Entity/Relationship-Datenmodellierung in axiomatischen Spezifikationssprachen

Rudolf Hettler

Fakultät für Informatik
der Technischen Universität München

Entity/Relationship-Datenmodellierung in axiomatischen Spezifikationssprachen

Rudolf Hettler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. A. Endres
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. M. Broy,
2.	Univ.-Prof. Dr. B. Mitschang

Die Dissertation wurde am 26. April 1995 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 13. Juni 1995 angenommen.

Zusammenfassung

Diese Arbeit integriert die Technik der Entity/Relationship-Datenmodellierung mit der axiomatischen Spezifikationstechnik SPECTRUM. Dies geschieht durch die formale Definition der Semantik des E/R-Modells in der Spezifikationssprache SPECTRUM. Dazu wird eine Vorschrift angegeben, die ein E/R-Schema in eine SPECTRUM-Spezifikation übersetzt, welche den durch das Schema beschriebenen Datenbestand repräsentiert. In der Folge wird untersucht, welche Vorteile sich aus dieser Integration für beide Techniken ergeben.

Auf der einen Seite ergibt sich die Möglichkeit, das E/R-Modell als Technik zur Datenmodellierung in die formale Softwareentwicklung mit SPECTRUM zu integrieren. Dazu wird die angegebene Übersetzungsvorschrift so weiterentwickelt, daß einem E/R-Schema eine Spezifikation zugeordnet wird, die zur weiteren Softwareentwicklung pragmatisch einsetzbar ist. Damit wird eine Schwäche der Sprache SPECTRUM behoben, die sich wie alle auf algebraischen Spezifikationstechniken basierenden Ansätze besser zur Beschreibung algorithmisch komplexer als stark datenorientierter Systeme eignet. Bei der Weiterentwicklung wird die formale Beziehung zwischen den durch die verschiedenen Übersetzungsvorschriften generierten Spezifikationen klar herausgearbeitet.

Auf der anderen Seite wird die in der Welt der formalen Softwareentwicklung vorhandene Idee der schrittweisen Verfeinerung von Spezifikationen in die Welt der E/R-Datenmodellierung übertragen. Die Arbeit untersucht, welche (informell definierten) Beziehungen zwischen Schemata in der Welt der E/R-Datenmodellierung Verwendung finden und wie sie sich zu den in SPECTRUM bekannten Beziehungen zwischen Spezifikationen verhalten. Insbesondere wird ein in der Literatur informell gegebener Satz von Transformationsregeln zur Entwicklung von E/R-Schemata auf seine Verträglichkeit mit dem in SPECTRUM definierten Begriff der Realisierungsbeziehung zwischen Spezifikationen untersucht. Darauf aufbauend wird ein mit dem SPECTRUM-Realisierungsbegriff verträglicher Entwicklungsbegriff für E/R-Schemata definiert. Mit Hilfe dieses Entwicklungsbegriffs wird eine Technik zur Integration verschiedener Schemata (*View Integration*) angegeben, die mit SPECTRUM verträglich ist.

Danksagung

Für wertvolle Kommentare zu Vorversionen dieser Arbeit bedanke ich mich bei Franz Huber, Heinrich Hußmann, Cornel Klein, Franz Regensburger und Mohsen Salhi. Mein Dank gilt darüberhinaus allen meinen Kolleginnen und Kollegen am Lehrstuhl für die stets offene und freundliche Atmosphäre, die fruchtbares Arbeiten erst möglich macht.

Besonderen Dank schulde ich Prof. Manfred Broy dafür, daß er diese Arbeit ermöglicht hat und mir während ihrer Anfertigung immer wieder seinen fachlichen Rat und Ermutigung zukommen ließ. Prof. Bernhard Mitschang gilt mein Dank für die fruchtbaren Anregungen, die ich in der Endphase meiner Arbeit von ihm erhalten habe.

Nicht zuletzt danke ich allen meinen Bekannten, meiner Familie und vor allem Gabi für die Geduld und Nachsicht, die sie während der Erstellung dieser Arbeit mit mir hatten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Entwicklung von Methoden zur effizienten Erstellung qualitativ hochwertiger, das heißt nutzungsadäquater, zuverlässiger, wohlstrukturierter und wartungsfreundlicher Software ist eines der vordringlichsten Probleme der Informatik. Forschung auf diesem Gebiet findet in zwei voneinander nahezu völlig unabhängigen Bereichen statt, dem Bereich der *pragmatischen Software-Entwicklungsmethoden* und dem Bereich der *formalen Methoden*.

Pragmatische Entwicklungsmethoden Pragmatische Software-Entwicklungsmethoden sind aus den praktischen Erfahrungen bei der Entwicklung großer Softwaresysteme entstanden. Sie versuchen, sowohl die Anforderungen an ein System als auch das Design mit Hilfe einer Reihe spezieller Darstellungstechniken (Diagramme, Formulare, Tabellen, ...) anschaulich zu präsentieren, um so die Komplexität des zu entwickelnden Systems beherrschbar zu machen. Prominente Vertreter solcher Darstellungstechniken sind etwa Entity/Relationship-Diagramme [Che76], Datenflußdiagramme [DeM79] oder Statecharts [Har87]. Oft werden diese Darstellungen mit erklärenden natürlichsprachlichen Texten ergänzt. Die Intention solcher Darstellungstechniken ist es, kompakte und leicht verständliche Sichten auf Teilaspekte des zu entwickelnden Systems (statische Sicht, dynamische Sicht, funktionale Sicht) angeben zu können. Sie erleichtern es nicht nur dem Entwickler, große und komplexe Systeme zu analysieren und zu beschreiben, sie vereinfachen auch die Kommunikation zwischen Entwickler und Auftraggeber, was gerade in den frühen Analysephasen zur Sicherstellung der Adäquatheit der Systembeschreibung unumgänglich ist. Ein oft sehr detailliertes Vorgehensmodell unterteilt den Entwicklungsprozeß in handhabbare Einzelschritte. Beispiele für pragmatische Methoden sind "Structured Analysis" [DeM79], SSADM [DCC92] oder die objektorientierte Methode OMT [RBP+91].

Der größte Schwachpunkt dieser pragmatischen Entwicklungsmethoden ist die fehlende mathematisch-präzise semantische Fundierung. Die Semantik der verwendeten Darstellungstechniken ist in den meisten Fällen informell gegeben. Durch dieses Fehlen einer ge-

meinsamen formalen Basis kann nicht angegeben werden, wie die Darstellungen der einzelnen Systemsichten zur Beschreibung des Gesamtsystems zusammenspielen. Darüberhinaus ist ohne formale Fundierung keine Möglichkeit zur *Verifikation* gegeben, das heißt, es ist nicht möglich, gewünschte Eigenschaften der erstellten Software nachzuweisen. Gerade für sicherheitskritische Anwendungen wird jedoch in den letzten Jahren die Forderung nach einem formalen Korrektheitsbeweis immer lauter. *Korrektheit* bedeutet in diesem Zusammenhang die Übereinstimmung des implementierten Programms mit der ursprünglichen Spezifikation.

Formale Methoden Formale Methoden, deren Entwicklung und Einsatz noch vorwiegend auf den universitären Bereich beschränkt ist, verstehen Software-Entwicklung dagegen als einen Vorgang, der vollständig innerhalb eines präzise definierten mathematisch-logischen Rahmens abläuft. Damit ist die Grundlage für die formale Sicherstellung der Korrektheit im obigen Sinne gegeben. Formale Techniken sind heute genügend ausgereift, um mit allen bei der Entwicklung von Software auftretenden Problemen umzugehen, wie zum Beispiel die Behandlung reaktiver und verteilter Systeme. Neben vielen anderen legen Bücher wie [BBS93, BW82, Bro89, Lee90, Dij76, Gri81, Jon86, LS87, Par90, TM87] ein solides Fundament für den Einsatz formaler Techniken bei der Programmentwicklung.

Die Nachteile der formalen Methoden liegen gerade in den Bereichen, in denen die pragmatischen Methoden ihre Stärken haben:

- Die verwendeten mathematischen und logischen Notationen sind nur Experten verständlich. Kommunikation mit Kunden ist auf Basis der so erstellten Dokumente nicht möglich.
- Durch den Zwang zur Präzision werden Spezifikationen oft mit einer Fülle von kleinen Details überladen, die selbst Experten bereits bei der Entwicklung von Programmen mittlerer Größe den Überblick verlieren lassen. Auch die bei vielen formalen Spezifikationssprachen vorhandenen mächtigen Modularisierungskonstrukte können dieses Problem nicht beseitigen. Ein skizzenhaftes Festhalten von Anforderungen in frühen Phasen der Analyse ist mit diesen Techniken ebenfalls nicht oder nur sehr eingeschränkt möglich.
- Die meisten formalen Methoden besitzen kein oder nur ein sehr schwach ausgeprägtes Vorgehensmodell, das zwar dem Entwickler große Freiheiten einräumt, damit aber auch die Gefahr erhöht, daß falsche oder unvorteilhafte Wege bei der Entwicklung beschritten werden.

Aufgrund der genannten Eigenschaften sind formale Techniken mehr zur Entwicklung algorithmisch komplexer, aber kleinerer Anwendungen als zur Entwicklung großer Informationssysteme prädestiniert. Im industriellen Bereich sind formale Methoden deshalb so gut wie bedeutungslos, mit Ausnahme bei der Entwicklung kleiner, sicherheitskritischer Anwendungen.

Integration der beiden Gebiete Die spezifischen Stärken und Schwächen der beiden Gebiete geben Anlaß zu der Hoffnung, daß sich durch eine geschickte Integration der beiden Ansätze die Situation im Bereich des Software Engineerings verbessern läßt. Es ist klar, daß bei einer solchen Integration die formale Welt die semantische Fundierung der Methode liefern muß, während die externe Repräsentation, also die verwendeten Darstellungstechniken wie auch das Vorgehensmodell, von pragmatischen Softwareengineering-Methoden beeinflußt wird.

Durch diese Integration können sich eine Reihe von Vorteilen für beide Welten ergeben. Formale Techniken können durch Darstellungsformen wie Diagramme oder Tabellen deutlich an Übersichtlichkeit und Akzeptanz gewinnen. Die pragmatischen Methoden erhalten durch die Integration mit formalen Techniken eine saubere semantische Fundierung, die zu einer Reihe von positiven Auswirkungen führen kann:

i) Durch die semantische Fundierung können die mit pragmatischen Methoden erstellten Modelle wesentlich besser auf ihre Konsistenz und Vollständigkeit hin überprüft werden. Die bisher in den pragmatischen Methoden informell gegebenen Konsistenzbedingungen können dann formal aufgeschrieben werden. Ein großer Teil davon wird dadurch maschinell überprüfbar. Damit lassen sich mächtigere Unterstützungswerkzeuge für die Methoden entwickeln, die es nicht nur gestatten, die erstellten Modelle auf Konsistenz und Vollständigkeit zu prüfen, sondern die Erstellung dieser Modelle bis hin zur Codegenerierung unterstützen können.

ii) Formale Techniken können Teil von pragmatischen Methoden werden. Lücken in pragmatischen Methoden, in denen die beschriebenen Modelle unvollständig sind, können durch formale Anteile gefüllt werden. Die erstellten Modelle werden also durch eine Mixtur aus informellen beziehungsweise semiformalen und formalen Dokumenten beschrieben.

iii) Formale Techniken können zur Analyse und Verbesserung der pragmatischen Methoden eingesetzt werden. Dabei wird nicht *innerhalb* der Methode mit formalen Techniken gearbeitet, sondern die Formalismen werden benutzt, um *über* die pragmatischen Methoden zu reden. Durch die Analyse bestehender Methoden mit formalen Mitteln lassen sich Einsichten gewinnen, die zur Entwicklung besserer pragmatischer Methoden führen.

1.2 Ziele der vorliegenden Arbeit

Die vorliegende Arbeit untersucht die Integration der Welt der pragmatischen Softwareentwicklung mit formalen Techniken am Beispiel der konzeptuellen Datenmodellierung, das heißt der problemnahen Beschreibung der für ein Informationssystem relevanten Daten eines Anwendungsgebiets. Damit wird ein bei der Spezifikation betrieblicher Informationssysteme zentraler Vorgang abgedeckt. Als konkrete Beschreibungstechnik zur konzeptuellen Datenmodellierung wird in dieser Arbeit das Entity/Relationship-Modell (E/R-Modell)

[Che76] betrachtet. Zur Integration mit dem E/R-Modell wird mit der axiomatischen Spezifikationssprache SPECTRUM [BFG+93a, BFG+93b] einer der modernsten und mächtigsten Vertreter aus der Entwicklungslinie der algebraischen Spezifikationstechniken ausgewählt.

Die Auswahl dieser beiden Techniken begründet sich zum einen aus ihrer Akzeptanz, was ihre Einsatzhäufigkeit und ihren Reifegrad betrifft, und zum anderen aus ihrer großen Gegensätzlichkeit, was die Anwendungsgebiete angeht.

Das Entity/Relationship-Modell (E/R-Modell) ist die mit Abstand am häufigsten eingesetzte pragmatische Technik. Eine Studie [BHS92], die 29 kommerzielle Softwareprojekte einbezog, zeigte, daß E/R-Modellierung in 24 Fällen eingesetzt wurde. Keine andere pragmatische Technik erfreut sich einer ähnlich hohen Akzeptanz. Die Technik ist ausgereift und hat ihre Eignung für die konzeptuelle Datenmodellierung unter Beweis gestellt. Sie bildet die Basis für viele andere semantische Datenmodelle, zum Beispiel das in [EGH+90, GH91, KG90, Hoh93, Gog94] eingesetzte erweiterte E/R-Modell (EERM) oder objektorientierte Datenmodelle wie OMT [RBP+91]. Obwohl es eine Vielzahl unterschiedlicher Notationen für E/R-Diagramme gibt, und obwohl die Semantik nicht formal definiert ist, hat sich eine eindeutige Interpretation der Diagramme entwickelt. Andere pragmatische Techniken wie zum Beispiel Datenflußdiagramme tragen im Gegensatz dazu in verschiedenen Methoden unterschiedliche Bedeutung. Diese eindeutige informelle Semantik der E/R-Diagramme erleichtert die Formalisierung erheblich.

Algebraische Spezifikationstechniken sind Forschungsgegenstand seit Mitte der 70er Jahre (vgl. [Gut75, GTWW75]). Ihre Grundlagen sind gut erforscht, zahlreiche Fallstudien haben die prinzipielle Eignung dieser Art formaler Techniken zur Spezifikation und Entwicklung von Software gezeigt. Die meisten dieser Fallstudien behandeln kleinere Beispiele, deren Komplexität eher in der Algorithmik als in der Struktur der verarbeiteten Daten liegt. Der Grund dafür ist, daß sich algebraische Techniken besser zur Spezifikation von Algorithmen als zur Beschreibung von Daten eignen, da alle Eigenschaften von Daten indirekt über die Eigenschaften von Funktionen ausgedrückt werden müssen, die auf den Daten operieren (zum Beispiel Konstruktor- und Selektorfunktionen). Beschreibt man auf diese Weise Daten, die wie im E/R-Modell komplex strukturiert sind und darüberhinaus zusätzliche Eigenschaften wie Schlüssel oder Kardinalitäten von Relationshiptypen aufweisen, entstehen sehr leicht große und unübersichtliche Spezifikationen.

Integration mit der E/R-Datenmodellierung kann dieses Problem algebraischer Spezifikationstechniken mildern. Die Integration erlaubt es, ein E/R-Schema als formales Dokument im Sinne der verwendeten formalen Technik (hier SPECTRUM) zu sehen. Man kann deshalb den statischen Anteil (Zustand) eines zu spezifizierenden Systems mit Hilfe eines E/R-Diagramms beschreiben, ohne den formalen Rahmen zu verlassen. Darauf aufbauend können dann die Systemfunktionen mit Hilfe der formalen Sprache spezifiziert werden. Dieser Ansatz wurde in Form einer Vorveröffentlichung zu dieser Dissertation am Beispiel der Spezifikation eines klinischen Informationssystems bereits erfolgreich eingesetzt (siehe [Het93], weitere Informationen zu dieser Fallstudie finden sich unter anderem in [Hußß93b, Nic93, SNM+93]).

Von der in dieser Arbeit durchgeführten Integration profitiert jedoch nicht nur die Welt der algebraischen Spezifikationstechniken. Neben einer formal definierten Semantik erfährt das E/R-Modell durch die Spezifikationssprache SPECTRUM eine natürliche Erweiterung, die es erlaubt, wesentlich allgemeinere statische Integritätsbedingungen für Daten zu beschreiben, als dies mit Hilfe von Schlüsseln und Kardinalitäten möglich ist. Damit wird auch die Klasse der bei der Normalisierung verwendeten funktionalen Abhängigkeiten als spezielle Form statischer Integritätsbedingungen vollständig in das konzeptuelle Schema integriert.

Die Integration mit SPECTRUM erlaubt darüberhinaus eine Übertragung der in der formalen Welt vorhandenen methodischen Erkenntnisse und Ergebnisse in die Welt der Datenmodellierung. Die Untersuchung der methodischen Implikationen der Integration und die Bereitstellung einer formal untermauerten Entwicklungsmethodik für E/R-Schemata ist ein weiteres wichtiges Ziel dieser Arbeit. In diesem Zusammenhang wird zum einen ein durch die formale Technik untermauerter Entwicklungsbegriff für E/R-Schemata angegeben, zum anderen wird untersucht, wie sich die Integration verschiedener E/R-Schemata zu einem einzigen konzeptuellen Schema im gegebenen formalen Rahmen durchführen läßt.

1.3 Vergleichbare Arbeiten

Die in der Motivation zu dieser Arbeit beschriebene Kluft zwischen pragmatischen und formalen Ansätzen zur Softwareentwicklung hat erst in jüngster Zeit Aufmerksamkeit als Forschungsgegenstand erhalten. Es gibt deshalb nur wenige Arbeiten, die sich mit diesem Thema beschäftigen. Darunter finden sich sowohl Arbeiten, die sich die formale Fundierung vollständiger pragmatischer Software-Engineering-Methoden zum Ziel gesetzt haben, als auch Arbeiten, die sich wie die vorliegende mit dem E/R-Modell auf eine einzelne Beschreibungstechnik spezialisieren und deren formale Fundierung im Detail untersuchen.

SAZ

Im Rahmen des an der University of York durchgeführten SAZ-Projekts [PWM93, PWM94] wurde die Software-Engineering Methode SSADM mit der Spezifikationstechnik Z integriert.

SSADM [DCC92, Eva92, CCT90] ist eine von der britischen "Central Computing and Telecommunications Agency" (CCTA) standardisierte Software-Engineering Methode, deren Einsatz seit 1983 für alle von der britischen Regierung in Auftrag gegebenen Software-Projekte zwingend vorgeschrieben ist. Aus Sicht der vorliegenden Arbeit ist im wesentlichen die zur Datenmodellierung verwendete Technik interessant (in SSADM wird diese Technik "Logical Data Modeling" genannt). Hier verwendet SSADM ein klassisches E/R-Modell.

Z [Spi92, Dil94] ist eine auf Konzepten der klassischen Mathematik (im wesentlichen Mengentheorie) basierende formale Spezifikationssprache. Diese Sprache unterscheidet sich von

den in SSADM eingesetzten Techniken nicht nur durch ihre mathematisch präzise semantische Fundierung, sie ermöglicht durch ihre Formalität auch das Führen mathematischer und sogar formaler (maschinell überprüfbarer) Beweise.

Das SAZ-Projekt sieht zwei verschiedene Arten vor, Z im Rahmen von SSADM einzusetzen:

Zur Qualitätsüberwachung SAZ beschreibt, wie sich die verschiedenen in SSADM erstellten Beschreibungen (Datenschemata, Datenflußdiagramme, Entity-Event Modelle, ...) nach Z übersetzen lassen. Anhand der so erstellten Z-Spezifikation läßt sich die Qualität (Widerspruchsfreiheit, Vollständigkeit) des in SSADM erstellten Modells auf einer einheitlichen Sprachebene auf formale Weise prüfen (vergleiche Punkt i) auf Seite 3).

Als zusätzliche Beschreibungstechnik Die Sprache Z kann im Sinne von Punkt ii) (Seite 3) als eigenständige, zusätzliche Beschreibungstechnik in SSADM verwendet werden. Sie kann dann dazu dienen, präzise Anforderungen an Stellen zu formulieren, an denen SSADM-Beschreibungen lückenhaft oder informell sind.

Der Anwender der Methode muß also in SAZ zumindest grundlegende Kenntnisse der Sprache Z besitzen, um die Vorteile von SAZ gegenüber SSADM nutzen zu können.

Im Bereich der konzeptuellen Datenmodellierung gibt SAZ ebenso wie die vorliegende Arbeit eine Übersetzung von E/R-Schemata in die formale Technik an, die auch durch ein Werkzeug (CADiZ) unterstützt wird. Damit erzielt SAZ Resultate, die den in dieser Arbeit (vorwiegend in Kapitel 4) als Konsequenzen einer derartigen Übersetzung vorgestellten sehr ähnlich sind:

- Die E/R-Modellierung kann als Beschreibungstechnik zur Spezifikation der Daten eines Informationssystems innerhalb der formalen Sprache benutzt werden,
- Mit Z steht ein Sprachmittel zur Verfügung, das es erlaubt, wesentlich allgemeinere statische Integritätsbedingungen zu spezifizieren, als dies im E/R-Modell möglich ist.

SAZ beschäftigt sich jedoch wie auch SSADM nicht mit der zur konzeptuellen Modellierung eingesetzten Methodik. Als Hilfsmittel zur Erstellung des konzeptuellen Schemas werden in SSADM mit der Analyse der Dokumentenflüsse und relationaler Analyse zwar Techniken gezeigt, es wird jedoch nicht darauf eingegangen, daß auch die konzeptuelle Datenmodellierung ein inkrementeller Vorgang ist, der in der Regel erst über eine Reihe von Zwischenschritten zum endgültigen Datenschema führt. Da diese Tatsache unberücksichtigt bleibt, ist es in SSADM auch nicht sinnvoll, über einen Entwicklungsbegriff für konzeptuelle Schemata oder die Integration verschiedener Schemata zu reden, wie es in Kapitel 5 der vorliegenden Arbeit geschieht. Da SAZ als formale Fundierung von SSADM gedacht ist, geht der Ansatz folgerichtig auch nicht auf die Auswirkungen der formalen Fundierung auf diese methodische Sichtweise der schrittweisen Entwicklung konzeptueller Schemata ein.

Formale Fundierung von SSADM mittels SPECTRUM

An der TU München wurde ein vollständig anderer Ansatz zur formalen Fundierung der Methode SSADM entwickelt [Huß94]. Die zur Fundierung verwendete Spezifikationstechnik ist in diesem Fall, wie auch in der vorliegenden Arbeit, die axiomatische Spezifikationssprache SPECTRUM (siehe Abschnitt 2.2).

SPECTRUM wird in [Huß94] jedoch nicht als Beschreibungstechnik *innerhalb* der Methode verwendet, die Spezifikationssprache wird vielmehr dazu benutzt, Aussagen *über* die Methode zu machen. Die semantische Fundierung von SSADM mittels SPECTRUM wird also durchgeführt, um das Zusammenspiel der einzelnen Beschreibungstechniken bei der Systembeschreibung zu verstehen und eventuell zu verbessern. Für den Anwender der so entstandenen Variante SSADM-F ist die formale Sprache SPECTRUM völlig unsichtbar, das heißt es sind zur Softwareentwicklung in SSADM-F keine SPECTRUM-Kenntnisse nötig. Diese Art des Einsatzes der formalen Technik entspricht der auf Seite 3 in Punkt iii) aufgeführten Variante, die Integration der pragmatischen mit der formalen Welt vorzunehmen.

Die Behandlung der konzeptuellen Datenmodellierung in [Huß94] unterscheidet sich von der vorliegenden Arbeit im wesentlichen in zwei Punkten:

- Im Gegensatz zur vorliegenden Arbeit ist die Zielsetzung in [Huß94] weder, SPECTRUM als Teil eines Beschreibungsmittels zu verwenden, noch ist es beabsichtigt, die Vorteile zu nutzen, die der Sprache SPECTRUM durch die Integration mit dem E/R-Modell entstehen. Deshalb wird dort eine Umsetzung von E/R-Schemata in SPECTRUM auf einer Abstraktionsebene angegeben, die sich in der vorliegenden Arbeit aus pragmatischen Gründen als hinderlich erweisen würde. Die Einführung dieser sehr abstrakten sogenannten *Methodenebene* ist jedoch hervorragend geeignet, um das Zusammenspiel des konzeptuellen Datenschemas, das in SSADM als *Logical Data Model* (LDM) bezeichnet wird, mit den anderen SSADM-Beschreibungstechniken zu untersuchen, wie es das wichtigste Ziel von [Huß94] ist.

- Aufgrund der in SSADM nur sehr schwach ausgeprägten Methodik zur konzeptuellen Datenmodellierung wird in [Huß94] wie auch in SAZ nicht darauf eingegangen, wie sich Datenschemata in kontrollierter Weise erstellen und weiterentwickeln lassen.

E/R-Modellierung in Z

An der Universität Oxford wurde, ähnlich wie im SAZ-Projekt, eine Übersetzungsvorschrift entwickelt, die es erlaubt, ein E/R-Schema in eine Menge von Z-Schemata zu transformieren [JRP91a, JRP91b]. Dieser Ansatz entspringt der gleichen Motivation, die in der vorliegenden Arbeit den Kapiteln 3 und 4 zugrundeliegt:

- Mit Hilfe der formalen Sprache kann der pragmatischen Notation des E/R-Modells eine präzise mathematische Semantik zugeordnet werden.

- Z erlaubt es, Invarianten für die modellierten Daten zu beschreiben, die im klassischen E/R-Modell nicht ausgedrückt werden können.
- Aufbauend auf der durch die Übersetzung entstehenden Spezifikation können Funktionen, die mit den modellierten Daten arbeiten, formal spezifiziert und entwickelt werden [Gin92].

Wie die beiden anderen vorgestellten Ansätze beschäftigt sich jedoch auch diese Arbeit nicht mit der Methodik der konzeptuellen Datenmodellierung. Die Frage, wie formale fundierte Entwicklungsschritte aussehen sollen, die zur Erstellung eines E/R-Schemas eingesetzt werden können, wird also auch in dieser Arbeit vollständig ignoriert.

Algebraische Semantik eines erweiterten E/R-Modells

Den zweifellos interessantesten Bezugspunkt für die vorliegende Arbeit stellt ein Ansatz dar, der in den letzten Jahren an der Universität Braunschweig entwickelt und ausführlich dokumentiert wurde [EGH+90, KG90, GH91, Hoh93, Gog94].

In dieser Arbeit wird eine algebraische Semantik für ein stark erweitertes Entity/Relationship-Modell definiert. Dieses sogenannte EERM enthält neben den im klassischen E/R-Modell vorhandenen Ausdrucksmitteln (Attribute, Entitytypen, Relationshiptypen) im wesentlichen folgende Erweiterungen:

Komponenten Das EERM erlaubt Attribute, die selbst Entities sind beziehungsweise Entities enthalten (Listen, Mengen von Entities, ...). Solche Attribute werden *Komponenten* genannt.

Typkonstruktionen Eine *Typkonstruktion* ist ein allgemeines sprachliches Mittel, um Enthaltenseinsbeziehungen zwischen Entitytypen auszudrücken. Damit kann insbesondere das in der Objektorientierung wichtige Konzept der Generalisierung (Vererbung) ausgedrückt werden.

Die Semantik eines EER-Schemas ist in diesem Ansatz definiert als Menge von Algebren über einer sogenannten *ER-Signatur*, die aus dem EER-Diagramm und den Signaturen der als abstrakte Datentypen spezifizierten Attributtypen generiert werden kann. In diesem Punkt ist der Ansatz sehr ähnlich zu der in Kapitel 3 gegeben Definition der statischen Semantik des E/R-Modells.

Basierend auf dieser semantischen Fundierung des EERM wird dann ein sogenannter *EER-Kalkül* definiert, der es in sehr flexibler Weise erlaubt, Anfragen an die mit dem Datenschema implizit definierte EER-Datenbank zu formulieren. Dieser formal definierte Kalkül kann als Grundlage zur semantischen Fundierung gebräuchlicher Datenbankabfragesprachen dienen. Der Ansatz kann einfach über das EERM hinaus auf andere Datenmodelle ausgeweitet werden. Ein in diesem Zusammenhang besonders hervorzuhebendes Ergebnis

ist die auf dem EER-Kalkül basierende Definition der Semantik für die in relationalen Datenbanksystemen weit verbreitete Abfragesprache SQL [KG90, Gog94].

In [Hoh93] wird eine Reihe von Abstraktionsebenen herausgearbeitet, die bei der konzeptuellen Datenmodellierung eine Rolle spielen. Es handelt sich dabei um die Spezifikation der Attributtypen, also der Wertebereiche der Attribute, die Beschreibung der Struktur der Daten als EER-Diagramm sowie die Spezifikation von statischen und dynamischen Integritätsbedingungen. Diese Sichtweise deckt sich mit der in der vorliegenden Arbeit gegebenen, jedoch wird hier die Beschreibung von dynamischen Integritäten nicht als unabdingbar gesehen. Die vorliegende Arbeit wird deshalb auf eine Formalisierung der Beschreibung dynamischer Integritäten verzichten.

Der Braunschweiger Ansatz unterscheidet sich in einer Reihe von Punkten von der vorliegenden Arbeit. Die Zielsetzung ist in diesem Ansatz nicht die Integration pragmatischer mit formalen Techniken, um so Vorteile für beide Welten zu erzielen. Vielmehr sollen formale Techniken eingesetzt werden, um eine präzise mathematische Semantik für Datenmodelle und Abfragesprachen anzugeben. Aus diesem Grund verwendet der Ansatz auch keine vorgegebene Spezifikationstechnik wie SPECTRUM, sondern definiert eine eigene, speziell auf das EERM zugeschnittene algebraische Semantik. Damit stellt sich im Braunschweiger Ansatz die in der vorliegenden Arbeit in Kapitel 4 untersuchte Frage nicht, inwieweit die formale Sprache Nutzen aus der Integration mit der pragmatischen Technik ziehen kann. Damit wird ein aus Sicht der vorliegenden Arbeit wesentlicher Aspekt der Integration pragmatischer mit formalen Techniken in diesem Ansatz außer acht gelassen. Der Vorteil dieses Vorgehens ist jedoch, daß der verwendete algebraische Ansatz exakt auf die Problemstellung zugeschnitten ist, während die vorliegende Arbeit auf Eigenschaften der Spezifikationssprache SPECTRUM Rücksicht nehmen muß, die als vielseitig einsetzbare Spezifikationssprache entwickelt und nicht speziell auf ihre Eignung zur semantischen Fundierung von Datenmodellen hin zugeschnitten ist.

Wie die drei anderen bereits vorgestellten Ansätze läßt auch der hier besprochene den methodischen Aspekt der konzeptuellen Datenmodellierung außer acht. Es wird weder ein formal definierter Entwicklungsbegriff angegeben noch wird auf die Punkte wie die Integration verschiedener Sichten (View Integration) eingegangen.

1.4 Gliederung der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Zunächst werden in einem Grundlagenkapitel (Kapitel 2) die beiden in der Arbeit verwendeten Techniken der Entity/Relationship-Modellierung und der formalen Spezifikation in SPECTRUM vorgestellt. Die Präsentation ist so detailliert, daß sie als Grundlage für das Verständnis der in der Arbeit entwickelten Ansätze dienen kann. Dabei wird auch auf vertiefende Literatur verwiesen, die ein weitergehendes Verständnis dieser Techniken ermöglicht.

In Kapitel 3 wird die Semantik des E/R-Modells definiert, indem eine Übersetzungsvorschrift angegeben wird, die es erlaubt, jedes E/R-Schema in eine SPECTRUM-Spezifikation zu transformieren. In einer Diskussion dieser Definition (Abschnitt 3.4) wird dabei insbesondere auf die Probleme eingegangen, die der praktischen Verwertbarkeit des vorgestellten Ansatzes im Wege stehen.

