen anhand zweier IsA-Strukturen dar. Betrachtet werden hierbei die Mengen ausgehend von der blau hervorgehobenen Entität. Im linken IsA-Konstrukt ist die Subtypenmenge in Grün und die Supertypenmenge in Rot darstellt. Das rechte IsA-Konstrukt zeigt die beeinflusste Typenmenge.

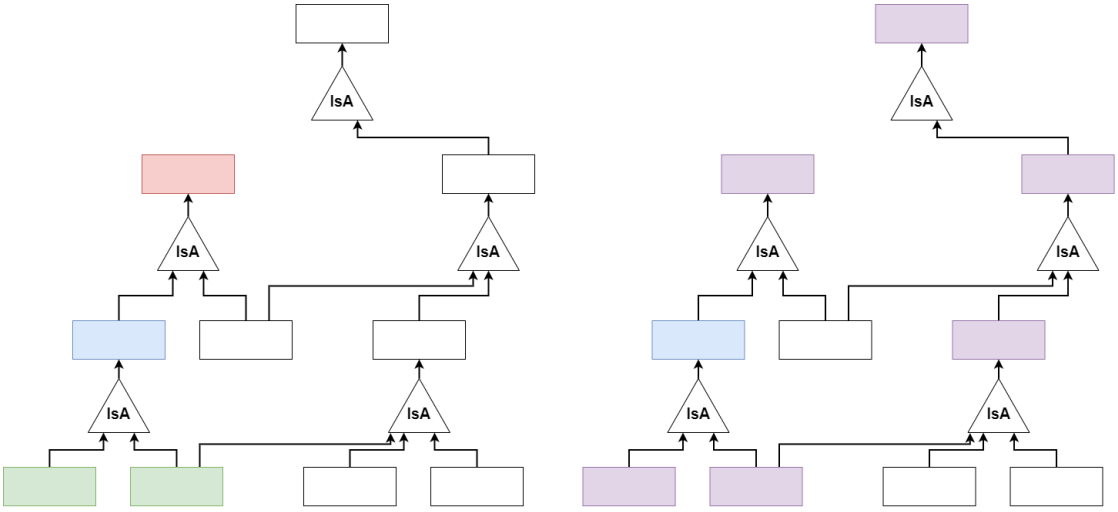

Abbildung 4.5: Darstellung der Sub-, Super- und beeinflussten Typenmenge

Die beeinflusste Typenmenge einer Entität entsprechen hierbei alle Entitäten, welche bei einer Verbindung dieser Entität zu einer IsA-Struktur als Untertyp eine Mehrfachvererbung widerfahren können.

Wird eine Entität mit einer IsA-Struktur als Obertyp verbunden, so kann die beeinflusste Typenmenge aller Untertypen eine Mehrfachvererbung widerfahren.

Mithilfe der definierten Mengen und unter der Berücksichtigung der Vermeidung von Mehrfachvererbung können folgend die Regeln festgelegt werden.

Regeln für IsA-Strukturen

1. Eine IsA-Struktur besitzt genau eine starke Entität als Obertyp und mindestens eine starke Entität als Untertyp.
2. Bei einer Verbindung einer Entität als Untertyp einer IsA-Struktur muss die beeinflusste Typenmenge der Entität disjunkt sein mit der Supertypenmenge des Obertyps der IsA-Struktur.
3. Bei einer Verbindung einer Entität als Obertyp einer IsA-Struktur muss die beeinflusste Typenmenge der Untertypen der IsA-Struktur disjunkt sein mit der Supertypenmenge der Entität.
4. Entitäten welche ein Untertyp einer IsA-Struktur sind, müssen kein identifizierendes Attribut besitzen, da dieses vom Obertyp vererbt wird.

4.6 Anmerkungen zu weiteren ER-Elementen

Es ist anzumerken, dass in ER-Modellen zwei weitere ER-Elementen des öfteren Verwendung finden. Hierbei handelt es sich um abgeleitete Attribute und assoziative Entitäten.

Ersteres wird im Modell dieser Arbeit nicht unterstützt. Dies liegt daran begründet, dass bei der Modellierung von relationalen Datenbanksystemen das ER-Modell in das relationale Modell überführt wird uns sich hierbei dieser Attributstyp gegenstandslos auflöst. Assoziative Entitäten finden im Rahmen dieses Modells zugunsten von n-ären Relationen ebenfalls keine Anwendung.

5 Konzeptionelle Umsetzung eines Validierungs- und Feedbacksystems

Im Kapitel Problem- und Anforderungsanalyse wurden Validierungs- und Feedbackanforderungen auf Basis der benutzergenerierten Entity-Relationship Modelle an die Software gestellt. Im Folgenden werden diese adressiert.

5.1 Ansätze zur Umsetzung der Validierungs- und Feedbackanforderungen

Ein erster nativer Ansatz ist es, in den Modellierungsprozess des ER-Diagramms im Zeichentool zunächst nicht einzugreifen und damit uneingeschränkt alle Verbindungen zwischen Elementen zu erlauben.

Hierbei kann die Validierung über einen, von den Übersetzungsalgorithmen getrennten, Validierungsalgorithmus durchgeführt werden. Wird das System aufgefordert, das ER-Modell in das relationale Modell zu überführen, durchläuft das gesamte ER-Modell den Validierungsalgorithmus. Dieser prüft das übergebene Modell dann auf die im vorherigen Abschnitt definierten Regeln. Hierbei hält der Algorithmus eine Liste, welche er mit Fehlernachrichten befüllt, wenn dieser einen Verstoß gegen eine Regel feststellt. Nach Abschluss des Algorithmus wird dann dem Benutzer ein Popup dargestellt, welches alle festgestellten Fehler des Modells beinhaltet. Wird kein Verstoß gegen eine Regel festgestellt, so können direkt die Übersetzungsalgorithmen ausgeführt werden.

Dieser Ansatz hat den Nachteil, dass unter bestimmten Umständen nicht alle Verstöße gegen Regeln ermittelt werden. Dieser Fall kann eintreten, wenn ein Algorithmus aufgrund eines strukturellen Fehlers des ER-Diagramms den Algorithmus nicht fortführen kann und somit vorzeitig abbrechen muss. Weitere Fehler, welche innerhalb des Algorithmus entdeckt werden würden, werden dann nicht in die Fehlerliste aufgenommen.

Zudem erhält bei diesem Ansatz der Benutzer erst spätes Feedback. Das verspätete Feedback kann hierbei zu zusätzlichem Aufwand für den Entwickler führen.

Da die Motivation der Software jedoch darin liegt, den Entwickler weitgehend zu unterstützen und den Aufwand zu verringern, ist stattdessen eine proaktivere Lösung erforderlich. Das in dieser Arbeit implementierte Validierungssystem findet daher unmittelbar bei der Modellierung der ER-Diagramme Anwendung.

Die grundsätzliche Idee dieses Systems ist dabei, proaktiv Aktionen innerhalb der Modellierung zu unterbinden, welche zwangsweise zu einem Verstoß einer Regel führen würde.

Die Herausforderung dieser Idee ist es, den Nutzer bei der Modellierung nicht einzuschränken. Würde für jede Aktion des Benutzers auf Einhaltung der Regeln geprüft werden, so wäre der Benutzer in seinen Möglichkeiten stark eingegrenzt. In diesem Falle könnte der Benutzer nur von einem validen Zustand eines ER-Diagramms in einen weiteren wechseln. Dieses Szenario wird in der Abbildung verdeutlicht 5.1.

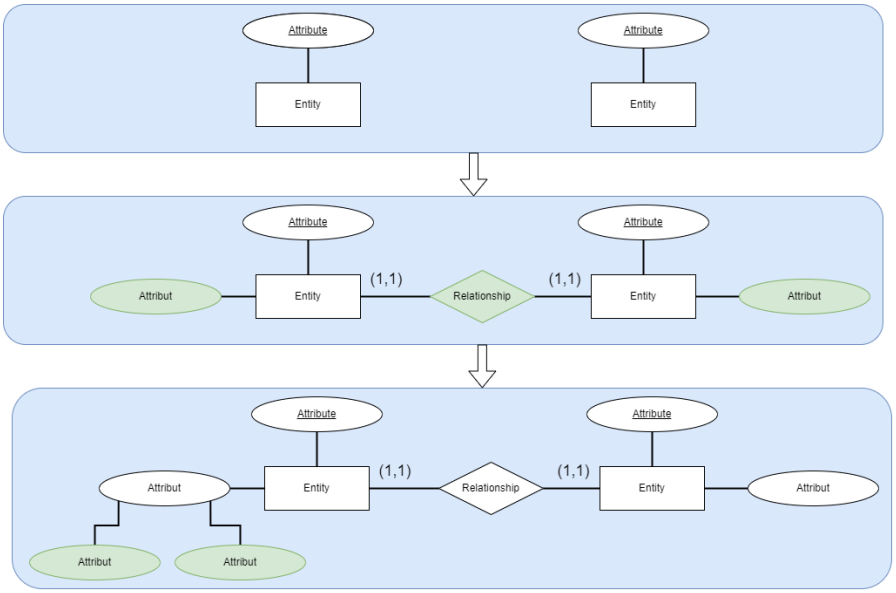

Abbildung 5.1: Modellierung eines ER-Diagramms über valide Zustände

Es wäre bei der Modellierung zum Beispiel nicht möglich, die Relation vor den Entitäten zu erstellen, da dies zu einer Verletzung der Relationsregeln führen würde. Darüber hinaus könnte sich der Benutzer beispielsweise nicht mehr dazu entscheiden, Attribute zuerst zu modellieren und diese erst zu einem späteren Zeitpunkt mit Entitäten zu assoziieren.

Eine freie Erstellung von beliebigen Elementen wäre somit nicht möglich. Die direkte Prüfung würde hierbei auch die Löschung von Elementen einschränken.

Die Problematik ist auf die Strenge der Regeln für ER-Diagramme zurückzuführen. Diese ist für die Übersetzung des Diagramms notwendig. Während des Modellierungsprozesses müssen jedoch mildere Regeln herangezogen werden.

5.2 Einführung des Begriffs der partiellen Konsistenz

In der vorhergehenden Sektion wurde festgestellt, dass eine proaktive Prüfung auf die in Kapitel 4 definierten Regeln zu einer Einschränkung bei der Modellierung führt.

Um diese Problematik zu lösen, wird im Folgenden der Begriff "partielle Konsistenz" definiert. Aufbauend auf dieser Definition wird ein Konzept vorgestellt, welches die Flexibilität bei der Modellierung erhält und dennoch die Einhaltung der Regeln gewährleistet.

Der Begriff "partielle Konsistenz" ist wie folgt definiert:

Ein Entity-Relationship Modell ist in einem partiell konsistenten Zustand, wenn es sich durch Einfügeoperationen, bestehend aus dem Hinzufügen von Elementen und Verbindungen, in einen Zustand überführen lässt, welcher alle definierten ER-Regeln erfüllt.

Bei dieser Definition eines Zustandes eines ER-Diagramms handelt es sich um eine Teilmenge der möglichen validen und invaliden Ausprägungen eines ER-Diagramms.

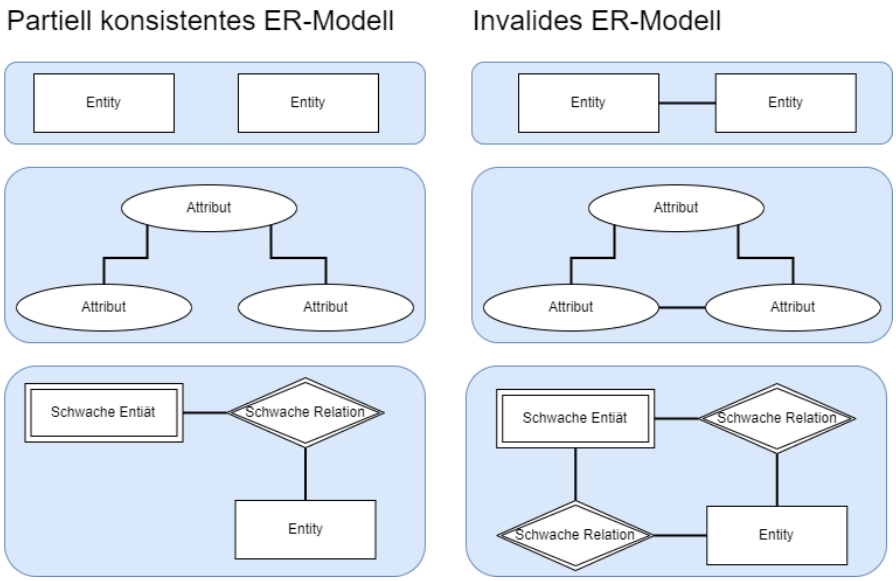

Abbildung 5.2: Vergleich partiell konsistenter und invalider ER-Modelle

Die Erstellung von ER-Modellen über partiell konsistente Zustände hinweg besitzt dabei Eigenschaften, welche folgend erläutert werden.

Die Modellierung mittels partieller Konsistenz schränkt das Einfügen von Elementen nicht ein, da ER-Diagramme grundsätzlich nicht endlich beschränkt sind und jedes Element in ein ER-Modell integriert werden kann.

Darüber hinaus wird das Löschen von Elementen in keinster Weise eingeschränkt.

Wenn sich der Ursprungszustand des ER-Diagramms in einem partiell konsistenten Zustand befindet, sind somit auch alle ER-Diagramme die sich durch Löschoperationen ergeben in einem partiell konsistenten Zustand.

Dies liegt darin begründet, dass mittels Einfügeoperationen das ER-Modell wieder in den Ursprungszustand versetzt werden kann. Da dieser Zustand partiell konsistent ist, kann das Modell gemäß der Definition mittels Einfügeoperationen in einen validen Zustand gebracht werden.

Somit können auch alle sich aus Löschoperationen ergebenen ER-Diagramme mittels Einfügeoperationen in einen validen Zustand überführt werden, was diese gemäß der Definition partiell konsistent macht.

Analog gilt dies für das Löschen von Verbindungen zwischen zwei Elementen eines ER-Modells.

Wird während der Modellierung für jede Aktion des Benutzers proaktiv auf partielle Konsistenz geprüft, so wird dem Benutzer eine maximale Flexibilität zugesprochen.

Dies geht auf Basis der genannten Eigenschaften hervor, da ausschließlich das Einfügen von Verbindungen eingeschränkt wird, welche das ER-Modell in einen invaliden Zustand bringen. Das Überführen des Modells von diesem invaliden Zustand in einen validen Zustand würde hierbei zwingend das Löschen der Verbindung oder anderer Elemente erfordern.

In der Abbildung 5.3 wird die Modellierung über partiell konsistente Zustände dargestellt.

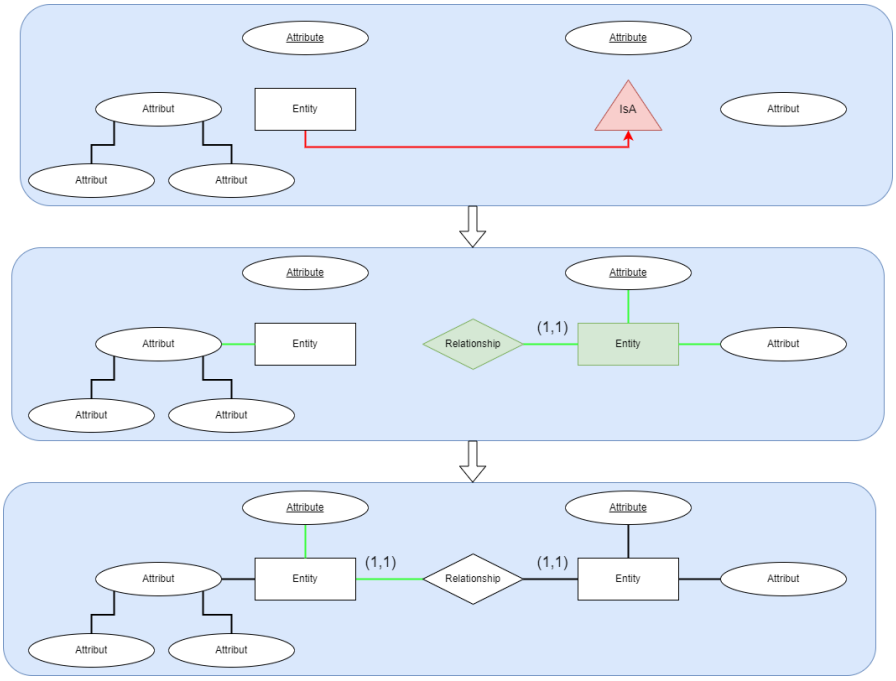

Abbildung 5.3: Modellierung eines ER-Diagramms über partiell konsistente Zustände

5.3 Konzeptionelle Umsetzung der Validierungs- und Feedbackanforderungen

Die Umsetzung der Validierungs- und Feedbackanforderungen auf Basis von partiell konsistenten Zuständen erfolgt über zwei voneinander getrennte Systeme.

Das erste System ist für die proaktive Erhaltung der partiellen Konsistenz verantwortlich. Dieses evaluiert hierbei, ob das Hinzufügen einer Verbindung zwischen ER-Elemente das bisher partiell konsistente ER-Diagramm in einen weiteren partiell konsistenten Zustand übergehen lässt. Dies geschieht auf Basis von Übergangsfunktionen. Eine Übergangsfunktion legt dabei für ein ER-Konstrukt fest, unter welchen Umständen dies mit einem anderen Element assoziiert werden kann.