Die Behebung dieser Probleme, das heißt die Weiterentwicklung der zur Semantikdefinition eingeführten Übersetzungsvorschrift in eine praktisch einsetzbare Spezifikationstechnik, ist das Ziel von Kapitel 4. Dabei wird mit der Verwendung logischer Formeln zur Beschreibung von Integritätsbedingungen eine für den gegebenen Ansatz zentrale Erweiterung des eingesetzten E/R-Modells eingeführt.

Nachdem in den beiden vorangegangenen Kapiteln die Technik der (im Sinne von SPECTRUM) formalen Entity/Relationship-Modellierung eingeführt wurde, beschäftigt sich Kapitel 5 mit den Auswirkungen dieser Technik auf die Methodik der konzeptuellen Datenmodellierung. Der vorgestellte Ansatz wird auf diese Weise auch methodisch untermauert.

Kapitel 6 schließt die Arbeit mit einer Zusammenfassung der erzielten Ergebnisse (Abschnitt 6.1) und Ausblicken auf mögliche weiterführende Arbeiten, die sich auf die erzielten Ergebnisse stützen können (Abschnitt 6.2).

Kapitel 2

Grundlagen

2.1 Das verwendete E/R-Modell

Das von Chen [Che76] entwickelte Entity/Relationship-Modell existiert in einer Vielzahl von Varianten, die zum Teil unterschiedliche Ausdrucksmächtigkeit aufweisen. Die Grenzen zu anderen Ansätzen wie objektorientierten Modellen sind dabei fließend.

Aus diesem Grund ist es nötig, das in dieser Arbeit verwendete E/R-Modell exakt festzulegen. Die Festlegung der verwendeten Variante des E/R-Modells geschieht unter folgenden Maximen:

- Die Arbeit beschränkt sich auf ein "klassisches" E/R-Modell. Auf Konstrukte wie zum Beispiel Generalisierung, die das klassische E/R-Modell in Richtung anderer semantischer (insbesondere objektorientierter) Datenmodelle erweitern, wird explizit verzichtet.

- Das vorgestellte klassische E/R-Modell soll möglichst allgemein sein, das heißt möglichst viele andere klassische E/R-Modelle sollen Spezialfälle des hier vorgestellten sein, ohne dadurch die Darstellung der in dieser Arbeit vorgestellten Ansätze unnötig zu komplizieren.

Die Grundelemente des E/R-Modells sind *Entitytypen*, *Relationshiptypen* und *Attribute*. Entitytypen beschreiben die für das jeweilige Datenschema relevanten Einheiten von Information, die *Entities*. Entities sind durch die Werte ihrer Attribute gegeben. Relationships sind (zwei- oder mehrstellige) Beziehungen zwischen Entities. Relationshiptypen legen fest, Entities welcher Entitytypen Beziehungen eingehen können.

Die Struktur des zu modellierenden Datenbestandes wird mit Hilfe dieser Grundelemente festgelegt und meist grafisch in Form eines *E/R-Diagramms* dargestellt. Die Form der grafischen Darstellung unterscheidet sich ebenfalls von E/R-Ansatz zu E/R-Ansatz. In dieser Arbeit werden Entitytypen durch Rechtecke dargestellt, die mit ihrem Namen beschriftet sind. Relationshiptypen werden durch (ebenfalls mit ihrem Namen beschriftete)

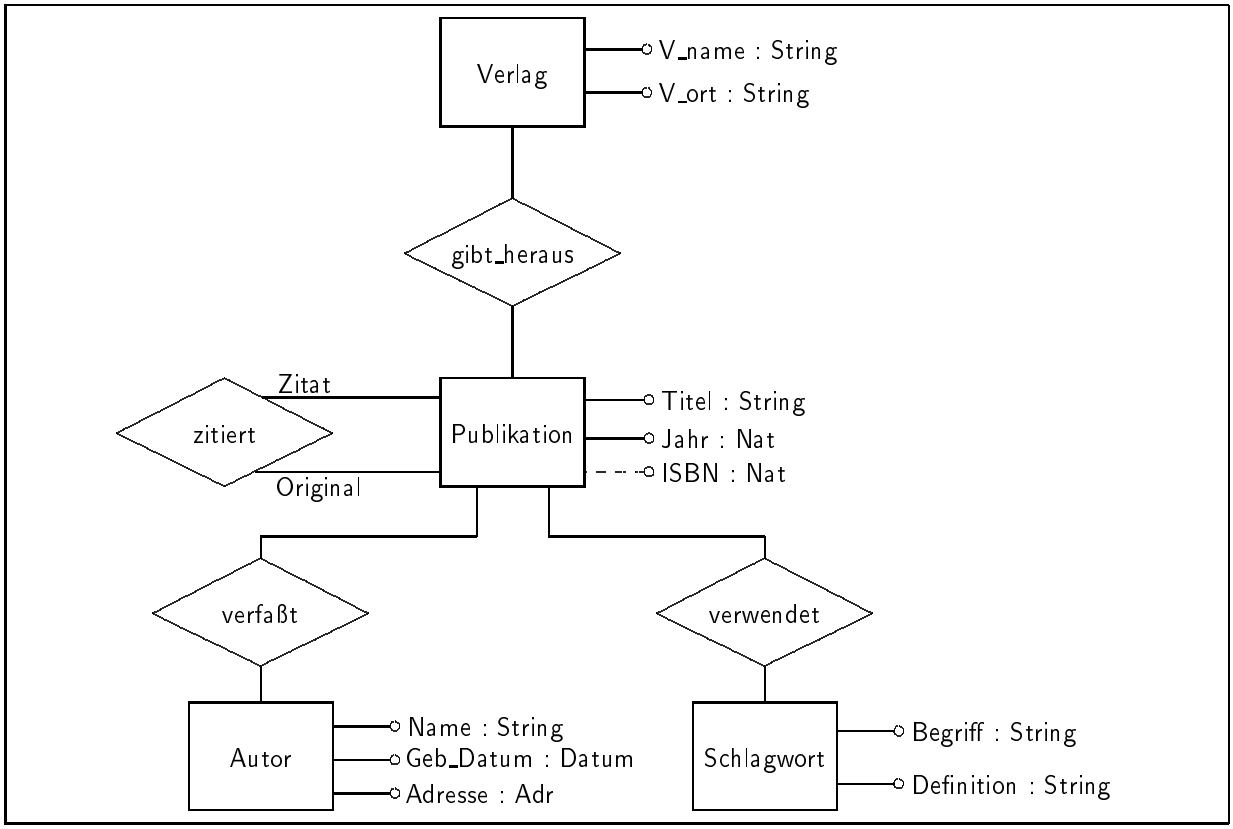


Abbildung 2.1: E/R-Diagramm

Rauten dargestellt, die durch Linien mit den Entitytypen verbunden sind, deren Entities eine Beziehung dieses Typs eingehen können. Jede dieser Linien kann optional mit einem Rollenbezeichner beschriftet sein, der angibt, in welcher Rolle der entsprechende Entitytyp am Relationshiptyp teilnimmt. Nimmt ein Entitytyp mehrfach an einem Relationshiptyp teil, so ist die Angabe von Rollenbezeichnern zwingend. Attribute werden durch kleine Kreise symbolisiert, die mit dem zugehörigen Entitytyp verbunden sind. Sie sind mit dem Attributbezeichner und ihrem Attributtyp beschriftet.

Beispiel 1 *Abbildung 2.1 zeigt die Struktur der Daten, die von einer fiktiven Applikation zur Unterstützung von Literaturrecherchen (im weiteren mit RECHERCHE bezeichnet) verwaltet werden. Zur besseren Präsentierbarkeit ist dieses Datenschema stark vereinfacht, es reicht jedoch aus, um die Beschreibung der Daten einer derartigen Applikation zu demonstrieren. Im Entitytyp* Schlagwort *verwaltet RECHERCHE eine Liste von Begriffen und (ähnlich einem Lexikon) deren Definitionen. Mit Hilfe des Relationshiptyps* verwendet *ist es möglich, Publikationen zu finden, die diese Schlagworte verwenden. Autoren und Verlag einer jeden Publikation werden in den entsprechenden Entitytypen verwaltet. Über* zitiert *können Querverweise zwischen den einzelnen Publikationen aufgefunden und ver-*

folgt werden. Das Attribut ISBN *des Entitytyps* Publikation *ist durch eine gestrichelte Linie als* optional *gekennzeichnet, das heißt daß dieses Attribut bei Entities des Typs* Publikation *nicht zwingend mit einem Wert belegt sein muß.*

*Es ist zu beachten, daß das E/R-Diagramm die Attributtypen (*Nat, String, *...) zwar verwendet, aber nicht näher charakterisiert. Diese fehlende Präzision wird bei der späteren Formalisierung des E/R-Modells zu beheben sein.* □

Neben der strukturellen Gliederung des zu modellierenden Datenbestandes in Entitytypen und Relationshiptypen erlaubt das E/R-Modell die Angabe von statischen Integritätsbedingungen. *Statische Integritätsbedingungen* sind Anforderungen, die der modellierte Datenbestand während seiner gesamten Lebensdauer zu erfüllen hat. Das E/R-Modell kennt typischerweise zwei Arten solcher statischen Integritätsbedingungen:

Grade von Relationshiptypen: Diese Art von Bedingungen erlaubt es festzulegen, wie oft eine Entity an Relationships eines gegebenen Relationshiptyps teilnehmen darf beziehungsweise muß. Zur Formulierung solcher Bedingungen existiert eine ganze Reihe unterschiedlicher Schreibweisen und grafischer Notationen (siehe z. B. [SS83]). In dieser Arbeit werden Relationshiptypen an jedem Ende mit einem Tupel (min, max) annotiert. Dabei stehen min und max für natürliche Zahlen, die die minimale bzw. maximale Partizipation von Entities des jeweiligen Entitytyps am gegebenen Relationshiptyp angeben. Ein $*$ anstelle von max gibt an, daß Entities beliebig oft an solchen Relationships beteiligt sein können.

Schlüssel: Schlüssel dienen zur eindeutigen Identifizierung von Entities eines Entitytyps *E* im modellierten Datenbestand. Sie bestehen typischerweise aus einer Reihe von Attributen, den *Schlüsselattributen*. Die mit einem Schlüssel assoziierte Integritätsbedingung ist, daß die Werte der Schlüsselattribute eine Entity innerhalb des Datenbestands eindeutig identifizieren. Das bedeutet, daß eine Datenbank, die dem E/R-Schema genügt, keine verschiedenen Entities enthalten kann, die in ihren Schlüsselattributen identisch sind. Darüberhinaus wird für Schlüssel im allgemeinen eine Minimalitätseigenschaft gefordert, die besagt, daß keine echte Untermenge der Schlüsselattribute ausreicht, um eine Entity eindeutig zu charakterisieren.

Wie im Ansatz von Batini, Ceri und Navathe [BCN92] soll auch in dieser Arbeit der Begriff des Schlüssels um sogenannte *externe Identifikatoren* erweitert werden. Bei diesem erweiterten Schlüsselbegriff werden zur Identifizierung einer Entity eines Entitytyps *E* nicht nur Schlüsselattribute herangezogen, sondern auch Relationships, die diese Entity bezüglich eines Relationshiptyps mit anderen Entities eingeht. Um den Schlüsselbegriff verständlich zu halten, sind für die Identifizierung jedoch keine beliebigen Relationshiptypen erlaubt, sondern ausschließlich zweistellige Relationshiptypen, an denen Entities des Typs *E* genau einmal $((min, max) = (1, 1))$ beteiligt sind.

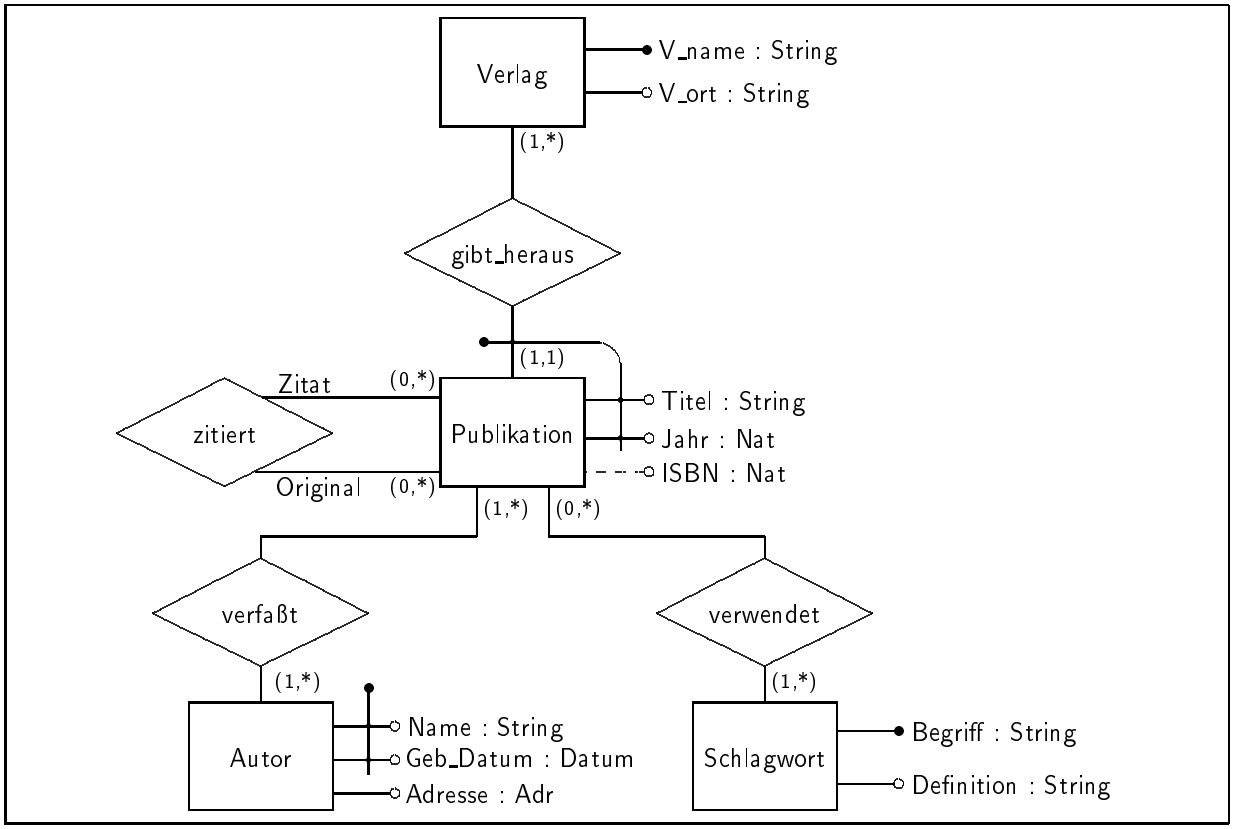


Abbildung 2.2: E/R-Diagramm mit Integritätsbedingungen

In der hier verwendeten graphischen Darstellung werden die Identifikatoren, die zusammen den Schlüssel eines Entitytyps ergeben, durch eine durchgezogene Linie verbunden, die in einem kleinen, ausgefüllten Kreis endet (siehe Abbildung 2.2). Besteht ein Schlüssel nur aus einem (internen) Identifikator, so wird er durch Ausfüllen des Kreises symbolisiert, der das Schlüsselattribut repräsentiert.

Beispiel 2 *Abbildung 2.2 zeigt das E/R-Diagramm von RECHERCHE aus Beispiel 1, angereichert um die statischen Integritätsbedingungen, die für diese Applikation sinnvoll sind. Der Grad des Relationshiptyps* verfaßt *legt zum Beispiel fest, daß jeder Autor mindestens eine Publikation verfaßt haben muß, das heißt, zu jeder im Datenbestand enthaltenen Entity des Typs* Autor *muß mindestens eine Entity des Typs* Publikation *enthalten sein, die mit ihr in der* verfaßt*-Beziehung steht. Die Entitytypen* Verlag *und* Schlagwort *haben einfache, nur aus einem einzigen Schlüsselattribut bestehende, Schlüssel. Der Entitytyp* Autor *besitzt einen zusammengesetzten Schlüssel bestehend aus den Attributen* Name *und* Geb_Datum*. Entities des Typs* Publikation *werden durch einen zusammengesetzten Schlüssel identifiziert, der neben den Attributen* Titel *und* Jahr *einen externen Identifikator über den Relationshiptyp* gibt_heraus *enthält. Dieser Schlüssel besagt, daß es keine verschiedenen Publikationen*

geben kann, die den gleichen Titel besitzen, im gleichen Jahr erschienen sind und vom gleichen Verlag herausgegeben wurden. Diese externe Identifikation über den Verlag, in dem die Arbeit publiziert wurde, ist möglich, weil gibt_heraus *ein zweistelliger Beziehungstyp ist und die Beteiligung von* Publikation *an* gibt_heraus *als* (1,1) *festgelegt ist.* □

Konventionen

Um die Darstellung der in dieser Arbeit gegebenen Ansätze zu vereinfachen, werden für die Vergabe von Bezeichnern in E/R-Schemata folgende Konventionen festgelegt:

- Alle verwendeten Bezeichner entsprechen der Syntax der Spezifikationssprache SPECTRUM [GHN+94].
- Kein Bezeichner wird mehrfach verwendet.

Es ist offensichtlich, daß das vorgestellte E/R-Modell durch diese Konventionen in seiner Allgemeinheit nicht beschränkt wird.

2.2 Formale Spezifikation mit SPECTRUM

2.2.1 Sprache

Die Spezifikationssprache SPECTRUM stellt eine Weiterentwicklung algebraischer Spezifikationstechniken dar [GTWW75, Gut75]. Im folgenden wird diese Sprache im Überblick vorgestellt. Es wird auf alle Eigenschaften der Sprache eingegangen, die im weiteren Verlauf der Arbeit von Bedeutung sein werden. Für eine detaillierte Darstellung von SPECTRUM wird auf [BFG+93a, BFG+93b, GHN+94, GR94] verwiesen.

SPECTRUM-Spezifikationen bestehen aus einem Signaturteil und einem Axiomenteil. Im Signaturteil werden die in einem System vorkommenden Sorten[1] und Funktionen deklariert, im Axiomenteil werden logische Gesetze gegeben, die von den in der Signatur enthaltenen Komponenten (Sorten und Funktionen) eingehalten werden müssen. Auf diese Weise kann eine deklarative Beschreibung der Eigenschaften eines Systems gegeben werden. Im Gegensatz zu vielen anderen algebraischen Ansätzen, in denen die Gesetze in Form von Gleichungen oder bedingten Gleichungen angegeben werden, steht in SPECTRUM die volle Prädikatenlogik erster Stufe zusammen mit einem zusätzlichen Termerzeugungsprinzip

[1] In algebraischen Spezifikationstechniken werden Typen und Typsysteme oft als *Sorten* und *Sortensysteme* bezeichnet, um dieses Konzept besser vom Konzept der *abstrakten Datentypen* zu unterscheiden, die oft auch abkürzend als Typen bezeichnet werden. Diese Begriffswahl wird auch in der vorliegenden Arbeit eingehalten. So wird im folgenden zum Beispiel sowohl von Entitytypen wie auch von Entitysorten die Rede sein. Dabei bezeichnet der Begriff *Entitytyp* das entsprechende Konzept aus dem E/R-Modell, während *Entitysorte* das SPECTRUM-Konstrukt bezeichnet, das zur Repräsentation von Entitytypen eingesetzt wird.

zur Verfügung. Mit dieser Logik können sehr abstrakte, deklarative Anforderungsspezifikationen formuliert werden. Mit Hilfe der eingebauten Bereichstheorie (Spezifikation von Funktionen als kleinste Fixpunkte von Funktionalen) können jedoch auch konstruktive Spezifikationen angegeben werden, die funktionalen Programmen entsprechen. SPECTRUM ist somit speziell zur Entwicklung von Programmen für funktionale Programmiersprachen geeignet.

Die Sprache kann zur Beschreibung strikter und totaler Funktionen ebenso eingesetzt werden wie zur Spezifikation partieller oder nichtstrikter Funktionen. Neben Funktionen, die immer als stetig angenommen werden, da sie in einer Programmiersprache implementiert werden sollen, enthält SPECTRUM das Konzept der *Mappings*. Mappings stellen allgemeine mathematische Abbildungen dar, die lediglich zu Spezifikationszwecken eingesetzt, aber nicht implementiert werden sollen. Sie müssen deshalb keinerlei Stetigkeits- oder Monotonieforderungen erfüllen.

SPECTRUM besitzt ein statisches Sortensystem, das parametrische Polymorphie und Sortenklassen unterstützt. Es ist an die in funktionalen Programmiersprachen gängigen Typsysteme angelehnt, insbesondere weist es starke Ähnlichkeit mit dem Typsystem der Sprache HASKELL [HPW92] auf. Obwohl die Sprache stark getypt ist, erlaubt ein Sorteninferenzalgorithmus weitgehend das Weglassen von Sorteninformation in Axiomen.

Eines der Hauptziele von SPECTRUM ist das Erstellen gut strukturierter Spezifikationen. Zu diesem Zweck bietet die Sprache eine Reihe von Operationen zum *Spezifizieren im Großen* an, die wesentlich von der Sprache ASL [SW83] beeinflußt sind. Die wichtigste dieser Operationen (`enriches`) erlaubt es Spezifikationen, sich hierarchisch auf andere, sogenannte *primitive* Spezifikationen abzustützen und stellt so ein sehr natürliches Modularisierungskonzept zur Verfügung.

Die Semantik von SPECTRUM ist lose. Jede Algebra, die die in der Spezifikation gegebenen Gesetze erfüllt, ist ein Modell der Spezifikation. Es erfolgt keine Beschränkung etwa auf initiale oder terminale Modelle. Die Semantik einer SPECTRUM-Spezifikation ist also im allgemeinen eine Menge von Algebren, die auch als die *Modellklasse* der Spezifikation bezeichnet wird. In diesem Rahmen kann Verfeinerung einfach als Einschränkung der Modellklasse definiert werden (siehe Abschnitt 2.2.2), was zum Beispiel durch Hinzunahme von Gesetzen zur Spezifikation geschehen kann.

Beispiel 3 *Abbildung 2.3 zeigt die polymorphe Spezifikation endlicher Mengen in* SPECTRUM, *die im weiteren Verlauf der Arbeit mehrfach Verwendung finden wird. Die folgenden Bemerkungen sollen zum Verständnis dieser Spezifikation dienen und so einen Eindruck von* SPECTRUM *geben.*

Die Spezifikation trägt den Namen `Set` *und stützt sich hierarchisch auf die Spezifikation* `Nat` *der natürlichen Zahlen (*`enriches Nat`*), das heißt alle in* `Nat` *spezifizierten Konzepte können in* `Set` *verwendet werden. Alle in* `Set` *spezifizierten Funktionen sind strikt und total (*`strict total`*). Die Spezifikation führt einen polymorphen Sortenkonstruktor* `Set` α *ein. Polymorph heißt, daß durch Anwendung des Konstruktors Mengen über beliebigen Elementsorten gebildet werden können (*`Set Nat, Set String, Set (Set Nat)`, ...*).*

```
Set = { enriches Nat;

strict total;

sort Set α;

∅       : α::EQ ⇒ Set α;
add     : α::EQ ⇒ α × Set α → Set α;
del     : α::EQ ⇒ α × Set α → Set α;
.∈.     : α::EQ ⇒ α × Set α → Bool prio 8;
card    : α::EQ ⇒ Set α → Nat;

Set α generated by ∅,add;

axioms α::EQ ⇒ ∀e, e':α, s: Set α in
add(e,add(e,s)) = add(e,s);
add(e',add(e,s)) = add(e,add(e',s));

del(e,∅) = ∅;
del(e,add(e',s)) = if e=e' then s else add(e',del(e,s)) endif;

¬(e ∈ ∅);
e ∈ add(e',s) = (e = e' ∨ e ∈ s);

card(∅) = 0;
¬(e ∈ s) ⇒ card(add(e,s)) = 1 + card(s);
e ∈ s ⇒ card(add(e,s)) = card(s);
endaxioms;
}
```

Abbildung 2.3: Spezifikation endlicher Mengen in SPECTRUM

Anschließend werden die Signaturen der in der Spezifikation definierten Funktionen $\emptyset$, `add`, `del`, `.∈.` *und* `card` *eingeführt. Die Funktion* `add` *ist zum Beispiel eine polymorphe Funktion, die ein Element einer Sorte* α *und eine endliche Menge über dieser Sorte als Parameter erhält und wieder eine solche Menge liefert. Man beachte, daß die leere Menge* $\emptyset$ *in* SPECTRUM *als nullstellige Funktion betrachtet wird. Das Enthaltenseinsprädikat* `.∈.` *ist als Infix-Funktionssymbol definiert (die Punkte repräsentieren Platzhalter für die Argumente der Funktion, mit dem Schlüsselwort* `prio` *wird die Angabe einer Bindungsstärke des Funktionssymbols eingeleitet). Der in den Signaturen verwendete* Sortenkontext α`::EQ` $\Rightarrow$ *ist durch eine Besonderheit der Spezifikationssprache bedingt. In* SPECTRUM *können Sorten im Gegensatz zu vielen anderen Sprachen auch durch nicht flach geordnete Bereiche interpretiert werden (zum Beispiel Funktionssorten). Mengen über nicht flach geordneten Sorten sind nicht sinnvoll, unter anderem weil für derartige Mengen das Enthaltenseinsprädikat* `.∈.` *nicht monoton und damit nicht implementierbar ist. Der Sortenkontext* α`::EQ` $\Rightarrow$ *schränkt die Sortenvariable* α *auf Sorten aus der in* SPECTRUM *vordefinierten Sortenklasse* `EQ` *ein, die exakt die Menge der flach geordneten Sorten bezeichnet.*

Das Axiom `Set` α `generated by ∅, add` *kennzeichnet* $\emptyset$ *und* `add` *als Konstruktoren der mit Hilfe des Sortenkonstruktors* `Set` *erzeugten Sorten und stellt so ein Induktionsprinzip zur Verfügung. Die daran anschließenden logischen Axiome (*`axioms...endaxioms`*) beschreiben die Eigenschaften, die die in der Signatur angegebenen Funktionen erfüllen müssen. Dabei spezifizieren die in den beiden ersten Axiomen gegebenen Gleichungen über den Konstruktoren die typischen Mengeneigenschaften (Mengen enthalten keine Duplikate von Elementen und die Reihenfolge des Einfügens ist irrelevant). Die Eigenschaften der restlichen Funktionen werden jeweils mit Hilfe einer vollständigen Fallunterscheidung über die Konstruktoren* $\emptyset$ *und* `add` *spezifiziert.*

□

2.2.2 Methodik

Neben der Bereitstellung einer mächtigen und flexiblen Sprache zur Spezifikation wird in SPECTRUM Wert auf eine gute methodische Unterstützung des Software-Entwicklungsprozesses gelegt. Eine Entwicklung wird in SPECTRUM als Graph verstanden, dessen Knoten Spezifikationen und dessen Kanten formal definierte Beziehungen zwischen den Knoten repräsentieren. Da die Beziehungen formal definiert sind, läßt sich ihre Gültigkeit ebenfalls formal, das heißt mit Hilfe eines logischen Kalküls nachweisen (siehe Abschnitt 2.2.3). Auf die verschiedenen in der SPECTRUM-Methodik verwendeten Beziehungen zwischen Spezifikationen wird im folgenden kurz eingegangen. Für eine detailliertere Darstellung und Beispiele wird auf [Wir92, BW93, PW95] verwiesen.

Die Stützrelation

Wie in Abschnitt 2.2.1 beschrieben enthält SPECTRUM eine `enriches`-Operation zur Modularisierung von Spezifikationen. Sie bedeutet das hierarchische Abstützen einer Spezifi-

kation auf eine andere. Dies kann als syntaktisch definierte Beziehung zwischen den beiden Spezifikationen verstanden werden, die auch *Stützrelation* genannt wird. Ein bezüglich der Stützrelation abgeschlossener Teil eines SPECTRUM-Entwicklungsgraphen wird auch *Konfiguration* genannt.

Die Verfeinerungsrelation

Die lose Semantik von SPECTRUM erlaubt es, in frühen Entwicklungsphasen mit Unterspezifikation zu arbeiten. Unterspezifikation bedeutet, daß ein System durch eine Spezifikation nicht vollständig festgelegt wird, sondern daß bestimmte Eigenschaften (vorläufig) offen bleiben. So ist es zum Beispiel bei der Beschreibung der grundlegenden Anforderungen an ein System oft nicht nötig, die Art und Weise festzulegen, in der mit fehlerhaften Benutzereingaben umgegangen werden soll. Ein Konzept zur Fehlerbehandlung kann später hinzugefügt werden. Auf diese Weise ergibt sich eine klare und verständliche Schichtung der Abstraktionsebenen. Semantisch betrachtet besitzt eine nicht vollständig festgelegte Spezifikation mehrere unterschiedliche Modelle und kann somit auch auf verschiedene Arten implementiert werden.

Verfeinerung bedeutet in diesem Rahmen das Verschärfen der Anforderungen an das spezifizierte System und/oder die Hinzunahme neuer Funktionen, das heißt die Aufnahme neuer Elemente in die Signatur. Verschärfen der Anforderungen bedeutet eine Konkretisierung der Spezifikation, das heißt eine Einschränkung der Modellklasse.

Formal nennt man eine Spezifikation $SP' = (\Sigma', Ax')$ eine Verfeinerung einer Spezifikation $SP = (\Sigma, Ax)$, wenn gilt:

1. $\Sigma \subseteq \Sigma$'
2. Für alle Axiome $ax \in Ax$ gilt $Ax' \blacktriangleright ax$

Das Zeichen $\blacktriangleright$ ist das im Logikkalkül von SPECTRUM verwendete Herleitungszeichen. Die oben gegebene Forderung $Ax' \blacktriangleright ax$ kann gelesen werden als

> Die Formel ax kann unter Verwendung des in SPECTRUM gegebenen logischen Kalküls aus der Formelmenge Ax' formal hergeleitet werden.

Man beachte, daß der so definierte Verfeinerungsbegriff nicht nur das Hinzufügen neuer Formeln (Anforderungen) zur Axiomenmenge gestattet. Er ermöglicht auch das Austauschen von Axiomen durch andere. Damit können nicht-konstruktive Axiome durch konstruktive ersetzt und so die Entwicklung in Richtung auf eine (funktionale) Programmiersprache vorangetrieben werden.

Der Datenstrukturwechsel

Der Begriff der Verfeinerung, wie er oben angegeben ist, reicht nicht in allen Fällen aus, um die Entwicklung eines Systems zu beschreiben. Er erlaubt es nicht, eine abstrakt spezifizierte Sorte durch eine konkretere, in funktionalen Sprachen darstellbare Sorte zu repräsentieren.

Beispiel 4 *Die in Beispiel 3 spezifizierte polymorphe Mengensorte ist in funktionalen Sprachen nicht definierbar. Hier lassen sich nur sogenannte* freie Datentypen *definieren, die keine Gleichungen auf Konstruktortermen erlauben, wie es bei den ersten beiden Axiomen der* `Set`*-Spezifikation der Fall ist. Bevor man also bei der Entwicklung von Software, zu deren Beschreibung man die Mengenspezifikation verwendet hat, den Schritt zur funktionalen Sprache machen kann, muß die Mengensorte durch eine andere, in der Programmiersprache ausdrückbare, Sorte ersetzt werden. Zur Repräsentation von Mengen werden dabei üblicherweise Listen eingesetzt.* □

Ein Wechsel der Datenstruktur bedeutet also das Ersetzen einer abstrakten Sorte durch eine sie repräsentierende konkrete Sorte, wobei alle auf der abstrakten Sorte definierten Operationen als Operationen auf der konkreten Sorte (mit denselben Eigenschaften) zur Verfügung stehen müssen. Es gibt eine Reihe von Ansätzen, diesen Wechsel der Datenstruktur formal zu bewältigen. Eine Variante soll am oben gegebenen Beispiel der Darstellung von Mengen durch Listen kurz vorgestellt werden.

Sollen Mengen durch Listen dargestellt werden, muß zunächst geklärt werden, auf welche Weise die Elemente der Listensorte die Mengenelemente repräsentieren. Dabei ist zu beachten, daß es Listen geben kann, die nicht zur Repräsentation der Mengen benutzt werden und daß möglicherweise mehrere verschiedene Listen Repräsentanten derselben Menge sind. Es gibt selbstverständlich eine Reihe unterschiedlicher Möglichkeiten, Mengen als Listen darzustellen. Im vorliegenden Beispiel wird folgender Ansatz gewählt:

1. Listen, die Duplikate enthalten, repräsentieren keine Mengen.

2. Sind zwei duplikatfreie Listen Permutationen voneinander, enthalten sie also die gleichen Elemente, so repräsentieren sie dieselbe Menge.

Gegeben sei also eine Spezifikation `List` von polymorphen Listen als abstrakter Datentyp (in SPECTRUM ist eine solche Spezifikation bereits vordefiniert, siehe [GHN+94]). Diese Spezifikation wird in einer hierarchisch darüberliegenden Spezifikation `Set_by_List` unter Verwendung der Signatur von `Set` um folgende Konzepte angereichert:

- Für die in der Signatur von `Set` enthaltenen Funktionen, die mit Mengen arbeiten, müssen auf Listen operierende Implementierungen angegeben werden. Im Beispiel sind also Funktionen

```
∅' : α::EQ ⇒ List α;
add' : α::EQ ⇒ α × List α → List α;
del' : α::EQ ⇒ α × List α → List α;
card' : α::EQ ⇒ List α → Nat;
.∈'. : α::EQ ⇒ α × List α → Bool;
```

zu spezifizieren, die den Funktionen ∅, `add`, `del`, `card` und `.∈.` auf Mengen entsprechen sollen[2].

- Die oben angegebene Bedingung 1 wird durch ein *Repräsentationsprädikat*

  ```
  is_Set : List α → Bool
  ```

  modelliert, das für duplikatfreie Listen `true` liefert. An dieser Stelle ist als erste Beweisverpflichtung die Abgeschlossenheit der Funktionen ∅', `add'`, `del'`, `card'` und `.∈'.` unter `is_Set` zu zeigen. Im Beispiel sind nur ∅', `add'` und `del'` betroffen, da nur diese Funktionen Listen als Ergebnis haben. Es ist also zu zeigen:

  ```
  is_Set(∅');
  ∀x,s. is_Set(s) ⇒ is_Set(add'(x,s));
  ∀x,s. is_Set(s) ⇒ is_Set(del'(x,s));
  ```

- Bedingung 2 wird modelliert durch Angabe einer *Kongruenzrelation*, die alle Listen gleichsetzt, die dieselbe Menge repräsentieren:

  ```
  .∼. : α::EQ ⇒ List α × List α → Bool;
  ```

  Es gilt $a \sim b$, genau dann wenn a und b Mengen repräsentieren und Permutationen voneinander sind. Hier ist zu beweisen, daß `.∼.` tatsächlich eine Kongruenzrelation ist, das heißt daß sie reflexiv, symmetrisch und transitiv ist und bezüglich `add'` und `del'` eine Kongruenz ist:

  ```
  ∀s. is_Set(s) ⇒ s∼s;
  ∀s,s'. is_Set(s) ∧ is_Set(s') ⇒ s∼s' = s'∼s;
  ∀s,s',s''. is_Set(s) ∧ is_Set(s') ∧ is_Set(s'') ⇒ s∼s' ∧ s'∼s'' ⇒ s∼s'';
  ∀x,s,s'. is_Set(s) ∧ is_Set(s') ∧ s∼s' ⇒ add'(x,s)∼add'(x,s');
  ∀x,s,s'. is_Set(s) ∧ is_Set(s') ∧ s∼s' ⇒ del'(x,s)∼del'(x,s');
  ```

- Nun kann die Beziehung zwischen den auf Mengen operierenden Funktionen und ihren Listen-Repräsentationen hergestellt werden. Dies geschieht durch Angabe eines Homomorphismus

[2]Um die Darstellung der Vorgehensweise knapp zu halten, wird hier darauf verzichtet, die Axiomatisierung dieser Funktionen anzugeben. Eine ausführliche Darstellung dieses Beispiels (Repräsentation von Mengen durch Listen in SPECTRUM) findet sich in [Slo95]. Pepper et. al. [PBDD94] geben mit der Realisierung von Mengen als Hashtabellen ein sehr ähnliches Beispiel an.

```
abs : List α → Set α;

Set α generated by abs;

axioms β::EQ ⇒ ∀x:α, s:List α, y:β, s':List β in
is_Set(s) ⇒ δ(abs(s));

abs(∅') = ∅;
abs(add'(x,s)) = add(x,abs(s));
abs(del'(x,s)) = del(x,abs(s));
card'(s) = card(abs(s));
y ∈' s' = y ∈ abs(s');
endaxioms;
```

Man beachte, daß `abs` auf allen Listen definiert ist, die eine Menge repräsentieren (erstes Axiom). Die Zeile `Set` α `generated by abs` besagt, daß die Repräsentation durch Listen vollständig ist, das heißt daß jede beliebige Menge als Abstraktion `abs` einer Liste darstellbar ist.

- Der Zusammenhang zwischen der auf Listen definierten Kongruenz .∼. und der Gleichheit auf Mengen wird spezifiziert durch das Axiom:

```
is_Set(s) ∧ is_Set(s') ⇒ s∼s' = (abs(s)=abs(s'));
```

Lassen sich aus den Axiomen der so gewonnenen Spezifikation `Set_by_List` die Axiome von `Set` mittels des SPECTRUM-Kalküls herleiten, so wurde eine korrekte Realisierung von Mengen durch Listen angegeben.

Es ist eine interessante Tatsache, daß der hier beschriebene Ansatz des Datenstrukturwechsels mit Hilfe der Stütz- und der Verfeinerungsrelation ausgedrückt werden kann. Eine detaillierte Darstellung dieses Ergebnisses findet sich in [Slo95]. Die beiden Begriffe der Verfeinerung und des Datenstrukturwechsels werden in SPECTRUM unter dem Begriff der *Realisierung* zusammengefaßt.

2.2.3 Kalkül

SPECTRUM-Spezifikationen beschreiben die Eigenschaften von Sorten und Funktionen mit Hilfe logischer Formeln der Prädikatenlogik erster Stufe mit Gleichheit und einem Erzeugungsprinzip. Die Definition eines logischen Kalküls, des SPECTRUM-Kalküls, erlaubt es nun, aus einer Menge solcher Formeln (Eigenschaften) weitere Eigenschaften herzuleiten. Die wichtigste Anwendung dieses Kalküls besteht im Nachweis der Verifikationsbedingungen, die beim Etablieren einer Verfeinerungsrelation oder beim Durchführen eines Datenstrukturwechsels anfallen.

Eine Besonderheit der Sprache SPECTRUM ist die in ihr vorgenommene Identifikation der Ebene der Formeln mit der Termebene. Formeln sind in SPECTRUM nichts anderes als Terme der Sorte `Bool`. Da `Bool` im Gegensatz zu den Wahrheitswerten dreiwertig ist (`true`, `false`, $\perp$), ist ein klassischer zweiwertiger prädikatenlogischer Kalkül zur Unterstützung der Verifikation in SPECTRUM ungeeignet. Es wurde deshalb ein dreiwertiger Kalkül definiert, der Gentzen's Kalkül des natürlichen Schließens [Gen35] ähnelt. Aufgrund der Dreiwertigkeit der Logik ergab sich ein sehr umfangreicher Kalkül mit 68 Regeln, der in [Reg94] ausführlich beschrieben ist.

Im folgenden wird die formale Verifikation mit dem SPECTRUM-Kalkül anhand eines Beispiels demonstriert, das sich, obwohl ohne praktische Relevanz, aufgrund seiner Kürze gut präsentieren läßt:

Beispiel 5 *Es soll aus der Spezifikation* `Set` *(Abbildung 2.3) hergeleitet werden, daß die Kardinalität der Menge, die entsteht, wenn zweimal dasselbe Element in die leere Menge eingetragen wird, gleich eins ist.*

Informell ist klar, wie der Beweis dieser einfachen Eigenschaft mit Hilfe der `Set`*-Axiome abläuft:*

$$
\begin{aligned}
&\texttt{∀x. card(add(x,add(x,∅)))} = \texttt{1} && \overset{(1.\ \texttt{Set}-Axiom)}{\Leftrightarrow}\\
&\texttt{∀x. card(add(x,∅))} = \texttt{1} && \overset{(8.\ \texttt{Set}-Axiom,\ x \notin \emptyset)}{\Leftrightarrow}\\
&\texttt{1 + card(∅)} = \texttt{1} && \overset{(5.\ \texttt{Set}-Axiom)}{\Leftrightarrow}\\
&\texttt{1 + 0} = \texttt{1} && \overset{(\texttt{Nat}-Axiome)}{\Leftrightarrow}\\
&\texttt{1} = \texttt{1} && \overset{(Refl.\ von\ =)}{\Leftrightarrow}\\
&\texttt{true}
\end{aligned}
$$

Führt man diesen Beweis anstatt in der oben angegebenen mathematischen Notation formal mit dem SPECTRUM*-Kalkül, so ist es möglich, die Korrektheit des Beweises automatisch überprüfen und — in einfachen Fällen wie dem gegebenen — auch automatisch finden zu lassen. Dazu existiert eine Instanz des generischen interaktiven Beweissystems* ISABELLE *[Pau94] für den* SPECTRUM*-Kalkül. Der Nachteil dieses formalen Vorgehens ist, daß der Beweis wesentlich detaillierter und damit umfangreicher wird, was auch die Verständlichkeit beeinträchtigen kann.*

Im SPECTRUM*-Kalkül wird ein formaler Beweis als sogenannter* Beweisbaum *dargestellt. Die Wurzel dieses Baums enthält das* Beweisziel *in Form einer sogenannten* Sequenz*. Im vorliegenden Beispiel ist die Sequenz* `Set`$\blacktriangleright$`∀x.card(add(x,add(x,∅)))=1` *zu beweisen. Das Zeichen* $\blacktriangleright$ *stellt dabei das in* SPECTRUM *verwendete Herleitungszeichen dar. Die gegebene Sequenz kann also gelesen werden als "Die Formel* `∀x.card(add(x,add(x,∅)))=1` *kann im* SPECTRUM*-Kalkül aus den Axiomen der Spezifikation* `Set` *hergeleitet werden". Der Beweisbaum für eine solche Sequenz setzt sich aus Anwendungen von Kalkülregeln der allgemeinen Form*

$$\frac{H_1 \blacktriangleright p_1 \quad \ldots \quad H_n \blacktriangleright p_n}{H \blacktriangleright p}\ (id)$$

*zusammen. Diese Regel trägt den Namen (id) und konstruiert einen Beweisbaum für die Sequenz $H \blacktriangleright p$ aus Beweisbäumen für die Sequenzen (*Prämissen*) $H_1 \blacktriangleright p_1, \ldots, H_n \blacktriangleright p_n$. Die Blätter des Beweisbaums werden durch Kalkülregeln ohne Vorbedingungen gebildet, also Regeln der Form*

$$\frac{}{H \blacktriangleright p}\ (id)$$

genügen. Der Beweisbaum für ein bestimmtes Beweisziel wird meist durch "Rückwärtsanwendung" von Kalkülregeln aufgebaut, das heißt, das Ziel wird solange in einfachere Teilziele aufgespalten, bis alle Teilziele Instanzen von Regeln ohne Vorbedingungen sind.

Im folgenden wird der Beweisbaum für das obige Beispiel angegeben. Aus Platzgründen wird er in eine Reihe von Teilbäumen zerlegt angegeben. Der Beweis wird nicht vollständig angegeben, sondern es wird angenommen, daß in der Theorie der natürlichen Zahlen bereits ein Beweis für

$$\texttt{Nat} \blacktriangleright \texttt{1+0=1}$$

gefunden wurde. Dann gilt:

$$\dfrac{\dfrac{\texttt{Nat} \blacktriangleright \texttt{1+0=1}}{\texttt{Set} \blacktriangleright \texttt{1+0=1}}\ (weak) \quad \dfrac{}{\texttt{Set} \blacktriangleright \texttt{card(}\emptyset\texttt{)=0}}\ (hyp)}{\texttt{Set} \blacktriangleright \texttt{1+card(}\emptyset\texttt{)=1}}\ (subst)$$

$$\dfrac{\dfrac{\dfrac{}{\texttt{Set} \blacktriangleright \forall \texttt{x}.\forall \texttt{s}.\neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)}}\ (hyp) \quad \dfrac{}{\emptyset \blacktriangleright (\forall x.p) = (\forall^{\perp} x.\delta(x) \Rightarrow p)}\ (ALL\_Def)}{\texttt{Set} \blacktriangleright \forall^{\perp} \texttt{x}.\delta\texttt{(x)} \Rightarrow \forall \texttt{s}.\neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)}}\ (subst)}{\texttt{Set} \blacktriangleright \delta\texttt{(x)} \Rightarrow \forall \texttt{s}.\neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)}}\ (ALLbE)$$

$$\dfrac{\texttt{Set} \blacktriangleright \delta\texttt{(x)} \Rightarrow \forall \texttt{s}.\neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)} \quad \dfrac{}{\delta\texttt{(x)} \blacktriangleright \delta\texttt{(x)}}\ (hyp)}{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \forall \texttt{s}.\neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)}}\ (mp)$$

$$\dfrac{\dfrac{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \forall \texttt{s}.\neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)} \quad \dfrac{}{\emptyset \blacktriangleright (\forall x.p) = (\forall^{\perp} x.\delta(x) \Rightarrow p)}\ (ALL\_Def)}{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \forall^{\perp} \texttt{s}.\delta\texttt{(s)} \Rightarrow \neg(\texttt{x} \in \texttt{s}) \Rightarrow \texttt{card(add(x,s))=1+card(s)}}\ (subst)}{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \delta(\emptyset) \Rightarrow \neg(\texttt{x} \in \emptyset) \Rightarrow \texttt{card(add(x,}\emptyset\texttt{))=1+card(}\emptyset\texttt{)}}\ (ALLbE)$$

$$\dfrac{\dfrac{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \delta(\emptyset) \Rightarrow \neg(\texttt{x} \in \emptyset) \Rightarrow \texttt{card(add(x,}\emptyset\texttt{))=1+card(}\emptyset\texttt{)} \quad \dfrac{}{\texttt{Set} \blacktriangleright \delta(\emptyset)}\ (hyp)}{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \neg(\texttt{x} \in \emptyset) \Rightarrow \texttt{card(add(x,}\emptyset\texttt{))=1+card(}\emptyset\texttt{)}}\ (mp) \quad \dfrac{}{\texttt{Set} \blacktriangleright \neg(\texttt{x} \in \emptyset)}\ (hyp)}{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \texttt{card(add(x,}\emptyset\texttt{))=1+card(}\emptyset\texttt{)}}\ (mp)$$

$$\dfrac{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \texttt{card(add(x,}\emptyset\texttt{))=1+card(}\emptyset\texttt{)} \quad \texttt{Set} \blacktriangleright \texttt{1+card(}\emptyset\texttt{)=1}}{\texttt{Set},\delta\texttt{(x)} \blacktriangleright \texttt{card(add(x,}\emptyset\texttt{))=1}}\ (subst)$$

$$\dfrac{\dfrac{\dfrac{}{\texttt{Set}\blacktriangleright\forall\texttt{x}.\,\forall\texttt{s}.\,\texttt{add(x,add(x,s))}=\texttt{add(x,s)}}\,(hyp)\quad\dfrac{}{\emptyset\blacktriangleright(\forall x.p)=(\forall^{\perp}x.\delta(x)\Rightarrow p)}\,(ALL\_Def)}{\texttt{Set}\blacktriangleright\forall^{\perp}\texttt{x}.\,\delta\texttt{(x)}\Rightarrow\forall\texttt{s}.\,\texttt{add(x,add(x,s))}=\texttt{add(x,s)}}\,(subst)}{\texttt{Set}\blacktriangleright\delta\texttt{(x)}\Rightarrow\forall\texttt{s}.\,\texttt{add(x,add(x,s))}=\texttt{add(x,s)}}\,(ALLbE)$$

$$\dfrac{\dfrac{\texttt{Set}\blacktriangleright\delta\texttt{(x)}\Rightarrow\forall\texttt{s}.\,\texttt{add(x,add(x,s))}=\texttt{add(x,s)}\quad\dfrac{}{\delta\texttt{(x)}\blacktriangleright\delta\texttt{(x)}}\,(hyp)}{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\forall\texttt{s}.\,\texttt{add(x,add(x,s))}=\texttt{add(x,s)}}\,(mp)$$

$$\dfrac{\dfrac{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\forall\texttt{s}.\,\texttt{add(x,add(x,s))}=\texttt{add(x,s)}\quad\dfrac{}{\emptyset\blacktriangleright(\forall x.p)=(\forall^{\perp}x.\delta(x)\Rightarrow p)}\,(ALL\_Def)}{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\forall^{\perp}\texttt{s}.\,\delta\texttt{(s)}\Rightarrow\texttt{add(x,add(x,s))}=\texttt{add(x,s)}}\,(subst)}{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\delta\texttt{(}\emptyset\texttt{)}\Rightarrow\texttt{add(x,add(x,}\emptyset\texttt{))}=\texttt{add(x,}\emptyset\texttt{)}}\,(ALLbE)$$

$$\dfrac{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\delta\texttt{(}\emptyset\texttt{)}\Rightarrow\texttt{add(x,add(x,}\emptyset\texttt{))}=\texttt{add(x,}\emptyset\texttt{)}\quad\dfrac{}{\texttt{Set}\blacktriangleright\delta\texttt{(}\emptyset\texttt{)}}\,(hyp)}{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\texttt{add(x,add(x,}\emptyset\texttt{))}=\texttt{add(x,}\emptyset\texttt{)}}\,(mp)$$

$$\dfrac{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\texttt{card(add(x,}\emptyset\texttt{))}=1\quad\texttt{Set},\delta\texttt{(x)}\blacktriangleright\texttt{add(x,add(x,}\emptyset\texttt{))}=\texttt{add(x,}\emptyset\texttt{)}}{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\texttt{card(add(x,add(x,}\emptyset\texttt{)))}=1}\,(subst)$$

$$\dfrac{\dfrac{\dfrac{\texttt{Set},\delta\texttt{(x)}\blacktriangleright\texttt{card(add(x,add(x,}\emptyset\texttt{)))}=1\quad\dfrac{}{\emptyset\blacktriangleright\delta(\delta(x))}\,(strong\_DEF)}{\texttt{Set}\blacktriangleright\delta\texttt{(x)}\Rightarrow\texttt{card(add(x,add(x,}\emptyset\texttt{)))}=1}\,(impI)}{\texttt{Set}\blacktriangleright\forall^{\perp}\texttt{x}.\,\delta\texttt{(x)}\Rightarrow\texttt{card(add(x,add(x,}\emptyset\texttt{)))}=1}\,(ALLbI)$$

$$\dfrac{\dfrac{\texttt{Set}\blacktriangleright\forall^{\perp}\texttt{x}.\,\delta\texttt{(x)}\Rightarrow\texttt{card(add(x,add(x,}\emptyset\texttt{)))}=1\quad\dfrac{}{\emptyset\blacktriangleright(\forall x.p)=(\forall^{\perp}x.\delta(x)\Rightarrow p)}\,(ALL\_Def)}{\texttt{Set}\blacktriangleright\forall\texttt{x}.\,\texttt{card(add(x,add(x,}\emptyset\texttt{)))}=1}\,(subst)$$

Durch Zusammensetzen aller dieser Teilbäume entsteht der Beweisbaum für die gesuchte Sequenz. Die Regelanwendungen in diesem Baum sind zur Verdeutlichung mit den in [Reg94] angegebenen Bezeichnern der verwendeten Regeln versehen. Auf eine detaillierte Erklärung der Regeln des SPECTRUM*-Kalküls wird in dieser Arbeit verzichtet. Stattdessen wird auf [Reg94] verwiesen, wo alle Regeln des Kalküls sowie eine Reihe von daraus abgeleiteten Regeln vorgestellt werden.* □

Kapitel 3

Eine axiomatische Semantik für das E/R-Modell

Wie in Abschnitt 2.1 gezeigt, erlaubt es das E/R-Modell, Daten als Mengen von Entities zu beschreiben, die durch Attribute charakterisiert werden und miteinander über Relationships in Beziehung stehen können. Ein E/R-Diagramm legt durch die in ihm enthaltenen Entity- und Relationshiptypen die Struktur der betrachteten Daten fest. Die ebenfalls im Diagramm gegebenen statischen Integritätsbedingungen schränken die Menge der erlaubten Zustände des modellierten Datenbestandes zusätzlich ein.

Die Semantik des E/R-Modells wird in der Regel wie in Abschnitt 2.1 informell gegeben. Regelwerke zur Umsetzung von E/R-Schemata in andere bekannte Datenmodelle wie das relationale Datenmodell, das hierarchische Datenmodell oder das Netzwerkmodell bewirken ebenfalls eine (teilweise) Festlegung der Semantik des E/R-Modells. Keiner der angegebenen Punkte eignet sich jedoch zur Festlegung einer vollständig formalen Semantik:

- Das Netzwerk- und das hierarchische Datenmodell besitzen selbst keine formale Semantik, sondern sind informell definiert. Darüberhinaus sind diese Datenmodelle weniger ausdrucksmächtig als das E/R-Modell, da sich die statischen Integritätsbedingungen nicht immer im Netzwerk- beziehungsweise im hierarchischen Modell darstellen lassen.
- Das relationale Datenmodell ist durch die ihm zugrundeliegende relationale Algebra mathematisch wesentlich besser fundiert als das Netzwerk- oder das hierarchische Modell. Jedoch lassen sich auch im relationalen Modell die statischen Integritätsbedingungen des E/R-Modells nicht vollständig ausdrücken.

Die Definition einer formalen Semantik für das E/R-Modell ist auf zwei unterschiedliche Weisen denkbar:

- Es kann eine Abbildung angegeben werden, die E/R-Schemata direkt auf mathematische Strukturen, wie zum Beispiel Algebren, abbildet. Dies ist der von der im Ab-

schnitt 1.3 angesprochenen Braunschweiger Arbeitsgruppe gewählte Weg [EGH+90, KG90, GH91, Hoh93, Gog94].

- Es kann eine (rein syntaktische) Abbildung in eine formal fundierte Beschreibungstechnik angegeben werden. Ein E/R-Schema erhält seine formale Semantik so durch die formale Semantik der dem Schema zugeordneten Beschreibung. So kann die Semantik des E/R-Modells zum Beispiel mit Hilfe einer Übersetzungsvorschrift definiert werden, die es erlaubt, E/R-Schemata in SPECTRUM-Spezifikationen zu übersetzen. Ein solches Vorgehen sieht auf den ersten Blick wie ein Umweg aus, es bringt aber neben einer formalen Semantikdefinition für das E/R-Modell auch Vorteile für die verwendete formale Technik. Durch die Abbildung wird das E/R-Modell als Beschreibungstechnik zur Datenmodellierung *innerhalb* der formalen Sprache zugänglich. Die Daten eines formal entwickelten Systems können dadurch in sehr klarer und verständlicher Weise als E/R-Schema ausgedrückt werden, ohne daß dadurch die Welt der formalen Software-Entwicklung verlassen würde. Da dies ein für die Welt der formalen Techniken sehr wichtiges Resultat ist, wurde in der vorliegenden Arbeit dieser zweite Weg gewählt.

Die formale Semantik des E/R-Modells wird im folgenden also durch Angabe einer Übersetzungsvorschrift definiert, die es erlaubt, E/R-Schemata nach SPECTRUM zu transformieren.

3.1 Konventionen

Um die Präsentation der vorgestellten Konzepte lesbar und verständlich zu halten, hält die vorliegende Arbeit eine Reihe von Konventionen ein. Diese sollen im folgenden kurz vorgestellt werden.

SPECTRUM-Spezifikationstext wird in `Schreibmaschinenschrift` dargestellt. Enthalten Spezifikationen (etwa bei der Präsentation des Übersetzungsschemas) Platzhalter für Bezeichner des E/R-Schemas (Entitytypen, Relationshiptypen, Attribute, ...), so werden diese durch *`Kursivschrift`* hervorgehoben. Die in konkreten E/R-Diagrammen auftretenden Bezeichner sind in serifenfreier Schrift dargestellt. Auf diese Weise können die aus den Diagrammen stammenden Bezeichner (etwa für Entitytypen) und ihre Entsprechungen in der zugehörigen SPECTRUM-Spezifikation (Bezeichner für Entitysorten) unterschieden werden. In den meisten Fällen ist diese subtile Unterscheidung jedoch irrelevant und kann beim Lesen vernachlässigt werden.

Es wird angenommen, daß alle in der folgenden Übersetzungsvorschrift neu eingeführten Bezeichner (`Db`, `db`, ...) im zu übersetzenden Schema nicht vorkommen. Sollte dies der Fall sein, so sind in der Übersetzung entsprechend andere, neue Bezeichner zu verwenden.

Bei E/R-Diagrammen ist das konkrete Layout der Darstellung, also zum Beispiel die Anordnung der Knoten und Kanten auf der Zeichenfläche, für die Bedeutung des Diagramms irrelevant. Bei der Übersetzung eines derartigen Diagramms in eine Textdarstellung kann

die Anordnung, das heißt die Reihenfolge, der vorkommenden Bezeichner oftmals bedeutsam werden. Als Beispiel soll ein zweistelliger Relationshiptyp liefert zwischen den Entitytypen Lieferant und Teil dienen. Im E/R-Diagramm ist es völlig ohne Bedeutung, wie dieser Relationshiptyp angeordnet ist. Repräsentiert man liefert jedoch in einer SPECTRUM-Spezifikation durch ein charakteristisches Prädikat, wie es in Abschnitt 3.3 geschehen wird, wird die Reihenfolge der Entitytypen relevant, da es syntaktisch einen Unterschied macht, ob das Prädikat die Signatur

```
liefert : Lieferant × Teil → Bool
```

oder die Signatur

```
liefert : Teil × Lieferant → Bool
```

besitzt. Aus diesem Grund wird vorausgesetzt, daß ein beliebiges, aber fest gewähltes und eindeutiges, Ordnungskriterium existiert, das es erlaubt, Aufzählungen von Bezeichnern aus dem E/R-Diagramm zu ordnen. In der angegebenen Übersetzungsvorschrift werden solche Aufzählungen[1] immer als gemäß diesem Kriterium geordnet angenommen.

In den angegebenen Beispielen wird die lexikographische Ordnung als Kriterium gewählt. Bei der Aufzählung der an einem Relationshiptyp beteiligten Entitytypen ist dieses Kriterium jedoch nicht eindeutig, deshalb werden die Entitytypen in diesem Fall nach den Bezeichnern der Rollen lexikographisch geordnet, in denen sie am Relationshiptyp teilnehmen.

3.2 Attribute

Entity/Relationship-Ansätze wie der in Abschnitt 2.1 vorgestellte machen im allgemeinen keine Aussagen über die Attributtypen, die dem Datenmodellierer zur Verfügung stehen. Für diese mangelnde Präzision gibt es verschiedene mögliche Gründe:

- In frühen Phasen der Softwareentwicklung möchte man sich noch nicht auf bestimmte Attributtypen festlegen. Man wählt also beliebige Bezeichner aus dem Anwendungskontext als Attributtypen und legt erst in späteren Phasen der Entwicklung fest, durch welche Datenstrukturen diese realisiert werden sollen. Jede Festlegung eines Typsystems für Attributtypen würde in dieser Phase eine Einschränkung und Verringerung des Abstraktionsgrades bedeuten.

- In späteren Phasen der Datenbankentwicklung werden die Attributtypen durch die Datenstrukturen implementiert, die das ausgewählte Datenbankmanagementsystem

[1] Aufzählungen werden in der Übersetzungsvorschrift meist in "Pünktchen-Notation" angegeben (a. . . z).

zur Verfügung stellt. Auch hier können also die Attributtypen nicht durch das E/R-Modell vorgegeben werden, sondern hängen von der Wahl des Datenbanksystems ab, das zur Implementierung des Datenschemas dient.

Um ein E/R-Datenschema als (vollständiges) formales Dokument im Sinne axiomatischer Spezifikation verstehen zu können, müssen auch die Attributtypen festgelegt werden. In der Spezifikationssprache SPECTRUM können die Attributtypen als abstrakte Datentypen, das heißt als Sorten zusammen mit charakteristischen Operationen, frei spezifiziert werden. In diesem Ansatz können Attributtypen sowohl unvollständig (Unterspezifikation in frühen Phasen) als auch detailliert und implementierungsnah (in späteren Phasen) spezifiziert werden.

Attributtypen werden in der zugehörigen SPECTRUM-Spezifikation also als Sorten repräsentiert (*Attributsorten*). Zur Spezifikation der Attributsorten wird das mächtige und sehr flexible Sortensystem der Sprache eingesetzt. Da das in SPECTRUM verfügbare Polymorphiekonzept in der E/R-Modellierung unbekannt ist, wird im weiteren auf polymorphe Attributsorten verzichtet. Attributsorten können in dieser Arbeit durch beliebige Sorten-Grundterme der Sprache SPECTRUM dargestellt werden. Damit sind Sorten wie `List Nat` als Attributsorten erlaubt, während polymorphe Sorten wie `List` α nicht als Attributsorten verwendet werden dürfen. Diese Einschränkung wird die Präsentation der in dieser Arbeit vorgestellten Konzepte wesentlich vereinfachen.

3.3 Statische Semantik

Im hier vorgestellten Ansatz werden die durch das E/R-Schema beschriebenen Daten so durch die Klasse der Modelle der zugehörigen (losen) Spezifikation repräsentiert, daß jede Algebra der Modellklasse genau einem (gemäß dem E/R-Schema) erlaubten Datenzustand entspricht. Die Sorten dieser Spezifikation entsprechen dabei den Entitytypen, während die Operationen gerade so spezifiziert sind, daß sie die Attribute und Relationshiptypen repräsentieren. Die statischen Integritätsbedingungen des Schemas werden in der Spezifikation durch Axiome wiedergegeben. Für die so definierte Semantik wird im weiteren der Begriff *statische Semantik des E/R-Modells* verwendet werden.

Der Begriff der statischen Semantik wird hier verwendet, obwohl er bereits im Bereich der Definition von Programmiersprachen mit einer klar definierten Bedeutung belegt ist. Dies kann durch folgende Analogie zwischen der Definition der statischen Semantik einer Programmiersprache und der E/R-Datenmodellierung gerechtfertigt werden: Die statische Semantik (oft auch kontextsensitive Syntax genannt) einer Programmiersprache beschreibt alle Eigenschaften von Programmen dieser Sprache, die statisch, das heißt zur Übersetzungszeit überprüft werden können. Das ist zum einen die Struktur der Sprache (kontextfreie Syntax) wie auch die zusätzlichen Kontextbedingungen (zum Beispiel Typkorrektheit). Ein E/R-Datenschema beschreibt alle Aspekte eines Informationssystems, die sich statisch überprüfen lassen. Statisch heißt hier, daß jeweils der aktuelle Zustand der

verwalteten Datenbank ausreicht, um die Bedingungen zu prüfen. Es ist dazu kein Wissen über die Vorgeschichte des aktuellen Datenzustands erforderlich. Das E/R-Schema beschreibt ebenfalls die Struktur der verwalteten Daten wie auch zusätzliche Anforderungen, die hier statische Integritätsbedingungen genannt werden. Der Vergleich mit der Definition von Programmiersprachen wird also durch eine andere Definition des Begriffes "statisch" im Falle von Informationssystemen ermöglicht. Dennoch erscheint es gerechtfertigt zu sagen, daß die in einem E/R-Schema enthaltene Information die statische Semantik eines Informationssystems beschreibt.

Während jedoch bei der Definition von Programmiersprachen die Ausdrucksmittel zur Beschreibung von Kontextbedingungen (als Ausdrücke über synthetisierten und vererbten Attributen der abstrakten Syntax) ausreichen, um die typischerweise an solche Sprachen gestellten statischen Anforderungen auszudrücken, ist dies beim E/R-Modell nicht der Fall. Es sind bei der Spezifikation von Informationssystemen sehr häufig statische Integritätsbedingungen denkbar, die im E/R-Modell nicht ausgedrückt werden können. Auf diesen Punkt wird im weiteren Verlauf der Arbeit noch eingegangen werden.