Mittels des zweiten Systems erhält der Nutzer Feedback über das aktuelle ER-Diagramm. Hierfür wird geprüft, ob sich das partiell konsistente ER-Diagramm in einem vollständig validen Zustand befindet. Dies geschieht auf Basis der festgelegten ER-Regeln, wobei diese durch die Eigenschaften von partiell konsistenten ER-Diagrammen vereinfacht und in Form von Endzuständen ausgedrückt werden können. Im Folgenden wird genauer auf diese Systeme eingegangen.

5.3.1 Proaktives Validierungssystem auf Basis von Übergangsfunktionen

Das Validierungssystem findet Anwendung bei der Erstellung einer Verbindung zwischen zwei Elementen. Um den Benutzer proaktiv zu unterstützen wird dieses System eingesetzt, wenn ein Benutzer ein Element selektiert.

Jeder ER-Typ erhält hierbei eine eigene Übergangsfunktion, welche aus einer Reihe von Bedingungen besteht. Jedes Element muss hierbei alle Bedingungen erfüllen.

Die Übergangsfunktion wird abhängig vom Typ des selektierten Elements ausgewählt.

Darauf folgend untersucht das System alle weiteren Elemente mittels der Übergangsfunktion auf eine mögliche Verbindung. Elemente zu denen eine Verbindung möglich ist, können anschließend in geeigneter Form hervorgehoben werden.

Übergangsfunktionen lassen sich dabei mittels eines Pipe-Filter Designs realisieren. In Abbildung 5.4 ist ein exemplarischer Aufbau einer Übergangsfunktion dargestellt.

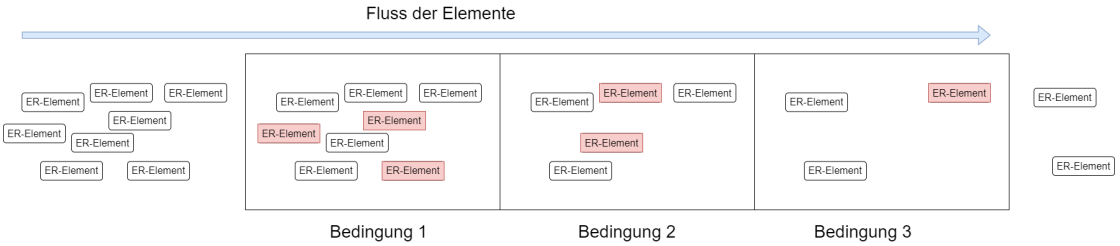

Abbildung 5.4: Exemplarischer Aufbau einer Übergangsfunktion

Fortfolgend werden die ER-Typ spezifischen Übergangsfunktionen festgelegt. Eine vollständige Erläuterung dieser, würde den Rahmen dieser Arbeit sprengen. Darauf folgend werden jedoch die wichtigsten querschnittlichen Konzepte bei der Erstellung dieser erläutert.

Übergangsfunktion von Entitäten

1. Starke Entitäten, schwache Entitäten und schwache identifizierende Attribute sind nicht erlaubt
2. Eine Verbindung zu einem Attribut ist nur dann erlaubt, wenn dieses noch nicht eindeutig zuordbar ist.
3. Eine Verbindung zu einer schwachen, identifizierenden Relation ist nicht erlaubt, wenn diese mit einer bereits zuordbaren schwachen Entität oder einer weiteren Entität verbunden ist.
4. Eine Verbindung zu einer schwachen, identifizierenden Relation ist nicht erlaubt, wenn diese bereits von einer weiteren starken Entität oder von zwei schwachen Entitäten referenziert wird.
5. Eine Verbindung zu einer IsA-Struktur ist nur erlaubt, wenn die 2. und 3. Regel von IsA-Strukturen erfüllt ist.

Übergangsfunktion von Relationen

1. Es sind nur die ER-Typen normales Attribut, mehrwertiges Attribut, starke Entität und schwache Entität erlaubt.
2. Eine Verbindung zu einem Attribut ist nur dann erlaubt, wenn dieses noch nicht eindeutig zuordbar ist.
3. Bei einer Verbindung zu einer starken Entität ist die 2. Relationenregel anzuwenden.

Übergangsfunktion von Attributen

1. Für ein normales, reguläres Attribut sind identifizierende Attribute, schwache identifizierende Attribute und IsA-Strukturen nicht erlaubt
2. Für ein mehrwertiges Attribut sind identifizierende Attribute, schwache identifizierende Attribute und IsA-Strukturen nicht erlaubt
3. Ein identifizierendes Attribut kann sich nur mit einer starken Entität verbinden.
4. Ein schwaches identifizierendes Attribut kann sich nur mit einer schwachen Entität verbinden.

5. Eine Verbindung zu einer Entität oder Relation ist nur dann erlaubt, wenn das Attribut nicht bereits eindeutig zuordbar ist.
6. Eine Verbindung zu einem weiteren Attribut ist nur dann erlaubt, wenn
   - Eines oder beide Attribute nicht zuordbar sind
   - Keines der Attribute zuordbar ist und diese nicht über einen Pfad an weiteren Attributen miteinander verbunden sind

Übergangsfunktionen schwacher Typen

Bei der Übergangsfunktion von schwachen Entitäten sind folgende Bedingungen definiert:

1. Starke Entitäten, schwache Entitäten, IsA-Strukturen und identifizierende Attribute sind nicht erlaubt
2. Eine Verbindung zu einem Attribut ist nur dann erlaubt, wenn dieses noch nicht eindeutig zuordbar ist.
3. Eine Verbindung zu einer schwachen, identifizierenden Relation ist nur dann erlaubt, wenn:
   - Die schwache Relation ist mit keiner Entität verbunden
   - Die schwache Relation ist mit einer starken Entität verbunden und die selektierte schwache Entität ist nicht zuordbar
   - Die schwache Relation ist mit einer nicht zuordbaren schwachen Entität verbunden
   - Die schwache Relation ist mit einer bereits zuordbaren schwachen Entität verbunden und die selektierte schwache Entität ist noch nicht zuordbar

Bei der Übergangsfunktion von schwachen Relationen sind folgende Bedingungen definiert:

1. (Schwache) Relationen, IsA-Strukturen, und (schwache) identifizierende Attribute sind nicht erlaubt
2. Eine Verbindung zu einem Attribut ist nur dann erlaubt, wenn dieses noch nicht eindeutig zuordbar ist.
3. Eine Verbindung mit (schwachen) Entitäten ist nur erlaubt, wenn die Relation noch keine zwei Verbindungen mit (schwachen) Entitäten besitzt.

4. Bei einer Verbindung zu einer schwachen Entität ist analog die 3. Bedingung von schwachen Entitäten anzuwenden.

5. Eine Verbindung zu einer starken Entität ist nicht erlaubt, wenn die selektierte Relation mit einer zuordbaren schwachen Entität oder einer weiteren starken Entität verbunden ist.

Übergangsfunktionen von IsA-Strukturen

1. Eine Verbindung ist nur zu starken Entitäten erlaubt

2. Eine Verbindung zu einer IsA-Struktur ist nur erlaubt, wenn die 2. und 3. Regel von IsA-Strukturen erfüllt ist.

5.3.2 Querschnittliche Konzepte der Übergangsfunktionen

Innerhalb der Festlegung der Übergangsfunktionen für ER-Elemente gibt es zwei nicht triviale Einschränkungen, welche im Folgenden näher erläutert werden.

Die erste Einschränkung betrifft die Zusammensetzung von Attributen und wird mittels der 6. Bedingung von Attributen durchgesetzt.

Sie ist hierbei auf die eindeutige Zuordnung von Attributen zu Entitäten zurückzuführen. Betrachtet werden die in Abbildung 5.5 abgebildeten Fälle. Rot dargestellte Attribute können hierbei noch keiner Entität zugeordnet werden. Im Gegensatz dazu sind grün hervorgehobene Attribute bereits einer Entität zugeordnet. Der Fokus der Abbildung liegt hierbei auf der Erstellung der blau hervorgehobenen Verbindung.

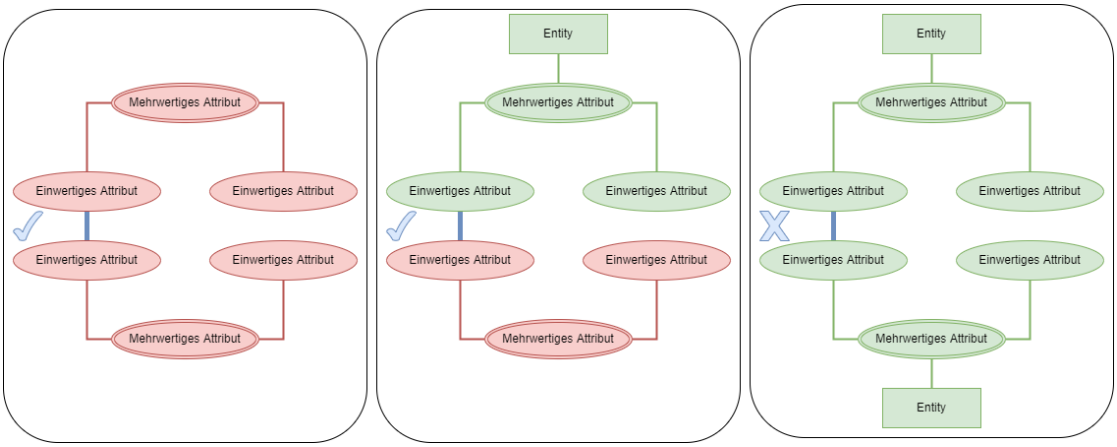

Abbildung 5.5: Zuordnung der Attribute zu Entitäten

Der Abbildung 5.5 ist zu entnehmen, dass in den ersten beiden Ausprägungen die Verbindung valide ist. Die Attribute in der ersten Ausprägung besitzen weiterhin keine Zuordnung zu einer Entität. Die Erstellung der Verbindung in der zweiten Ausprägung verursacht, dass alle rot hervorgehobenen Attribute der oben dargestellten Entität zugeordnet werden.

Die Verbindung der dritten Ausprägung muss hingegen untersagt werden, da mittels dieser von einem Attribut ausgehend zwei Entitäten erreichbar sind und somit eine eindeutige Zuordnung nicht mehr ermöglicht ist.

Die zweite Einschränkung betrifft hierbei die Beziehung von schwachen Typen mit starken Entitäten. Genauer sind hiervon die Bedingungen 4 und 5 von starken Entitäten, die Bedingung 3 von schwachen Entitäten und die Bedingungen 4 und 5 von schwachen Relationen betroffen.

Diese Einschränkung ist auf die eindeutige Zuordnung der Abhängigkeit von schwachen Entitäten zurückzuführen. Konzeptuell tritt hierbei die gleiche Problematik wie bei Attributsstrukturen, bedingt durch den gleichen strukturellen Aufbau dieser auf.

In der Abbildung 5.6 wird die Notwendigkeit der Einschränkung verdeutlicht. Rot hervorgehobene Elemente besitzen hierbei noch keine starke Entität, von der diese abhängen können. Grün hervorgehobene Elemente hängen eindeutig von einer starke Entität ab.

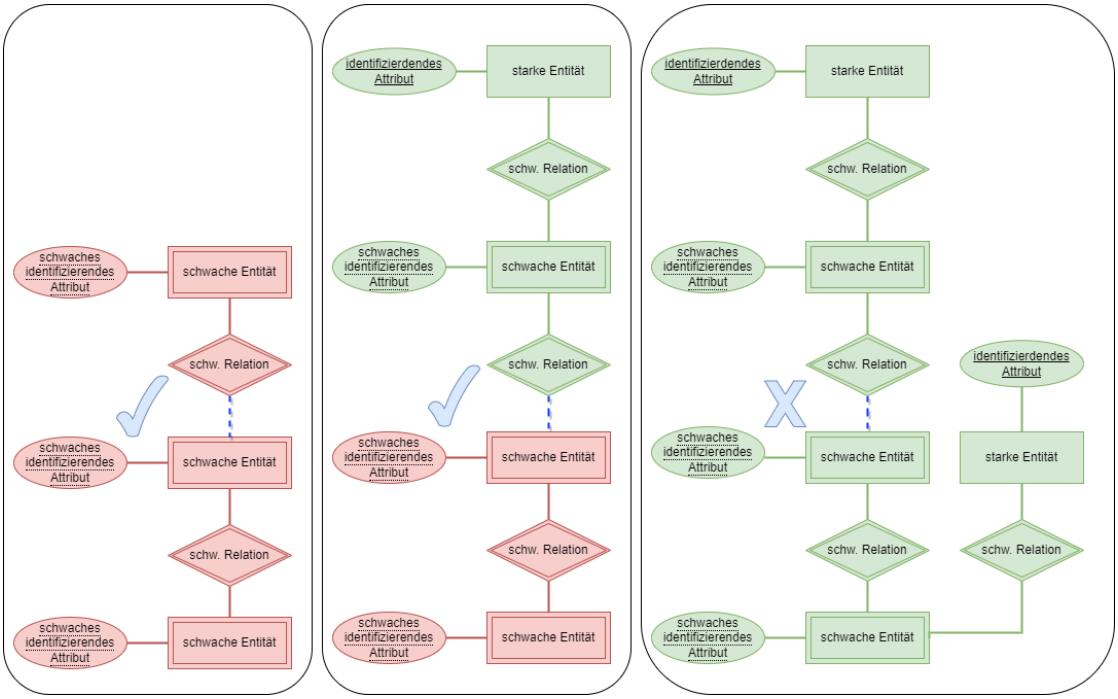

Abbildung 5.6: Zuordnung der Abhängigkeiten schwacher Typen

Wird die Erstellung der blau hervorgehobenen Verbindung betrachtet, so zeigt sich, dass analog zu den Attributskonstrukten, Strukturen vollständig untersucht werden müssen um festzustellen ob diese Verbindung valide ist.

Besitzen beide Strukturen keine starken Entitäten, so entsteht auch in der zusammengesetzten Struktur keine Zuordnung. Wird jedoch eine Struktur mit einer starken Entität mit einer Weiteren ohne starke Entität verbunden, so wird allen schwachen Entitäten der zweiten Struktur ein abhängiger Typ zugeordnet.

Werden zwei Strukturen mit starken Entitäten verbunden, so kann jeder schwachen Entität innerhalb der Struktur nicht mehr eindeutig der abhängige Typ zugeordnet werden. Diese Verbindung muss somit untersagt werden.

Die zu betrachtenden Strukturen sind hierbei immer abgegrenzt durch schwache und starke Entitäten, welche mittels schwachen Relationen zusammenhängen.

5.3.3 Proaktives Feedbacksystem auf Basis von Endbedingungen

Mittels des Validierungssystems ist gegeben, dass sich das ER-Diagramm immer in einem partiell konsistenten Zustand befindet.

Die Umsetzung des Feedbacksystems, auf konzeptioneller Ebene, ist daher sehr einfach umzusetzen, da alle Elemente nur auf eine Endbedingung, abhängig vom ER-Typ, geprüft werden müssen.

Diese Endbedingungen sind im Folgenden genannt.

1. Alle Entitäten besitzen mind. ein identifizierendes Attribut oder sind Untertyp einer IsA-Struktur.
2. Jede (schwache) Relation ist mit mind. zwei (schwachen) Entitäten verbunden.
3. Alle Attribute sind zuordbar.
4. Jede schwache Entität ist zuordbar.
5. Jede IsA-Struktur besitzt mind. einen Untertyp und genau einen Obertyp.

Die Prüfung der Elemente auf die Endbedingung erfolgt unmittelbar nachdem der Benutzer eine Aktion durchgeführt hat. Verstößt ein Element über eine der genannten Bedingungen, so kann diesem der Verstoß in geeigneter Form bekannt gemacht werden.

Wird für kein Element ein Verstoß festgestellt, so ist das ER-Diagramm valide und eine Überführung in das relationale Modell ist möglich.

6 Übersetzung des Entity-Relationship Modells

6.1 Übersetzung in das relationale Modell

Innerhalb der vorangegangen Sektionen wurden Regeln für Entity-Relationship Modelle festgelegt und dessen Einhaltung mittels des Validierungs- und Feedbacksystems gesichert.

Auf Basis dieser Grundlagen wird im Folgenden ein Konzept zur Übersetzung des Entity-Relationship Modells in das relationale Modell vorgestellt.

Hierfür wird zunächst der strukturelle Aufbau von ER-Modellen näher betrachtet.

6.1.1 Strukturelles Datenmodell von Entity-Relationship Modellen

Um algorithmische Ansätze zur Übersetzung des Modells zu ermitteln, ist ein grundsätzliches Datenmodell erforderlich, auf dem diese arbeiten können. Um dieses zu ermitteln, wird daher im Folgenden die Struktur von ER-Modellen untersucht.

Hierfür wird zunächst das ER-Diagram in Abbildung 6.1 betrachtet. Zur Übersichtlichkeit werden Kardinalitäten nicht dargestellt.