In der folgenden Definition (Definition 1) wird eine Übersetzungsvorschrift angegeben, die es erlaubt, ein Entity/Relationship-Schema *ER* in eine SPECTRUM-Spezifikation zu übersetzen. In dieser Vorschrift werden die folgenden Platzhalter für die aus dem Schema stammenden Bezeichner verwendet: $E_1, \ldots, E_n$ bezeichnen die in *ER* vorkommenden Entitytypen und $R_1, \ldots, R_m$ die Relationshiptypen. Die Attribute der Entitytypen werden mit $attr_j^{E_i}$ ($1 \leq i \leq n$, $1 \leq j \leq$ *Anzahl der Attribute von* E_i) bezeichnet, die zugehörigen Attributtypen mit $Attr_j^{E_i}$. Dabei bezeichne $attr_j^{E_i}{:}Attr_j^{E_i}$ das j-te Attribut des Entitytyps E_j.

Definition 1 setzt die Existenz einer Spezifikation voraus, in der die Attributsorten $Attr_j^{E_i}$ definiert sind. Darüberhinaus wird eine polymorphe Mengenspezifikation mit einem Enthaltenseinsprädikat .∈. und einer Funktion `card` zur Berechnung der Kardinalität gefordert (siehe Abbildung 2.3). Zur Modellierung der Optionalität von Attributen wird in Definition 1 die folgende Spezifikation `Opt` verwendet:

```
Opt = {
data Opt α = UNDEF | attr(!α);
}
```

`Opt` spezifiziert einen Sortenkonstruktor `Opt`, mit dessen Hilfe eine beliebige Sorte α um ein neues Element `UNDEF` erweitert wird. Die Definition dieses Sortenkonstruktors erfolgt mit Hilfe des `data`-Konstruktes. Dieses Konstrukt dient wie in funktionalen Programmiersprachen zur Definition von freien Datentypen. Die obige Definition entspricht sogar syntaktisch mit einer kleinen Ausnahme der entsprechenden Definition in der funktionalen Sprache HASKELL. Diese Ausnahme betrifft die Striktheit von Konstruktorfunktionen. Während in HASKELL Konstruktoren immer nichtstrikt sind, ist es in SPECTRUM möglich, die Striktheit dieser Funktionen zu spezifizieren. Dies geschieht durch das Ausrufezeichen in obiger Spezifikation. Es verlangt die Striktheit der Konstruktorfunktion `attr`.

Definition 1 — Statische Semantik eines E/R-Schemas —
Die Signatur der zu einem E/R-Schema ER *gehörenden Spezifikation enthält zu jedem in* ER *enthaltenen Entitytyp eine Sorte. Da die Elemente dieser Sorten später mit Hilfe endlicher Mengen verwaltet werden sollen, werden sie bereits hier als flach geordnet, also als Elemente der Sortenklasse* EQ*, spezifiziert.*

sort $E_1, \ldots, E_n$;
$E_1, \ldots, E_n$::EQ;

Attribute der Entitytypen werden durch strikte und totale Zugriffsfunktionen auf diesen Sorten repräsentiert. Es existieren also für alle Entitytypen und Attribute Funktionsdeklarationen der Form

$attr_j^{E_i}$: $E_i \rightarrow A_j^{E_i}$;
$attr_j^{E_i}$ strict total;

Optionalität eines Attributes wird mit Hilfe des oben eingeführten Sortenkonstruktors Opt *modelliert, der den zusätzlichen Wert* UNDEF *zur Verfügung stellt. Sei also der Typ des Attributs* $attr_j^{E_i}$ *mit* $Attr_j^{E_i}$ *bezeichnet und* $attr_j^{E_i}$ *ein optionales Attribut, so steht* $A_j^{E_i}$ *in obiger Signatur für* Opt $Attr_j^{E_i}$*, andernfalls für* $Attr_j^{E_i}$*.*

Relationshiptypen werden durch Prädikate dargestellt, die entscheiden, ob zwischen Entities der beteiligten Entitytypen eine Beziehung besteht oder nicht.

Ist R_i *ein* n_i*-stelliger Relationshiptyp, an dem die Entitytypen* $E_1^{R_i}, \ldots, E_{n_i}^{R_i}$ *beteiligt sind, so enthält die Signatur der E/R-Spezifikation folgende Funktionsdeklaration:*

R_i : $E_1^{R_i} \times \ldots \times E_{n_i}^{R_i} \rightarrow$ Bool;
R_i strict total;

Die statischen Integritätsbedingungen des E/R-Schemas spiegeln sich in den Axiomen der E/R-Spezifikation wieder. Diese Axiome sind so zu wählen, daß der durch die Spezifikation beschriebene Zustandsraum (Modellklasse) genauso eingeschränkt wird, wie dies durch die Integritätsbedingungen gefordert wird. Wie in Abschnitt 2.1 gezeigt lassen sich beim E/R-Modell zwei unterschiedliche Arten von Integritätsbedingungen angeben:

1. ***Schlüsselbedingungen:*** *Schlüssel dienen der eindeutigen Identifizierung von Entities eines Entitytyps. Stimmen zwei Entities* e_1, e_2 *eines Entitytyps* E_i *in allen Komponenten des Schlüssels überein, so gilt* $e_1 = e_2$*. Hat also* E_i *einen Schlüssel, der aus k Komponenten besteht, so ergibt sich ein Axiom der Form*

   $\forall e_1, e_2 : E_i .\ EQID_1 \wedge \ldots \wedge EQID_k \Rightarrow e_1 = e_2$

   Für welchen Term das Konstrukt $EQID_j$ *steht, hängt dabei von der Art der j-ten Schlüsselkomponente ab. Handelt es sich dabei um einen internen Identifikator, so steht* $EQID_j$ *für*

$attr_j^{E_i}(e_1) = attr_j^{E_i}(e_2)$

Ist die j-te Komponente ein externer Identifikator, also ein zweistelliger Relationshiptyp R mit (1,1)-Beteiligung, an dem E_i teilnimmt, so steht $EQID_j$ für:

$\forall e' : E'.\ R\ (e', e_1) = R\ (e', e_2)$

Dabei bezeichne E' den zweiten an R beteiligten Entitytyp.

2. ***Beteiligung an Relationshiptypen:*** *Für jede Rolle gibt ein Tupel (min,max) an, wie oft Entities des zugehörigen Entitytyps E_1 an einem Relationshiptyp R_j mindestens teilnehmen müssen beziehungsweise höchstens teilnehmen dürfen. Die E/R-Spezifikation muß also für jede Rolle ein Axiom enthalten, das diese Eigenschaft ausdrückt. Im folgenden seien die übrigen an R_j beteiligten Entitytypen mit E_2, ..., E_k bezeichnet. Dann ergibt sich folgendes Axiom:*

$$\forall e_1 : E_1.\ \exists s : Set\ (E_1 \times E_2 \times \ldots \times E_k).\ \forall e_2 : E_2, \ldots, e_k : E_k.$$
$$((e_1, \ldots, e_k) \in s \Leftrightarrow R_j\ (e_1, \ldots, e_k)) \wedge min \leq card(s) \wedge card(s) \leq max$$

Ist die maximale Beteiligung $max = *$, so vereinfacht sich das Axiom zu:

$$\forall e_1 : E_1.\ \exists s : Set\ (E_1 \times E_2 \times \ldots \times E_k).\ \forall e_2 : E_2, \ldots, e_k : E_k.$$
$$((e_1, \ldots, e_k) \in s \Leftrightarrow R_j\ (e_1, \ldots, e_k)) \wedge min \leq card(s)$$

Eine Beteiligung von (0,$*$) bedeutet für einen Entitytyp keine Beschränkung bezüglich der Beteiligung an einem Relationshiptyp. Deshalb wird in diesem Fall kein Axiom generiert.

□

Es ist eine interessante Beobachtung, daß die obige Definition bei der Übersetzung von Schlüsselbedingungen keinen Gebrauch von der geforderten (1,1)-Beteiligung von Entitytypen an externen Identifikatoren macht. Die Formalisierung würde also eine Erweiterung des Begriffs "externer Identifikator" auf beliebige zweistellige Relationshiptypen zulassen.

Die oben informell angegebene Umsetzung von E/R-Schemata in zugehörige SPECTRUM-Spezifikationen ist vollständig mechanisch durchführbar. Ein Werkzeug, das zu diesem Zweck eingesetzt werden kann, ist in [Hub94] beschrieben. Eine Implementierung der in der vorliegenden Arbeit gegebenen Übersetzungsvorschriften wurde im Rahmen eines Fortgeschrittenenpraktikums [Sal95] mit diesem Werkzeug erstellt.

Beispiel 6 *Die Spezifikationen der natürlichen Zahlen, der polymorphen Mengen sowie der Attributsorten seien mit* `Nat`, `Set` *beziehungsweise* `AttrSort` *bezeichnet. Dann ergibt sich für das Datenschema der RECHERCHE-Applikation aus Beispiel 2 gemäß obiger Übersetzungsvorschrift folgende Spezifikation*[2]*:*

[2]Die Zeile `strict total` zu Beginn dieser Spezifikation ist eine abkürzende Schreibweise und fordert (wie in obigem Übersetzungsschema verlangt) Striktheit und Totalität von allen eingeführten Funktionen.

```
RECHERCHE = { enriches Opt + Nat + Set + AttrSort;

strict total;

sort Autor, Publikation, Schlagwort, Verlag;
Autor, Publikation, Schlagwort, Verlag :: EQ;

V_name          : Verlag → String;
V_ort           : Verlag → String;
Titel           : Publikation → String;
Jahr            : Publikation → Nat;
ISBN            : Publikation → Opt Nat;
Name            : Autor → String;
Geb_Datum       : Autor → Datum;
Adresse         : Autor → Adr;
Begriff         : Schlagwort → String;
Definition      : Schlagwort → String;

gibt_heraus     : Publikation × Verlag → Bool;
zitiert         : Publikation × Publikation → Bool;
verfaßt         : Autor × Publikation → Bool;
verwendet       : Publikation × Schlagwort → Bool;

axioms
-- Schlüsselbedingungen
∀v1,v2:Verlag. V_name v1 = V_name v2 ⇒ v1 = v2;
∀a1,a2:Autor.  Name a1 = Name a2 ∧ Geb_Datum a1 = Geb_Datum a2 ⇒ a1 = a2;
∀s1,s2:Schlagwort. Begriff s1 = Begriff s2 ⇒ s1 = s2;
∀p1,p2:Publikation. Titel p1 = Titel p2 ∧ Jahr p1 = Jahr p2 ∧
              (∀v:Verlag. gibt_heraus (p1,v) = gibt_heraus (p2,v)) ⇒ p1 = p2;

-- Beteiligung an Relationshiptypen
∀v:Verlag. ∃s: Set (Publikation×Verlag). ∀p:Publikation.
  ((p,v) ∈ s ⇔ gibt_heraus(p,v)) ∧ 1 ≤ card(s);
∀p:Publikation. ∃s: Set (Publikation×Verlag). ∀v:Verlag.
  ((p,v) ∈ s ⇔ gibt_heraus(p,v)) ∧ 1 ≤ card(s) ∧ card(s) ≤ 1;
∀p:Publikation. ∃s: Set (Autor×Publikation). ∀a:Autor.
  ((a,p) ∈ s ⇔ verfaßt (a,p)) ∧ 1 ≤ card(s);
∀a:Autor. ∃s: Set (Autor×Publikation). ∀p:Publikation.
  ((a,p) ∈ s ⇔ verfaßt (a,p)) ∧ 1 ≤ card(s);
∀sw:Schlagwort. ∃s: Set (Publikation×Schlagwort). ∀p:Publikation.
  ((p,sw) ∈ s ⇔ verwendet (p,sw)) ∧ 1 ≤ card(s);
endaxioms;
}
```

□

3.4 Diskussion

Repräsentation der modellierten Daten

Durch die angegebene Übersetzung wird einem E/R-Schema eine lose SPECTRUM-Spezifikation so zugeordnet, daß zwischen der Menge der durch das Schema beschriebenen Datenbestände und der Menge der Modelle der Spezifikation eine Bijektion existiert. Jedem integren Datenbestand ist also genau ein Modell der Spezifikation zugeordnet und umgekehrt. Bei der Repräsentation der Datenzustände durch die Modellklasse ist eine technische Besonderheit zu beachten, die auf die Definition der Semantik der Sprache SPECTRUM zurückzuführen ist: Alle in Modellen von SPECTRUM-Spezifikationen enthaltenen Trägermengen enthalten das ausgezeichnete Element $\perp$[3]. Die den Entitysorten zugeordneten Trägermengen enthalten also neben den Elementen, die Entities repräsentieren, ein zusätzliches $\perp$-Element. Insbesondere werden leere Entitymengen in den Modellen durch einelementige Trägermengen repräsentiert, die $\perp$ als einziges Element enthalten. Auf diese Besonderheit wird im weiteren Verlauf der Arbeit zu achten sein, um alle Konzepte technisch sauber zu präsentieren.

Durch die Angabe der Semantik eines E/R-Schemas als axiomatische Spezifikation ist es möglich, die auf diesem Gebiet vorhandenen Forschungsergebnisse in die Welt der Datenmodellierung zu übertragen. Das betrifft insbesondere die Erkenntnisse über Modularisierung und Verfeinerung von Spezifikationen (spezifizieren im Großen). Kapitel 5 wird sich ausführlich mit dieser Thematik beschäftigen.

Konsistenz der statischen Semantik

Wird wie in diesem Kapitel eine abstrakte Spezifikation zur Modellierung bestimmter Konzepte angegeben, so stellt sich automatisch die Frage nach der Konsistenz dieser Spezifikation. Konsistenz einer Beschreibung beziehungsweise Spezifikation läßt sich unter anderem durch Angabe eines Modells zeigen, das heißt durch Angabe einer Struktur, die alle in der Beschreibung gegebenen Anforderungen erfüllt. Um die Konsistenz eines E/R-Schemas zu zeigen, ist also ein Daten(bank)zustand anzugeben, der den statischen Integritätsbedingungen genügt. Zum Nachweis der Konsistenz der statischen Semantik eines E/R-Schemas ist eine Algebra anzugeben, die Modell dieser Spezifikation ist.

Im vorliegenden Fall sind zwei Fragen zu beantworten:

1. Stellen E/R-Schemata immer konsistente Beschreibungen von Daten dar?
2. Ist die einem konsistenten E/R-Schema zugeordnete statische Semantik konsistent? Dies ist natürlich eine Grundvoraussetzung, um die Definition der statischen Semantik adäquat nennen zu können.

[3] Dieses Element repräsentiert undefinierte Zustände und wird zum Beispiel dazu verwendet, um nichtterminierende Berechnungen zu modellieren.

Es ist offensichtlich, daß eine leere Datenbank, also ein Datenzustand, in dem alle durch das E/R-Schema beschriebenen Entitymengen leer sind, die im Schema gegebenen statischen Integritätsbedingungen immer erfüllt. Damit ist jedes beliebige E/R-Schema erfüllbar. Das E/R-Modell ist also eine Beschreibungstechnik, die nur konsistente Beschreibungen von Daten erlaubt.

Im folgenden wird gezeigt, wie sich neben dem trivialen Modell der leeren Datenbank auch ein nichtleeres Modell konstruieren läßt: Man startet mit einem Zustand, in dem jede Entitysorte genau ein Element enthält und in dem keine Beziehungen zwischen Entities existieren. In diesen Zustand fügt man solange Beziehungen und bei Bedarf auch neue Entities ein, bis alle durch die statischen Integritätsbedingungen geforderten minimalen Beteiligungen an Relationshiptypen erfüllt sind. Diese Konstruktion liefert immer ein nichtleeres Modell des E/R-Schemas und durch die zugehörige Algebra auch ein Modell der statischen Semantik, wenn folgende Bedingungen erfüllt sind:

(i) Die Spezifikationen der Attributsorten sind konsistent.

(ii) Die als Schlüsselsorten verwendeten Attributsorten enthalten hinreichend viele Elemente, so daß bei jeder Hinzunahme einer neuen Entity in eine Entitymenge ein noch nicht verwendeter Schlüssel zur Verfügung steht. Da die in der Praxis für Schlüsselattribute verwendeten Sorten (wie die natürlichen Zahlen oder Zeichenreihen) in der Regel unendlich viele verschiedene Elemente enthalten, ist diese Einschränkung eher theoretischer Natur.

Zum Nachweis der Konsistenz der statischen Semantik eines E/R-Schemas reicht es somit aus, die Bedingungen *(i)* und *(ii)* zu zeigen.

Verändernder Zugriff auf die Daten

Aus Sicht der formalen Spezifikationstechnik stellt sich die Frage, welche Vorteile die Integration mit dem E/R-Modell bringt. Das dabei angestrebte Ziel ist, die Daten eines Systems (z.B. eines betrieblichen Informationssystems) mit Hilfe des E/R-Modells zu beschreiben und die Systemfunktionen mit Hilfe der Spezifikationssprache (hier SPECTRUM) zu entwickeln. Es ist also zu untersuchen, ob sich die aus einem E/R-Schema generierte Spezifikation zu diesem Zweck eignet.

Zunächst ist festzustellen, daß die gegebene Spezifikation rein beobachtungsorientiert ist. Sie definiert also nur Beobachtungsfunktionen, aber keine Konstruktorfunktionen für die definierten Entitysorten. Damit können auch die Funktionen des Systems, die auf den modellierten Datenbestand zugreifen, nur abstrakt beschrieben werden. In späteren Phasen des Software-Entwicklungsprozesses ist es jedoch nötig, diese Funktionen konkreter, das heißt implementierungsnäher zu beschreiben. Dazu müssen auf dem Datenschema Konstruktorfunktionen vorhanden sein, um etwa Entities explizit angeben zu können oder Relationships zwischen Entities herzustellen. Soll das Entity/Relationship-Modell also als

"Front-End" zur Beschreibung statischer Anteile von Systemen dienen, die mittels axiomatischer Spezifikationstechniken entwickelt werden, so sind aus pragmatischen Gründen Konstruktorfunktionen für Entitytypen nötig. Das folgende Beispiel zeigt, daß die Erweiterung der statischen Semantik um solche Konstruktorfunktionen problematisch ist und ohne spezielle Vorkehrungen zu Inkonsistenzen führen kann.

Beispiel 7 *Reichert man die Spezifikation aus Beispiel 6 um Konstruktoren für die Entitysorten an, so erhält man etwa für den Entitytyp* Verlag *die Konstruktorfunktion*

`mkVerlag: String` $\times$ `String` $\rightarrow$ `Verlag`

Für die Selektoren ergeben sich die Gesetze

```
V_name(mkVerlag(n,o)) = n;
V_ort(mkVerlag(n,o)) = o;
```

Mit Hilfe der Konstruktorfunktionen können nun konkrete Entities aufgebaut werden, etwa `mkVerlag(¨Springer¨,¨Berlin¨)` *und* `mkVerlag(¨Springer¨,¨Heidelberg¨)`. *Da das Attribut* V_name *Schlüssel des Entitytyps* Verlag *ist, gilt*

`mkVerlag(¨Springer¨,¨Berlin¨) = mkVerlag(¨Springer¨,¨Heidelberg¨)`

Damit gilt aber auch

`V_ort(mkVerlag(¨Springer¨,¨Berlin¨)) = V_ort(mkVerlag(¨Springer¨,¨Heidelberg¨))`

und somit

`¨Berlin¨ = ¨Heidelberg¨`

Durch Hinzufügen des Konstruktors `mkVerlag` *ist also eine inkonsistente Spezifikation entstanden.* □

Das im obigen Beispiel geschilderte Problem ist rein technischer Natur. Es hat seine Ursache in bestimmten Eigenschaften der verwendeten Spezifikationssprache. Kapitel 4 wird sich mit der Lösung dieses Problems beschäftigen.

Bei der Entwicklung von Informationssystemen wird oft unterschieden zwischen Funktionen, die auf den verwalteten Datenbestand rein lesend zugreifen (die Software Engineering Methode SSADM nennt diese Funktionen *report-Funktionen*), und solchen, die den Datenbestand verändern (*update-Funktionen*). Da der modellierte Datenbestand durch die Modellklasse der in Abschnitt 3.3 gegebenen Spezifikation repräsentiert wird, sind Funktionen, die die Daten verändern können, Abbildungen zwischen Modellen der statischen

Semantik, also zwischen Algebren. Derartige Funktionen zwischen Modellen einer Spezifikation können in SPECTRUM nicht innerhalb der Sprache spezifiziert werden. Es ist also im gegebenen Ansatz nicht möglich, die Funktionen eines Informationssystems basierend auf der statischen Semantik in SPECTRUM zu spezifizieren und zu entwickeln.

Der vorliegende Abschnitt hat gezeigt, daß die statische Semantik gut geeignet ist, eine formale Semantik für das E/R-Modell zu geben, da sie nicht nur leicht verständlich ist, sondern es auch erlaubt, alle aus der Welt der algebraischen Spezifikation bekannten Forschungsergebnisse in die Welt der Datenmodellierung zu übertragen. Ein weiteres Ziel der Integration dieser beiden Techniken war es jedoch, die E/R-Modellierung als Technik zur Beschreibung von Daten in SPECTRUM einzusetzen. Zu diesem Zweck ist die statische Semantik nicht geeignet, da es, wie oben festgestellt, nicht möglich ist, in SPECTRUM Funktionen zu spezifizieren, die die in der statischen Semantik beschriebenen Daten verändern. Kapitel 4 wird sich damit beschäftigen, wie sich aus der statischen Semantik eine Beschreibung der Daten gewinnen läßt, die zu diesem Zweck geeignet ist.

Kapitel 4

Das E/R-Modell als axiomatische Spezifikationstechnik

Algebraische beziehungsweise axiomatische Spezifikationstechniken beschreiben ein Softwaresystem als Menge von Funktionen, die auf im allgemeinen nicht näher charakterisierten Daten operieren. Die Eigenschaften des Systems werden also durch die Eigenschaften seiner Funktionen ausgedrückt. Auch komplex strukturierte Daten können nur durch die Eigenschaften der auf ihnen operierenden Funktionen beschrieben werden. Die Spezifikation von stark datenorientierten Systemen wie zum Beispiel betrieblichen Informationssystemen wird also dadurch erschwert, daß die Struktur der Daten nur indirekt beschrieben werden kann. Dadurch werden Spezifikationen von Systemen, die auf komplex strukturierten Daten arbeiten, häufig sehr komplex und schwer verständlich.

Neuere Spezifikationstechniken tragen diesem Umstand durch Bereitstellung mächtigerer Sortensysteme Rechnung. So bietet die in dieser Arbeit verwendete Sprache SPECTRUM ein polymorphes Sortensystem mit Sortenklassen, wie es zum Beispiel von der funktionalen Programmiersprache HASKELL [HPW92] her bekannt ist. In diesem Sortensystem lassen sich beispielsweise recordartige Strukturen mit Hilfe des `data`-Konstrukts als sogenannte freie Datentypen spezifizieren. Dieses `data`-Konstrukt stellt dabei lediglich eine Schreibabkürzung dar, da es in eine (`data`-freie) SPECTRUM-Spezifikation übersetzt werden kann (in [BFG+93a] wird die Semantik dieses Konstrukts gerade durch eine solche Übersetzung erklärt).

Zur Beschreibung der relevanten Daten eines betrieblichen Informationssystems eignet sich jedoch auch ein Sortensystem wie das von SPECTRUM nur sehr eingeschränkt. Als dafür sehr geeignet hat sich in der Praxis das Entity/Relationship-Modell herausgestellt [BHS92]. Die in Abschnitt 3.3 gegebene Übersetzung nach SPECTRUM zeigt, daß das E/R-Modell ähnlich wie das `data`-Konstrukt als abkürzende Schreibweise zur Spezifikation komplexer Daten innerhalb der Spezifikationssprache verstanden werden kann. Ziel dieses Kapitels ist es, die in Abschnitt 3.4 identifizierten Probleme zu lösen, die diese Übersetzung im praktischen Einsatz mit sich bringt, und so das Entity/Relationship-Modell als pragmatisch

nutzbares Front-End zur Beschreibung von Daten in die Spezifikationssprache SPECTRUM zu integrieren.

4.1 Internalisierung der Semantik

In Abschnitt 3.4 wurde festgestellt, daß es in SPECTRUM nicht möglich ist, update-Funktionen basierend auf der in der statischen Semantik gegebenen Beschreibung der Daten eines Informationssystems zu spezifizieren. Sollen Funktionen, die den Datenbestand verändern, innerhalb der Spezifikationssprache SPECTRUM spezifiziert werden können, ist es deshalb nötig, die statische Semantik so weiterzuentwickeln, daß der Datenbestand innerhalb der Spezifikation in Form einer *Datenbanksorte* zugreifbar ist. Eine solche Weiterentwicklung der statischen Semantik wird im weiteren *internalisierte Semantik* genannt. Eine update-Funktion ist dann eine Funktion, die auf dieser Datenbanksorte operiert. Dabei ist wichtig, daß durch dieses Vorgehen nicht einfach eine weitere axiomatische Semantik für das E/R-Modell gegeben wird, die ohne Zusammenhang neben der statischen Semantik steht. Vielmehr wird die internalisierte Semantik aus der statischen Semantik durch klar definierte Schritte gewonnen. Damit kann das Verhältnis beider Semantiken zueinander formal und eindeutig beschrieben werden.

Informell kann die Datenbanksorte als Struktur verstanden werden, die es erlaubt, die Zustände des modellierten Datenbestandes, also Entities der gegebenen Entitytypen sowie Beziehungen zwischen diesen Entities gemäß den gegebenen Relationshiptypen, zu speichern.

Formal ist die Datenbanksorte eine Sorte, die die Modellklasse der statischen Semantik repräsentiert, da die Modelle der statischen Semantik genau den Datenzuständen entsprechen. Die Konstruktion der internalisierten Semantik aus der statischen Semantik ist also ein Spezialfall einer Konstruktion, die zu einer gegebenen Spezifikation SP die Spezifikation ihrer Modellklasse in Form einer Sorte angibt. Eine allgemeine Lösung dieses Problems könnte aus folgenden zwei Schritten bestehen:

1. Benütze die Spezifikationssprache, um den Algebrenbegriff zu spezifizieren, der in der Semantikdefinition der Sprache verwendet wird. Das Ergebnis dieses Schrittes ist die Spezifikation SP^{mod} einer Sorte Mod zusammen mit ihren Konstruktor- und Selektorfunktionen, deren Elemente genau den zu SP passenden Σ-Algebren entsprechen.

2. Generiere aus den Axiomen von SP Axiome für SP^{mod}, die sicherstellen, daß Mod nur die Σ-Algebren enthält, die Modelle von SP sind.

Die so spezifizierte Sorte Mod entspricht im vorliegenden Fall der gesuchten Datenbanksorte, die Spezifikation SP^{mod} der internalisierten Semantik. Führt man jedoch diese formale Konstruktion für die Sprache SPECTRUM durch, so genügt das erreichte Ergebnis den Anforderungen, die aus pragmatischen Gründen an die internalisierte Semantik gestellt werden müssen, aus mehreren Gründen nicht:

- SPECTRUM ist eine sehr mächtige Sprache, die dem Anwender viele Möglichkeiten wie ein komplexes polymorphes Sortensystem, Funktionen höherer Ordnung und vieles mehr bietet. Aus diesem Grund ist zur Definition der Semantik von SPECTRUM ein sehr komplexer Algebrenbegriff nötig. Es entsteht bei Durchführung obiger Konstruktion folglich eine sehr komplexe Datenbanksorte, die mit der informellen Vorstellung, die mit dieser Sorte verbunden wird, nicht übereinstimmt.

- Die Modelle der statischen Semantik enthalten nicht nur Trägermengen für die Entitysorten und Prädikate, die den Relationshiptypen entsprechen. Sie beinhalten ebenso die Interpretationen der primitiven Konzepte, auf die sich die Spezifikation stützt, wie zum Beispiel die Wahrheitswerte, die natürlichen Zahlen oder die Attributsorten. Alle diese Konzepte sind gemäß obiger Konstruktion in den Elementen der Datenbanksorte enthalten, was nicht mit der informellen Vorstellung übereinstimmt, daß Datenbankzustände aus Entities und Beziehungen zwischen ihnen bestehen.

Um zu einer leicht verständlichen internalisierten Semantik zu kommen, die mit der oben gegebenen informellen Charakterisierung übereinstimmt, werden einige Vereinfachungen vorgenommen.

Die Datenbanksorte `Db` wird bei diesem Ansatz so spezifiziert, daß ihre Elemente Teile von Algebren widerspiegeln, die in der Semantik von SPECTRUM Verwendung finden. Als Vorbild für die Definition von `Db` wird in dieser Arbeit jedoch nicht der komplizierte Algebrenbegriff verwendet, der der SPECTRUM-Semantik zugrundeliegt. Stattdessen wird von einem wesentlich einfacheren Algebrenbegriff ausgegangen, wie er zum Beispiel in [BG84] definiert wird. Eine Signatur $\Sigma = (S, F)$ besteht dabei aus einer Menge S von Sortenbezeichnern und einer (mit Sorten indizierten) Menge F von Funktionssymbolen. Eine Σ-Algebra $\mathcal{A}$ enthält für jeden Sortenbezeichner $s \in S$ eine Trägermenge $s^{\mathcal{A}}$ und und für jeden Funktionsbezeichner $f : s_1 \times \ldots \times s_n \rightarrow s$ eine Funktion $f^{\mathcal{A}} : s_1^{\mathcal{A}} \times \ldots \times s_n^{\mathcal{A}} \rightarrow s^{\mathcal{A}}$. Dieser im Vergleich zur SPECTRUM-Semantikdefinition vereinfachte Algebrenbegriff darf angenommen werden, weil die Sprachkonstrukte wie Polymorphie etc., die für SPECTRUM einen komplizierteren Algebrenbegriff notwendig machen, zur Definition der Entity- und Relationshiptypen in der statischen Semantik nicht verwendet werden.

Des weiteren enthält die Datenbanksorte nicht die vollständigen Algebren aus der Modellklasse der statischen Semantik, sondern lediglich den Anteil, der den Entitytypen und Relationshiptypen entspricht. Dieser Anteil ist selbst keine Algebra.

Unter Berücksichtigung dieser Entscheidungen läßt sich die internalisierte Semantik durch folgende zwei Schritte konstruieren:

1. Führe eine Datenbanksorte `Db` ein, deren Elemente die möglichen Zustände der modellierten Datenbank repräsentieren. `Db` ist eine Sorte, die als Behälter für folgende Komponenten dient:

   - Für jede Entitysorte E_i eine Menge von Elementen dieser Sorte

- Für jeden Relationshiptyp R_j ein ihn repräsentierendes Prädikat

`Db` wird zusammen mit seinen Konstruktor- und Selektorfunktionen spezifiziert und geeignet axiomatisiert.

Die spezifizierte Datenbank enthält also eine Reihe von Entitymengen und Prädikate, die die Beziehungen zwischen den in diesen Mengen enthaltenen Entities repräsentieren. Als Mengenbegriff legt die vorliegende Arbeit den klassischen, in Abschnitt 2.2.1 (Abbildung 2.3) spezifizierten Begriff der endlichen Mengen zugrunde. Die verwendeten Mengen können so zwar beliebig groß sein, sie sind aber immer endlich. Deshalb gehen beim Schritt von der statischen zur internalisierten Semantik alle echt unendlichen Modelle der statischen Semantik verloren. Dies ist hinnehmbar, weil diese Modelle echt unendlich große Datenbanken repräsentieren, die in der Praxis natürlich ohne Bedeutung sind. Die Alternative zu diesem Vorgehen wäre gewesen, statt des Konzepts der endlichen Mengen eine andere Datenstruktur zu wählen, die auch echt unendliche Zustände repräsentieren kann. Eine solche Struktur wäre jedoch deutlich weniger verständlich gewesen als endliche Mengen. Um die Darstellung des Ansatzes lesbar und verständlich zu halten, wurde auf diese Alternative verzichtet.

2. Anstatt die Sorte `Db` durch Axiome so zu beschränken, daß sie nur integre Datenbankzustände als Elemente enthält, wird ein Prädikat `OK` spezifiziert, das integre von nicht integren Datenbankzuständen unterscheiden kann. Die Axiome für `OK` werden aus der Axiomatisierung der statischen Semantik gewonnen.

   Bei dieser Vorgehensweise enthält die Sorte `Db` auch Elemente, die nicht integre Datenbankzustände repräsentieren. Das `OK`-Prädikat dient zum Erkennen der eigentlich gewünschten integren Zustände. Die abschließende Diskussion dieses Kapitels (Abschnitt 4.4) wird auf die Vorteile dieses Ansatzes und mögliche Alternativen eingehen.

Konstruktion der internalisierten Semantik

Die statische Semantik eines E/R-Schemas stellt den Ausgangspunkt dieser Konstruktion dar. Sie ist eine SPECTRUM-Spezifikation $SP^s = (\Sigma^s, Ax^s)$[1].

Die polymorphe Signatur $\Sigma^s = (\Omega^s, F^s, O^s)$ besteht aus:

- Einer Sortensignatur $\Omega^s = (K^s, \leq^s, SC^s)$, die sich zusammensetzt aus einer partiellen Ordnung $(K^s, \leq^s)$ von *Kinds* (Typklassen) sowie einer Menge SC^s von Sortenkonstruktoren. SC^s enthält insbesondere die Teilmenge $SC^s_E = \{E_1, \ldots, E_n\}$ der in der statischen Semantik definierten Entitysorten.

[1] Die Definition der Konstruktion orientiert sich an der abstrakten Struktur von SPECTRUM-Spezifikationen, wie sie in [BFG+93a, BFG+93b, GR94] angegeben ist. Für Details wird auf die angegebene Literatur verwiesen.

- Einer Menge F^s von Funktionsbezeichnern, die insbesondere die Teilmenge $F^s_R = \{R_1 : \tau_1, \ldots, R_m : \tau_m\}$ der Prädikate beinhaltet, die die Relationshiptypen repräsentieren.

- Einer Menge O^s von (Bezeichnern für) nicht-stetige Funktionen (Mappings).

Die Menge Ax^s enthält insbesondere die Menge $Ax^s_{Int} = \{ax_1, \ldots, ax_k\}$ der Axiome, die die statischen Integritätsbedingungen des E/R-Schemas repräsentieren.

Aufbauend auf dieser Definition ergibt sich für die internalisierte Semantik eine Spezifikation $SP^i = (\Sigma^i, Ax^i)$ mit den im folgenden dargestellten Eigenschaften.

Signatur der internalisierten Semantik Die internalisierte Semantik hat die Signatur $\Sigma^i = (\Omega^i, F^i, O^i)$, wobei die Sortensignatur $\Omega^i = (K^s, \leq^s, SC^s \cup \{\texttt{Db}\})$ im Vergleich zu der der statischen Semantik um einen Sortenbezeichner `Db` für die Datenbanksorte erweitert wird.

Die Menge der Funktionsbezeichner F^i enthält Konstruktor- und Selektorfunktionen für die neue Sorte `Db`:

$$
\begin{aligned}
F^i &= (F^s \setminus F^s_R) \cup F_{Db} \\
F_{Db} &= \{\texttt{mkdb:Set } E_1 \times \ldots \times \texttt{Set } E_n \times \tau_1 \times \ldots \times \tau_m \rightarrow \texttt{Db}\} \\
&\quad \cup \{\texttt{ent'}E_i\texttt{:Db} \rightarrow \texttt{Set } E_i | 1 \leq i \leq n\}^2 \\
&\quad \cup \{R_i\texttt{:Db} \rightarrow \tau_i | R_i : \tau_i \in F^s_R\}
\end{aligned}
$$

Die Signatur des `OK`-Prädikats wird zur Menge der Signaturen allgemeiner mathematischer Abbildungen (Mappings) hinzugenommen[3]:

$$O^i = O^s \cup \{\texttt{OK: Db to Bool}\}$$

Axiome der internalisierten Semantik In der internalisierten Semantik werden die statischen Integritätsbedingungen in das `OK`-Prädikat "verpackt". Darüberhinaus müssen Axiome angegeben werden, die den Zusammenhang zwischen Konstruktor- und Selektorfunktionen der Sorte `Db` beschreiben.

$$Ax^i = (Ax^s \setminus Ax^s_{Int}) \cup Ax_{Db} \cup Ax_{OK}$$

[2]Mengen von Signaturelementen beziehungsweise Axiomen werden im folgenden häufig als *Mengenkomprehension* aufgeschrieben. Eine Mengenkomprehension $\{T[x]|P[x]\}$ bezeichnet die Menge aller $T[x]$, für die x das Prädikat P erfüllt. $\{\texttt{ent'}E_i\texttt{:Db} \rightarrow \texttt{Set } E_i | 1 \leq i \leq n\}$ steht also zum Beispiel für die Menge $\{\texttt{ent'}E_1\texttt{:Db} \rightarrow \texttt{Set } E_1, \ldots, \texttt{ent'}E_n\texttt{:Db} \rightarrow \texttt{Set } E_n\}$.

[3]Das `OK`-Prädikat ist keine Funktion, die implementiert werden soll. Es stellt vielmehr ein reines Spezifikations-Hilfskonstrukt dar, das im wesentlichen zur Generierung von Beweisverpflichtungen verwendet werden wird. Es wird deshalb in SPECTRUM als *Mapping* spezifiziert (siehe auch [BFG+93a]).

Die Menge Ax_{Db} enthält die Axiome, die die Eigenschaften der Datenbanksorte festlegen. Dazu gehört sowohl der direkte Zusammenhang zwischen Konstruktoren und Selektoren als auch eine Abgeschlossenheitseigenschaft, die durch die Tatsache bedingt ist, daß die Datenbanksorte die (teilweise) Spezifikation einer Klasse von Algebren ist. Die Anwendung von Funktionen einer Algebra ist nur definiert auf Elementen, die in Trägermengen der Algebra enthalten sind. Diese Eigenschaft ist auch für die in der Sorte `Db` enthaltenen Relationship-Prädikate R_j zu fordern.

$$(1)\quad \forall \mathtt{db{:}Db}.\ \forall \mathtt{e}_1^{R_j}{:}E_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j}{:}E_{n_j}^{R_j}.\ \delta(R_j\ \mathtt{db}\ (\mathtt{e}_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j})) \Rightarrow \mathtt{e}_1^{R_j} \in \mathtt{ent'}E_1^{R_j}\ \mathtt{db} \wedge \ldots \wedge \mathtt{e}_{n_j}^{R_j} \in \mathtt{ent'}E_{n_j}^{R_j}\ \mathtt{db}$$

Diese Abgeschlossenheitsforderung gilt bei Algebren natürlich auch für die Ergebnisse der Funktionen, das heißt die Funktionen dürfen nur Resultate liefern, die selbst wieder in der Algebra enthalten sind. Im vorliegenden Fall liefern die Relationship-Prädikate Werte der Sorte `Bool`. Diese Sorte ist keine Entitysorte und somit nicht in `Db` enthalten. Deshalb ist für diesen Teil der Abgeschlossenheitseigenschaft nichts zu fordern.

Die Selektorfunktionen R_j der Sorte `Db` repräsentieren die genauso bezeichneten Relationship-Prädikate aus der statischen Semantik. Deshalb müssen auch die in der statischen Semantik geforderten Eigenschaften der Striktheit und Totalität auf diese Selektoren übertragen werden. Hier ist es wiederum ausreichend, die Striktheit und Totalität auf die Elemente der in `Db` enthaltenen Trägermengen zu beschränken.

$$(2)\quad \forall \mathtt{db{:}Db}.\ \forall^{\perp} \mathtt{e}_1^{R_j}{:}E_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j}{:}E_{n_j}^{R_j}.\ \mathtt{e}_1^{R_j} \in \mathtt{ent'}E_1^{R_j}\ \mathtt{db} \wedge \ldots \wedge \mathtt{e}_{n_j}^{R_j} \in \mathtt{ent'}E_{n_j}^{R_j}\ \mathtt{db} \Rightarrow \delta(R_j\ \mathtt{db}\ (\mathtt{e}_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j})) \Rightarrow \delta(\mathtt{e}_1^{R_j}) \wedge \ldots \wedge \delta(\mathtt{e}_{n_j}^{R_j})$$

$$(3)\quad \forall \mathtt{db{:}Db}.\ \forall^{\perp} \mathtt{e}_1^{R_j}{:}E_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j}{:}E_{n_j}^{R_j}.\ \mathtt{e}_1^{R_j} \in \mathtt{ent'}E_1^{R_j}\ \mathtt{db} \wedge \ldots \wedge \mathtt{e}_{n_j}^{R_j} \in \mathtt{ent'}E_{n_j}^{R_j}\ \mathtt{db} \Rightarrow \delta(\mathtt{e}_1^{R_j}) \wedge \ldots \wedge \delta(\mathtt{e}_{n_j}^{R_j}) \Rightarrow \delta(R_j\ \mathtt{db}\ (\mathtt{e}_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j}))$$

Der in dieser Arbeit vorausgesetzte Mengenbegriff ist der Standardbegriff, der endliche, strikte Mengen beschreibt. Deshalb gilt für die in den Trägermengen der Sorte `Db` enthaltenen Elemente

$$(4)\quad \forall \mathtt{db{:}Db}.\ \forall \mathtt{e}{:}E_i.\ \mathtt{e} \in \mathtt{ent'}E_i\ \mathtt{db} \Rightarrow \delta(\mathtt{e})$$

Mit der Eigenschaft (4) lassen sich (1)–(3) offensichtlich zusammenfassen zu

$$(5)\quad \forall \mathtt{db{:}Db}.\ \forall \mathtt{e}_1^{R_j}{:}E_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j}{:}E_{n_j}^{R_j}.\ \delta(R_j\ \mathtt{db}\ (\mathtt{e}_1^{R_j},\ldots,\mathtt{e}_{n_j}^{R_j})) \Leftrightarrow \mathtt{e}_1^{R_j} \in \mathtt{ent'}E_1^{R_j}\ \mathtt{db} \wedge \ldots \wedge \mathtt{e}_{n_j}^{R_j} \in \mathtt{ent'}E_{n_j}^{R_j}\ \mathtt{db}$$

Zusätzliche Eigenschaften sind die Striktheit und Totalität der Selektorfunktionen der Datenbanksorte `Db` sowie die Tatsache, daß `mkdb` der einzige Konstruktor für `Db` ist:

$\{\texttt{ent'}s\ \texttt{strict total}\ |s{\in}SC_E^S\}$
$\{R_j\ \texttt{strict total}\ |R_j{:}\texttt{Db}{\rightarrow}(E_1^{R_j}{\times}\ldots{\times}E_{n_j}^{R_j})\ \in\ F_{Db}\}$
$\{\texttt{Db freely generated by mkdb}\}$

Damit ergibt sich die Axiomenmenge Ax_{Db} zu:

$$
\begin{array}{l}
Ax_{Db} = \\
\quad \{\forall \texttt{db:Db}.\forall \texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{r}_m\,. \\
\qquad \texttt{db} = \texttt{mkdb}(\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{r}_m) \Rightarrow \\
\qquad\quad \texttt{ent'}E_1\ \texttt{db} = \texttt{e}_1 \wedge \ldots \wedge \texttt{ent'}E_n\ \texttt{db} = \texttt{e}_n \wedge \\
\qquad\quad R_1\ \texttt{db} = \texttt{r}_1 \wedge \ldots \wedge R_1\ \texttt{db} = \texttt{r}_1\} \\
\quad \cup \\
\quad \{\forall \texttt{db:Db}.\ \forall \texttt{e}_1^{R_j}{:}E_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j}{:}E_{n_j}^{R_j}\,. \\
\qquad \delta(R_j\ \texttt{db}\ (\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j})) \Leftrightarrow \texttt{e}_1^{R_j} \in \texttt{ent'}E_1^{R_j}\ \texttt{db} \wedge \ldots \wedge\ \texttt{e}_{n_j}^{R_j} \in \texttt{ent'}E_{n_j}^{R_j}\ \texttt{db} \\
\quad\ |\ R_j{:}\texttt{Db}{\rightarrow}(E_1^{R_j}{\times}\ldots{\times}\texttt{e}_{n_j}^{R_j}{:}E_{n_j}^{R_j})\ \in\ F_{Db}\} \\
\quad \cup \\
\quad \{\texttt{ent'}s\ \texttt{strict total}\ |s{\in}SC_E^S\} \\
\quad \cup \\
\quad \{R_j\ \texttt{strict total}\ |R_j{:}\texttt{Db}{\rightarrow}(E_1^{R_j}{\times}\ldots{\times}E_{n_j}^{R_j})\ \in\ F_{Db}\} \\
\quad \cup \\
\quad \{\texttt{Db freely generated by mkdb}\}
\end{array}
$$

Man beachte, daß die aus der Welt der algebraischen Spezifikation herrührende Abgeschlossenheitseigenschaft der Sorte `Db` (die oben angegebene Eigenschaft (1)) exakt das im Datenbankbereich bekannte Konzept der *referentiellen Integrität* beschreibt. Referentielle Integrität bedeutet, daß eine Datenbank nur Beziehungen zwischen Entitäten enthalten kann, die selbst in der Datenbank vorhanden sind. Im Falle von relationalen Datenbanken heißt dies konkret, daß Fremdschlüsselattribute immer auf vorhandene Tupel zeigen müssen. Eine Reihe von Implementierungen relationaler Datenbanksysteme ist bereits in der Lage, diese Eigenschaft selbständig zu überwachen.

Das `OK`-Prädikat testet einen Datenbankzustand auf statische Integrität. Die Axiomatisierung dieses Prädikats kann aus der Menge Ax_{Int}^s der Axiome konstruiert werden, die in der statischen Semantik die statischen Integritätsbedingungen beschreiben. Die aus den statischen Integritätsbedingungen gewonnenen Teilprädikate sind nur sinnvoll, wenn sie für jeden Datenbankzustand einen definierten Wert (`true` oder `false`) liefern. Aus diesem Grund ist Totalität für das `OK`-Prädikat zu fordern.

$$
\begin{array}{l}
Ax_{OK} = \{\forall \texttt{db:Db. OK db} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k])\ |\{ax_1,\ldots,ax_k\} = Ax_{Int}^s\} \\
\qquad \cup\ \{\texttt{OK total}\}
\end{array}
$$

In dieser Definition wird ein syntaktischer Umbauoperator φ für SPECTRUM-Axiome eingesetzt, der induktiv über den Aufbau von SPECTRUM-Termen definiert ist[4]. Die Termsyntax

[4] Axiome sind in SPECTRUM nichts anderes als Terme der Sorte `Bool`. Die Definition von φ muß sich also an der Termsprache von SPECTRUM orientieren.

⟨exp1⟩	::=	⟨exp2⟩	
	|	$\forall^{\perp}$⟨id⟩:⟨sort⟩.⟨exp1⟩	*($\forall^{\perp}$-Quantification)*
	|	$\exists^{\perp}$⟨id⟩:⟨sort⟩.⟨exp1⟩	*($\exists^{\perp}$-Quantification)*
	|	**λ**⟨pat⟩.⟨exp1⟩	*(λ-Abstraction)*
⟨exp2⟩	::=	⟨aexp⟩	
	|	⟨exp2⟩⟨aexp⟩	*(Application)*
⟨aexp⟩	::=	⟨id⟩	*(Identifier)*
	|	(⟨exp1⟩{,⟨exp1⟩}$^{+}$)	*(Tuples)*
	|	(⟨exp1⟩)	*(Grouping)*
	|	⟨aexp⟩:⟨asort⟩	*(Sorted Expression)*

Abbildung 4.1: Die Termstruktur der SPECTRUM-Kernsprache

der Kernsprache[5] von SPECTRUM zeigt Abbildung 4.1[6].

Die Definition des Operators φ ist in Abbildung 4.2 gegeben. Er erfüllt im wesentlichen die beiden im folgenden dargestellten Aufgaben.

- Die in der statischen Semantik gegebenen Axiome beschreiben Einschränkungen, denen die modellierten Daten zu gehorchen haben. In der internalisierten Semantik werden die Daten durch die Datenbanksorte `Db` modelliert. Die Einschränkungen beziehen sich hier also nur auf Elemente, die in der Datenbanksorte enthalten sind. Dies betrifft insbesondere die in der Datenbank enthaltenen Entities. Es müssen also alle Quantoren, die über Elemente von Entitysorten laufen, eingeschränkt werden auf Entities, die in der Datenbank enthalten sind. Dies wird oft als *Sortenrelativierung* bezeichnet.

  Das zur Relativierung verwendete Prädikat muß $x{\in}\texttt{ent'}\tau(\texttt{db})$ lauten, um die Elemente einer Entitysorte τ auf die in der Datenbank `db` enthaltenen Entities zu beschränken. Wie in Abschnitt 3.4 erläutert, enthalten die in der statischen Semantik den Entitysorten zugeordneten Trägermengen das zusätzliche Element $\perp$. Dieses Element muß im Sortenrelativierungsprädikat gesondert betrachtet werden. Deshalb wird in der Definition von φ in Abbildung 4.2 das Relativierungsprädikat $\delta(x){\Rightarrow}x{\in}\texttt{ent'}\tau(\texttt{db})$ verwendet. Im weiteren wird dieses Prädikat auch mit ρ[x] abgekürzt.

- Die zur Modellierung von Relationshiptypen verwendeten Prädikate sind in der internalisierten Semantik ebenfalls in der Datenbank gespeichert. Auf sie muß also nun

[5] SPECTRUM enthält einen schmalen Sprachkern, auf den die restlichen Sprachkonstrukte als definitorische Erweiterungen gestützt sind. Aus diesem Grund ist es ausreichend, für die Definition von φ diese Kernsprache zugrunde zu legen.

[6] Die Quantoren werden in SPECTRUM mit $\forall^{\perp}$ und $\exists^{\perp}$ bezeichnet. Die übliche Quantorschreibweise $\forall x.P$ (beziehungsweise $\exists x.P$) existiert ebenfalls und stellt eine Abkürzung für $\forall^{\perp}x.\delta(x){\Rightarrow}P$ ($\exists^{\perp}x.\delta(x){\wedge}P$) dar.

$$\varphi[\forall^{\perp} x:s.P] = \begin{cases} \forall^{\perp} x:s.\varphi[P] & falls \quad s \in (SC^s \setminus SC^s_E) \\ \forall^{\perp} x:s.(\delta(x) \Rightarrow x \in \texttt{ent's(db)}) \Rightarrow \varphi[P] & falls \quad s \in SC^s_E \end{cases}$$

$$\varphi[\exists^{\perp} x:s.P] = \begin{cases} \exists^{\perp} x:s.\varphi[P] & falls \quad s \in (SC^s \setminus SC^s_E) \\ \exists^{\perp} x:s.(\delta(x) \Rightarrow x \in \texttt{ent's(db)}) \wedge \varphi[P] & falls \quad s \in SC^s_E \end{cases}$$

$$\varphi[\lambda p.t] = \lambda p.\varphi[t]$$

$$\varphi[(t_1, \ldots, t_n)] = (\varphi[t_1], \ldots, \varphi[t_n])$$

$$\varphi[(t)] = (\varphi[t])$$

$$\varphi[t:s] = \varphi[t] : s$$

$$\varphi[t_1\ t_2] = \varphi[t_1]\ \varphi[t_2]$$

$$\varphi[x] = \begin{cases} x(db) & falls \quad x:\tau \in F^s_R \\ x & falls \quad x:\tau \in (F^s \setminus F^s_R) \vee x:\tau \in O^s \\ x & falls \quad x \in \chi\ Variable \end{cases}$$

Abbildung 4.2: Der Umbauoperator $\varphi : W_{\Sigma^s}(\chi) \rightarrow W_{\Sigma^i}(\chi)$

über die entsprechende Selektorfunktion der Sorte `Db` zugegriffen werden. Jede Anwendung eines solchen Prädikats auf ein Tupel von Entities $R(e_1,\ldots,e_i)$ ist deshalb zu ersetzen durch $R\ \texttt{db}\ (e_1,\ldots,e_i)$, wobei `db` die bei der Definition von Ax_{OK} eingeführte Datenbank-Variable ist.

Beispiel 8 *Die internalisierte Semantik des Beispiel-Datenschemas RECHERCHE hat gemäß obiger Konstruktion folgende Gestalt:*

```
RECHERCHE_i = { enriches Opt + Nat + Set + AttrSort;

sort Autor, Publikation, Schlagwort, Verlag, Db;
Autor, Publikation, Schlagwort, Verlag, Db :: EQ;

V_name          : Verlag → String;
V_ort           : Verlag → String;
Titel           : Publikation → String;
Jahr            : Publikation → Nat;
ISBN            : Publikation → Opt Nat;
Name            : Autor → String;
Geb_Datum       : Autor → Datum;
```

```
Adresse         : Autor → Adr;
Begriff         : Schlagwort → String;
Definition      : Schlagwort → String;
V_name,V_ort,Titel,Jahr,ISBN,Name,
Geb_Datum,Adresse,Begriff,Definition strict total;

mkdb : Set Autor × Set Publikation × Set Schlagwort × Set Verlag ×
       (Publikation × Verlag → Bool) ×
       (Autor × Publikation → Bool) ×
       (Publikation × Schlagwort → Bool) ×
       (Publikation × Publikation → Bool)
       → Db;
mkdb strict;

Db freely generated by mkdb;

ent'Autor       : Db → Set Autor;
ent'Publikation : Db → Set Publikation;
ent'Schlagwort  : Db → Set Schlagwort;
ent'Verlag      : Db → Set Verlag;
ent'Autor,ent'Publikation,ent'Schlagwort,ent'Verlag strict total;

gibt_heraus     : Db → (Publikation × Verlag → Bool);
verfaßt         : Db → (Autor × Publikation → Bool);
verwendet       : Db → (Publikation × Schlagwort → Bool);
zitiert         : Db → (Publikation × Publikation → Bool);
gibt_heraus,verfaßt,verwendet,zitiert strict total;

axioms
-- Beziehung zwischen Konstruktor und Selektoren
∀db:Db. ∀a:Set Autor, p:Set Publikation, s:Set Schlagwort, v:Set Verlag,
         g:Publikation × Verlag → Bool,
         vf:Autor × Publikation → Bool,
         vw:Publikation × Schlagwort → Bool,
         z:Publikation × Publikation → Bool.
   db = mkdb(a,p,s,v,g,vf,vw,z) ⇒ ent'Autor db = a ∧
                                   ent'Publikation db = p ∧
                                   ent'Schlagwort db = s ∧
                                   ent'Verlag db = v ∧
                                   gibt_heraus db = g ∧
                                   verfaßt db = vf ∧
                                   verwendet db = vw ∧
                                   zitiert db = z;
```

```
∀db:Db. ∀p:Publikation, v:Verlag. δ(gibt_heraus db (p,v)) ⇔
    p ∈ ent'Publikation db ∧ v ∈ ent'Verlag db;
∀db:Db. ∀a:Autor, p:Publikation. δ(verfaßt db (a,p)) ⇔
    a ∈ ent'Autor db ∧ p ∈ ent'Publikation db;
∀db:Db. ∀p:Publikation, s:Schlagwort. δ(verwendet db (p,s)) ⇔
    p ∈ ent'Publikation db ∧ s ∈ ent'Schlagwort db;
∀db:Db. ∀p1:Publikation, p2:Publikation. δ(zitiert db (p1,p2)) ⇔
    p1 ∈ ent'Publikation db ∧ p2 ∈ ent'Publikation db;
endaxioms;

OK : Db to Bool;
OK total;

axioms
∀db:Db. OK db ⇔
 (∀v1,v2: Verlag. v1 ∈ ent'Verlag db ∧ v2 ∈ ent'Verlag db ⇒
    V_name v1 = V_name v2 ⇒ v1 = v2) ∧
 (∀a1,a2:Autor. a1 ∈ ent'Autor db ∧ a2 ∈ ent'Autor db ⇒
    Name a1 = Name a2 ∧ Geb_Datum a1 = Geb_Datum a2 ⇒ a1 = a2) ∧
 (∀s1,s2:Schlagwort. s1 ∈ ent'Schlagwort db ∧ s2 ∈ ent'Schlagwort db ⇒
    Begriff s1 = Begriff s2 ⇒ s1 = s2) ∧
 (∀p1,p2:Publikation. p1 ∈ ent'Publikation db ∧ p2 ∈ ent'Publikation db ⇒
    Titel p1 = Titel p2 ∧ Jahr p1 = Jahr p2 ∧
    (∀v:Verlag. v ∈ ent'Verlag db ⇒
                gibt_heraus db (p1,v) = gibt_heraus db (p2,v)) ⇒ p1 = p2) ∧
 (∀v:Verlag. v ∈ ent'Verlag db ⇒
  (∃s: Set (Publikation×Verlag).  ∀p:Publikation. p ∈ ent'Publikation db ⇒
    ((p,v) ∈ s ⇔ gibt_heraus db (p,v)) ∧ 1 ≤ card(s))) ∧
 (∀p:Publikation. p ∈ ent'Publikation db ⇒
  (∃s: Set (Publikation×Verlag). ∀v:Verlag. v ∈ ent'Verlag db ⇒
    ((p,v) ∈ s ⇔ gibt_heraus db (p,v)) ∧ 1 ≤ card(s) ∧ card(s) ≤ 1)) ∧
 (∀p:Publikation. p ∈ ent'Publikation db ⇒
  (∃s: Set (Autor×Publikation). ∀a:Autor. a ∈ ent'Autor db ⇒
    ((a,p) ∈ s ⇔ verfaßt db (a,p)) ∧ 1 ≤ card(s))) ∧
 (∀a:Autor. a ∈ ent'Autor db ⇒
  (∃s: Set (Autor×Publikation). ∀p:Publikation. p ∈ ent'Publikation db ⇒
    ((a,p) ∈ s ⇔ verfaßt db (a,p)) ∧ 1 ≤ card(s))) ∧
 (∀sw:Schlagwort. sw ∈ ent'Schlagwort db ⇒
  (∃s: Set (Publikation×Schlagwort). ∀p:Publikation. p ∈ ent'Publikation db ⇒
    ((p,sw) ∈ s ⇔ verwendet (p,sw)) ∧ 1 ≤ card(s)));
endaxioms;
}
```

□

Anmerkung In Beispiel 8 werden Quantifizierungen über Entityvariablen $x{:}\tau$ nur mittels $x{\in}\texttt{ent'}\tau\texttt{(db)}$ eingeschränkt statt mit $\delta(x){\Rightarrow}x{\in}\texttt{ent'}\tau\texttt{(db)}$. Dies ist eine abkürzende Schreibweise, die logisch folgendermaßen begründet werden kann:

Eine Quantifizierung $\forall x{:}\tau.P$ ist in SPECTRUM eine Abkürzung für $\forall^{\perp} x{:}\tau.\delta(x){\Rightarrow}P$. Gemäß der Definition von φ (siehe Abbildung 4.2) wird diese Quantifizierung übersetzt in $\forall^{\perp} x{:}\tau.(\delta(x){\Rightarrow}\texttt{ent'}\tau\texttt{(db)}){\Rightarrow}\delta(x){\Rightarrow}P$. Durch einfache aussagenlogische Umformungen kann gezeigt werden, daß diese Formel äquivalent ist zu $\forall^{\perp} x{:}\tau.\delta(x){\Rightarrow}\texttt{ent'}\tau\texttt{(db)}{\Rightarrow}P$ und damit zu $\forall x{:}\tau.\texttt{ent'}\tau\texttt{(db)}{\Rightarrow}P$. Für den $\exists$-Quantor kann auf analoge Weise gezeigt werden, daß $\exists^{\perp} x{:}\tau.(\delta(x){\Rightarrow}\texttt{ent'}\tau\texttt{(db)}){\wedge}\delta(x){\wedge}P$ äquivalent ist zu $\exists x{:}\tau.\texttt{ent'}\tau\texttt{(db)}{\wedge}P$.

4.2 Formale Beziehung zur statischen Semantik

In Abschnitt 3.3 wurde dem E/R-Modell eine formale Semantik zugewiesen, indem eine Übersetzungsvorschrift angegeben wurde, die es erlaubt, ein E/R-Schema in eine SPECTRUM-Spezifikation zu übersetzen. Die dabei entstehende Spezifikation wurde als statische Semantik des Schemas bezeichnet. Durch die in Abschnitt 4.1 vorgenommene Internalisierung wurde aus der statischen Semantik eine Spezifikation gewonnen, die als Grundlage für einen praktischen Einsatz der E/R-Modellierung im Rahmen der formalen Software-Entwicklung mit SPECTRUM dienen kann. Auch diese Spezifikation kann nun als Definition der Semantik des E/R-Modells verstanden werden. Sie trägt deshalb die Bezeichnung internalisierte Semantik. Es stellt sich nun die Frage, in welcher formalen Beziehung die internalisierte zur statischen Semantik steht, aus der sie hervorgegangen ist.

Es ist leicht einzusehen, daß der in SPECTRUM verwendete Verfeinerungsbegriff, der im wesentlichen auf der Inklusion der Modellklassen der beiden Spezifikationen basiert, nicht geeignet ist, die Beziehung zwischen statischer und internalisierter Semantik zu beschreiben. Die in den beiden Modellklassen enthaltenen Algebren sind strukturell zu unterschiedlich, als daß sich eine solche Inklusionsbeziehung herstellen ließe, selbst wenn man Konzepte wie Signaturmorphismen und Reduktbildungen mit in Betracht zieht. Diese starke Unterschiedlichkeit der Algebren der beiden Spezifikationen macht jeden Vergleich auf Modellebene kompliziert. Die Beziehung wird jedoch klar, wenn man statt der Modellebene die Ebene der von den Spezifikationen aufgespannten logischen Theorien betrachtet.

Die prädikatenlogische Spezifikationssprache SPECTRUM besitzt einen logischen Kalkül [Reg94], der es erlaubt, Eigenschaften aus SPECTRUM-Formeln abzuleiten (siehe auch Abschnitt 2.2.3). Die *logische Theorie* einer Spezifikation besteht aus allen Sätzen (geschlossene Formeln), die mit Hilfe dieses Kalküls aus den Axiomen der Spezifikation abgeleitet werden können. Die Modelle der Spezifikation sind alle die Algebren, die die Axiome und damit auch alle anderen in der logischen Theorie enthaltenen Formeln erfüllen. Entwickelt man also eine SPECTRUM-Spezifikation weiter, indem man neue Axiome hinzufügt, vergrößert man im allgemeinen die logische Theorie, während man die Modellklasse der Spezifikation verkleinert.

Betrachtet man nun die logische Theorie $\mathcal{TH}^{stat}$ der statischen Semantik und die Theorie $\mathcal{TH}^{int}$ der internalisierten Semantik, so läßt sich eine Abbildung $\Phi : \mathcal{TH}^{stat} \rightarrow \mathcal{TH}^{int}$ angeben, die eine *Theorieinterpretation* im Sinne von [End72] darstellt. Eine solche Abbildung Φ ist dann eine Theorieinterpretation von $\mathcal{TH}^{stat}$ in $\mathcal{TH}^{int}$, wenn gilt:

$$\mathcal{TH}^{stat} \subseteq \Phi^{-1}[\mathcal{TH}^{int}]$$

Es wird also verlangt, daß jede Formel aus $\mathcal{TH}^{stat}$ sich nach Übersetzung mittels Φ in $\mathcal{TH}^{int}$ wiederfindet. Informell heißt das, daß der logische Gehalt der Spezifikation, die $\mathcal{TH}^{stat}$ aufspannt, also im vorliegenden Fall der statischen Semantik, im logischen Gehalt der Zielspezifikation, hier also der internalisierten Semantik enthalten ist.

Im folgenden wird eine solche Abbildung Φ zwischen den Formeln der statischen und der internalisierten Semantik angegeben und gezeigt, daß diese Abbildung die Eigenschaften einer Theorieinterpretation besitzt.