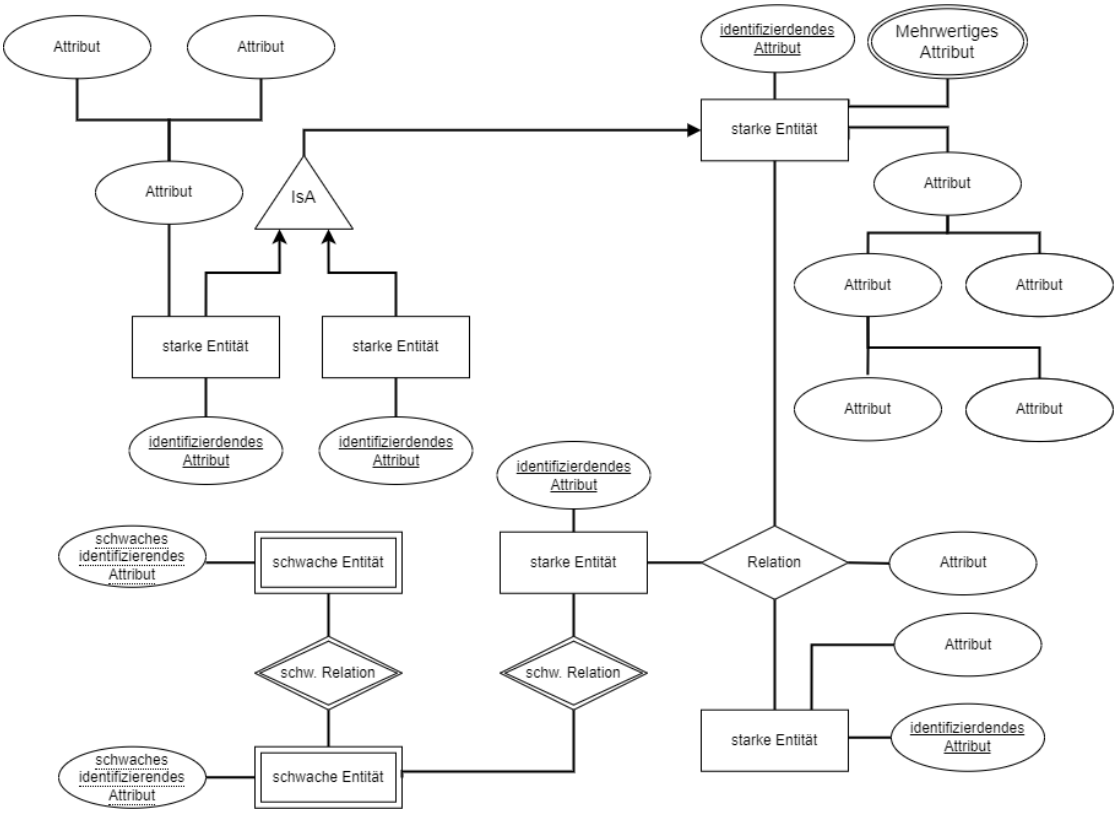

Abbildung 6.1: Darstellung eines ER-Diagramms

Wird das ER-Diagramm betrachtet, so besteht dieses im Wesentlichen aus Elementen und Verbindungen. Daher kann die Struktur direkt mittels eines Graphen ausgedrückt werden.

Die Graphdarstellung des oben gezeigten Modells ist in Abbildung 6.2 abgebildet. Rot hervorgehobene Knoten entsprechen einer Entität, während blaue Knoten Relationen darstellen. Grün hervorgehobene Elemente stellen die Attribute des ER-Diagramms dar. Bei dem nicht gefärbten Knoten handelt es sich um die IsA-Struktur.

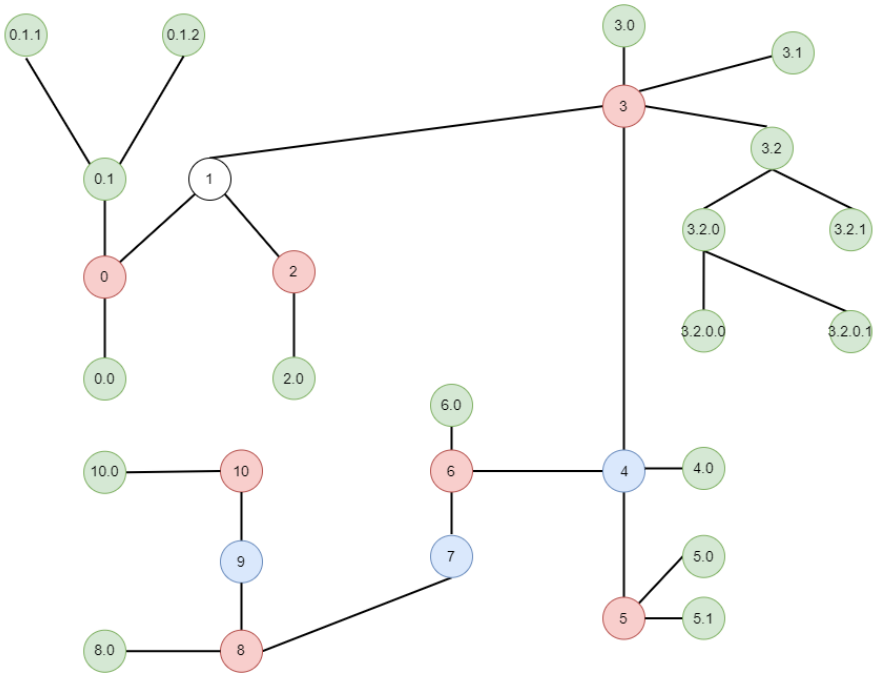

Abbildung 6.2: Graphdarstellung eines ER-Diagramms

Die in Abbildung 6.2 gezeigte Darstellung des ER-Diagramms als Graph ist sehr allgemein und lässt sich weiter einschränken.

In der Definition von ER-Diagrammen wurde erläutert, dass sich Attribute immer in Form von Bäumen ausdrücken. Dies ermöglicht es, den Graphen weiter einzuschränken. Aufgrund dessen, dass alle Entitäten und Relationen Attribute besitzen können, agiert jedes dieser Elemente als Wurzel eines Baumes.

Des Weiteren können Informationen über die Kardinalität zwischen einer Relation und einer Entität in den Kanten zwischen den abwechselnden blauen und roten Knoten abgelegt werden. Die vom nicht gefärbten Knoten ausgehenden Kanten erhalten Informationen darüber, ob es sich bei dem verbundenen Knoten um einen Obertyp oder Untertyp der IsA-Struktur handelt.

Um diese Zusammenhänge zu verdeutlichen, wurden in der Abbildung 6.3 die Knoten des Graphen verschoben. Gelbe Ovale stellen hierbei einzelne Baumstrukturen innerhalb des Graphen dar.

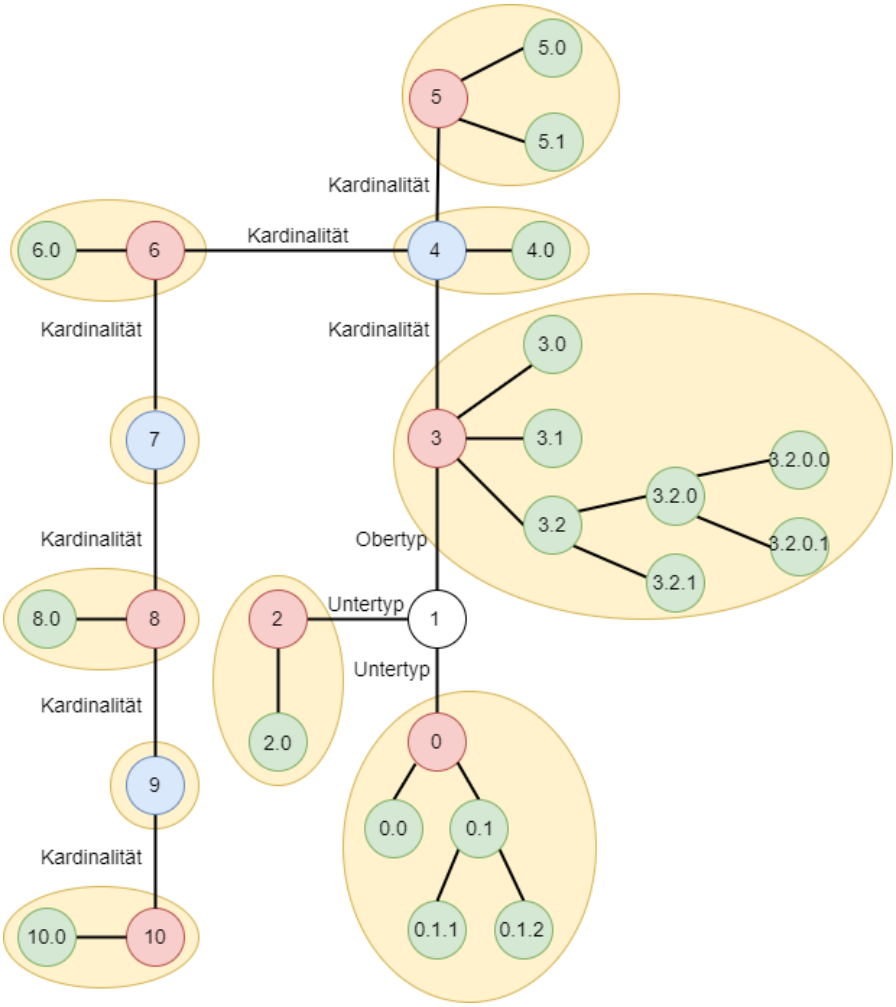

Abbildung 6.3: Weitergehende Analyse des Graphen

Alle Informationen, welche für die Übersetzung von Entitäten, Relationen und IsA-Strukturen benötigt werden, befinden sich somit innerhalb des Subgraphen, bestehend aus den blauen, roten und ungefärbten Knoten.

Währenddessen befinden sich alle notwendigen Attributsinformationen in den Bäumen.

Ein Übersetzungsalgorithmus, welches sich auf die Attribute bezieht, muss sich daher ausschließlich auf die Bäume beziehen. Alle weiteren Algorithmen können auf Basis des Subgraphen durchgeführt werden.

6.2 Transformation von Attributen

Aus der vorherigen Sektion ist bekannt, dass nur Bäume, bestehend aus Attributen, bei der Übersetzung beachtet werden müssen. Ziel der Übersetzung ist es hierbei, eine Attributsstruktur mittels Relationen auszudrücken.

Wird ein Baum, bestehend aus ausschließlich einwertigen Attributen betrachtet, so ist dies trivial lösbar, indem eine Relation für die Entität erstellt wird und jedes Blatt des Baumes der Relation hinzugefügt wird.

Algorithmisch kann hierbei der Baum im Postorder durchlaufen werden und geprüft werden, ob es sich bei dem aktuellen Knoten um ein Blatt handelt.

Dieser Durchlauf ist in der Abbildung 6.4 dargestellt.

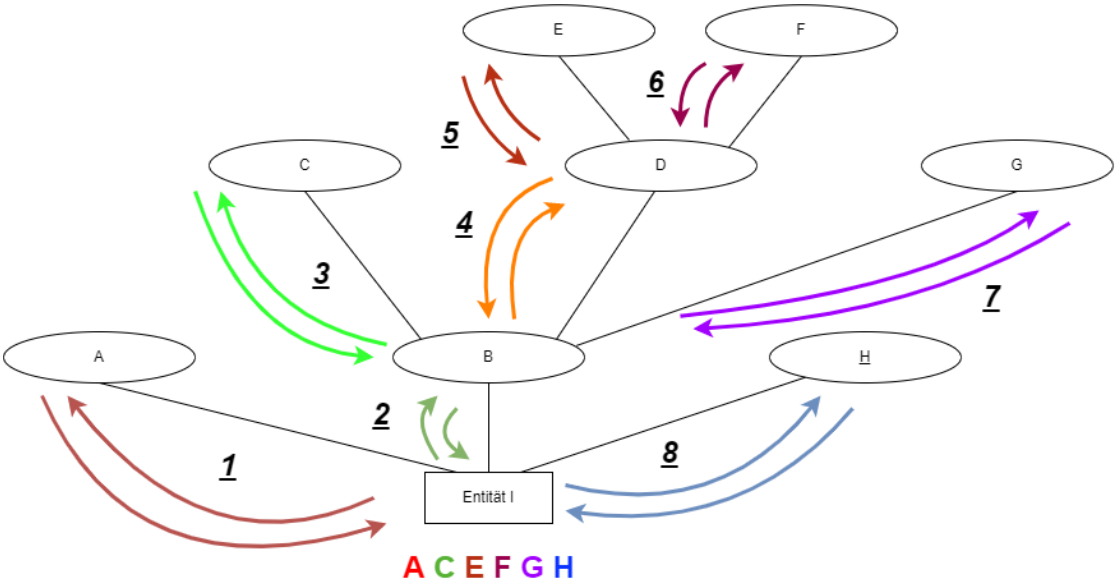

Abbildung 6.4: Übersetzung von einwertigen Attributen

Beinhaltet ein Baum jedoch mehrwertige Attribute, so können diese der Relation nicht hinzugefügt werden, da eine Relation ein festes Schema besitzt und mehrwertige Attribute beliebig viele Werte annehmen können.

In diesem Fall ist eine eigenständige Relation zu erstellen und eine Referenz zwischen dieser und der Relation der Entität wird erstellt.

Bei zusammengesetzten Attributsstrukturen muss auf den Spezialfall geachtet werden, zwischen welchen Relationen eine Referenz erstellt wird. Hierbei muss es sich nicht zwangsweise um die Relation der Entität handeln.

Algorithmisch kann hierfür zu Beginn für jedes Attribut eine Relation erstellt werden. Wird darauffolgend der Baum im Postorder durchlaufen, so wird bei einem mehrwertigen Attribut eine Referenz zwischen dieser und der Elternrelation erstellt. Bei einwertigen Attributen wird die Relation mit der Relation des Elternknotens zusammengeführt. Hierbei werden alle Attribute und Referenzen der Kindrelation in die Relation des Elterknotens übertragen.

In der Abbildung 6.5 ist dieser algorithmische Durchlauf abgebildet. Der Spezialfall tritt in dieser Abbildung nicht auf.

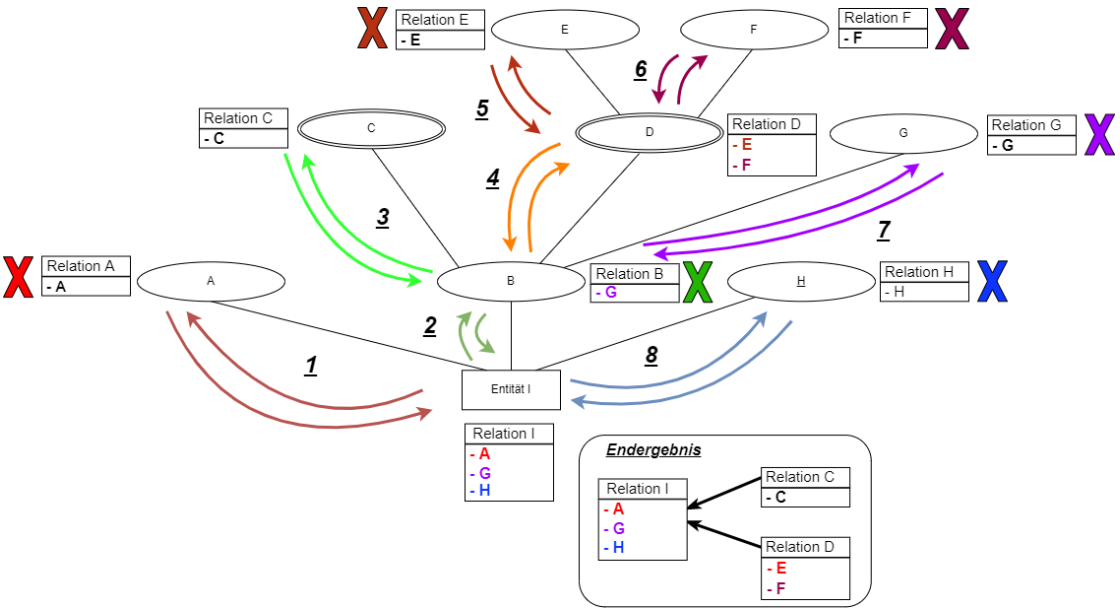

Abbildung 6.5: Übersetzung von ein- und mehrwertigen Attributen

Wird das einwertige Attribut F durch ein mehrwertiges Attribut ersetzt, so tritt der oben genannte Spezialfall auf. Der Spezialfall ist in Abbildung 6.6 abgebildet.