Definition 2 — Die Abbildung $\Phi : W^{Bool}_{\Sigma^s}(\chi) \rightarrow W^{Bool}_{\Sigma^i}(\chi)$ —
Mit $W^{Bool}_{\Sigma^s}(\chi)$ ($W^{Bool}_{\Sigma^i}(\chi)$) sei die Menge der boole'schen Terme (Formeln) bezeichnet, die mit Hilfe der Signatur Σ^s (Σ^i) und der Menge von Variablenbezeichnern χ gebildet werden können. Sei $f \in W^{Bool}_{\Sigma^s}(\chi)$ eine beliebige Σ^s-Formel. Dann ist $\Phi[f]$ definiert als

$$\Phi[f] = \forall \texttt{db:Db. OK db} \Rightarrow \varphi[f]$$

Dabei bezeichnet φ den in Abbildung 4.2 definierten Umbauoperator auf SPECTRUM*-Formeln. Offensichtlich ist $\Phi[f] \in W^{Bool}_{\Sigma^i}(\chi)$ eine Formel, die über der Signatur der internalisierten Semantik definiert ist. Die Fortsetzung von Φ auf Mengen von Formeln wird ebenfalls mit Φ bezeichnet:*

$$\Phi[\{f_1, \ldots, f_n\}] = \{\Phi[f_1], \ldots, \Phi[f_n]\}$$

□

Im folgenden wird gezeigt, daß Φ tatsächlich eine Theorieinterpretation definiert. Dazu werden zunächst in den Sätzen 1 – 3 vorbereitend einige Eigenschaften der Abbildung Φ gezeigt. Satz 4 schließlich faßt die Ergebnisse dieser Hilfssätze zum Beweis der gewünschten Eigenschaft zusammen.

Satz 1 *Die Abbildung Φ ist injektiv.*

Beweis Siehe Anhang A. □

Satz 2 *Die Axiome der statischen Semantik Ax^s können (nach Umbau mittels Φ) aus den Axiomen der internalisierten Semantik hergeleitet werden:*

$$\Phi[Ax^s] \subseteq \mathcal{TH}^{int}$$

Beweis Siehe Anhang A. □

Satz 3 *Sei $f \in W^{Bool}_{\Sigma^s}(\chi)$ eine Σ^s-Formel, $H \subseteq W^{Bool}_{\Sigma^s}(\chi)$ eine Menge von Σ^s-Formeln und B ein Beweisbaum für $H \blacktriangleright f$. Dann gibt es einen Beweisbaum $\Psi[B]$, der mit*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f] \blacktriangleright \Phi[f]$$

endet. Informell bedeutet diese Aussage, daß jeder Beweis, der über der Signatur der statischen Semantik geführt werden kann, nach Übersetzung aller beteiligten Formeln mittels Φ und unter Zuhilfenahme der Axiome der internalisierten Semantik über der Signatur der internalisierten Semantik "nachgespielt" werden kann. Die Abbildung Φ nimmt im wesentlichen eine Sortenrelativierung aller gebundenen Entityvariablen $x : \tau$ ($\tau \in SC^s_E$) mit dem Relativierungsprädikat

$$\rho[x] \stackrel{def}{=} (\delta(x) \Rightarrow x \in \texttt{ent'}\tau\texttt{(db)})$$

vor. Im Beweis $\Psi[B]$ müssen jedoch auch alle frei auftretenden Variablen für Entities mittels ρ eingeschränkt werden. Zu diesem Zweck wird in den Prämissen der auftretenden Sequenzen eine Formelmenge

$$\mathcal{ENV}[H, f] = \{\rho[x] | x : \tau \in (FV[H] \cup FV[f]) \wedge \tau \in SC^s_E\}$$

mitgeführt. Die Prämissenmenge $\mathcal{ENV}[H, f]$ enthält also Formeln, die genau für alle in H und f frei auftretenden Entityvariablen die Relativierung mittels ρ vornehmen.

Beweis Siehe Anhang A. □

Die Aussagen der bisher bewiesenen drei Sätze erlauben es nun, den Zusammenhang zwischen statischer und internalisierter Semantik des E/R-Modells formal zu beschreiben.

Satz 4 *Die Abbildung Φ beschreibt eine Theorieinterpretation der Theorie $\mathcal{TH}^{stat}$ in die Theorie $\mathcal{TH}^{int}$.*

Beweis *Nach [End72] ist zu zeigen*

$$\mathcal{TH}^{stat} \subseteq \Phi^{-1}[\mathcal{TH}^{int}]$$

Um diese Inklusion sicherzustellen, muß für jede geschlossene Σ^s-Formel f mit $Ax^s \blacktriangleright f$ gezeigt werden:

(i) *$\Phi[f]$ ist wieder geschlossen. Dies ist aufgrund der Definition von Φ offensichtlich.*

(ii) *$Ax^i \blacktriangleright \Phi[f]$*

Da es einen Beweis für $Ax^s \blacktriangleright f$ *gibt, gibt es nach Satz 3 auch einen Beweis für*

$$Ax^i, \Phi[Ax^s], \mathcal{ENV}[Ax^s, f] \blacktriangleright \Phi[f]$$

Da sowohl die Axiome in Ax^s *als auch die Formel* f *geschlossen sind, ist* $\mathcal{ENV}[Ax^s, f] = \emptyset$. *Deshalb gilt auch*

$$Ax^i, \Phi[Ax^s] \blacktriangleright \Phi[f]$$

Wegen Satz 2 gilt $\Phi[Ax^s] \subseteq \mathcal{TH}^{int}$, *das heißt alle Axiome aus* Ax^s *lassen sich aus* Ax^i *ableiten. Deshalb gilt auch*

$$Ax^i \blacktriangleright \Phi[f]$$

Damit ist auch Bedingung ***(ii)*** *gezeigt und nachgewiesen, daß die Abbildung* Φ *tatsächlich eine Theorieinterpretation definiert.* □

Die in Satz 4 gezeigte Eigenschaft der Abbildung Φ stellt alleine noch nicht sicher, daß der "logische Gehalt" von $\mathcal{TH}^{stat}$ vollständig in $\mathcal{TH}^{int}$ wiedergefunden werden kann, denn sie schließt nicht aus, daß verschiedene Formeln aus $\mathcal{TH}^{stat}$ in die gleiche $\mathcal{TH}^{int}$-Formel übersetzt werden. Im Extremfall wäre sogar eine Abbildung denkbar, die alle $\mathcal{TH}^{stat}$-Formeln nach `true` übersetzt und dennoch die in Satz 4 nachgewiesene Eigenschaft der Theorieinterpretation aufweist, obwohl die in $\mathcal{TH}^{stat}$ enthaltene Information durch diese Abbildung vollständig verloren geht.

Durch die in Satz 1 gezeigte Injektivität von Φ ist jedoch sichergestellt, daß unterschiedliche Formeln auch nach der Übersetzung unterschieden werden können und somit der "logische Gehalt" von $\mathcal{TH}^{stat}$ bei der Übersetzung mittels Φ vollständig erhalten bleibt. Damit ist die Definition der internalisierten Semantik neben der statischen Semantik auch formal gerechtfertigt.

4.3 Eine praktisch einsetzbare Zugriffsschicht

Die Konstruktion der internalisierten Semantik hat mit der Spezifikation der Datenbanksorte `Db` die durch das zugehörige E/R-Schema beschriebenen Daten innerhalb der Spezifikationssprache SPECTRUM verfügbar gemacht. Damit wurde die Möglichkeit geschaffen, das E/R-Modell als Technik zur Beschreibung strukturierter Daten in den formalen Software-Entwicklungsprozeß mit dieser Spezifikationssprache zu integrieren.

Um diese Integration auch praktikabel, also praktisch einsetzbar, zu machen, reicht die Spezifikation der Datenbanksorte, wie sie in der internalisierten Semantik gegeben ist, jedoch nicht aus. Es wird eine Zugriffs-Schnittstelle zur Datenbank benötigt, das heißt eine Menge von Funktionen, die einen einfachen — sowohl lesenden als auch verändernden — Zugriff auf die in der Datenbank enthaltenen Informationen erlauben.

Im Datenbankbereich werden für den Zugriff auf Datenbanken spezielle Sprachen, sogenannte *Data Manipulation Languages* (DML), eingesetzt. Eine der bekanntesten DML ist

die Sprache SQL [Ins86], die bei vielen relationalen Datenbanksystemen verwendet wird. Die Arbeiten der Braunschweiger Gruppe haben unter anderem eine vollständig formale, algebraische Formalisierung dieser DML gegeben [KG90, Gog94]. SQL ist jedoch eine ungetypte Sprache, die nur mit Schwierigkeiten mit dem starken Typsystem der Sprache SPECTRUM in Einklang zu bringen ist. Um Operatoren wie zum Beispiel den Join-Operator von SQL in SPECTRUM spezifizieren zu können, müßten einige Maßnahmen ergriffen werden, die die starke Typung von SPECTRUM umgehen, wie zum Beispiel die Verwendung eines *Attributuniversums* als variante Sorte, die alle Attributsorten umfaßt, und die Modellierung von Entitytypen als Listen von Attributen. Dies würde die Anwendung der so spezifizierten SQL-Kommandos bei der praktischen Softwareentwicklung mit SPECTRUM dadurch erschweren, daß alle Attribute und Entities vor der Anwendung eines SQL-Kommandos in diese ungetypte Welt übersetzt und die Ergebnisse wieder rückübersetzt werden müßten.

Obwohl es also prinzipiell möglich ist, SQL beziehungsweise eine für das E/R-Modell angepaßte Variante davon in den vorliegenden Ansatz zu integrieren, wird aus Gründen der Praktikabilität darauf verzichtet und stattdessen eine Zugriffsschicht angegeben, die aus einer Reihe von Funktionen besteht, die es im wesentlichen erlauben, Entities und Beziehungen zwischen ihnen aufzubauen, in die Datenbank einzutragen, abzufragen oder aus der Datenbank zu löschen. Eine derartige Zugriffsschicht ist zwar deutlich einfacher und weniger flexibel als zum Beispiel SQL, hat aber den Vorteil, daß sie sich nahtlos in die formale Anwendungsentwicklung mit SPECTRUM einfügt.

Die um die angesprochenen Zugriffsfunktionen erweiterte internalisierte Semantik wird im weiteren als *Zugriffsspezifikation* bezeichnet. Die folgenden Abschnitte befassen sich mit der Entwicklung einer solchen Zugriffsspezifikation aus der internalisierten Semantik. Dies geschieht in vier Schritten:

- Spezifikation der Entitysorten als freie Datentypen,
- Implementierung der Relationshiptypen als Mengen von Entity-Tupeln,
- Erweitern der statischen Integritätsbedingungen um allgemeine SPECTRUM-Formeln,
- Anreichern um Zugriffsfunktionen, die das Verändern des Datenbankzustands erlauben.

Alle diese Schritte sind Realisierungen im Sinne der SPECTRUM-Methodik (siehe Abschnitt 2.2.2). Da dies in allen vier Fällen leicht einzusehen ist, stellen die folgenden Abschnitte zwar die Entwicklungsschritte dar, verzichten aber auf einen formalen Beweis der Realisierungseigenschaft.

Die Zugriffsspezifikation wird als hierarchische Spezifikation angegeben. Insgesamt ergibt sich folgende Situation: Um einen Datenbestand mit der Technik der E/R-Modellierung in SPECTRUM zu beschreiben, erstellt der Software-Entwickler ein E/R-Diagramm der Daten (evtl. mit zusätzlichen statischen Integritätsbedingungen als SPECTRUM-Formeln) sowie eine Spezifikation der Attributsorten. Abbildung 4.3 stellt diese Situation dar. Diesen

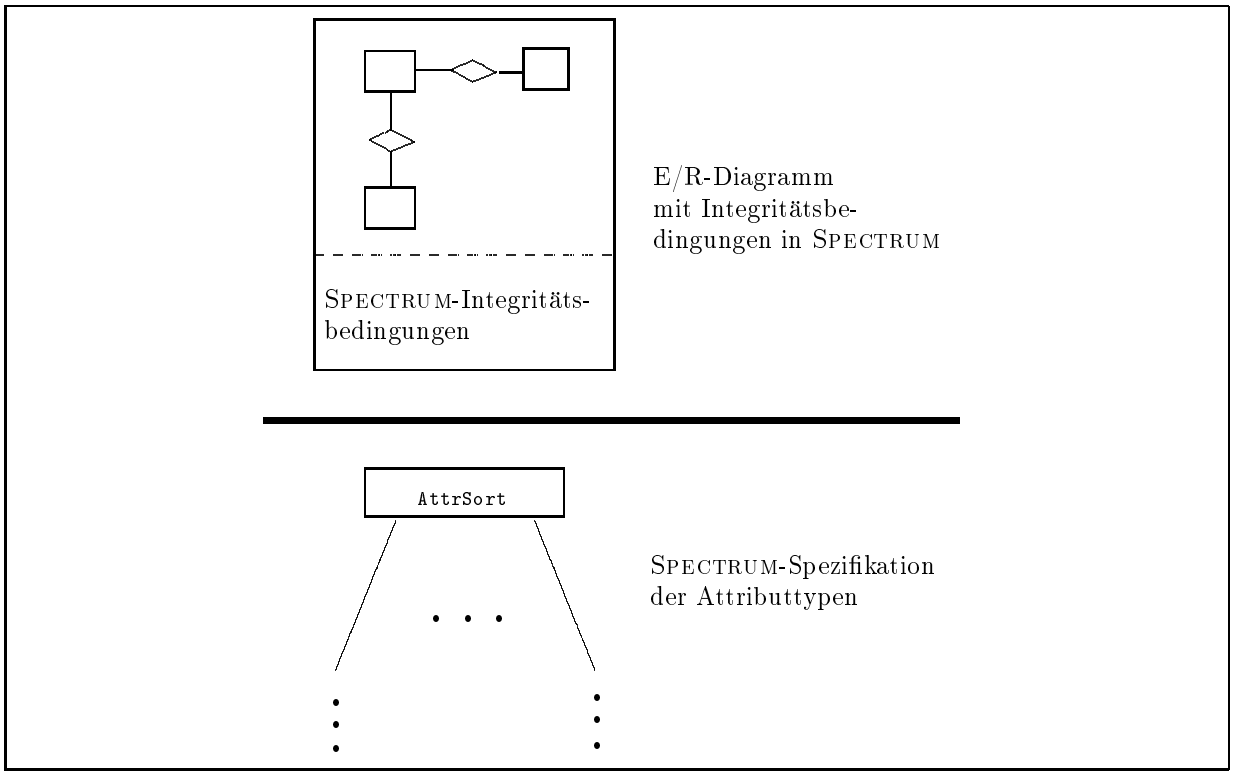


Abbildung 4.3: Vom Entwickler erstelltes Datenschema

beiden Komponenten wird eine hierarchische Zugriffsspezifikation (siehe Abbildung 4.4) zugeordnet, die, basierend auf der Spezifikation der Attributsorten, zunächst die Entitysorten und darauf aufbauend die Datenbanksorte und die darüberliegende Zugriffsschicht spezifiziert.

In den folgenden Abschnitten werden zunächst die vier Entwicklungsschritte, die zur Zugriffsspezifikation führen, detailliert vorgestellt. Danach wird als Beispiel die vollständige Zugriffsspezifikation des RECHERCHE-Datenschemas angegeben.

4.3.1 Entitysorten

Die Entitysorten sind in der internalisierten Semantik sehr lose spezifiziert. Die einzige Forderung, die an eine Entitysorte gestellt wird, ist die Existenz von Selektorfunktionen für die Attribute des modellierten Entitytyps. Diese schwache Forderung war ausreichend, um dem E/R-Modell eine formale Semantik zu geben. Im praktischen Einsatz bei der Software-Entwicklung ist es darüber hinaus jedoch nötig, Entities konkret aufschreiben zu können, das heißt es werden Konstruktorfunktionen benötigt, die es erlauben, aus den zugehörigen Attributwerten Entities aufzubauen.

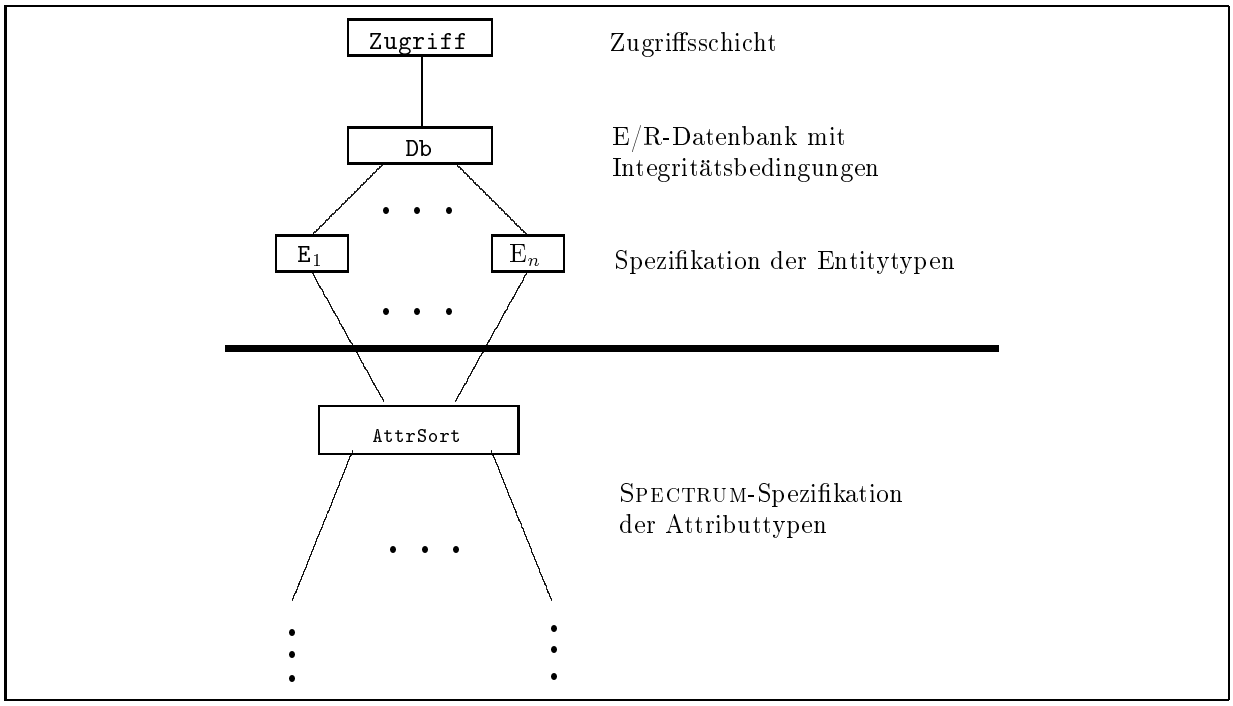


Abbildung 4.4: Dem Datenschema zugeordnete Zugriffsspezifikation

Beispiel 7 hat gezeigt, daß die Erweiterung um Konstruktoren für Entitysorten bei der statischen Semantik im allgemeinen zu inkonsistenten Spezifikationen führt. Bei der internalisierten Semantik ist diese Gefahr jedoch nicht gegeben, da hier das E/R-Schema nicht durch die Modelle der Spezifikation, sondern durch die Elemente der Datenbanksorte `Db` repräsentiert wird. Eine Verletzung der statischen Integrität durch die Möglichkeit, unterschiedliche Entities mit den gleichen Schlüsselattributen zu konstruieren, wirkt sich hier also lediglich auf die Elemente der Datenbanksorte aus, wo sie mit Hilfe des `OK`-Prädikats leicht überprüft werden kann. Es können dadurch aber keine Inkonsistenzen entstehen.

Die konkrete Vorstellung, von der beim Einsatz des E/R-Modells ausgegangen wird, ist, daß Entities durch ihre Attribute bereits *vollständig* charakterisiert sind, daß sie also neben den Attributwerten keine weiteren Eigenschaften besitzen. Sie werden also als Records verstanden, deren Felder die Attribute sind.

In SPECTRUM können solche Records als freie Datentypen mit dem `data`-Konstrukt beschrieben werden. Ein Entitytyp E mit den Attributen $attr_1$, ..., $attr_n$ wird dann als freier Datentyp

`data` E = `mk'`E`(!`$attr_1$`:`A_1`, ..., !`$attr_n$`:`A_n`)`

spezifiziert. Hier bezeichnen A_1,...,A_n die Sorten der jeweiligen Attribute (im Fall optionaler Attribute wie bereits in Abschnitt 3.3 gezeigt mit dem zusätzlichen Konstruktor `Opt`

versehen).

Zusätzlich sind für das Arbeiten mit Entities Funktionen nützlich, die ein selektives Verändern einzelner Attributwerte gestatten. Die Spezifikation des Entitytyps E wird deshalb ergänzt um die Funktionen

```
set'attr_1 : E × A_1 → E;
           ⋮
set'attr_n : E × A_n → E;
set'attr_1,...,set'attr_n strict total;

axioms ∀ e, a_1, ..., a_n in
set'attr_1(e,a_1) = mk'E(a_1, attr_2 e,...,attr_n e);
           ⋮
set'attr_n(e,a_n) = mk'E(attr_1 e,...,attr_{n-1} e, a_n);
endaxioms;
```

Auf diese Weise ist eine praktikable Zugriffsschnittstelle zu den Entities des modellierten Datenbestands gegeben.

4.3.2 Implementierung der Relationshiptypen

Ein Relationshiptyp R, der die Entitytypen E_1^R, ..., E_k^R in Beziehung setzt, wird in der internalisierten Semantik durch ein Prädikat repräsentiert, das entscheidet, ob konkrete Entities der beteiligten Typen in der Beziehung R stehen oder nicht. Dieses Prädikat ist in der entsprechenden Komponente der Datenbanksorte `Db` enthalten.

Sollen in der Datenbankzugriffsschicht Funktionen spezifiziert werden, die es erlauben, Beziehungen zwischen Entities explizit einzuführen beziehungsweise zu löschen, müssen diese Beobachtungsprädikate entsprechend geändert werden können. Solche Update-Operationen auf den Relationship-Prädikaten können mit der Sprache SPECTRUM problemlos spezifiziert werden. Alternativ, und leichter verständlich, kann durch einen Datenstrukturwechsel die Repräsentation von Relationshiptypen so verändert werden, daß die Update-Operationen trivial werden. Dieser Weg wird in der Zugriffsspezifikation verfolgt.

Statt Relationshiptypen durch Prädikate zu repräsentieren, werden sie in der Zugriffsspezifikation durch Mengen von Tupeln der beteiligten Entitytypen dargestellt. Die Entities $\mathtt{e}_1$, ..., $\mathtt{e}_k$ sind genau dann in der Beziehung R, wenn das Tupel $(\mathtt{e}_1,\ldots,\mathtt{e}_k)$ in der entsprechenden Menge enthalten ist. Die Signatur der Konstruktorfunktion `mkdb` verändert sich durch diesen Wechsel der Datenstruktur zu

```
mkdb: Set E_1 × ... × Set E_n ×
      Set (E_1^{R_1} × ... × E_{n_1}^{R_1}) ×
                ⋮
```

$$\texttt{Set}\ (E_1^{R_m} \times \ldots \times E_{n_m}^{R_m}) \rightarrow \texttt{Db}$$

Dabei bezeichnen die $E_j^{R_i}$ die am Relationshiptyp R_i beteiligten Entitytypen. Für die Selektoren R_i ergibt sich durch diesen Wechsel der Datenstruktur folgende Implementierung:

$$R_i\texttt{: Db} \rightarrow (E_1^{R_i} \times \ldots \times E_{n_i}^{R_i} \rightarrow \texttt{Bool});$$
$$R_i\ \texttt{strict total;}$$

$$\texttt{axioms}\ \forall \texttt{db}, \texttt{e}_1, \ldots, \texttt{e}_n, \texttt{r}_1, \ldots, \texttt{r}_m, \texttt{e}_1^{R_i}, \ldots, \texttt{e}_{n_i}^{R_i}\ \texttt{in}$$
$$\texttt{db} = \texttt{mkdb}(\texttt{e}_1, \ldots, \texttt{e}_n, \texttt{r}_1, \ldots, \texttt{r}_m) \Rightarrow R_i\ \texttt{db}\ (\texttt{e}_1^{R_i}, \ldots, \texttt{e}_{n_i}^{R_i}) = ((\texttt{e}_1^{R_i}, \ldots, \texttt{e}_{n_i}^{R_i}) \in \texttt{r}_i);$$
$$\texttt{endaxioms;}$$

Durch diesen Wechsel der Datenstruktur ist das Herstellen beziehungsweise Auflösen von Relationships auf das Hinzufügen beziehungsweise Löschen von Elementen in Mengen zurückgeführt.

Die neue Datenstruktur zur Repräsentation von Relationshiptypen ist sehr ähnlich der Struktur, die beim Überführen des E/R-Schemas in ein relationales Schema entsteht. Dort werden Relationshiptypen im allgemeinen durch spezielle Tabellen dargestellt, in die die in Beziehung stehenden Entities (Tupel) eingetragen werden. Allerdings werden dort nur die Schlüssel der betroffenen Tupel als Fremdschlüssel eingetragen. Für spezielle Arten von Relationships (sogenannte 1:1- und 1:n-Beziehungen) ist im relationalen Schema überhaupt keine spezielle Tabelle nötig. In diesen Fällen werden die Fremdschlüssel in eine der an der Beziehung beteiligten Tabellen eingetragen. In der vorliegenden Arbeit wird auf die Betrachtung solcher Effizienzgesichtspunkte verzichtet. Der Übergang von der verwendeten Darstellung beispielsweise zur Verwendung der Schlüssel anstelle der ganzen Entities ist durch einen weiteren Datenstrukturwechsel leicht möglich.

4.3.3 Allgemeine statische Integritätsbedingungen

Statische Integritätsbedingungen beschreiben eine Invariante des modellierten Datenbestands. Mit ihrer Hilfe lassen sich Bedingungen in das Datenschema aufnehmen, die jeder Zustand der Datenbank erfüllen muß. Bei der E/R-Datenmodellierung werden die statischen Integritäten mit Hilfe spezieller graphischer Notationen dargestellt. Dabei gibt es in den verschiedenen Ansätzen zur E/R-Modellierung eine große Zahl unterschiedlicher Notationen, die zum Teil auch unterschiedliche Ausdrucksmächtigkeit aufweisen.

Bei der Definition einer graphischen Notation zur Beschreibung statischer Integritätsbedingungen in einem E/R-Schema spielt eine Reihe von Kriterien eine Rolle:

Verständlichkeit Die Verständlichkeit und leichte Kommunizierbarkeit des E/R-Diagramms darf nicht leiden. Aus diesem Grund ist eine notationelle Überfrachtung der Diagrammdarstellung zu vermeiden.

Mächtigkeit Gerade bei der Analyse der Anforderungen an ein zu entwickelndes Softwaresystem sollten alle Anforderungen, die nur die Daten des Systems betreffen, zusammen mit dem Datenschema notiert werden können. Es sollte also möglich sein, alle diese Anforderungen als statische Integritätsbedingungen zu notieren. Dieser Punkt widerspricht in gewisser Weise dem ersten Punkt, da eine mächtige Sprache zur Formulierung statischer Integritätsbedingungen leicht zu einem schwer verständlichen, überfrachteten Datenschema führen kann.

Implementierbarkeit Das erstellte Datenschema muß in einer späteren Entwicklungsphase (meist in Form einer relationalen Datenbank) implementiert werden. Heutige Datenbanksysteme können in eingeschränkter Form statische Integritätsbedingungen überwachen. Diese Möglichkeit beschränkt sich jedoch meist auf Schlüsselbedingungen und in sehr eingeschränkter Form auf die Überwachung von Graden der modellierten Relationshiptypen.

Die Frage der Implementierbarkeit sollte jedoch bei der Festlegung einer Sprache für statische Integritätsbedingungen ohnehin nicht im Vordergrund stehen. Im in dieser Arbeit vorgestellten Ansatz werden statische Integritätsbedingungen nicht als Einschränkungen gesehen, die zur Laufzeit vom Datenbanksystem überwacht werden müssen, sondern als Beweisverpflichtungen während der Entwicklung einer Applikation. Es muß also für alle Funktionen des entwickelten Systems nachgewiesen werden, daß sie die statische Integrität der Datenbank erhalten. Abschnitt 4.4 wird ein Beispiel für diese Sichtweise geben. Für die Verwendung der statischen Integritätsbedingungen als Beweisverpflichtungen ist es jedoch unerheblich, ob die Bedingungen in einem konkreten Datenbanksystem implementierbar sind.

Durch die Einbettung des E/R-Modells in die Welt der formalen Spezifikation steht mit der Sprache SPECTRUM ein mächtiges Ausdrucksmittel zur Formulierung sehr allgemeiner statischer Integritätsbedingungen zur Verfügung. Um die dadurch gegebenen Möglichkeiten zu nutzen, wird der in Abschnitt 2.1 vorgestellte Ansatz zur E/R-Datenmodellierung wie folgt erweitert:

E/R-Schemata dürfen mit beliebig vielen SPECTRUM-Formeln annotiert werden, die über der Signatur Σ^s der statischen Semantik formuliert sind. Diese Formeln werden in die statische Semantik als zusätzliche Axiome aufgenommen. Mit der Internalisierung der Semantik gelangen sie (nach Umbau durch den Operator φ aus Abbildung 4.2) auch in das `OK`-Prädikat.

Beispiel 9 *In der ISBN-Nummer einer Publikation ist unter anderem ein eindeutiger dreistelliger Code für den herausgebenden Verlag enthalten [Inb83]. Datenbankzustände des RECHERCHE-Schemas (Beispiel 2) sind also nur korrekt, wenn der zu einer Publikation mittels des Relationshiptyps* gibt_heraus *gespeicherte Verlag mit dem in ihrer ISBN-Nummer enthaltenen Verlags-Code übereinstimmt (oder wenn die Publikation keine ISBN-Nummer hat). Diese statische Integritätsbedingung läßt sich mit den Möglichkeiten des*

vorgestellten E/R-Modells nicht ausdrücken. Mit der Spezifikationssprache SPECTRUM *ist es jedoch leicht möglich, diese Bedingung zu formulieren:*

```
∀p:Publikation,v:Verlag.ISBN(p)=UNDEF∨(gibt_heraus(p,v)⇒consistent(ISBN(p),v))
```

Dabei soll die Funktion `consistent` *die oben informell geschilderte Überprüfung vornehmen.* □

Abschnitt 4.4 wird aufzeigen, wie diese erweiterten statischen Integritätsbedingungen als Beweisverpflichtungen bei der Verifikation des entwickelten Softwaresystems genutzt werden können.

4.3.4 Vervollständigung der Zugriffsschicht

Durch die auf der Datenbanksorte definierten Selektorfunktionen besteht in der internalisierten Semantik bereits die Möglichkeit, lesend auf die Datenbank zuzugreifen, das heißt zu entscheiden, ob gegebene Entities in ihr enthalten sind und ob Beziehungen zwischen diesen Entities existieren. Um die Spezifikation in der praktischen Entwicklung einsetzen zu können, werden zusätzliche Funktionen für die folgenden Aufgaben benötigt:

Eintragen einer Entity Für das Hinzufügen einer Entity zur Datenbank wird für jeden Entitytyp E_i eine Funktion put'E_i definiert:

```
put'E_i: E_i × Db → Db;
put'E_i strict total;

axioms ∀db,e,e_1,...,e_n,r_1,...,r_m in
db=mkdb(e_1,...,e_i,...,e_n,r_1,...,r_m) ⇒
  put'E_i(e,db)=mkdb(e_1,...,add(e,e_i),...,e_n,r_1,...,r_m);
endaxioms;
```

Die Funktion `add` bezeichnet dabei die Operation, die das Einfügen eines Elements in eine Menge realisiert.

Löschen einer Entity Analog zum Einfügen einer Entity in die Datenbank wird das Löschen definiert:

```
del'E_i: E_i × Db → Db;
del'E_i strict;

axioms ∀db,e,e_1,...,e_n,r_1,...,r_m in
db=mkdb(e_1,...,e_i,...,e_n,r_1,...,r_m) ⇒
  del'E_i(e,db)=mkdb(e_1,...,del(e,e_i),...,e_n,r_1,...,r_m);
endaxioms;
```

Hier bezeichnet `del` die Funktion zum Löschen eines Elements aus einer Menge. Man beachte, daß die Funktion `del'`E_i nicht total ist. Durch das Löschen einer Entity kann die referentielle Integrität der Datenbank verletzt werden,das heißt es kann das auf Seite 44 für alle definierten Datenbankzustände geforderte Axiom (1) verletzt werden. Ist das der Fall, liefern der Konstruktor `mkdb` und damit auch die Funktion `del'`E_i die undefinierte Datenbank $\perp$.

Etablieren einer Beziehung Durch den in Abschnitt 4.3.2 vorgenommenen Datenstrukturwechsel geht das Eintragen einer Beziehung in die Datenbank analog zum Eintragen einer Entity:

$$\texttt{est'}R_j\texttt{:}\ E_1^{R_j} \times \ldots \times E_{n_j}^{R_j} \times \texttt{Db} \rightarrow \texttt{Db};$$
$$\texttt{est'}R_j\ \texttt{strict};$$

$$\texttt{axioms}\ \forall \texttt{db},\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{r}_m,\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j}\ \texttt{in}$$
$$\texttt{db}=\texttt{mkdb}(\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{r}_j,\ldots,\texttt{r}_m) \Rightarrow$$
$$\texttt{est'}R_j(\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j},\texttt{db})=\texttt{mkdb}(\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{add}((\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j}),\texttt{r}_j),\ldots,\texttt{r}_m);$$
$$\texttt{endaxioms};$$

Wie beim Löschen einer Entity kann auch beim Etablieren einer Beziehung die referentielle Integrität der Datenbank verletzt werden. Aus diesem Grund ist die Funktion `est'`R_j wie auch die Funktion `del'`E_i nicht total.

Entfernen einer Beziehung Das Entfernen einer Beziehung aus der Datenbank kann die referentielle Integrität nicht verletzen. Aus diesem Grund ist die dafür vorgesehene Funktion `rel'`R_j total:

$$\texttt{rel'}R_j\texttt{:}\ E_1^{R_j} \times \ldots \times E_{n_j}^{R_j} \times \texttt{Db} \rightarrow \texttt{Db};$$
$$\texttt{rel'}R_j\ \texttt{strict total};$$

$$\texttt{axioms}\ \forall \texttt{db},\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{r}_m,\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j}\ \texttt{in}$$
$$\texttt{db}=\texttt{mkdb}(\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{r}_j,\ldots,\texttt{r}_m) \Rightarrow$$
$$\texttt{rel'}R_j(\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j},\texttt{db})=\texttt{mkdb}(\texttt{e}_1,\ldots,\texttt{e}_n,\texttt{r}_1,\ldots,\texttt{del}((\texttt{e}_1^{R_j},\ldots,\texttt{e}_{n_j}^{R_j}),\texttt{r}_j),\ldots,\texttt{r}_m);$$
$$\texttt{endaxioms};$$

Zugriff auf Entities über ihren Schlüssel Die wichtigste Funktionalität, die eine Datenbank ihren Nutzern zur Verfügung stellt, ist der Zugriff auf Entities über ihre Schlüssel. Zu diesem Zweck wird in der Zugriffsspezifikation für jeden Entitytyp eine Funktion `get'`E_i spezifiziert:

$$\texttt{get'}E_i\texttt{:}\ \textit{KEY[}E_i\textit{]} \times \texttt{Db} \rightarrow E_i;$$
$$\texttt{get'}E_i\ \texttt{strict};$$

$$\texttt{axioms}\ \forall \texttt{db},\texttt{k}\ \texttt{in}$$

```
δ(get'E_i(k,db))=∃e. e ∈ ent'E_i(db) ∧ key'E_i(e) = k;
get'E_i(k,db)=e ⇔ e ∈ ent'E_i(db) ∧ key'E_i(e) = k;
endaxioms;
```

Dabei steht *KEY[E_i]* für die Sorte des Schlüssels des Entitytyps E_i. Die Funktion key'E_i ist eine Projektionsfunktion, die zu einer Entity den zugehörigen Schlüssel liefert. Der Schlüssel ist im allgemeinen ein Tupel von Schlüsselkomponenten. Man beachte, daß in dem in Abschnitt 2.1 vorgestellten E/R-Modell nicht nur Attributwerte Schlüsselkomponenten sein können, sondern (im Fall externer Identifikatoren) auch Entities.

4.3.5 Die Zugriffsspezifikation von RECHERCHE

Um die in diesem Abschnitt vorgestellten Schritte zu verdeutlichen, soll nun die dem Beispiel-Datenschema RECHERCHE zugeordnete Zugriffsspezifikation vollständig angegeben werden. Dabei wird wie in Beispiel 8 davon ausgegangen, daß die Spezifikation der Attributsorten in einer Spezifikation `AttrSort` gegeben ist.

Basierend auf `AttrSort` werden zunächst die Entitysorten spezifiziert:

```
Autor_RECHERCHE = { enriches Opt + AttrSort;

strict total;

data Autor = mk'Autor(!Adresse:Adr,
                      !Geb_Datum:Datum,
                      !Name:String);
Autor :: EQ;

set'Adresse     : Autor × Adr → Autor;
set'Geb_Datum   : Autor × Datum → Autor;
set'Name        : Autor × String → Autor;

axioms ∀a:Autor,ad:Adr,g:Datum,n:String in
set'Adresse(a,ad)=mk'Autor(ad,Geb_Datum(a),Name(a));
set'Geb_Datum(a,g)=mk'Autor(Adresse(a),g,Name(a));
set'Name(a,n)=mk'Autor(Adresse(a),Geb_Datum(a),n);
endaxioms;
}

Publikation_RECHERCHE = { enriches Opt + AttrSort;

strict total;

data Publikation = mk'Publikation(!ISBN:Opt Nat,
```

```
                                    !Jahr:Nat,
                                    !Titel:String);
Publikation :: EQ;

set'ISBN        : Publikation × Opt Nat → Publikation;
set'Jahr        : Publikation × Nat → Publikation;
set'Titel       : Publikation × String → Publikation;

axioms ∀p:Publikation,i:Opt Nat,j:Nat,t:String in
set'ISBN(p,i)=mk'Publikation(i,Jahr(p),Titel(p));
set'Jahr(p,j)=mk'Publikation(ISBN(p),j,Titel(p));
set'Titel(p,t)=mk'Publikation(ISBN(p),Jahr(p),t);
endaxioms;
}

Schlagwort_RECHERCHE = { enriches Opt + AttrSort;

strict total;

data Schlagwort = mk'Schlagwort(!Begriff:String,
                                !Definition:String);
Schlagwort :: EQ;

set'Begriff     : Schlagwort × String → Schlagwort;
set'Definition  : Schlagwort × String → Schlagwort;

axioms ∀s:Schlagwort,b:String,d:String in
set'Begriff(s,b)=mk'Schlagwort(b,Definition(s));
set'Definition(s,d)=mk'Schlagwort(Begriff(s),d);
endaxioms;
}

Verlag_RECHERCHE = { enriches Opt + AttrSort;

strict total;

data Verlag = mk'Verlag(!V_name:String,
                        !V_ort:String);
Verlag :: EQ;

set'V_name      : Verlag × String → Verlag;
set'V_ort       : Verlag × String → Verlag;

axioms ∀v:Verlag,vn:String,vo:String in
set'V_name(v,vn)=mk'Verlag(vn,V_ort(v));
```

```
set'V_ort(v,vo)=mk'Verlag(V_name(v),vo);
endaxioms;
}
```

Gestützt auf die so gegebene Spezifikation der Entitysorten kann nun in der Spezifikation $\mathtt{Db}_{RECHERCHE}$ die Datenbanksorte Db sowie das OK-Prädikat spezifiziert werden.

```
Db_RECHERCHE = { enriches Nat + Set + Autor_RECHERCHE + Publikation_RECHERCHE
                                   + Schlagwort_RECHERCHE + Verlag_RECHERCHE;

mkdb : Set Autor × Set Publikation × Set Schlagwort × Set Verlag ×
       Set (Publikation × Verlag) ×
       Set (Autor × Publikation) ×
       Set (Publikation × Schlagwort) ×
       Set (Publikation × Publikation)
       → Db;
mkdb strict;

Db freely generated by mkdb;

ent'Autor       : Db → Set Autor;
ent'Publikation : Db → Set Publikation;
ent'Schlagwort  : Db → Set Schlagwort;
ent'Verlag      : Db → Set Verlag;
ent'Autor,ent'Publikation,ent'Schlagwort,ent'Verlag strict total;

gibt_heraus     : Db → (Publikation × Verlag → Bool);
verfaßt         : Db → (Autor × Publikation → Bool);
verwendet       : Db → (Publikation × Schlagwort → Bool);
zitiert         : Db → (Publikation × Publikation → Bool);
gibt_heraus,verfaßt,verwendet,zitiert strict total;

axioms
-- Beziehung zwischen Konstruktor und Selektoren
∀db:Db. ∀a:Set Autor, p:Set Publikation, s:Set Schlagwort, v:Set Verlag,
         g:Set (Publikation × Verlag),
         vf:Set (Autor × Publikation),
         vw:Set (Publikation × Schlagwort),
         z:Set (Publikation × Publikation),
         a':Autor,p':Publikation,p'':Publikation,s':Schlagwort,v':Verlag.
   db = mkdb(a,p,s,v,g,vf,vw,z) ⇒ ent'Autor db = a ∧
                                   ent'Publikation db = p ∧
                                   ent'Schlagwort db = s ∧
                                   ent'Verlag db = v ∧
                                   gibt_heraus db (p',v') = ((p',v')∈g) ∧
                                   verfaßt db (a',p') = ((a',p')∈vf) ∧
```

```
                                    verwendet db (p',s') = ((p',s')∈vw) ∧
                                    zitiert db (p',p'') = ((p',p'')∈z);

∀db:Db. ∀p:Publikation, v:Verlag. δ(gibt_heraus db (p,v)) ⇔
    p ∈ ent'Publikation db ∧ v ∈ ent'Verlag db;
∀db:Db. ∀a:Autor, p:Publikation. δ(verfaßt db (a,p)) ⇔
    a ∈ ent'Autor db ∧ p ∈ ent'Publikation db;
∀db:Db. ∀p:Publikation, s:Schlagwort. δ(verwendet db (p,s)) ⇔
    p ∈ ent'Publikation db ∧ s ∈ ent'Schlagwort db;
∀db:Db. ∀p1:Publikation, p2:Publikation. δ(zitiert db (p1,p2)) ⇔
    p1 ∈ ent'Publikation db ∧ p2 ∈ ent'Publikation db;
endaxioms;


OK : Db to Bool;
OK total;

axioms
∀db:Db. OK(db) ⇔
 -- Aus dem E/R-Diagramm generierte Bedingungen
 (∀v1,v2: Verlag. v1 ∈ ent'Verlag db ∧ v2 ∈ ent'Verlag db ⇒
    V_name v1 = V_name v2 ⇒ v1 = v2) ∧
 (∀a1,a2:Autor. a1 ∈ ent'Autor db ∧ a2 ∈ ent'Autor db ⇒
    Name a1 = Name a2 ∧ Geb_Datum a1 = Geb_Datum a2 ⇒ a1 = a2) ∧
 (∀s1,s2:Schlagwort. s1 ∈ ent'Schlagwort db ∧ s2 ∈ ent'Schlagwort db ⇒
    Begriff s1 = Begriff s2 ⇒ s1 = s2) ∧
 (∀p1,p2:Publikation. p1 ∈ ent'Publikation db ∧ p2 ∈ ent'Publikation db ⇒
    Titel p1 = Titel p2 ∧ Jahr p1 = Jahr p2 ∧
    (∀v:Verlag. v ∈ ent'Verlag db ⇒
                gibt_heraus db (p1,v) = gibt_heraus db (p2,v)) ⇒ p1 = p2) ∧
 (∀v:Verlag. v ∈ ent'Verlag db ⇒
  (∃s: Set (Publikation×Verlag).  ∀p:Publikation. p ∈ ent'Publikation db ⇒
    ((p,v) ∈ s ⇔ gibt_heraus db (p,v)) ∧ 1 ≤ card(s))) ∧
 (∀p:Publikation. p ∈ ent'Publikation db ⇒
  (∃s: Set (Publikation×Verlag). ∀v:Verlag. v ∈ ent'Verlag db ⇒
    ((p,v) ∈ s ⇔ gibt_heraus db (p,v)) ∧ 1 ≤ card(s) ∧ card(s) ≤ 1)) ∧
 (∀p:Publikation. p ∈ ent'Publikation db ⇒
  (∃s: Set (Autor×Publikation). ∀a:Autor. a ∈ ent'Autor db ⇒
    ((a,p) ∈ s ⇔ verfaßt db (a,p)) ∧ 1 ≤ card(s))) ∧
 (∀a:Autor. a ∈ ent'Autor db ⇒
  (∃s: Set (Autor×Publikation). ∀p:Publikation. p ∈ ent'Publikation db ⇒
    ((a,p) ∈ s ⇔ verfaßt db (a,p)) ∧ 1 ≤ card(s))) ∧
 (∀sw:Schlagwort. sw ∈ ent'Schlagwort db ⇒
  (∃s: Set (Publikation×Schlagwort). ∀p:Publikation. p ∈ ent'Publikation db ⇒
    ((p,sw) ∈ s ⇔ verwendet (p,sw)) ∧ 1 ≤ card(s))) ∧
```

```
 -- Allgemeinere SPECTRUM-Bedingungen
 (∀p:Publikation,v:Verlag.ISBN(p)= UNDEF ∨
                                   (gibt_heraus(p,v)⇒consistent(ISBN(p),v));
endaxioms;
}
```

Nun können als oberste Schicht der Zugriffsspezifikation die Zugriffsfunktionen spezifiziert werden.

```
Zugriff_RECHERCHE = { enriches Db_RECHERCHE;

put'Autor       : Autor × Db → Db;
put'Publikation : Publikation × Db → Db;
put'Schlagwort  : Schlagwort × Db → Db;
put'Verlag      : Verlag × Db → Db;
put'Autor,put'Publikation,put'Schlagwort,put'Verlag strict total;
del'Autor       : Autor × Db → Db;
del'Publikation : Publikation × Db → Db;
del'Schlagwort  : Schlagwort × Db → Db;
del'Verlag      : Verlag × Db → Db;
del'Autor,del'Publikation,del'Schlagwort,del'Verlag strict;

axioms ∀db:Db,a:Set Autor, p:Set Publikation, s:Set Schlagwort, v:Set Verlag,
        g:Set (Publikation × Verlag),
        vf:Set (Autor × Publikation),
        vw:Set (Publikation × Schlagwort),
        z:Set (Publikation × Publikation),
        a':Autor,p':Publikation,s':Schlagwort,v':Verlag in
db=mkdb(a,p,s,v,g,vf,vw,z) ⇒
  put'Autor(a',db)=mkdb(add(a',a),p,s,v,g,vf,vw,z) ∧
  put'Publikation(p',db)=mkdb(a,add(p',p),s,v,g,vf,vw,z) ∧
  put'Schlagwort(s',db)=mkdb(a,p,add(s',s),v,g,vf,vw,z) ∧
  put'Verlag(v',db)=mkdb(a,p,s,add(v',v),g,vf,vw,z) ∧
  del'Autor(a',db)=mkdb(del(a',a),p,s,v,g,vf,vw,z) ∧
  del'Publikation(p',db)=mkdb(a,del(p',p),s,v,g,vf,vw,z) ∧
  del'Schlagwort(s',db)=mkdb(a,p,del(s',s),v,g,vf,vw,z) ∧
  del'Verlag(v',db)=mkdb(a,p,s,del(v',v),g,vf,vw,z);
endaxioms;

est'gibt_heraus : Publikation × Verlag × Db → Db;
est'verfaßt     : Autor × Publikation × Db → Db;
est'verwendet   : Publikation × Schlagwort × Db → Db;
est'zitiert     : Publikation × Publikation × Db → Db;
est'gibt_heraus,est'verfaßt,est'verwendet,est'zitiert strict;
rel'gibt_heraus : Publikation × Verlag × Db → Db;
```

```
rel'verfaßt     : Autor × Publikation × Db → Db;
rel'verwendet   : Publikation × Schlagwort × Db → Db;
rel'zitiert     : Publikation × Publikation × Db → Db;
rel'gibt_heraus,rel'verfaßt,rel'verwendet,rel'zitiert strict total;

axioms ∀db:Db,a:Set Autor, p:Set Publikation, s:Set Schlagwort, v:Set Verlag,
        g:Set (Publikation × Verlag),
        vf:Set (Autor × Publikation),
        vw:Set (Publikation × Schlagwort),
        z:Set (Publikation × Publikation),
        a':Autor,p':Publikation,p'':Publikation,s':Schlagwort,v':Verlag in
db=mkdb(a,p,s,v,g,vf,vw,z) ⇒
  est'gibt_heraus(p',v',db)=mkdb(a,p,s,v,add((p',v'),g),vf,vw,z) ∧
  est'verfaßt(a',p',db)=mkdb(a,p,s,v,g,add((a',p'),vf),vw,z) ∧
  est'verwendet(p',s',db)=mkdb(a,p,s,v,g,vf,add((p',s'),vw),z) ∧
  est'zitiert(p',p'',db)=mkdb(a,p,s,v,g,vf,vw,add((p',p''),z)) ∧
  rel'gibt_heraus(p',v',db)=mkdb(a,p,s,v,del((p',v'),g),vf,vw,z) ∧
  rel'verfaßt(a',p',db)=mkdb(a,p,s,v,g,del((a',p'),vf),vw,z) ∧
  rel'verwendet(p',s',db)=mkdb(a,p,s,v,g,vf,del((p',s'),vw),z) ∧
  rel'zitiert(p',p'',db)=mkdb(a,p,s,v,g,vf,vw,del((p',p''),z));
endaxioms;

get'Autor       : (Datum × String) × Db → Db;
get'Publikation : (Nat × String × Verlag) × Db → Db;
get'Schlagwort  : String × Db → Db;
get'Verlag      : String × Db → Db;
get'Autor,get'Publikation,get'Schlagwort,get'Verlag strict;

axioms ∀db:Db,ka:Datum×String,kp:Nat×String×Verlag,ks:String,kv:String in
δ(get'Autor(ka,db))=∃a:Autor. a∈ent'Autor(db) ∧ key'Autor(a)=ka;
get'Autor(ka,db)=a ⇔ a∈ent'Autor(db) ∧ key'Autor(a)=ka;
δ(get'Publikation(kp,db))=
  ∃p:Publikation. p∈ent'Publikation(db) ∧ key'Publikation(p)=kp;
get'Publikation(kp,db)=p ⇔ p∈ent'Publikation(db) ∧ key'Publikation(p)=kp;
δ(get'Schlagwort(ks,db))=
  ∃s:Schlagwort. s∈ent'Schlagwort(db) ∧ key'Schlagwort(s)=ks;
get'Schlagwort(ks,db)=s ⇔ s∈ent'Schlagwort(db) ∧ key'Schlagwort(s)=ks;
δ(get'Verlag(kv,db))=∃v:Verlag. v∈ent'Verlag(db) ∧ key'Verlag(v)=kv;
get'Verlag(kv,db)=v ⇔ v∈ent'Verlag(db) ∧ key'Verlag(v)=kv;
endaxioms;
}
```

4.4 Diskussion

Im vorliegenden Kapitel wurde die statische Semantik so weiterentwickelt, daß das E/R-Modell als Technik zur Datenmodellierung bei der Software-Entwicklung mit SPECTRUM eingesetzt werden kann. Eine Variante dieser Zugriffsspezifikation wurde als Vorveröffentlichung zu dieser Dissertation im Rahmen einer Fallstudie zur Spezifikation der Anforderungen an ein Klinik-Informationssystem erfolgreich eingesetzt [Het93, SNM+93, Huß93b]. Einen umfassenden Überblick über diese Fallstudie mit einer vollständigen Literaturliste gibt [CHL95].

An dieser Stelle werden nun die wichtigsten Punkte der angegebenen Entwicklung herausgestellt und diskutiert. Die dabei getroffenen Entscheidungen werden begründet und mögliche Alternativen vorgestellt und bewertet.

Internalisierung

Die in Kapitel 3 angegebene statische Semantik des E/R-Modells konnte nicht direkt als Technik zur Integration der E/R-Datenmodellierung in die formale Spezifikationssprache verstanden werden, weil sie den Zustandsraum der modellierten Daten als Modellklasse einer SPECTRUM-Spezifikation, also außerhalb der Sprache, repräsentierte. Um die Transaktionen eines Informationssystems spezifizieren zu können, muß es jedoch möglich sein, Funktionen anzugeben, die auf diesem Zustandsraum arbeiten und ihn verändern können.

Durch die Technik der Internalisierung wurde aus der statischen Semantik eine neue Spezifikation erzeugt, die die relevanten Teile der Modelle der statischen Semantik in Form einer Sorte (der Datenbanksorte `Db`) beschreibt. Damit wird der Zustandsraum der modellierten Daten für in der formalen Sprache beschriebene Funktionen zugreifbar. Die Technik der Internalisierung orientiert sich dabei an der in algebraischen Spezifikationstechniken üblichen Definition des Begriffes der Algebra. Ein zentraler Aspekt dieses Ansatzes ist, daß die internalisierte Semantik aus der statischen Semantik konstruiert wurde und daß dadurch eine formale Beziehung zwischen diesen beiden Semantiken des E/R-Modells besteht, die mit Hilfe einer Theorieinterpretation beschrieben werden kann.

Als Alternative zu diesem Vorgehen wäre es möglich gewesen, die Transaktionen als Funktionen zwischen Algebren zu spezifizieren. Diese Funktionen müßten also auf der Modellklasse der statischen Semantik operieren. Da es mit der Sprache SPECTRUM nicht möglich ist, Funktionen zwischen Modellen zu beschreiben, müßte dafür eine andere Spezifikationstechnik gefunden werden, wie etwa die Technik der *Evolving Algebras* [Gur91]. Mit einem solchen Ansatz würde jedoch das ursprünglich gesteckte Ziel, die Software-Entwicklung mit SPECTRUM zu unterstützen, verfehlt. Deshalb wurde in dieser Arbeit die Internalisierung der statischen Semantik als der geeignetere Weg gewählt.

Statische Integritätsbedingungen

Die statischen Integritätsbedingungen eines gegebenen E/R-Schemas werden in der internalisierten Semantik durch das `OK`-Prädikat repräsentiert, dessen Axiomatisierung aus den Axiomen der statischen Semantik gewonnen wird und das in der Lage ist, Datenbankzustände auf statische Integrität zu untersuchen. Die Datenbanksorte enthält also auch nicht integre Datenbankzustände als Elemente.

Eine Alternative dazu wäre die Beschränkung der Datenbanksorte auf ausschließlich integre Zustände unter Verzicht auf das `OK`-Prädikat. Aus den Axiomen der statischen Semantik würden in diesem Fall Axiome generiert, die diese Beschränkung der Datenbanksorte vornehmen. Dieser Ansatz stellt allerdings hohe Anforderungen an die Datenbankschnittstelle. Die Sprache zur Manipulation der Daten muß in diesem Fall die Erhaltung der statischen Integrität der Datenbank garantieren. Mit steigender Komplexität der Integritätsbedingungen wird zwangsläufig auch die Datenbankschnittstelle komplexer, wenn sie in der Lage sein soll, die statische Integrität der Datenbank zu erhalten. Läßt man, wie in der Zugriffsspezifikation, beliebige allgemeine statische Integritätsbedingungen zu, ist es nicht mehr möglich, eine Datenbankschnittstelle zu definieren, die die statische Integrität als Invariante garantiert.

Ein Blick auf das Gebiet der relationalen Datenbanksysteme zeigt, daß die gewählte Modellierung (Zulassen nicht integrer Datenzustände und Spezifikation des `OK`-Prädikats) mit der Wirklichkeit gut übereinstimmt. Die bei relationalen Datenbanksystemen vorherrschende Sprache SQL garantiert die Erhaltung der statischen Integrität — ebenso wie die in dieser Arbeit vorgestellte Zugriffsspezifikation — nicht. Die Datenbankmanagementsysteme lassen deshalb auch nicht integre Datenbankzustände zu. Um das Problem der Datenbankintegrität dennoch zufriedenstellend in den Griff zu bekommen, stellen diese Datenbanksysteme ein *Transaktionskonzept* zur Verfügung. Eine Transaktion faßt eine Reihe von Datenbankzugriffen zusammen, die insgesamt eine integritätserhaltende Manipulation der Datenbank darstellen. Die einzelnen innerhalb der Transaktion stattfindenden Datenbankzugriffe dürfen die statische Integrität jedoch kurzfristig verletzen. Mit Hilfe des `OK`-Prädikats ist diese Eigenschaft von Transaktionen offenkundig leicht zu formulieren.

Bei der Definition der Zugriffsspezifikation wurde das verwendete E/R-Modell um die Möglichkeit erweitert, allgemeine statische Integritätsbedingungen in Form von SPECTRUM-Formeln anzugeben. Damit können bei der Software-Entwicklung alle Anforderungen an die Daten des zu modellierenden Systems in einem einzigen Dokument zusammengefaßt werden, das bei der weiteren Entwicklung eine zentrale Rolle spielt. Dieses Dokument besteht aus einem E/R-Diagramm und den angesprochenen SPECTRUM-Formeln und ist aufgrund der ihm zugeordneten Zugriffsspezifikation vollständig formal zu verstehen. Da insbesondere die Integritätsbedingungen formal angegeben sind, können sie zur Verifikation der entwickelten Systemfunktionen verwendet werden. Für jede spezifizierte Transaktion

$\texttt{t}\colon I \times \texttt{Db} \to O \times \texttt{Db}$

ist folgende Formel zu zeigen:

```
∀i:I,o:O,db:Db,db':Db. OK(db) ∧ t(i,db)=(o,db') ⇒ OK(db')
```

Gelingt es, diese Aussage zu zeigen, ist nachgewiesen, daß die spezifizierte Transaktion die statische Integrität der Datenbank nicht verletzen kann.

Beispiel 10 *Eine Transaktion, die in der RECHERCHE-Applikation aus Beispiel 2 eine neue Publikation in die Datenbank einträgt, muß sicherstellen, daß die Beziehungen zum Verlag und den Autoren ebenfalls eingetragen werden. Eine derartige Funktion kann zum Beispiel durch ihre Auswirkungen auf die einzelnen Komponenten der Datenbank spezifiziert werden:*

```
enter : Publikation × Verlag × Set Autor × Db → Db;
enter strict total;

axioms ∀db,db':Db, p:Publikation, v:Verlag, sa:Set Autor in
enter(p,v,sa,db) = db' ⇒
  ent'Schlagwort(db') = ent'Schlagwort(db) ∧
  ent'Publikation(db') = add(p,ent'Publikation(db)) ∧
  ent'Verlag(db') = add(v,ent'Verlag(db)) ∧
  ent'Autor(db') = ent'Autor(db) ∪ sa ∧
  verwendet(db') = verwendet(db) ∧
  zitiert(db') = zitiert(db) ∧
  (∀p':Publikation,v':Verlag.
    gibt_heraus db' (p',v') = (gibt_heraus db (p',v') ∨ (p' = p ∧ v' = v))) ∧
  (∀p':Publikation,a':Autor.
    verfaßt db' (a',p') = (verfaßt db (a',p') ∨ (p' = p ∧ a' ∈ sa)));
endaxioms
```

Wird nun eine konkrete Implementierung für `enter` *angegeben, zum Beispiel mit Hilfe in dieser Arbeit angegebenen Funktionen der Zugriffsschicht (in diesem Fall* `put'Verlag`, `est'gibt_heraus`, *...), so sind zum Nachweis der Korrektheit dieses Implementierungsschritts folgende Beweisverpflichtungen zu zeigen:*

- *Das oben als deskriptive Spezifikation von* `enter` *angegebene Axiom*
- *Die Formel*

```
∀db,db':Db, p:Publikation, v:Verlag, sa:Set Autor.
  OK(db) ∧ enter(p,v,sa,db) = db' ⇒ OK(db')
```

Die zweite Beweisverpflichtung stellt sicher, daß die Transaktion `enter` *die statische Integrität der Datenbank erhält.* □

Eine wichtige Klasse statischer Integritätsbedingungen, die mit dem E/R-Modell normalerweise nicht formuliert werden können, die jedoch in der weiteren Entwicklung hin zur Datenbank wesentlich sind, sind *funktionale Abhängigkeiten* zwischen Attributen. Diese werden bei der Normalisierung des Datenschemas benötigt (siehe zum Beispiel [SS83]). Es ist deshalb sinnvoll, sie gleich ins konzeptuelle Schema als zusätzliche statische Integritäten aufzunehmen.

In Abschnitt 3.4 wurden einfache Kriterien angegeben, die es erlauben, die Konsistenz der statischen Semantik eines E/R-Schemas ohne allgemeine statische Integritätsbedingungen zu überprüfen. Es ist klar, daß durch das Zulassen beliebiger SPECTRUM-Formeln zur Beschreibung statischer Integritäten diese Kriterien nicht mehr ausreichen, um die Konsistenz der statischen Semantik sicherzustellen. Hier muß auf klassische Methoden wie die explizite Konstruktion eines Modells zurückgegriffen werden.

Zugriffsspezifikation

Die in Abschnitt 4.3 gegebene Zugriffsspezifikation stellt eine Reihe einfacher Funktionen zur Verfügung, die sowohl lesenden als auch verändernden Zugriff auf die Datenbank erlauben. Obwohl diese Schnittstelle wesentlich einfacher ist als herkömmliche Datenbankzugriffssprachen (Data Manipulation Languages) wie zum Beispiel SQL, wurde die praktische Einsetzbarkeit in der oben angesprochenen Fallstudie gezeigt.

Es ist zu beachten, daß die Größe der Zugriffsspezifikation linear mit der Anzahl der im zugehörigen E/R-Schema definierten Komponenten (Entitytypen, Relationshiptypen, Attribute) wächst. Die Spezifikation bleibt so auch für größere Anwendungen als das in dieser Arbeit gegebene RECHERCHE-Beispiel beherrschbar.

Schrittweise Entwicklung von E/R-Schemata

Kapitel 3 und 4 haben sich damit beschäftigt, wie dem E/R-Modell durch Angabe einer Übersetzung nach SPECTRUM eine formale Semantik zugewiesen werden kann und wie sich aus der so erzeugten Spezifikation eine für den praktischen Einsatz in der formalen Anwendungsentwicklung geeignete Beschreibung der Daten eines Informationssystems gewinnen läßt. Diese beiden Kapitel berücksichtigen jedoch nicht, daß das betrachtete E/R-Schema selbst das Produkt eines Entwicklungsprozesses ist, der oft als *konzeptuelle Datenmodellierung* bezeichnet wird.

Methodisch ist die gegebene Übersetzung eines E/R-Schemas nach SPECTRUM nur sinnvoll, wenn die zur Erzeugung des Schemas eingesetzten Vorgehensweisen mit der SPECTRUM-Methodik (Abschnitt 2.2.2) verträglich sind. Diese Forderung bedeutet, daß alle bei der Entwicklung des Datenschemas verwendeten Schritte auf der Ebene der generierten SPECTRUM-Spezifikationen Realisierungsschritten entsprechen sollten. Das folgende Kapitel wird deshalb auf den Vorgang der konzeptuellen Datenmodellierung eingehen und

untersuchen, inwieweit dieser Vorgang mit der Entwicklungsmethodik von SPECTRUM verträglich ist.

Kapitel 5

Konzeptuelle Datenmodellierung mit dem E/R-Modell

Im vorangegangenen Kapitel wurde gezeigt, wie die E/R-Datenmodellierung als Technik zur Unterstützung der Entwicklung stark datenorientierter Systeme im Rahmen der formalen Software-Entwicklungsmethodik von SPECTRUM eingesetzt werden kann. Im folgenden soll nun die Rolle des E/R-Modells bei der Spezifikation und Entwicklung von Datenbanken geklärt und insbesondere der Vorgang der Erstellung von E/R-Schemata untersucht werden.