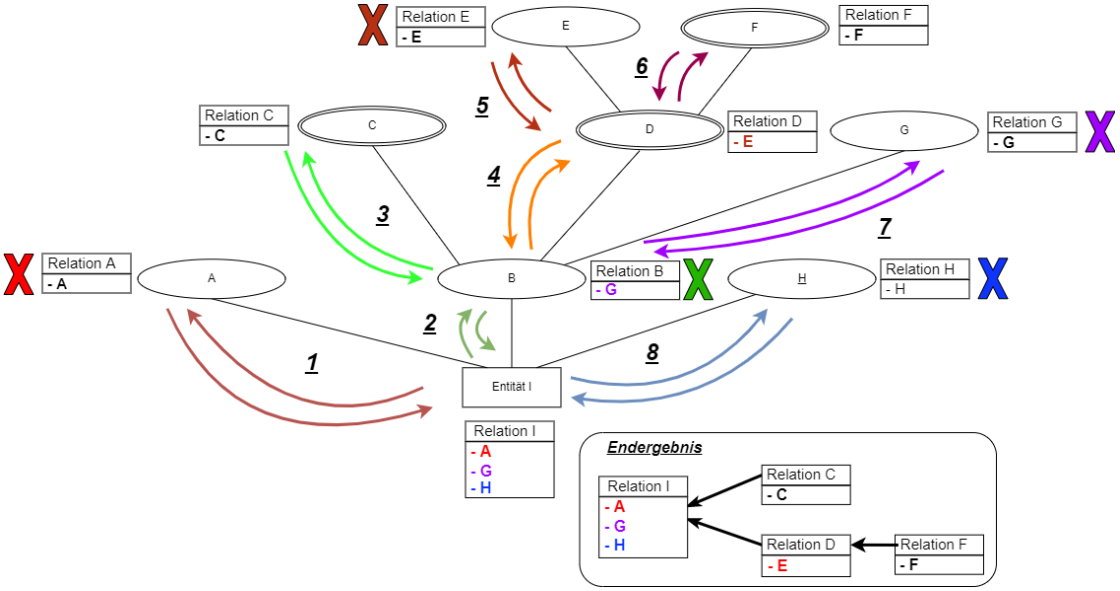

Abbildung 6.6: Übersetzung von komplexen Attributsstrukturen

Ein Attributswert D besteht hierbei aus vielen Attributswerten F und einem Arttributswert E. Eine direkte Referenz zwischen der Entitätrelation und der Relation F könnte diesen Sachverhalt nicht darstellen.

Im Algorithmus kann die Erweiterung vorgenommen werden, dass zusammengesetzte Attribute, welche nur aus einem weiteren mehrwertigen Attribut bestehen übersprungen werden.

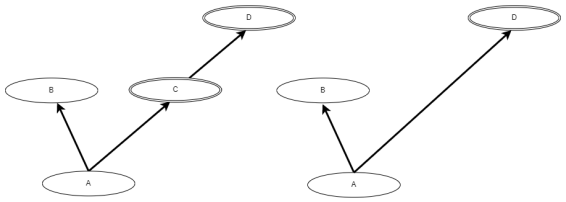

Abbildung 6.7: Überspringen von einfachen zusammengesetzten Attributen

Im Quelltext 6.1 ist der Algorithmus zur Übersetzung von Attributen im Pseudocode dargestellt. Der Algorithmus wird hierbei für jede Entität und jede Beziehung im ER-Graphen ausgeführt.

```
1  Function TransformAttributeTree (Parent)
2
3      //Execute postorder traversal
4      For Each Child in TreeNode
5          TransformAttributeTree (Child)
6      End For
7
8      If TreeNode is Multivalued Attribute Then
9          TreeNode.Table <- marked
10     End If
11
12     //Recursion resolution
13     If TreeNode is Leaf Then
14         Return
15     End If
16
17     For Each Child in TreeNode
18
19         If Child.Table is marked Then
20             If TreeNode.Children.Size = 1 Then
21                 //Handling of "forwarding" attributes
22                 MergeChildTableInParentTable(TreeNode.Table,
                       Child.Table)
23                 TreeNode.Table <- marked
24             Else
25                 TreeNode.Table.References.Add(Child.Table)
26             End If
27
28         Else
29             MergeChildTableInParentTable(TreeNode.Table,
                   Child.Table)
30         End If
31     End For
32
33 End Function
```

Quelltext 6.1: Transformation von Attributen

Um die Übersetzung der Attribute in das relationale Modell vollständig abzuschließen, müssen die Referenzen zwischen den Tabellen in Form von Fremdschlüsselabhängigenkeiten abgebildet werden.

Hierfür können die verbleibenden Relationen im Preorder durchlaufen werden und den Relationen Primärschlüssel hinzugefügt werden. Diese agieren zudem als Fremdschlüssel auf die Primärschlüssel der referenzierten Relation. Dieses kaskadieren der Fremdschlüssel ist im Quelltext 6.2 im Pseudocode dargestellt.

```
Function CascadeForeignKeys(ParentTable)

    For Each ChildTable in ParentTable.References
        For Each PrimaryKey in ChildTable

            //Add foreign key as primary key to the child table,
            //referencing the parenttables primary key
            AddForeignKeyAsPrimaryKey(ParentTable, ChildTable)
        End For

        //Execute preorder traversal
        CascadeForeignKeys(ChildTable)
    End For
End Function
```

Quelltext 6.2: Kaskadieren der Primary Keys

6.3 Transformation von IsA-Strukturen

Die Transformation von IsA-Strukturen in das relationale Modell wird mittels Fremdschlüsselabhängigkeiten zwischen Unter- und Obertypen der IsA-Struktur realisiert.

Hierbei ist zu beachten, dass die Relationen der Entitäten existieren müssen und sich darin die Primärschlüssel befinden. Aufgrund dessen muss die Transformation der Attribute vor der Transformation von IsA-Strukturen erfolgen.

Jeder Untertyp einer IsA-Struktur erbt alle Primärschlüssel des Obertyps. Zusätzlich verweisen diese als Fremdschlüssel auf den Obertyp. Verkettungen an IsA-Strukturen müssen hierbei von "Oben nach Unten" übersetzt werden, um sicherzustellen, dass Entitäten unterer Ebenen alle Primärschlüssel erhalten.

Die Abbildung 6.8 stellt auf der linken Seite das Entity-Relationship Modell und auf der rechten Seite das relationale Modell dar.

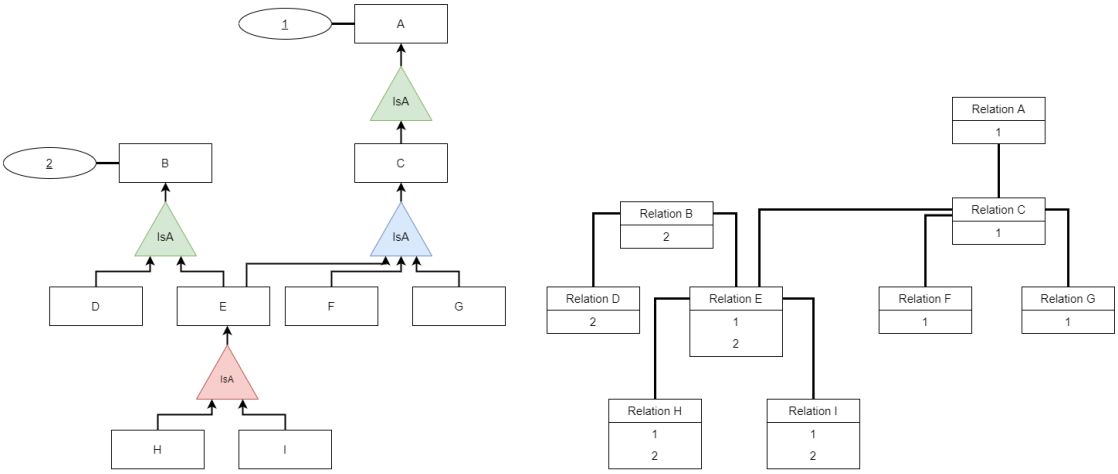

Abbildung 6.8: Übersetzung von IsA-Strukturen

Bei der Übersetzung der gezeigten Grafik bedeutet die Übersetzung von "Oben nach Unten", dass die rot hervorgehobene IsA-Struktur erst nach den blauen und grünen IsA-Strukturen übersetzt werden darf. Die Blaue hingegen erst nach der grünen IsA-Struktur.

Algorithmisch kann dies einfach realisiert werden, indem alle IsA-Strukturen bis zu N mal durchlaufen werden. N ist hierbei die Anzahl der IsA-Strukturen. Erbt der Obertyp der ausgewählten IsA-Struktur von weiteren IsA-Strukturen und sind diese noch nicht übersetzt, so wird der aktuelle Durchlauf übersprungen. Dies gilt auch, wenn die ausgewählte IsA-Struktur bereits übersetzt wurde.

Im Quelltext 6.3 ist der algorithmische Ablauf der Übersetzung als Pseudocode dargestellt.

```
1  Function TransformIsAs(Graph)
2      i <- 0
3      AmountIsAs = Graph.IsAStructures.Size
4      While i < AmountIsAs do
5          For Each IsAStructure in Graph
6              TransformIsAStructure(IsAStructure)
7          End For
8          i++
9      End While
10 End Function
11
12 Function TransformIsAStructure(IsAStructure)
13
14     If IsAStructure is transformed Then
15         Return
16     End If
17
18     UpperLayerIsAs <- GetInheritedIsAs(IsAStructure.SuperType)
19     UnhandledUpperLayerIsAs <- UpperLayerIsAs which are not
           transformed
20
21     If UnhandledUpperLayerIsAs not empty Then
22         Return
23     End If
24
25     For Each SubType in IsAStructure.Subtypes
26
27         For Each PrimaryKey in SuperType
28             AddForeignKeyAsPrimaryKey(Supertype, Subtype)
29         End For
30     End For
31
32     IsAStructure <- isTransformed
33 End Function
```

Quelltext 6.3: Übersetzung von IsA-Strukturen

6.4 Transformation schwacher Typen

Für die Übersetzung von schwachen Typen in das relationale Modell ist Voraussetzung, dass für alle Entitäten und Beziehungen Relationen existieren und sich darin bereits alle Attribute befinden. Es ist daher die Transformation von Attributen und IsA-Strukturen zuvor durchzuführen. IsA-Strukturen müssen hierbei zuvor übersetzt werden, da eine starke Entität, von der eine schwache Entität abhängt, während der Übersetzung weitere Primärschlüssel erhalten kann.

Der Übersetzungsvorgang von schwachen Typen wird anhand des folgenden ER-Diagramms in Abbildung 6.9 erläutert.

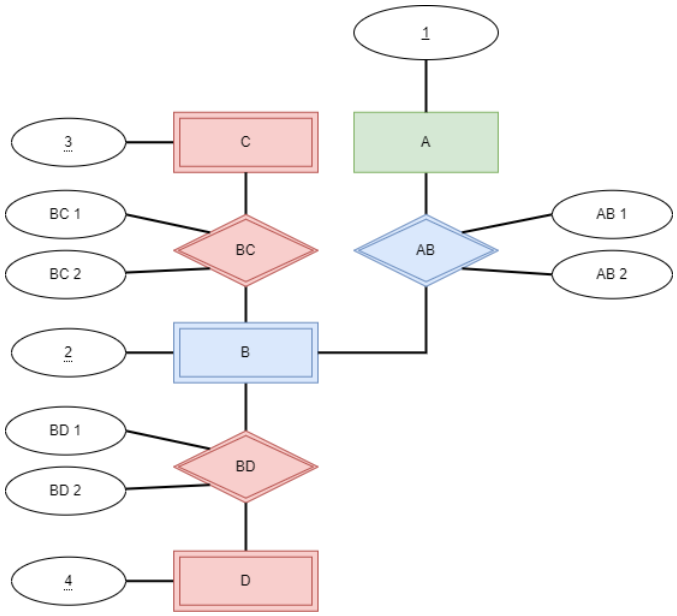

Abbildung 6.9: Ausgangsdiagramm zur Übersetzung schwacher Typen

Zuerst ist bei der Transformation die Übersetzungsreihenfolge zu beachten. Der Algorithmus setzt sich hierbei nach der Übersetzung der grün hervorgehobenen, starken Entität A an.

Die Übersetzung ist hierbei immer ausgehend von schwachen Entitäten, welche mittels einer schwachen Beziehung eine Verbindung zu einer starke Entität oder einer bereits übersetzten schwachen Entität besitzen. Die Relation der schwachen Beziehung wird hierbei immer mit der Relation der ausgehenden Entität zusammengeführt. Folglich werden in der oberen Abbildung zunächst die blauen Elemente und drauffolgend die roten Elemente übersetzt.

Während der Übersetzung hält die Relation der zu übersetzenden Entität dabei eine Referenz zu der starken oder bereits übersetzten Entität. Mittels dieser können im Anschluss die Fremdschlüsselabhängigkeiten erstellt werden.

Der Übersetzungsvorgang des in Abbildung 6.9 darstellten ER-Diagramms ist in der Abbildung 6.10 abgebildet. Die Abbildung setzt unmittelbar nach der Ausführung der Algorithmen zur Übersetzung von Attributen und IsA-Strukturen an.

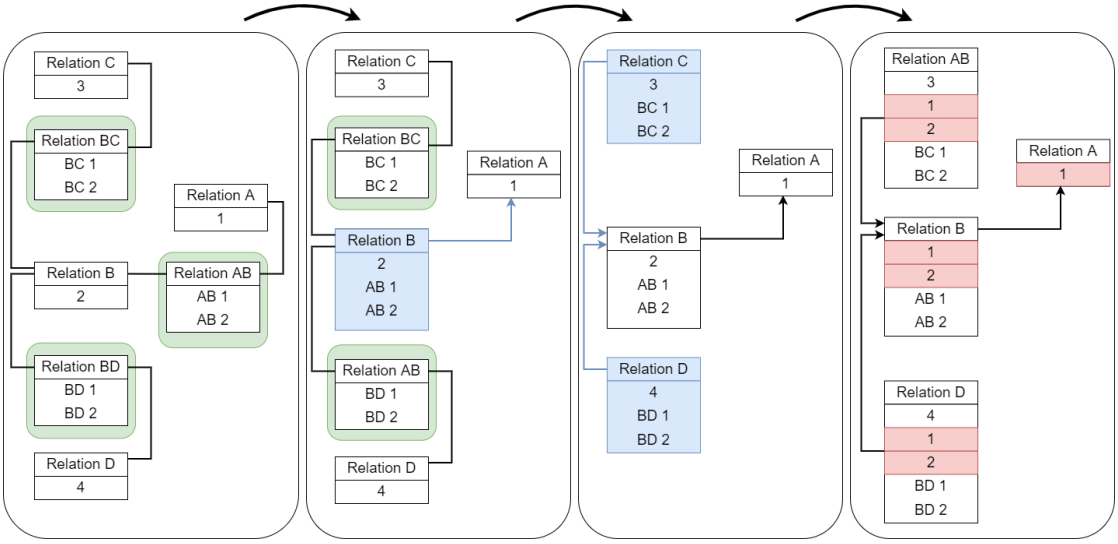

Abbildung 6.10: Übersetzungsvorgang schwacher Typen

Algorithmisch ist die Zusammenführung der Relationen und die Referenzerstellung aus Abbildung 6.10 im Quelltext 6.4 als Pseudocode dargestellt. Die Einhaltung der Reihenfolge erfolgt in diesem, analog zur Übersetzung von IsA-Strukturen, durch mehrmaliges Überprüfen aller schwachen Entitäten.

```
1  //Execute function for each weak entity, N times
2  //With N equal to the amount of weak entites in the graph
3  Function TransformWeakEntity(WeakEntity)
4      If WeakEntity is transformed Then
5          Return
6      End If
7
8      WeakRelations <- GetConnectedWeakRelations(WeakEntity)
9      For Each WeakRelation in WeakRelations
10
11         OtherEntity <- GetOtherEntity(WeakEntity)
12         If OtherEntity is no StrongEntity and is not transformed
               Then
13             Continue
14         End If
15
16         WeakEntity.References.Add = OtherEntity
17         MergeTables(WeakEntity,  IdentifyingRelation)
18         Return
19     End For
20 End Function
```

Quelltext 6.4: Übersetzung von schwachen Typen

Die Kardinalitäten der schwachen Beziehung sind für die grundlegende Überführung nicht zu beachten, da es sich bei diesen nur um 1:1 und N:1 Beziehungen zum identifizierenden Typen hin handeln kann. Beide Funktionalitäten werden mittels einer Fremdschlüsselabhängigkeit realisiert.

Hierbei ist anzumerken, dass in einem weiteren, über diese Arbeit hinausgehenden, Schritt auf Basis der Kardinalitäten, weitere Inklusionsabhängigkeiten erstellt werden können.

6.5 Transformation von Beziehungen

Die Transformation von Beziehungen setzt die vorherige Durchführung aller vorhergehenden Algorithmen voraus, da alle Relationen mit den vollständigen Primärschlüsseln für Beziehungen, Entitäten und schwache Typen Voraussetzung zur Übersetzung von Beziehungen sind.

Bei der Übersetzung sind dabei drei Fälle zu beachten.

Besitzt eine Beziehung mehrere Entitäten oder ist die Funktionalität N:M, so werden der Beziehung die Primärschlüssel aller verbundenen Entitäten hinzugefügt. Diese verweisen sodann als Fremdschlüssel auf die Primärschlüssel der Entitäten.

Ist die Funktionalität der Beziehung 1:N, so wird die Beziehung aufgelöst. Hierbei wird die Beziehung mit der Entität auf der N-Seite zusammengeführt. Zudem erhält diese zusammengeführte Relation alle Primärschlüssel der gegenüberliegenden Entität als normale Attribute. Diese agieren als Fremdschlüssel auf die gegenüberliegende Entität.

Im letzten Fall ist die Beziehung 1:1. Die Übersetzung ist hierbei analog zu 1:N Beziehungen durchzuführen, wobei die Entität, welche die Fremdschlüssel und Beziehungsattribute erhält, hierfür festgelegt werden muss. Da in dem ER-Diagramm dieser Arbeit, die Min-Max Notation verwendet wird, ist die Optionalität zu beachten um Nullwerte zu vermeiden. Beschreibt eine der Kardinalitäten eine Optionalität, so werden der Entität auf der anderen Seite die Attribute der Beziehung und Fremdschlüssel hinzugefügt. Beschreiben beide oder keine der Kardinalitäten eine Optionalität, so ist diese beliebig wählbar.

In der Abbildung 6.11 sind die genannten Fälle dargestellt. Auf der linken Seite befindet sich das ER-Modell. Auf der rechten Seite das, durch die Übersetzung der ER-Modells entstehende, relationale Modell.

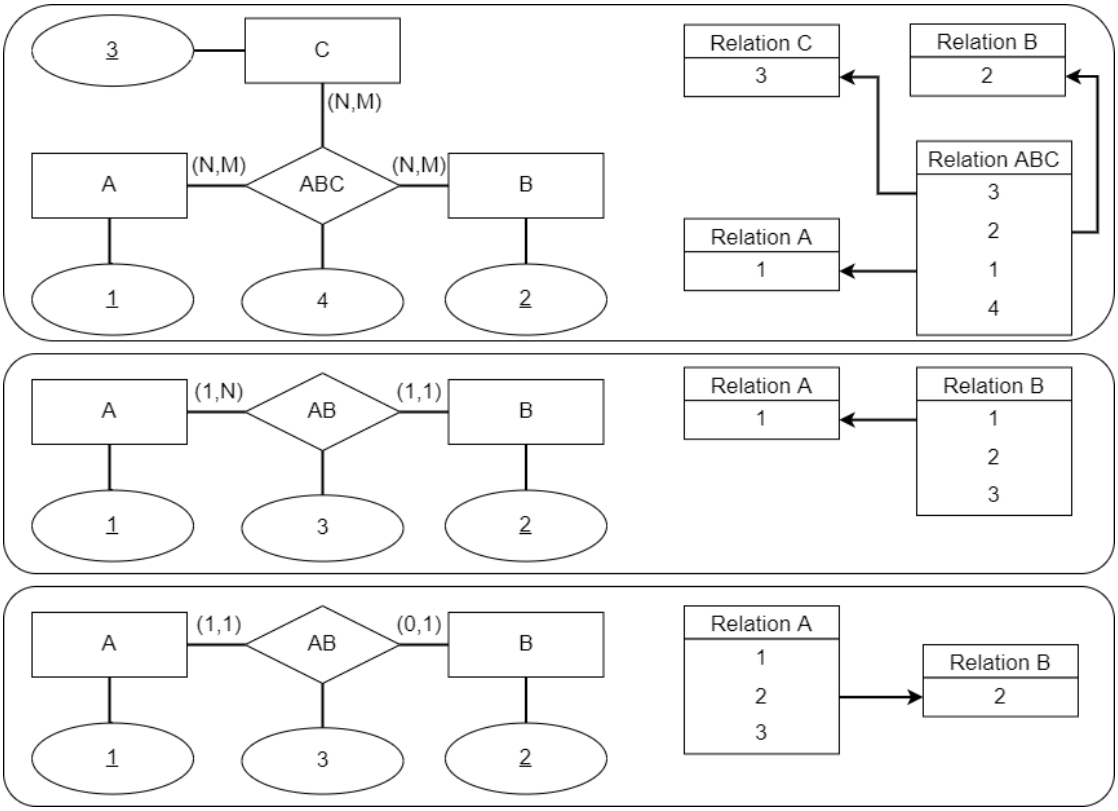

Abbildung 6.11: Übersetzung von Beziehungen

6.6 SQL-Code Generierung auf Basis des relationalen Modells

Mittels der, in den vorhergehenden Abschnitten erläuterten, Algorithmen ist es möglich das Entity-Relationship Modell in ein relationales Modell zu überführen.

Auf Basis des relationalen Modells und einer zusätzlichen Angabe von Datentypen durch den Benutzer ist es möglich einen SQL-Code Generator zu entwickeln.

Zur Erstellung des SQL-Codes für eine Tabelle kann eine Art "Template" verwendet werden. Eine aus dem relationalen Modell entnommene Relation befüllt dieses mit den Daten, wie dem Attributsnamen oder dem Datentyp. Im Quelltext 6.5 ist ein beispielhaftes Template abgebildet.

```
1  CREATE TABLE IF NOT EXISTS <Tablename> (
2    <Attributename> <Datatype>,
3    <Attributename> <Datatype>,
4    ...
5    <Attributename> <Datatype>,
6    PRIMARY KEY (<Attributename>, ... ,<Attributename>),
7    FOREIGN KEY (<Attributename>)
8      REFERENCES <ReferencedTable> (<Attributename>),
9    ...
10   FOREIGN KEY (<Attributename>)
11     REFERENCES <ReferencedTable> (<Attributename>)
12 );
```

Quelltext 6.5: Beispiel eines SQL-Templates

Um eine SQL-Datei mit allen Tabellendefinitionen zu erzeugen ist es notwendig, die einzelnen SQL-Codeteile in eine richtige Reihenfolge zu bringen.

Dies liegt darin begründet, dass die SQL-Statements Fremdschlüssel beinhalten können, welche andere Tabellen referenzieren. Die referenzierten Tabellen müssen hierbei bereits zuvor mittels eines SQL-Statements erstellt worden sein.

Um dies zu erreichen kann die im Grundlagenkapitel vorgestellte topologische Sortierung verwendet werden. Hierfür wird ein Graph erstellt, bei dem jede Relation des relationalen Modells einen Knoten repräsentiert und jede Fremdschlüsselabhängigkeit eine gerichtete Kante zu einem Knoten.

Das Ergebnis des Algorithmus ist dann eine Reihenfolge an Tabellen, bei der jede Tabelle vorkommt, bevor diese durch Fremdschlüsselabhängigkeiten anderer Tabellen referenziert wird, sofern dies möglich ist.

Die Tabellen können anschließend, unter Einhaltung der Sortierreihenfolge, verwendet werden, um das oben genannte Template zu befüllen. Die daraus entstehenden SQL-Statements können dann unmittelbar aneinander angehängt werden.

7 Implementierung

7.1 Architekturentscheidungen

Aus der vorangegangen Anforderungsanalyse wurde festgelegt, dass die Anwendung als Webanwendung zu realisieren ist. Des Weiteren wurde die nicht funktionale Anforderung der Modularität an die Anwendung gestellt.

Aufgrund dessen wurde die Anwendung mittels einer Client-Server Architektur implementiert. Die Clientanwendung implementiert hierbei das Zeichentool, mit welcher der User interagiert. Die Serveranwendung stellt für die Clientanwendung HTTP-Endpunkte zur Verfügung, mittels denen die Übersetzungsvorgänge durchgeführt werden.

Diese Architektur bietet den Vorteil, dass die Übersetzungslogik vom Entity-Relationship Modell in das relationale Modell und die darauf aufbauende SQL-Generierung vollständig von der Logik des Zeichentools getrennt wird.

Bezogen auf die nicht funktionale Anforderung ermöglicht dies des Weiteren den Übergang dieser Architektur in eine Service- oder Microservicearchitektur. Die Implementierung neuer Anforderungen wie zum Beispiel weitere Übersetzungen des Entity-Relationship Modells in andere Modelle kann hierbei durch die Implementierung weiterer eigenständiger Serveranwendungen erfolgen.

Hierdurch kann der Umfang und die Komplexität der Clientanwendung und der einzelnen Serveranwendungen gering gehalten werden. Darüber hinaus ermöglicht dies die Implementierung neuer Anforderungen in einer hierfür geeigneten Sprache.

7.2 Clientanwendung

Die Implementierung des Zeichentools erfolgte unter der Verwendung von Javascript, der JavaScript Bibliothek React und dem Zustandscontainer Redux. Die Clientanwendung wurde nicht für mobile Endgeräte optimiert. Der in der Entwicklung genutzte und unterstützte Browser ist Google-Chrome der neuesten Version 104.0.5112.81.

Mittels React ist es möglich, das Zeichentool in einzelne Komponenten zu zerlegen. Durch die Entwicklung wiederverwendbarer Komponenten kann auch innerhalb des Zeichentools eine hohe Modularität gewährleistet werden. Durch die Verwendung des Redux Store ist es möglich, den Zustand der Anwendung von den React-Komponenten loszulösen.

Spezifische, darstellungsunabhängige Logik wird mittels Services implementiert. Diese exportieren jeweils ein Javascript Objekt, welches notwendige Funktionen zur Verfügung stellt.

Folgend wird zunächst ein erster Prototyp mit den wichtigsten Interaktionsmöglichkeiten vorgestellt. Daraufhin erfolgt eine Zerlegung des Zeichentools in React-Komponenten. Darauf aufbauend werden Zusammenhänge dieser Komponenten erläutert.

7.2.1 Prototyp des Zeichentools

Um die Anforderungen an das Zeichentool grafisch festzuhalten und grundlegende Interkationsmöglichkeiten festzustellen, wurde ein Prototyp der Benutzerschnittstelle mittels des Wireframe-Tools Figma[7] entwickelt. Bei dem Design des Prototypen wurde sich an bestehenden populären Modellierungstools wie dem Enterprise Architect[21] oder DrawIo[9] orientiert.

Die Abbildung 7.1 zeigt einen Auschnitt des UI-Prototypen.

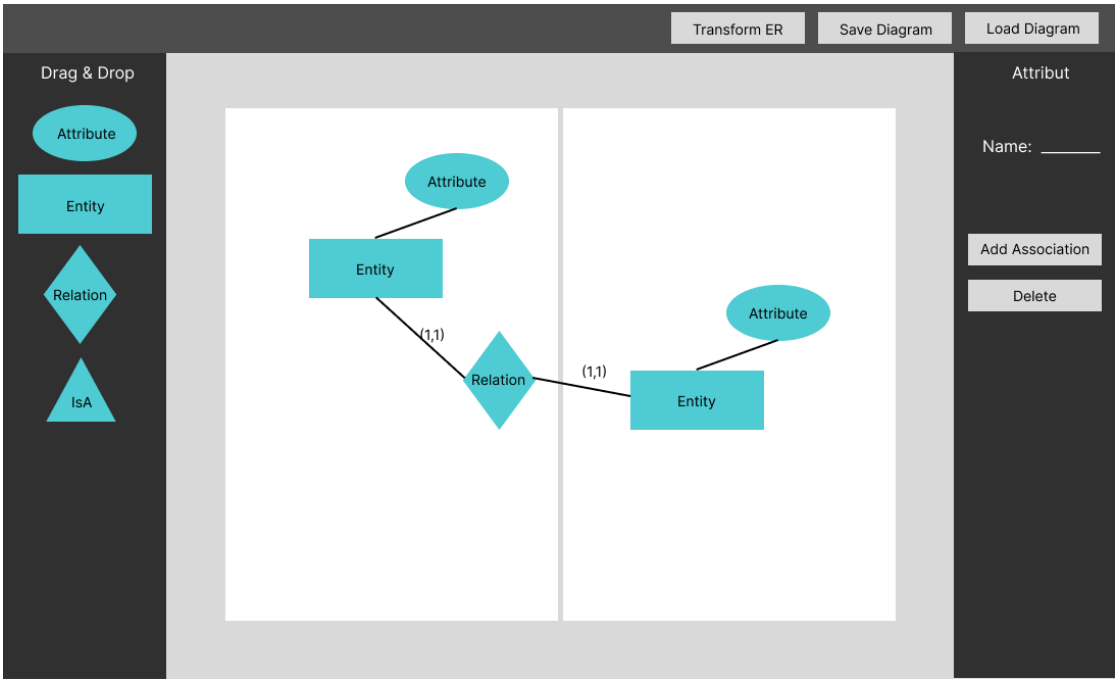

Abbildung 7.1: Prototyp des Zeichentools

In der Abbildung ist zunächst auf der linken Seite eine Darg & Drop Bar zu sehen. Mittels dieser sollen Elemente in die rechts daneben liegende Zeichenfläche gezogen werden.

Mit der Verwendung der Drag & Drop Bar soll damit eine schnelle und intuitive Erstellung von Elementen ermöglicht werden.

Wird ein Element selektiert, so wird auf der linken Seite eine Seitenbar sichtbar, welche zusätzliche Optionen bereitstellt. Bei diesen handelt es sich zum Beispiel um das vergeben eines Namens, das Löschen des Elements oder das Erstellen einer Verbindung zu einem anderen Element.

Oberhalb des Zeichenbretts sind auf der rechten Seite drei Buttons zu sehen. Mit dem ersten Button soll es ermöglicht werden, das gezeichnete ER-Diagramm in ein relationales Modell zu transformieren. Das relationale Modell wird anschließend in einem weiteren Tab dem Benutzer dargestellt.

Mittels des zweiten Buttons soll das aktuelle Diagramm in Form einer Textdatei abgespeichert werden. Durch Verwendung des dritten Buttons kann die Textdatei zu einem späteren Zeitpunkt wieder importiert werden.

Es wurde sich für das Speichern und Laden in Form von Textdateien entschieden, da somit keine Benutzerverwaltung implementiert werden muss.

Hierdurch entsteht der Vorteil, dass die Verwendung dieser Funktionalität erlaubt werden kann, ohne einen Benutzeraccount anlegen zu müssen. Zusätzlich wird es damit ermöglicht, die in den Textdateien abgespeicherten benutzererstellten Modelle über externe Medien an weitere Personen zu verteilen.

Für die Editierung des relationalen Modells wurde kein Prototyp entworfen, da sich diese auf die Selektion einer Tabelle mit anschließender Eingabe von Datentypen beschränkt. Die Anzeige des SQL-Codes erfolgt über ein einfaches Pop-Up, welches über einen Button in der relationalen Ansicht geöffnet werden kann.

7.2.2 React-Komponenten

Folgend wird die Zerlegung der Zeichentools in React-Komponenten erläutert. Da diese in der Regel grafisch in der Anwendung wiederzufinden sind, ist zunächst die implementierte Benutzerschnittstelle der Clientanwendung zu betrachten.

Diese ist auf Interaktionsbasis gleich mit dem bereits gezeigten Prototypen. Es wurden zusätzlich Tabs auf der oberen linken Seite eingefügt, um direkt zwischen den Modellen zu wechseln. Unterhalb der Zeichenfläche wurde zudem ein Icon hinzugefügt, welches den Benutzer darüber informiert, ob das ER-Diagramm valide ist oder falls Verletzungen vorliegen, diese dem Benutzer über ein Tooltip anzeigt. Das Design ist während der Implementierung schrittweise verfeinert worden.

Die implementierte Benutzerschnittstelle ist in der Abbildung 7.2 und 7.3 dargestellt. Erstere zeigt die Ansicht des ER-Modells. Die Zweite die Ansicht des relationale Modells.

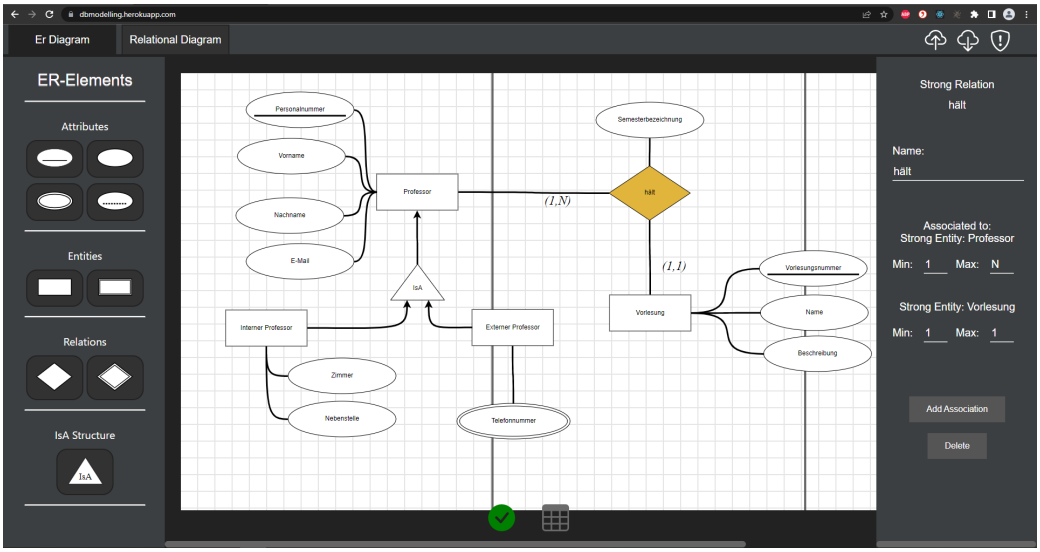

Abbildung 7.2: Ausschnitt des Zeichentools, Ansicht des ER-Modells

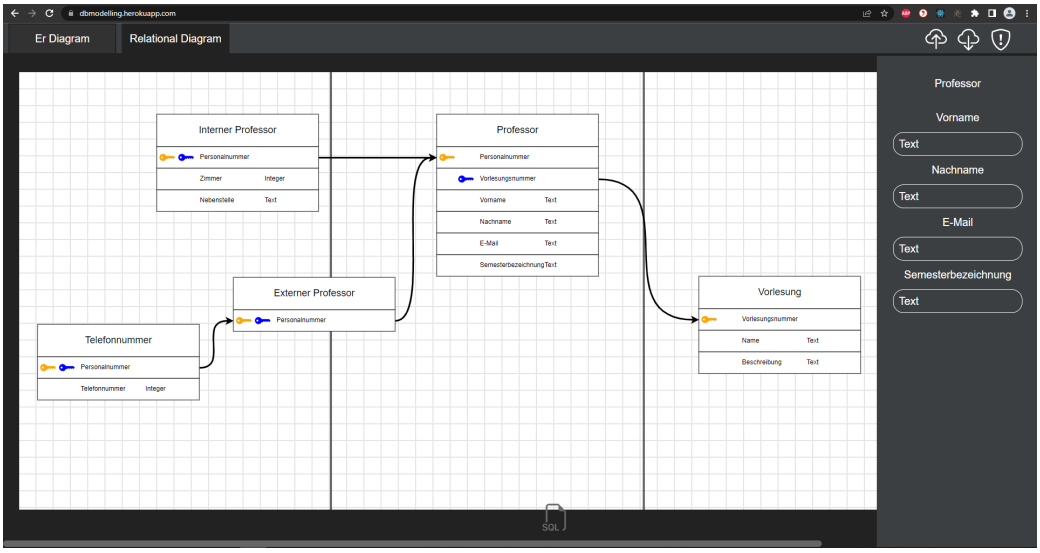

Abbildung 7.3: Ausschnitt des Zeichentools, Ansicht des relationalen Modells

Die gezeigten Benutzerschnittstellen wurden mittels der React-Komponenten aus Abbildung 7.4. implementiert. Grün hervorgehobene Komponenten sind exklusiv Teil der ER-Ansicht. Rot hervorgehobene Elemente Teil der relationalen Ansicht. Die Angabe unterhalb der Komponenten gibt an, wie oft dieses gerendert wird.

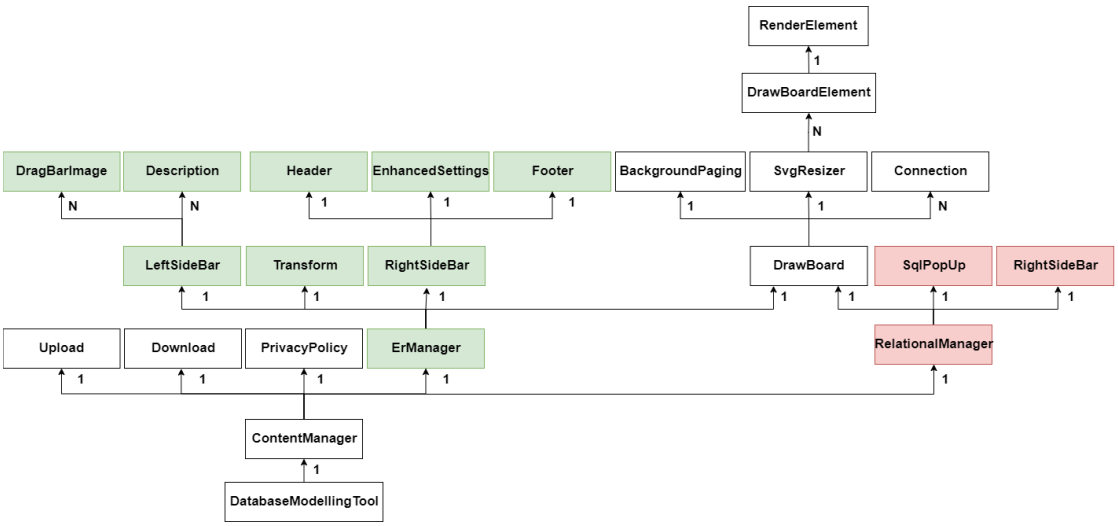

Abbildung 7.4: React-Komponenten Darstellung des Zeichentools

In Abbildung 7.5 werden die React-Komponenten visuell anhand der Ansicht des ER-Modells gezeigt.

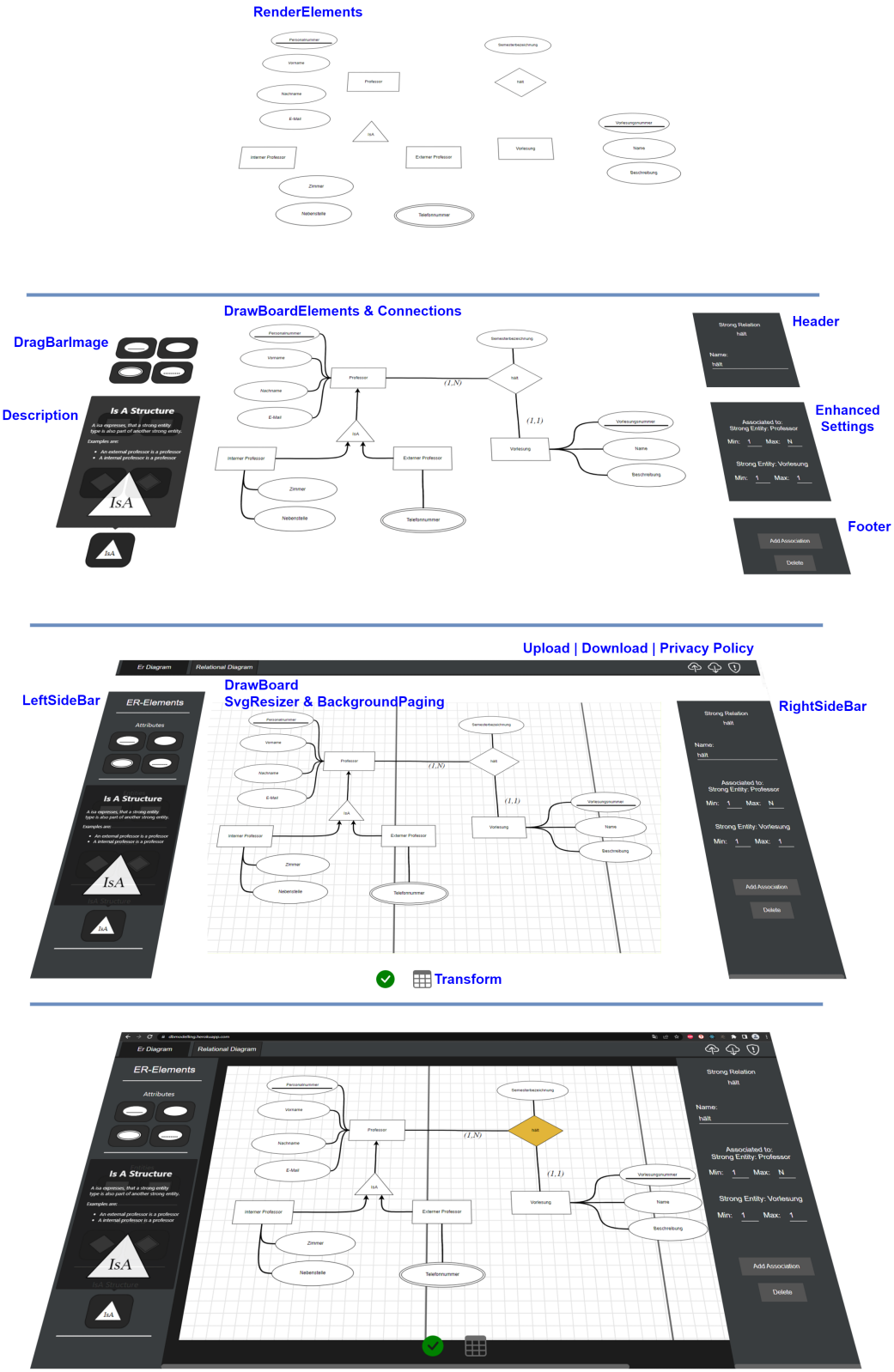

Abbildung 7.5: Visuelle Zerlegung des Zeichentools in Komponenten

7.2.3 Beziehungen zwischen den React-Komponenten

Alle React-Komponenten werden ausgehend von der Komponente DatabaseModellingTool aufgerufen. Dieses legt fest, welches Modell angezeigt werden soll und gibt die Information an die Kindkomponente ContentManager weiter.

Der ContentManager wickelt alle Aufrufe an Endpunkte des Servers ab. Zudem ist dieser für die Download- und Uploadfunktionalität mittels einer Textdatei verantwortlich.

Zur Implementierung dieser Funktionalitäten greift dieser auf den Redux-Store zu. Der Store besteht hierbei aus zwei voneinander getrennten Slices. Der Erste hält die Informationen für DrawBoardElemente und Connections im ER-Modell und bietet eine Reihe von Funktionen an, um die Informationen zu modifizieren. DrawBoardElemente repräsentieren im ER-Modell ER-Elemente wie z.B. starke Entitäten oder mehrwertige Attribute. Connections entsprechen Assoziationen, Verbindungen von Attributen, sowie Unter- und Obertypverbindungen von IsA-Strukturen. Der Zweite Slice ist für den Zustand des relationalen Modells verantwortlich und ist analog aufgebaut. DrawBoardElemente repräsentieren in diesem Modell Tabellen und Connections Fremdschlüsselabhängigkeiten zwischen den Tabellen.

Der ContentManager rendert den ErManager oder RelationalManager auf Basis der übergegebenen Informationen des DatabaseModellingTools. Beide Komponenten sind hierbei für die Businesslogik des jeweiligen Modells verantwortlich. Die Komponenten orchestrieren hierfür eine Reihe von Kindkomponenten.

Darüber hinaus implementieren die ErManager und RelationalManager Komponenten Funktionen unter der Verwendung des jeweiligen Redux-Slice. Diese Funktionen handhaben alle Interaktionsmöglichkeiten mit dem korrespondierenden Modell. Im Falle des ER-Modells sind hierbei z.B. Funktionen für das Hinzufügen von Verbindungen und Elementen implementiert. Innerhalb dieser Funktionen werden auch Services referenziert, welche z.B. die Logik des Feedback- und Validierungssystems aus Kapitel 5.3 implementieren.

Die Kindkomponenten werden von der ErManager oder RelationalManager Komponente mittels der implementierten Funktionen und den Daten des Redux-Stores parametrisiert. Die Kindkomponenten selbst besitzen keinen Zugriff auf den Redux-Store. Diese rendern visuell sichtbare HTML-Elemente, registrieren die übergebenen Funktionen als EventHandler auf die gerenderten Elemente und stellen übergebene Daten visuell dar.

Dieser Zusammenhang wird in der Abbildung 7.6 anhand der ER-Modell Komponenten dargestellt.

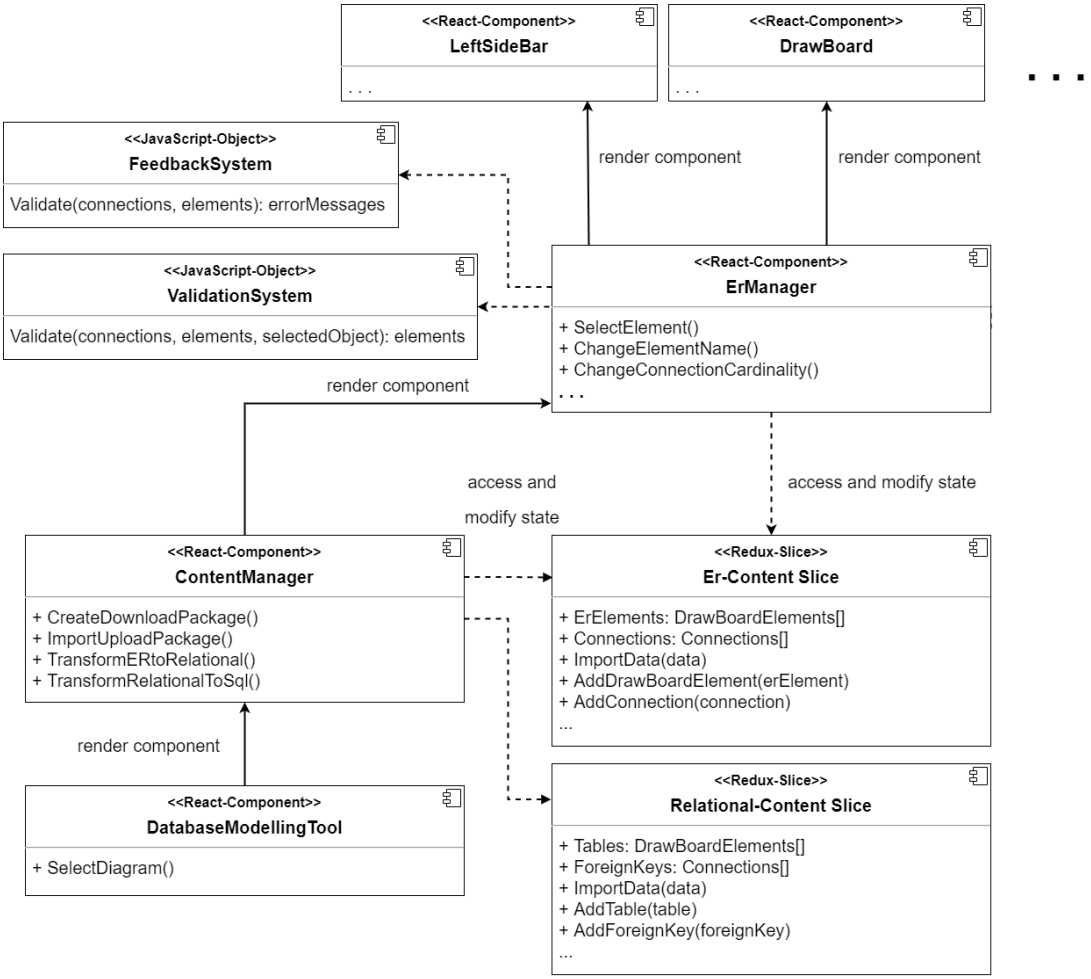

Abbildung 7.6: Beziehungen zwischen den Komponenten

Das Zeichenbrett selbst mit der Darstellung der Elemente und Verbindungen ist mittels einer Reihe von Komponenten realisiert. Dieses wurde aufgrund der Modularitätsanforderung generisch implementiert und wird von den beiden Komponenten ErManager und RelationalManager genutzt.

Die Schnittstelle zum Zeichenbrett ist mittels einer Komponente, dem sogenannten DrawBoard realisiert. Zum Verwenden dieser Schnittstelle sind Javascript-Objekte erforderlich. Diese Objekte benötigen hierbei eine eindeutige Id.

Objekte, welche als Verbindung dargestellt werden sollen, benötigen zusätzlich die Ids der zu verbindenden Objekte. Objekte, welche als Element im Zeichenbrett dargestellt werden, benötigen darüber hinaus einen Typ, welcher einer React-Komponente zugeordnet werden kann. Im Falle des ER-Modells ist dies z.B. eine Komponente zur Darstellung von starken Entitäten. Die im Zeichenbrett darstellbaren Komponenten beschränken sich hierbei auf die Verwendung von SVG-Elementen.

Des Weiteren können die Objekte weitere Attribute besitzen, welche in der Darstellung berücksichtigt werden. Für jedes Element und jede Verbindung ist es zudem möglich, sich auf eine Reihe von Handlern zu registrieren, welche ausgeführt werden, wenn ein korrespondierendes Event geworfen wird.

In der Abbildung 7.7 ist die Struktur der Javascript Objekte darstellt. Im Quelltext 7.1. wird ein exemplarischer Aufruf der DrawBoard-Komponente gezeigt.

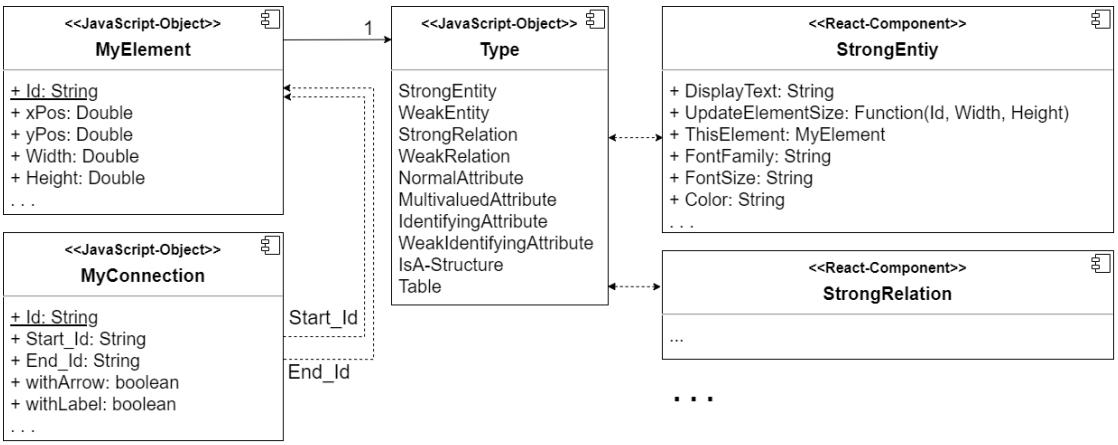

Abbildung 7.7: Struktur von Javascript-Objekten zur Darstellung mittels der DrawBoard Komponente

```
//Render function of a arbitrary component
return (
  <DrawBoard onDropHandler        ={myHandler}
             drawBoardElements    ={AllElements}
             //Apply of additional styles, e.g. a padding of 30px
             drawBoardBorderOffset={30}

    {AllElements.map((myElement) => (
      <DrawBoardElement key={myElement.id}
                        thisElement={myElement}

                    //Register handler for elements
                    onDrawBoardElementSelected     ={myHandler}
                    updateDrawBoardElementPosition ={myHandler}
                    updateDrawBoardElementSize     ={myHandler}
                    ...
      />
    ))}
    {AllConnections.map((myConnection) => (
      <ConnectionElement key={myConnection.id}
                         thisConnection={myConnection}
                         connections={AllConnections}

                    //Register handler for connections
                    onConnectionSelected           ={myHandler}
                    ...
      />
    ))}
  </DrawBoard>
);
```

Quelltext 7.1: Aufruf des DrawBoards

Der interne Aufbau der DrawBoard Komponente erfolgt mittels einer Reihe weiterer Komponenten.

Eine Komponente ist hierbei die BackgroundPaging Komponente. Diese ist dafür zuständig, den Hintergrund des DrawBoards mit weißen, kartierten Seiten zu befüllen. Hierfür ermittelt diese auf Basis der Positionen und Größen der übergebenen Elemente den benötigten Raum und passt die Seitenanzahl entsprechend an. Die Seitenanzahl ist hierbei nach Unten und nach Rechts prinzipiell endlos erweiterbar. Können nicht alle Seiten auf einem Desktop angezeigt werden, so werden Scrollbars zur Navigation eingeblendet. Werden Elemente gelöscht oder verschoben, sodass weniger Raum benötigt wird, wird die Seitenanzahl entsprechend reduziert.

Die Darstellung der Seiten orientiert sich hierbei an Zeichentools wie DrawIo und vermeidet, dass Elemente durch verschieben in einem prinzipiell endlos großen Zeichenbrett verloren gehen.

Die zweite Komponete ist der SVG-Resizer. Dieser rendert die SVG-Viewbox und registriert sich auf Größenänderungen des Bildschirms, sowie auf die Anzahl der dargestellten Seitenanzahlen der ersten Komponente. Bei jeder Änderung prüft dieser, ob die SVG-Viewbox auf dem aktuellen Bildschirm die korrekten Maße besitzt. Dies ist notwendig, da sonst gerenderte Elemente innerhalb des Zeichenbretts nicht dargestellt werden.

Diese Komponente erhält ebenso alle in der Schnittstelle übergebene DrawBoard-Element Komponenten und rendert diese innerhalb der SVG-Viewbox.

Die DrawBoardElemente ermitteln anhand des Typs des übergebenen Javascript-Objekts die darzustellende Komponente. Diese Komponente wird in einen dragbaren Container verpackt und die in der Schnittstelle übergebenen Handler registriert. Des Weiteren wird von dieser Komponente aus auch ein grundlegendes Design an die darzustellende Komponente übergeben.

An die DrawBoard-Komponente übergebene Connection-Komponenten benötigen keine weiteren Anpassungen und können direkt gerendert werden, da die Connection-Komponenten intern die Bibliothek React-X-Arrows [17] nutzen.

In der Abbildung 7.8 ist der beschriebene interne Aufbau dargestellt.

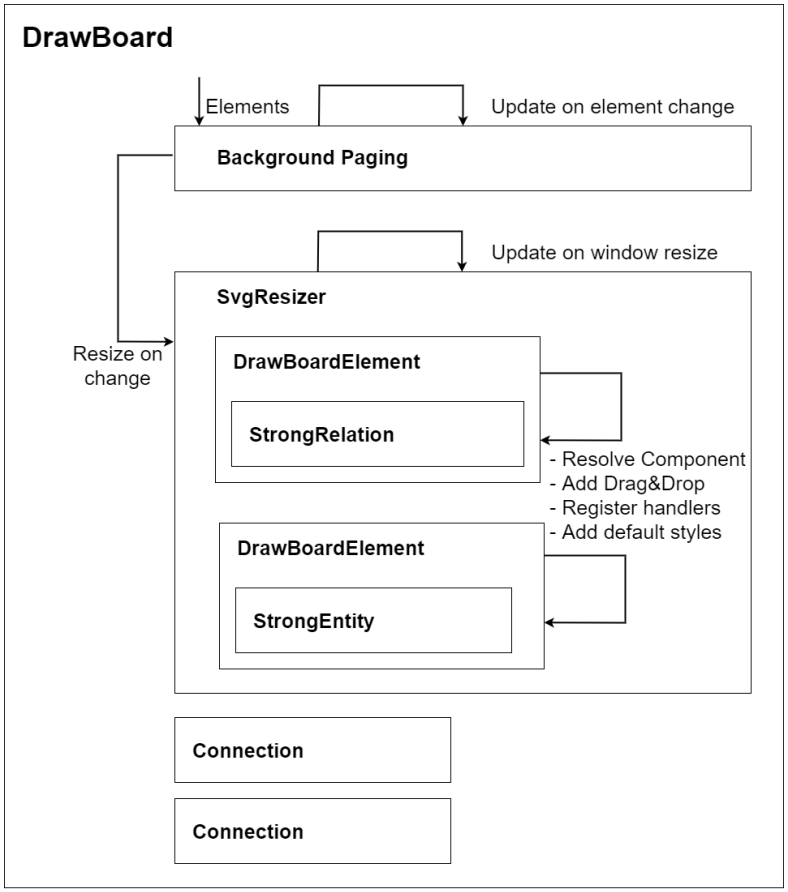

Abbildung 7.8: Interner Aufbau der DrawBoard-Komponente

7.3 Serveranwendung

Die in dieser Arbeit implementierte Serveranwendung zur Überführung des Entity-Relationship Modells in das relationale Modells und die darauf aufbauende SQL-Generierung wurde mittels Java unter der Verwendung von SpringBoot entwickelt. Die in Java gebotene starke Typisierung unterstützt hierbei eine einfachere fehlerunanfälligere Implementierung der Übersetzungslogik.

Die Serveranwendung bietet hierbei für beide Übersetzungsvorgänge jeweils einen Endpunkt an.

Die Implementierung der Endpunkte wird in den folgenden Sektionen erläutert.

7.3.1 Implementierung der Übersetzung des ER-Modells in das relationale Modell

Die Überführung des ER-Modells in das relationale Modell lässt sich grundsätzlich in zwei Phasen unterteilen. In einer ersten Phase wird auf Basis des vom Client übergebenen ER-Modells, in Form eines Data-Transfer-Objects kurz DTO, eine Datenstruktur aufgebaut. Die hierfür genutzte Datenstruktur wurde in Kapitel 6.1 vorgestellt. In der zweiten Phase werden die in dem Kapitel 6.1 entwickelten Übersetzungsalgorithmen auf die Datenstruktur angewandt und das relationale Modell über ein weiteres DTO an die Clientanwendung gesendet.

Im Folgenden werden diese beiden Phasen näher erläutert.

Aufbau der Datenstruktur

Der Client versendet sein ER-Modell in Form eines ConecptionalModelDto an den Server. Nachdem der Server dieses erhalten hat, ruft dieser einen ErToRelationalModelTransformer mit dem erhaltenen DTO auf.

Diese Klasse verwaltet den gesamten Prozess des Übersetzungsvorgangs und delegiert alle Teilschritte an weitere Klassen. Die Erstellung der Datenstruktur delegiert diese an die Klasse ErTreeGraphFactory.

Die Factory erstellt die Datenstruktur für ER-Modelle unter der Verwendung einer Graph- und Baumdatenstruktur.

Die Graphdatenstruktur wurde generisch implementiert, um somit beliebige Daten in den Kanten und Knoten des Graphen zu halten. Die Implementierung des Graphen ist dabei nicht gerichtet und es kann von Kanten und Knoten ausgehend der Graph traversiert werden.

Die Baumdatenstruktur wurde ebenfalls generisch implementiert und besteht aus TreeNodes, welche sich gegenseitig referenzieren. Eine TreeNode beinhaltet Logik zur Verwaltung von Kind- und Elternknoten, sowie weitere Hilfsmethoden, um einfach durch den Baum zu navigieren. Mittels des generischen Typs können beliebige Objekte als Daten der TreeNode hinzugefügt werden.

Die ErTreeGraphFactory erstellt für Entitäten, Relationen und IsA-Strukturen einen Knoten im Graphen. Die Knoten des Graphen erhalten hierbei TreeNodes. Diese TreeNodes entsprechen jeweils der Wurzel eines Baumes. Für alle Attribute im ER-Modell wird ebenfalls eine TreeNode erstellt und an entsprechende TreeNodes angehängt.

Die Daten aller ER-Elemente werden den TreeNodes in Form eines EntityRelationElement-Objekts bekannt gemacht. Informationen, welche nicht für die Übersetzung selbst, aber nach der Übersetzung vom Client benötigt werden, sind dem EntityRelationElement-Objekten als ElementMetaInformation-Objekt hinzugefügt. Daten dieser Art beinhalten z.B. den Namen oder die Position eines ER-Elements.

Für Assoziationen zwischen Entitäten, Relationen und IsA-Strukturen im übergebenen ER-Modell werden Kanten im Graphen eingefügt. In den Kanten sind hierbei zusätzliche Informationen in Form eines EntityRelationAssociation-Objekts abgelegt.

Zusätzliche Informationen sind hierbei unter anderem, ob eine Entität als Unter- oder Obertyp mit einer IsA-Struktur verbunden ist oder die Kardinalitätsangabe einer Assoziation zwischen einer Entität und Relation.

Die Abbildung 7.9 zeigt den genannten Zusammenhang der Komponenten.

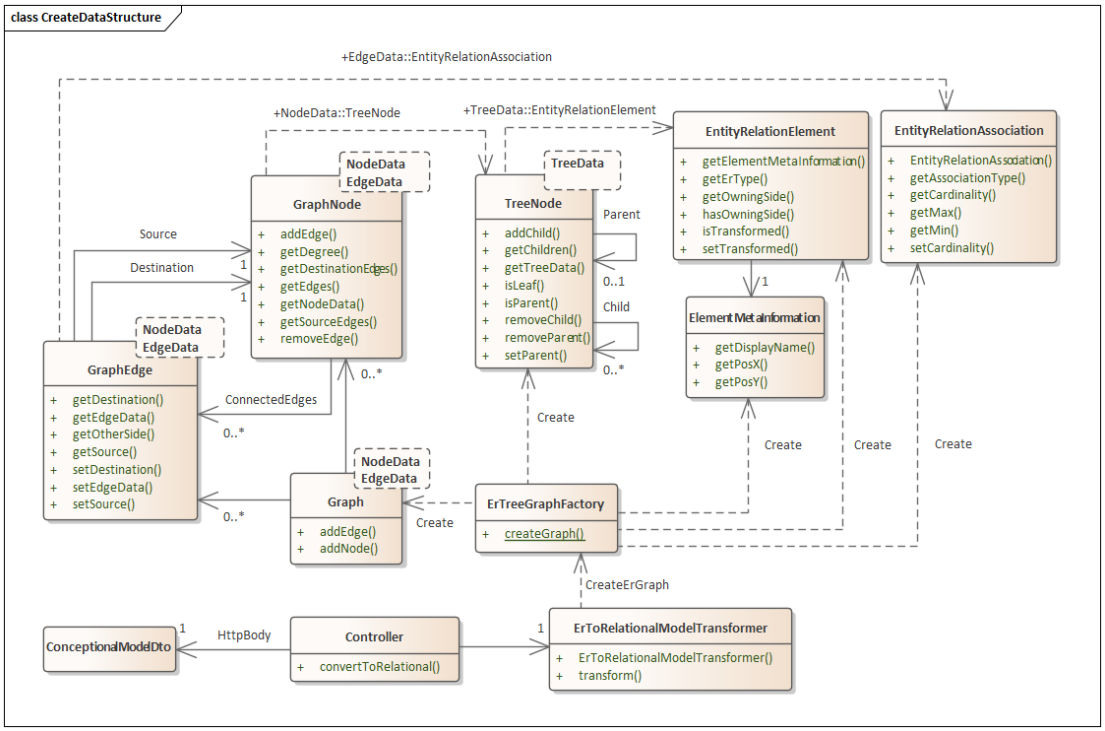

Abbildung 7.9: Aufbau der ER-Modell Datenstruktur

In der konzeptionellen Übersetzung des ER-Modells in das relationale Modell werden Tabellen mittels des Algorithmus zur Übersetzung von Attributen erstellt und modifiziert. Alle weiteren Übersetzungsalgorithmen modifizieren die Tabellen weiter.

In der Implementierung wurde sich dafür entschieden, die Logik zum erstellen, zugreifen und modifizieren der Tabellen mittels eines TableManagers von den Algorithmen zu trennen, um diese leichtgewichtig zu halten.

Der ErToRelationalTransformer hält daher eine Referenz zu einem TableManager und führt nach der Erstellung des ER-Graphen einen sogenannten TableCreatorService aus.

Dieser erstellt mittels eines übergebenen TableManagers für alle TreeNodes, welche Entitäten, Relationen oder Attribute repräsentieren eine Tabelle und fügt sie den Elementen hinzu. Der TableManager hält hierbei für einen einfachen Zugriff auf die Tabellen zusätzlich ein Register aller Tabellen.

Die Metainformationen der ER-Elemente innerhalb der TreeNodes werden an die Tabellen angehängt, damit diese nach der Übersetzung an den Client zurückgegeben werden können.

In der Abbildung 7.10 ist der statische Aufbau des Tabellenmanagements dargestellt.

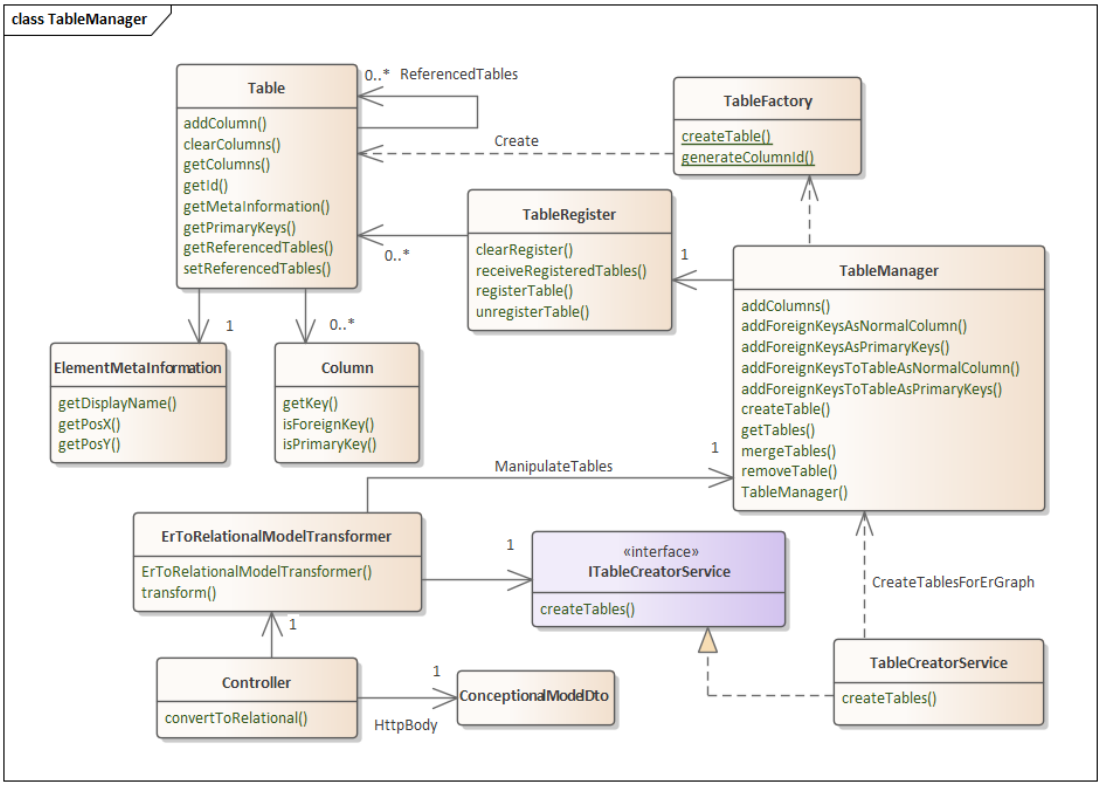

Abbildung 7.10: Aufbau des Tabellenmanagements

Ausführung der Übersetzungsalgorithmen

Mit der vorangegangen Erstellung des ER-Graphen und aller Tabellen, können alle im Kapitel 6.1 vorgestellten Algorithmen direkt ausgeführt werden.

Jeder Algorithmus wurde als eigenständiger Service implementiert. Diese arbeiten auf Basis eines übergebenen ER-Graphen und editieren die Tabellen unter der Verwendung eines übergebenen TableManagers. Aufgerufen werden diese mittels des ErToRelationalModelTransformers.

Nach der Ausführung der Algorithmen wird für die verbleibenden Tabellen ein DTO erstellt, welches an den Client zurückgesendet wird.

In der Abbildung 7.11 ist dieser Zusammenhang darstellt. Die Services werden dabei von Oben nach Unten aufgerufen.

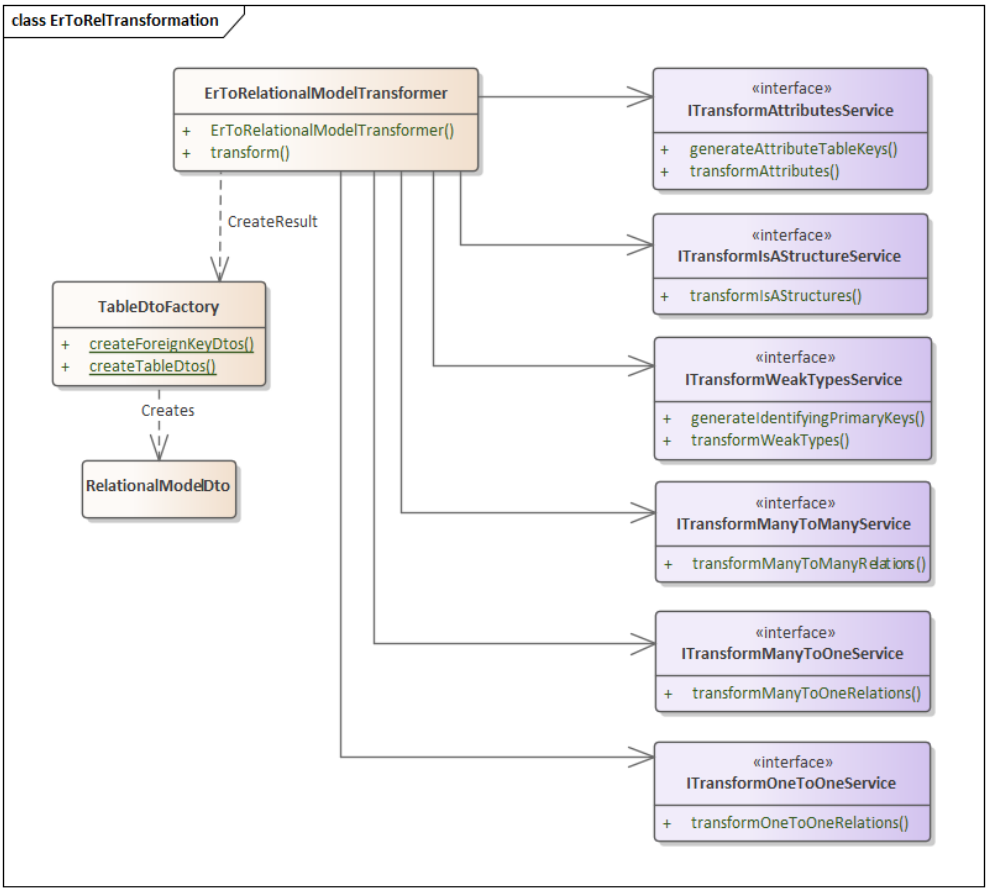

Abbildung 7.11: Ausführung der Übersetzungsalgorithmen

7.3.2 Implementierung der SQL-Generierung auf Basis des relationalen Modells

Der Client versendet das relationale Modell in Form eines RelationalModelDto an den Server. Dieses beinhaltet eine Liste an Tabellen, sowie eine Liste an Fremdschlüsselabhängigkeiten.

Nachdem der Server das DTO erhalten hat, ruft dieser einen RelationalModelToSqlTranslator mit dem erhaltenen DTO auf. Dieses koordiniert, analog zum ErToRelationalModelTransformer bei der Übersetzung des ER-Modells, die gesamte Überführung. Im Kapitel 6.2 wurden die durchzuführenden Schritte erläutert. Das Zusammenspiel der hierfür notwendigen Klassen ist in Abbildung 7.12 dargestellt und wird im Folgenden näher erläutert.

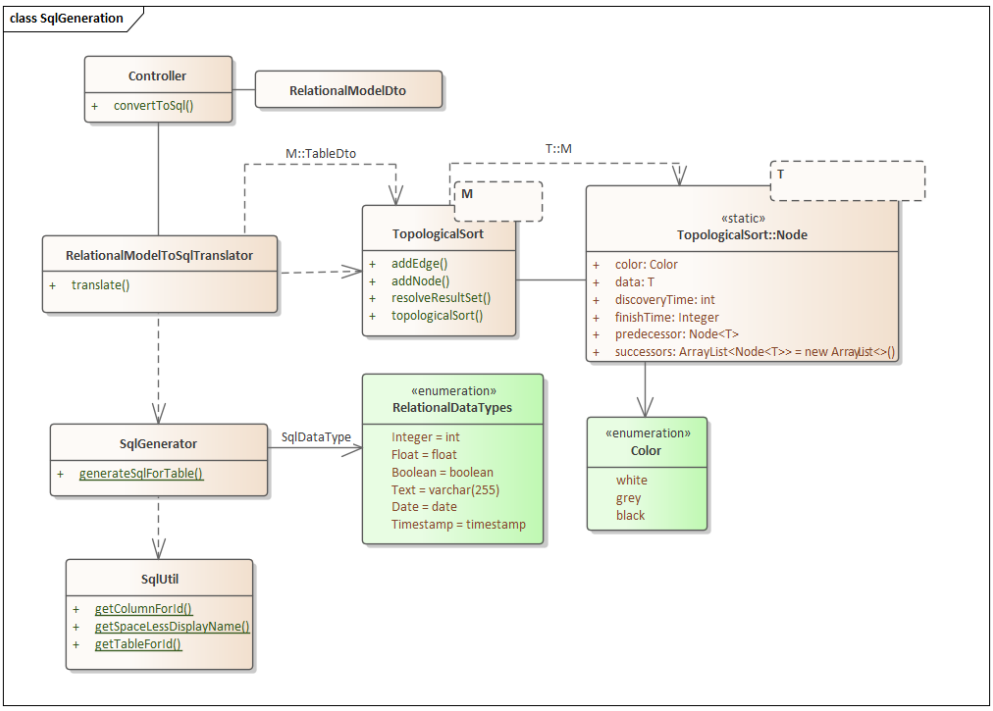

Abbildung 7.12: Aufbau SQL Generierung

Der Translator muss in einem ersten Schritt eine topologische Sortierung der Tabellen im RelationalModelDto durchführen.

Hierfür wurde der im Grundlagenkapitel vorgestellte Algorithmus zur topologischen Sortierung generisch implementiert. Die Implementierung erhält beliebige Objekte eines Typs, welche sortiert werden sollen. Darüber hinaus wird dem Algorithmus bekannt gegeben, welche Objekte voneinander abhängen.

Der Translator sortiert die Tabellen, indem er der Implementierung der topologischen Sortierung die Tabellen übergibt und für jede Fremdschlüsselabhängigkeit im RelationalModelDto eine Abhängigkeit zwischen zwei Tabellen einfügt.

Nach der Ausführung der topologischen Sortierung werden die sortierten Tabellen einem SQL-Generator übergeben. Dieser erstellt für jede Tabelle ein SQL-Statement. Dabei werden die Datentypen der vom Client übergebenen Tabellen durch SQL-spezifische Datentypen ersetzt. Die generierten SQL-Statements werden anschließend zusammengeführt und an den Client übergeben.

8 Inbetriebnahme

Die Inbetriebnahme der entwickelten Software erfolgte mittels Heroku[11].

Heroku ist ein cloudbasierter Plattform as a Service Anbieter. Der Vorteil dieses Anbieters ist es, dass er ein kostenloses Modell für das Hosting der Software und der DNS anbietet.

Darüber hinaus konnte mittels Heroku ein automatisches Deployment eingerichtet werden. Hierfür wurden zwei Heroku-Projekte angelegt und mit dem in diesem Projekt verwendeten GitHub[10] Repository verknüpft. Ein Heroku-Projekt hostet dabei eine Anwendung zum Bereitstellen des Clientcodes und ein weiteres Heroku-Projekt die Serveranwendung. Die Teilanwendungen laufen hierbei auf virtualisierten Containern, sogenannten Dynos.

Hiermit wird es ermöglicht, dass bei jedem Commit in das GitHub-Repository ein sofortiges Re-Deployment ausgeführt wird, wodurch kleinere Änderungen oder Fehlerbehebungen einfach und ohne größeren Aufwand durchgeführt werden können.

Der Nachteil dieses Anbieters ist es, dass die Dynos in denen die Anwendungen laufen in einen Schlafzustand geraten, wenn die Anwendungen über einen bestimmten Zeitraum nicht verwendet werden[3]. Ein Aufruf der Webseite dauert in diesem Fall ca. drei Sekunden. Die Serveranwendung ist hiervon ebenfalls betroffen. Daher ist es möglich, dass der erste Übersetzungsvorgang des ER-Modells in das relationale Modell oder die erste Generierung des SQL-Codes ebenfalls ca. drei Sekunden dauert. Alle weiteren darauffolgenden Anfragen sind hiervon nicht betroffen.

Zudem bietet Heroku in der kostenlosen Version insgesamt ein Kontingent von 550 Stunden pro Monat an, in denen die Dynos aktiv sind[8]. Da dies einer aktiven Nutzungszeit von 11,45 Tagen für beide Anwendungen entspricht ist dies für die Anwendung jedoch ausreichend.

Die Anwendung ist unter `https://dbmodelling.herokuapp.com` aufrufbar.

Für den lokalen Betrieb der Clientanwendung wurde Create React App [2] verwendet. Zum Starten der Clientanwendung ist der Packagemanager Npm[13] erforderlich. Die Clientanwendung kann gestartet werden indem in den Ordner "frontend/reactapp" navigiert wird und der Befehl "npm install" und "npm start" ausgeführt wird. Daraufhin startet der Developmentserver und die Anwendung ist unter der Url `localhost:3000` erreichbar.

Zum Betrieb der Serveranwendung ist Maven[15] und die Java OpenJDK 15[1] erforderlich. Zum Starten der Anwendung kann das Projekt im Ordner "backend" in einer IDE geöffnet werden und nach dem ausführen des Befehls "mvn clean install" gestartet werden.

9 Evaluierung

Die Evaluierung der entwickelten Software geschieht auf Basis der in Kapitel 3.7 zusammengefassten Anforderungen.

Die Software ist als Webanwendung realisiert und in Betrieb worden, wodurch die Anforderung Nr. 2 umgesetzt ist. Die Anforderung der Modularität Nr. 3, wurde durch die Architektur der Software, wiederverwendbare Komponenten der Clientanwendung und generische Implementierungen in der Serveranwendung adressiert.

Alle funktionalen Interaktionsanforderungen an das Zeichentool, genauer Nr. 4, 5, 6, 7 und 8 wurden mit dem Prototypen in Kapitel 7.2.1 abgedeckt. Diese werden in der darauf aufbauenden Implementierung unterstützt.

Die Applikation ist mittels des in Kapitel 5 entworfenen und in Kapitel 7 implementierten Validierungs- und Feedbacksystems benutzerfreundlich gestaltet worden. Mittels diesen Systemen wird auch die Durchsetzung der in Kapitel 4 definierten Regeln für ER-Diagramme gewährleistet. Hiermit werden die Anforderungen Nr. 9, 10 und 11 umgesetzt.

Die Benutzerfreundlichkeit wird durch das Design und die intuitiven Interaktionsmöglichkeiten des Zeichentools gefördert. In Kombination mit dem proaktiven Validierungs- und Feedbacksystems kann die Anwendung als intuitiv bedienbar gewertet werden, wodurch die Anforderung Nr. 1 adressiert wurde.

Die Übersetzung der ER-Modells in das Relationenmodell erfolgt über deterministische Algorithmen, wodurch sichergestellt wird, das ein ER-Modell immer in das gleiche relationale Modell überführt wird. Die entwickelten Algorithmen bedürfen hierbei neben dem ER-Modell keine weitere Einwirkung des Benutzers. Hierdurch wurden die Anforderungen Nr. 12 und 13 umgesetzt.

Die Darstellung des relationalen Modells erfolgt über einen separaten Tab innerhalb des Zeichentools. Über diesen kann der Benutzer Datentypen für das Modell eingeben. Die Anforderung zur Generierung von SQL-Code wurde mittels eines SQL-Generators in der Serveranwendung realisiert. Die SQL-Generierung benötigt, neben der Eingabe der Datentypen, kein weiteres Einwirken des Benutzers. Der generierte Code wird mittels eines Pop-Ups in der Clientanwendung dargestellt. Die Applikation erfüllt somit die Anforderungen Nr. 14, 15, 16 und 17.

Alle, an die Software gestellten Anforderungen sind somit umgesetzt worden.

10 Zusammenfassung und Ausblick

10.1 Erreichte Ergebnisse

Im Rahmen dieser Arbeit wurde eine Webanwendung entwickelt und in Betrieb genommen, welche den Benutzer über ein Zeichentool die Modellierung von Entity-Relationship Diagrammen ermöglicht und den Benutzer dabei proaktiv unterstützt.

Die Definition des in dieser Arbeit unterstützten ER-Modells erfolgte unter der Ermittlung wenig eingrenzender Regeln. Für die technische Durchsetzung der Regeln wurden benutzerfreundliche und proaktive Konzepte entwickelt und implementiert.

Des Weiteren wurden Algorithmen zur Überführung des ER-Modells in das relationale Modell erarbeitet und in der Software integriert. Darüber hinaus wurde ein SQL-Generator implementiert, welcher auf Basis des ermittelten relationalen Modells und unter der Eingabe von Datentypen durch den Benutzer SQL-Code erzeugt.

Durch Erreichen dieser Ergebnisse ist die entwickelte Software im Stande einen Benutzer bei dem gesamten Entwurfsprozess von relationalen Datenbanksystemen zu unterstützen.

10.2 Ausblick

Das Entity-Relationship Modell ist grundsätzlich ein konzeptionelles Modell, welches nicht zwangsweise in ein relationales Modell übergehen muss. In einer weitergehenden Arbeit können daher auf diesem Modell aufbauende Modelle untersucht oder entwickelt werden, mit denen Datenbanksysteme anderer Art modelliert werden können.

Durch die bisher existierende Unterstützung bei der Modellierung ist es auch denkbar, dass die Software sich zu einer Lernplattform weiterentwickelt. Diese kann sich sowohl tiefgreifender auf relationale Systeme konzentrieren als auch über verschiedene Datenbanksysteme erstrecken.

Literatur

[1] *Archived OpenJDK GA Releases*. 2022. URL: https://jdk.java.net/archive/ (besucht am 08. 08. 2022).

[2] *Create React App*. 2022. URL: https://create-react-app.dev/ (besucht am 08. 08. 2022).

[3] *Dynos and the Dyno Manager | Heroku Dev Center*. 2022. URL: https://devcenter.heroku.com/articles/dynos#dyno-sleeping (besucht am 08. 08. 2022).

[4] R. Elmasri, J. Weeldreyer und A. Hevner. „The category concept: An extension to the entity-relationship model“. In: *Data & Knowledge Engineering* 1.1 (1985), S. 75–116. ISSN: 0169023X. DOI: 10.1016/0169-023X(85)90027-8.

[5] Ramez Elmasri und Shamkant B. Navathe. *Grundlagen von Datenbanksystemen*. 3., überarb. Aufl., Ausg. Grundstudium, 2. Druck. I Datenbanken. München: Pearson Studium, 2007. ISBN: 3827370213.

[6] DB-Engines. *DB-Engines Ranking*. 2022. URL: https://db-engines.com/en/ranking (besucht am 19. 07. 2022).

[7] *Figma: the collaborative interface design tool*. 2022. URL: https://www.figma.com/ (besucht am 08. 08. 2022).

[8] *Free Dyno Hours | Heroku Dev Center*. 2022. URL: https://devcenter.heroku.com/articles/free-dyno-hours#usage (besucht am 08. 08. 2022).

[9] GitHub. *GitHub - jgraph/drawio: Source to app.diagrams.net*. 2022. URL: https://github.com/jgraph/drawio (besucht am 08. 08. 2022).

[10] GitHub. *GitHub: Where the world builds software*. 2022. URL: https://github.com/ (besucht am 08. 08. 2022).

[11] *Heroku*. 2022. URL: https://www.heroku.com/ (besucht am 08. 08. 2022).

[12] Alfons Heinrich Kemper und André Eickler. *Datenbanksysteme: Eine Einführung*. 10., erweiterte und aktualisierte Auflage. De Gruyter Oldenbourg Studium. Berlin und Boston: De Gruyter Studium, 2015. ISBN: 3110443759.

[13] *npm*. 2022. URL: https://www.npmjs.com/ (besucht am 08. 08. 2022).

[14] Peter Pin-Shan Chen. „The Entity-Relationship Model - Toward a Unified View of Data“. In: *ACM JournalsACM Transactions on Database SystemsVol. 1, No. 1* (1976). DOI: 10.1145/320434.320440.

[15] Brett Porter, Jason van Zyl und Olivier Lamy. *Maven – Welcome to Apache Maven*. 2022. URL: https://maven.apache.org/ (besucht am 08. 08. 2022).

[16] *React – A JavaScript library for building user interfaces*. 2022. URL: https://reactjs.org/ (besucht am 08. 08. 2022).

[17] *react-xarrows*. 2021. URL: https://eliav2.github.io/react-xarrows/ (besucht am 08. 08. 2022).

[18] *Redux - A predictable state container for JavaScript apps. | Redux*. 2022. URL: https://redux.js.org/ (besucht am 08. 08. 2022).

[19] Gunter Saake. *Algorithmen und Datenstrukturen: Eine Einführung mit Java*. 6., überarbeitete und erweiterte Auflage. Heidelberg: dpunkt.verlag, 2021. ISBN: 3969100666. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6379309.

[20] Gunter Saake, Kai-Uwe Sattler und Andreas Heuer. *Datenbanken: Konzepte und Sprachen*. 5. Aufl. Heidelberg: mitp, 2013. ISBN: 9783826694530.

[21] *Sparxsystems: UML, SysML, BPMN, Togaf, Updm vereint in Enterprise Architect*. 2022. URL: https://www.sparxsystems.de/ (besucht am 08. 08. 2022).

[22] Spring. *Spring makes Java simple*. 2022. URL: https://spring.io/ (besucht am 08. 08. 2022).

[23] Toby J. Theory, Dongqing Yang, James P. Fry. „A Logical Design Methodology for Relational Datbases Using the Extended EntityRelationship Model“. In: *Computing Surveys, Vol. 18, No. 2, June 1986* (1986).

[24] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 5. Aufl. München: Oldenbourg, 2005. ISBN: 3486253395.