Abbildung 5.1 zeigt das weithin akzeptierte Modell der "ANSI/SPARC/Study Group — Database Management Systems" [ANS78, TK81] zur Spezifikation von Datenbanken auf drei Abstraktionsebenen. Die Daten der realen Welt (eines Anwendungsgebiets) werden in einem *konzeptuellen Datenschema* modelliert. Zur Formulierung dieses Schemas wird häufig das E/R-Modell (oder ein anderes semantisches Datenmodell) benutzt. Da die gängigen Datenbankmanagementsysteme (DBMS) auf anderen, einfacheren Datenmodellen (relatio-

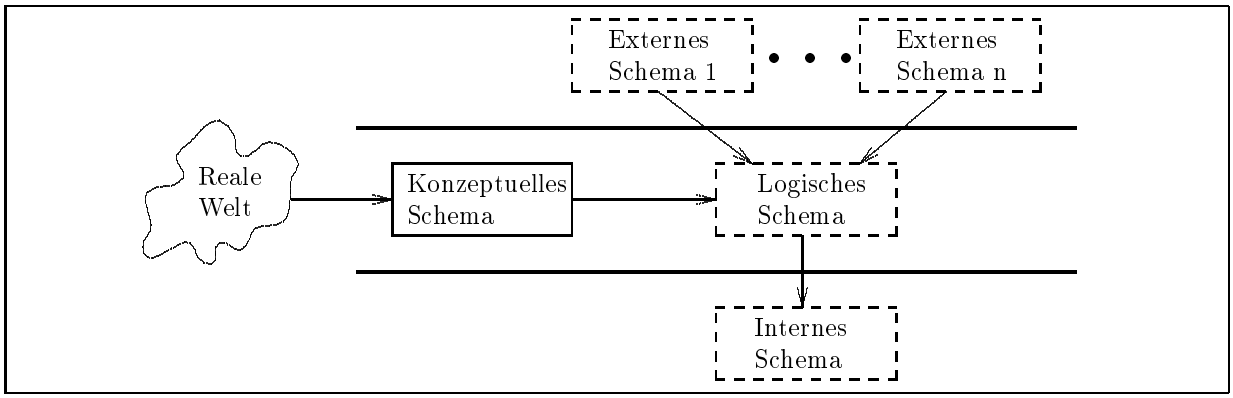


Abbildung 5.1: Das ANSI/X3/SPARC 3-Schichten-Modell

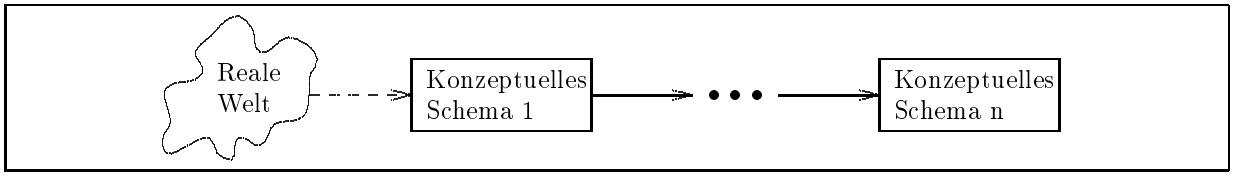


Abbildung 5.2: Schrittweise Entwicklung des konzeptuellen Schemas

nales Modell, Netzwerkmodell, hierarchisches Modell) beruhen als dem E/R-Modell, wird das konzeptuelle Schema in das Datenmodell des DBMS übertragen. Das so entstehende Schema wird als *logisches Schema* bezeichnet. Auf der Basis des logischen Schemas werden eine Reihe *externer Schemata* definiert, die die Sichten der verschiedenen Nutzer der Datenbank darstellen. Die externen Schemata werden im gleichen Datenmodell notiert wie das logische Schema. Das *interne Schema* stellt schließlich die Verbindung zum konkreten DBMS her. Es formuliert die im logischen Schema gegebene Beschreibung der Daten mit Hilfe der *Data Definition Language* (DDL) des Datenbanksystems (zum Beispiel SQL) und berücksichtigt dabei physikalische Aspekte der Datenspeicherung wie Indexe und Datencluster. Von allen an der Datenbankspezifikation beteiligten Schemata wird also nur das konzeptuelle potentiell mit Hilfe des E/R-Modells beschrieben. Deshalb beschäftigt sich die vorliegende Arbeit ausschließlich mit der konzeptuellen Datenmodellierung, das heißt mit der Erstellung des konzeptuellen Schemas.

Die Bedeutung der in Abbildung 5.1 vorkommenden Pfeile wurde in der Erläuterung bewußt unklar gelassen. Sie können zunächst als '*entsteht aus*' beziehungsweise '*steht in Beziehung zu*' gelesen werden. Es ist Aufgabe dieses Kapitels, die Bedeutung dieser Pfeile zu untersuchen, soweit sie mit der Erstellung des konzeptuellen Schemas zu tun haben. Es kommt also für die weitere Untersuchung nur der Pfeil zwischen den Knoten '*Reale Welt*' und '*Konzeptuelles Schema*' in Frage. Dieser Pfeil bezeichnet die Erfassung von informell gegebenen Anforderungen der realen Welt im konzeptuellen Schema, das in dieser Arbeit als E/R-Schema verstanden wird. Deshalb sieht es zunächst so aus, als ließe sich der durch diesen Pfeil symbolisierte Übergang nicht formal fassen. Es ist jedoch illusorisch anzunehmen, daß die Erstellung des konzeptuellen Schemas in einem einzigen Schritt vor sich geht, wie es von Abbildung 5.1 suggeriert wird. Hinter diesem Übergang steht vielmehr ein Entwicklungsprozeß, der eine nähere Untersuchung rechtfertigt. Für die Entwicklung des konzeptuellen Schemas sind zwei verschiedene Ansätze denkbar, die beide in der Praxis der Datenmodellierung Anwendung finden.

Die Sichtweise der schrittweisen Entwicklung des konzeptuellen Schemas stellt Abbildung 5.2 dar. Nach einer ersten, formal nicht beschreibbaren Erfassung der Daten des Anwendungsgebiets wird das so entstandene Schema schrittweise weiterentwickelt und verfeinert, bis ein als vollständig und adäquat betrachtetes konzeptuelles Schema entstanden ist. Die zwischen den verschiedenen Zwischenversionen bestehenden Beziehungen sind Gegenstand der in Abschnitt 5.1 angestellten Untersuchungen.

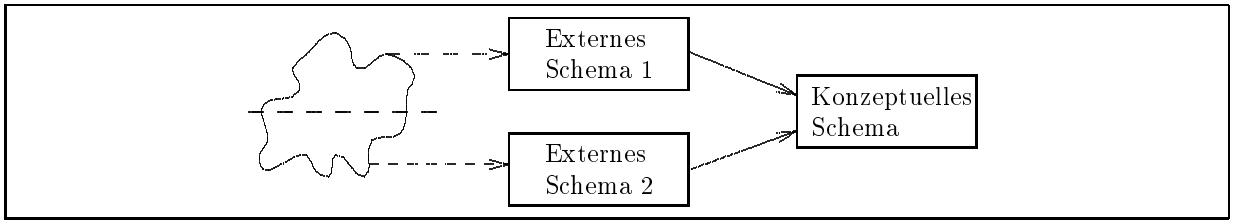


Abbildung 5.3: Integration externer Schemata

Einen anderen Ansatz zur Entwicklung des konzeptuellen Schemas stellt Abbildung 5.3 dar. Bei einem großen, klar gegliederten Anwendungsgebiet (zum Beispiel Betrieb mit mehreren Abteilungen) ist es oftmals sinnvoll, die Anforderungen der verschiedenen Teile des Anwendungsgebiets einzeln in unterschiedlichen Datenschemata zu erfassen, die als externe Schemata beziehungsweise *Views* bezeichnet werden. Diese externen Schemata werden dann in einem oft als *View Integration* bezeichneten Vorgang zum konzeptuellen Schema des gesamten Anwendungsbereichs verschmolzen. Abschnitt 5.2 wird sich im Detail mit einer Möglichkeit beschäftigen, die Formalisierung des E/R-Modells in SPECTRUM zur formalen Integration von externen Schemata zu einem konzeptuellen Schema einzusetzen.

5.1 Verfeinerung konzeptueller Schemata

Der Prozeß der konzeptuellen Datenmodellierung wird in den meisten Software-Engineering-Methoden nur sehr unzureichend unterstützt. Bekannte Methoden wie zum Beispiel SSADM [DCC92] oder OMT [RBP+91] räumen zwar dem konzeptuellen Datenschema (in Form eines E/R-Schemas in SSADM beziehungsweise eines Objektmodells in OMT) eine zentrale Rolle ein, das Vorgehen bei der Erstellung dieses Schemas wird jedoch methodisch kaum unterstützt[1]. Es wird in diesen Methoden zwar anerkannt, daß die Entwicklung des konzeptuellen Schemas ein evolutionärer Vorgang ist, der (wie in Abbildung 5.2 dargestellt) in einer Reihe von Schritten abläuft, über die Art der zwischen den einzelnen Zwischenversionen liegenden Entwicklungsschritte wird jedoch keine Aussage gemacht. Damit ist jede Veränderung eines Schemas ein gültiger Entwicklungsschritt. Insbesondere sind so im Rahmen der Entwicklung auch Schritte erlaubt, die vorhergehende Entwicklungsschritte zurücknehmen, indem sie zum Beispiel Teile des bereits entwickelten Schemas weglassen. Solche *Revisionen* im Entwicklungsprozeß sollten jedoch methodisch klar von den "normalen" Entwicklungsschritten unterschieden werden.

Die in SPECTRUM definierte Methodik stellt einen Gegenpol zu dieser sehr liberalen Sicht auf den Entwicklungsprozeß dar. Hier sind nur solche Entwicklungsschritte erlaubt, die eine formale Realisierungsbeziehung zwischen den beteiligten Spezifikationen etablieren (siehe Abschnitt 2.2.2). Revision bedeutet in diesem Ansatz explizites Zurücksetzen im bisherigen

[1] In der Diskussion der Methode SAZ in Abschnitt 1.3 wird dieser Punkt etwas genauer ausgeführt.

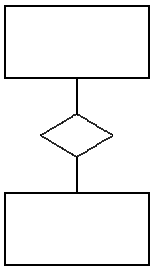
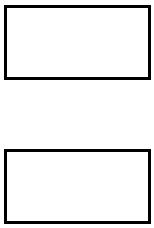
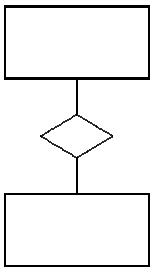
Primitive	Starting Schema	Resulting Schema
T_1: Entity → Related Entities		
B_2: Relationship Generation		

Abbildung 5.4: Beispiele für Schema-Transformationsregeln nach Batini

Entwicklungspfad, das heißt Wiederaufnahme der Entwicklung an einem früheren Knoten. Somit läßt sich jede Entwicklung in SPECTRUM als Folge von Realisierungsschritten ohne Revision darstellen.

Dem rigiden Entwicklungsbegriff von SPECTRUM kommen im Bereich der Datenmodellierung Ansätze nahe, die die Entwicklung konzeptueller Schemata ausschließlich über die Anwendung einer Reihe von (vordefinierten) primitiven Transformationen gestatten. In [BCN92] beispielsweise geben Batini, Ceri und Navathe eine Methodik zur Datenmodellierung an, die auf einem Satz von Transformationsregeln für (Teile von) E/R-Schemata basiert. Im folgenden wird diese Methodik kurz vorgestellt und untersucht, wieweit sie sich als Grundlage zur Definition einer formalen, mit dem SPECTRUM-Realisierungsbegriff verträglichen Methodik zur Datenmodellierung eignet.

Die Transformationsregeln werden in [BCN92] graphisch angegeben und mit einer informellen Erläuterung versehen. Abbildung 5.4 zeigt exemplarisch die Definition von zwei derartigen Transformationsregeln. Zu diesen Regeln sind in [BCN92] folgende Erläuterungen gegeben:

> Primitive T_1 refines an entity into a relationship between two or more entities.
>
> Primitive B_2 generates a new relationship between previously defined entities.

Ein Vorteil dieses Ansatzes ist die Beschränkung auf einen fest vorgegebenen Satz von (sinnvollen) Entwicklungsschritten. Revision im Sinne eines Rücksetzens der Entwicklung in einen früheren Zustand ist damit nicht als Entwicklungsschritt zugelassen[2]. Jedes Datenschema entsteht durch Anwendung einer Transformationsregel auf das Vorgängerschema. Damit ist ein eindeutiger Entwicklungsbegriff festgelegt.

Als Nachteil der in [BCN92] angegebenen Transformationsregeln ist ihre informelle Definition zu sehen, die lediglich aus einer graphischen Notation und einem kurzen erläuternden Text besteht. Damit werden die Regeln nicht eindeutig festgelegt. So läßt zum Beispiel die Definition der Regel T_1 aus Abbildung 5.4 eine ganze Reihe von Fragen offen:

- Was geschieht mit den Attributen des Entitytyps, auf den T_1 angewendet wird, oder ist diese Regel nur auf Entitytypen anwendbar, für die noch keine Attribute festgelegt sind?
- Falls der Entitytyp aus dem Ausgangsschema an Relationshiptypen teilnimmt, mit welchem Entitytyp des Ergebnisschemas werden diese Relationshiptypen verbunden?
- Worauf ist bezüglich der statischen Integritätsbedingungen zu achten, insbesondere was wird aus eventuell auf dem Start-Entitytyp definierten Schlüsselbedingungen und welchen Beschränkungen unterliegt der neu entstandene Relationshiptyp?

Insbesondere der letzte Punkt weist auf eine gravierende Schwäche der gesamten Methodik hin. Die Transformationsregeln beschäftigen sich lediglich mit der Struktur der Daten, nicht jedoch mit den statischen Integritäten. Es wird im Buch lediglich darauf hingewiesen, daß Integritätsbedingungen ein Thema darstellen, auf das im Zusammenhang mit Schema-Transformationen geachtet werden muß. Mit der dort gegebenen Methodik ist es möglich, daß Entwicklungen im Bereich der statischen Integritätsbedingungen nicht monoton sind, das heißt daß einmal eingeführte Bedingungen wieder zurückgenommen werden. Die SPECTRUM-Methodik legt im Gegensatz dazu großen Wert darauf, daß die Anforderungen an ein System in Verlauf der Entwicklung monoton stärker werden.

Batini, Ceri und Navathe unterscheiden bei dem von ihnen angegebenen Regelsatz zwischen sogenannten *Top-Down-Regeln* und *Bottom-Up-Regeln*. Top-Down-Regeln dienen zur Verfeinerung von bereits im Schema vorhandenen Konzepten. Der Begriff der Verfeinerung eines Schemas ist nicht formal definiert. Er bezeichnet das Ersetzen von Konstrukten[3] eines Schemas durch andere, detailliertere Konstrukte, die das gleiche Konzept aus dem Anwendungsgebiet modellieren. Die Regel T_1 aus Abbildung 5.4 ist ein Beispiel für eine solche Top-Down-Regel. Sie ersetzt einen Entitytyp durch zwei Entitytypen, die durch einen Relationshiptyp verbunden sind. Bottom-Up-Regeln hingegen dienen der Einführung

[2] Selbstverständlich ist es in jeder Methode möglich, eine Entwicklung in einem früheren Zustand neu aufzunehmen, das heißt eine Revision vorzunehmen. Dies kann aber hier nur als Schritt außerhalb der Methode verstanden und begründet werden. Somit ist jede Entwicklung ohne Revision darstellbar.

[3] Der Begriff *Konstrukt* dient im weiteren als Oberbegriff für die im E/R-Modell vorhandenen Konzepte Entitytyp, Relationshiptyp und Attribut.

neuer Konzepte in ein Schema. Ein Beispiel hierfür ist die Regel B_2. Sie führt einen neuen Relationshiptyp zwischen zwei bereits vorhandenen Entitytypen ein. Die Menge der gegebenen Bottom-Up-Regeln ist bei Batini, Ceri und Navathe minimal und vollständig, während die Menge der Top-Down-Regeln diese Eigenschaften nicht aufweist. Minimalität bedeutet in diesem Zusammenhang, daß keine der Regeln mit Hilfe der anderen ausgedrückt werden kann. Vollständigkeit besagt, daß jedes beliebige E/R-Schema mit Hilfe der gegebenen Regeln erzeugbar ist.

Betrachtet man die Auswirkungen, die die Anwendung derartiger Transformationsregeln auf die statische Semantik eines E/R-Schemas hat, so läßt sich folgendes feststellen:

- Das Hinzufügen neuer Konzepte (Entitytypen, Relationshiptypen, Attribute) durch Bottom-Up-Regeln bewirkt auf der Ebene der statischen Semantik lediglich eine Erweiterung der Signatur. Diese Art von Regeln etabliert somit eine SPECTRUM-Realisierungsbeziehung.

- Top-Down-Regeln hingegen *ersetzen* Konzepte eines Schemas, das heißt zwischen den Signaturen der beteiligten Semantiken existiert keine Inklusionsbeziehung. Damit sind diese Regeln nicht mit dem SPECTRUM-Realisierungsbegriff verträglich.

Zusammenfassend erscheint es sinnvoll, für das in dieser Arbeit verwendete E/R-Modell einen Satz von Verfeinerungsregeln anzugeben, die nach Batini, Ceri und Navathe als Bottom-Up-Regeln bezeichnet werden können und die durch den auf der statischen Semantik vorhandenen SPECTRUM-Realisierungsbegriff formal gerechtfertigt sind.

Definition 3 Realisierung von E/R-Schemata
Zur Realisierung, das heißt Weiterentwicklung von E/R-Schemata sind folgende Schritte zugelassen:

(a) ***Umbenennen von Konstrukten des Schemas*** *Im Schema (auch in den Spezifikationen der Attributsorten) vorkommende Konstrukte dürfen umbenannt werden, solange die in Abschnitt 2.1 gegebenen Konventionen eingehalten werden.*

(b) ***Realisierung auf Attributebene*** *Für die Spezifikation einer Attributsorte darf eine Realisierung angegeben werden. Damit ist nicht nur Verfeinerung, sondern auch Datenstrukturwechsel zur Entwicklung von Attributsorten erlaubt. Die Realisierung von mengenwertigen Attributen durch listenwertige Attribute könnte damit wie in Abschnitt 2.2.2 exemplarisch angegeben erfolgen.*

(c) ***Verfeinerung der Struktur***

1. ***Hinzufügen eines Entitytyps*** *Es wird ein neuer, noch nicht im Schema vorhandener Entitytyp zum E/R-Diagramm hinzugenommen. Der so entstandene Entitytyp besitzt weder Attribute noch nimmt er an Beziehungen teil.*

2. ***Hinzufügen eines Relationshiptyps*** *Zwischen mehreren im Schema bereits vorhandenen Entitytypen wird ein neuer Relationshiptyp* `R` *eingeführt. Die Beteiligung der Entitytypen an* `R` *ist (0, ∗).*

3. ***Hinzufügen eines Attributs*** *Ein bereits im Schema vorhandener Entitytyp wird um ein zusätzliches, neues Attribut erweitert. Ist die Sorte des Attributs noch nicht spezifiziert, muß in diesem Schritt auch eine Spezifikation dieser Sorte als primitive Sorte angegeben werden. Das Attribut kann sowohl optional als auch zwingend sein.*

(d) ***Verfeinerung der statischen Integritätsbedingungen*** *Die im Schema vorhandene Menge von statischen Integritätsbedingungen Ax_{Int} wird durch eine stärkere Menge Ax'_{Int} ersetzt:*

$$Ax'_{Int} \blacktriangleright Ax_{Int}$$

Dies schließt insbesondere das Hinzufügen und Verschärfen von Schlüsselbedingungen mit ein, sowie das Verschärfen der Anforderungen an die Beteiligung von Entitytypen an Beziehungen, also die mit Hilfe des klassischen E/R-Modells graphisch darstellbaren Integritätsbedingungen.

□

Die konzeptuelle Datenmodellierung beginnt also bei diesem Realisierungsbegriff immer mit dem leeren Schema und produziert das endgültige konzeptuelle Schema durch iterative Anwendung der oben angegebenen Entwicklungsschritte.

Auswirkungen auf die statische Semantik

Die obige Definition des Begriffs der Realisierung von E/R-Schemata wurde gegeben in der Absicht, einen mit der Technik der formalen Softwareentwicklung in SPECTRUM verträglichen Entwicklungsbegriff zu definieren. Der folgende Satz zeigt, daß dieses Ziel auf der Ebene der statischen Semantik, die in Kapitel 3 als formale Semantik des E/R-Modells gegeben wurde, erreicht wurde.

Satz 5 *Stehen zwei E/R-Schemata in einer Realisierungsbeziehung gemäß Definition 3, so besteht zwischen den zugehörigen statischen Semantiken eine* SPECTRUM*-Realisierungsbeziehung (siehe Abschnitt 2.2.2).*

Beweis *Für diese relativ offensichtliche Aussage wird kein formaler Beweis angegeben. Die Gültigkeit der Aussage wird stattdessen in informeller Weise motiviert. Dazu werden die unterschiedlichen möglichen Entwicklungsschritte gemäß Definition 3 betrachtet:*

(a) ***Umbenennen von Konstrukten des Schemas*** *Dieser Schritt bewirkt ein Umbenennen von Elementen der Signatur der statischen Semantik. Obwohl offensichtlich ist, daß auf diesem Weg eine äquivalente Spezifikation entsteht, erfüllt diese Umbenennung nicht die Bedingungen für das Vorliegen einer* SPECTRUM*-Verfeinerung*[4]*. Jedoch kann der Schritt durch einen trivialen Datenstrukturwechsel realisiert werden, in dem alle Funktionen durch identisch axiomatisierte, aber anders benannte Funktionen realisiert werden.*

(b) ***Realisierung auf Attributebene*** *Besteht eine Realisierung auf Attributebene, so unterscheiden sich die beiden statischen Semantiken lediglich in einer primitiven Spezifikation, wobei zwischen den beiden Versionen der primitiven Spezifikation eine Realisierung besteht. Diese Realisierungsbeziehung setzt sich damit auch auf die statischen Semantiken fort.*

(c) ***Verfeinerung der Struktur*** *Hinzufügen eines Attributs beziehungsweise eines Entity- oder Relationshiptyps bedeutet auf der Ebene der statischen Semantik eine Erweiterung der Signatur. Die Axiome werden durch einen derartigen Entwicklungsschritt nicht verändert. Ein solcher Schritt etabliert also eine Verfeinerungsbeziehung zwischen den beiden statischen Semantiken.*

(d) ***Verfeinerung der statischen Integritätsbedingungen*** *Die Anwendung dieses Entwicklungsschritts auf die statische Semantik eines E/R-Schemas läßt die Signatur unverändert und ersetzt die Axiome, die die statischen Integritätsbedingungen beschreiben, durch eine Menge logisch stärkerer Axiome. Damit besteht zwischen den beiden statischen Semantiken eine Verfeinerungsbeziehung.*

□

Auswirkungen auf die Zugriffsspezifikation

Der in Kapitel 4 gegebene Ansatz erlaubt es, ein E/R-Schema als formale Beschreibung des statischen Anteils eines Informationssystems zu sehen und die Entwicklung der Systemfunktionen darauf basierend in SPECTRUM vorzunehmen. Die Gesamtspezifikation des Systems besteht also aus einem E/R-Schema, das möglicherweise um SPECTRUM-Formeln zur Beschreibung allgemeinerer Integritätsbedingungen angereichert ist, und der SPECTRUM-Spezifikationen der Systemfunktionen. Semantisch kann diese zusammengesetzte Spezifikation als homogene SPECTRUM-Spezifikation verstanden werden, da dem E/R-Schema mittels Übersetzungsvorschrift eine SPECTRUM-Spezifikation, die Zugriffsspezifikation, zugeordnet ist, die die modellierten Daten in Form einer 'E/R-Datenbank' beschreibt. Aus

[4] Dies ist eine technische Einschränkung, die aus der sehr restriktiven Definition des Verfeinerungsbegriffs in SPECTRUM folgt. Wie der Definition in Abschnitt 2.2.2 entnommen werden kann, läßt der in SPECTRUM definierte Verfeinerungsbegriff keine Umbenennung von Signaturelementen zu.

diesem Grund sollte die Weiterentwicklung der Gesamtspezifikation konform zur SPECTRUM-Methodik (Abschnitt 2.2.2) erfolgen. Dies ist sicher der Fall, wenn das Datenschema während der Entwicklung der Applikation unverändert bleibt, wenn also lediglich die Systemfunktionen in SPECTRUM entwickelt werden. Es wurde jedoch nicht untersucht, in welcher Weise das E/R-Datenschema weiterentwickelt werden darf, damit die gewünschte Verträglichkeit mit dem SPECTRUM-Realisierungsbegriff gewahrt bleibt. Im folgenden wird nun untersucht, wie sich die Anwendung der in Definition 3 gegebenen Entwicklungsschritte auf der Ebene der Zugriffsspezifikation auswirkt, das heißt ob und wie diese Entwicklungsschritte im Rahmen des in Kapitel 4 beschriebenen Ansatzes verwendet werden können.

Der wesentliche Unterschied zwischen der Zugriffsspezifikation und der statischen Semantik ist die Verwendung der Datenbanksorte `Db` zur Modellierung der Daten. Diese ist bereits in der internalisierten Semantik vorhanden. Die Zugriffsspezifikation stellt lediglich eine Weiterentwicklung der internalisierten Semantik im Sinne des SPECTRUM-Realisierungsbegriffs dar. Im weiteren wird deshalb statt der Zugriffsspezifikation die (einfachere) internalisierten Semantik untersucht. Die Auswirkungen von E/R-Realisierungsschritten werden im folgenden Satz zusammengefaßt:

Satz 6 *Alle in Definition 3 gegebenen E/R-Realisierungsschritte außer dem Schritt "Verfeinerung der statischen Integritätsbedingungen" entsprechen auf der Ebene der internalisierten Semantik* SPECTRUM*-Realisierungsschritten.*

Beweis *Wie im Beweis von Satz 5 werden auch hier die möglichen Entwicklungsschritte separat betrachtet.*

(a) ***Umbenennen von Konstrukten des Schemas*** *Mit der gleichen Argumentation wie bei Satz 5 wird durch diesen Schritt eine Realisierung (in Form eines Wechsels der Datenstruktur) etabliert.*

(b) ***Realisierung auf Attributebene*** *Die Realisierung einer primitiven Attributspezifikation setzt sich wie bei der statischen Semantik auf die Zugriffsspezifikation fort.*

(c) ***Verfeinerung der Struktur***

1. ***Hinzufügen eines Entitytyps*** *Die Hinzunahme eines neuen Entitytyps bedeutet auf der Ebene der statischen Semantik lediglich die Erweiterung der Signatur um eine neue Entitysorte. Auf der Ebene der internalisierten Semantik erhält außerdem der Konstruktor* `mkdb` *der Datenbanksorte einen zusätzlichen Parameter. Damit kann dieser Entwicklungsschritt auf der Ebene der internalisierten Semantik nicht als Verfeinerung beschrieben werden. Er wird stattdessen als Wechsel der Datenstruktur verstanden, bei dem die ursprüngliche Datenbanksorte* $\texttt{Db}^{alt}$ *durch eine neue Datenbanksorte* $\texttt{Db}^{neu}$ *realisiert wird. Eine einfache Möglichkeit, die Elemente von* $\texttt{Db}^{alt}$ *durch* $\texttt{Db}^{neu}$*-Elemente zu repräsentieren, ist gegeben durch:*

i) Jedes $\mathtt{Db}^{neu}$*-Element repräsentiert ein* $\mathtt{Db}^{alt}$*-Element.*

ii) Zwei $\mathtt{Db}^{neu}$*-Elemente, die sich nur in der den neuen Entitytyp repräsentierenden Menge unterscheiden, stehen für dasselbe* $\mathtt{Db}^{alt}$*-Element.*

Sei S^{alt} *ein E/R-Schema, das die Entitytypen* $E_1, \ldots, E_n$ *sowie die Relationshiptypen* $R_1, \ldots, R_m$ *enthält.* S^{neu} *entstehe aus* S^{alt} *durch Hinzufügen des neuen Entitytyps* E*. Dann ist der Wechsel der Datenstruktur von* $\mathtt{Db}^{alt}$ *zu* $\mathtt{Db}^{neu}$ *gegeben durch (siehe Abschnitt 2.2.2):*

- *Die Konstruktor- und Selektorfunktionen von* $\mathtt{Db}^{alt}$ *(*$\mathtt{mkdb}^{alt}$, $\mathtt{ent'}E_i^{alt}$, R_j^{alt}*) werden durch die entsprechenden Funktionen auf* $\mathtt{Db}^{neu}$ *(*$\mathtt{mkdb}^{neu}$, $\mathtt{ent'}E_i^{neu}$, R_j^{neu}*) realisiert.*
- *Das Repräsentationsprädikat wird definiert als*

  $\mathtt{is\_Db}^{alt}\mathtt{: Db}^{neu} \rightarrow \mathtt{Bool;}$

  $\mathtt{axioms}$
  $\forall \mathtt{db}^{neu}\mathtt{: Db}^{neu}\mathtt{.\ is\_Db}^{alt}(\mathtt{db}^{neu}) = \mathtt{true;}$
  $\mathtt{endaxioms;}$
- *Die Kongruenzrelation* $.\sim.$ *ist gegeben durch*

  $.\sim. : \mathtt{Db}^{neu} \times \mathtt{Db}^{neu} \rightarrow \mathtt{Bool;}$

  $\mathtt{axioms}$
  $$\forall \mathtt{db}_1^{neu}, \mathtt{db}_2^{neu}\mathtt{: Db}^{neu}\mathtt{.\ db}_1^{neu} \sim \mathtt{db}_2^{neu} \Leftrightarrow \begin{array}{l} \mathtt{ent'}E_1^{neu}(\mathtt{db}_1^{neu}) = \mathtt{ent'}E_1^{neu}(\mathtt{db}_2^{neu}) \wedge \\ \qquad\qquad \vdots \\ \mathtt{ent'}E_n^{neu}(\mathtt{db}_1^{neu}) = \mathtt{ent'}E_n^{neu}(\mathtt{db}_2^{neu}) \wedge \\ R_1^{neu}(\mathtt{db}_1^{neu}) = R_1^{neu}(\mathtt{db}_2^{neu}) \wedge \\ \qquad\qquad \vdots \\ R_m^{neu}(\mathtt{db}_1^{neu}) = R_m^{neu}(\mathtt{db}_2^{neu}); \end{array}$$
  $\mathtt{endaxioms;}$

Es ist leicht einzusehen, daß durch diese Definitionen ein Datenstrukturwechsel von $\mathtt{Db}^{alt}$ *nach* $\mathtt{Db}^{neu}$ *gegeben ist, das heißt daß die in Abschnitt 2.2.2 für einen derartigen Entwicklungsschritt angegebenen Beweisverpflichtungen gelten. Auf einen formalen Beweis dieses Umstands wird hier verzichtet.*

2. ***Hinzufügen eines Relationshiptyps*** *Auch die Hinzunahme eines neuen Relationshiptyps erweitert den Konstruktor der Datenbanksorte um einen Parameter. Eine Realisierung auf der Ebene der internalisierten Semantik ist somit durch den gleichen Datenstrukturwechsel gegeben, der bereits beim Hinzufügen eines Entitytyps angewendet wurde.*

3. ***Hinzufügen eines Attributs*** *Die Hinzunahme eines neuen Attributs bedeutet wie bei der statischen Semantik Erweiterung der Spezifikation um eine Selektorfunktion. Dadurch wird also wieder eine Verfeinerungsbeziehung zwischen den internalisierten Semantiken etabliert.*

(d) ***Verfeinerung der statischen Integritätsbedingungen*** *Verfeinern der statischen Integritätsbedingungen, das heißt Ersetzen der Formelmenge Ax_{int}^{alt}, die die statischen Integritätsbedingungen eines Schemas repräsentiert, durch eine neue, logisch stärkere Formelmenge Ax_{int}^{neu}, ändert die Axiomatisierung des* `OK`*-Prädikats. Die aus Ax_{int}^{alt} beziehungsweise Ax_{int}^{neu} gewonnene Formel zur Spezifikation von* `OK` *wird im folgenden mit ax_{int}^{alt}[db] beziehungsweise ax_{int}^{neu}[db] bezeichnet*[5]*. Ist die Menge Ax_{int}^{neu} logisch echt stärker als Ax_{int}^{alt}, so setzt sich diese Eigenschaft auf ax_{int}^{neu}[db] und ax_{int}^{alt}[db] fort, das heißt es gilt*

(i) $\quad \forall$db:Db. ax_{int}^{neu}[db] $\Rightarrow$ ax_{int}^{alt}[db]
(ii) $\quad \exists$db:Db. ax_{int}^{alt}[db] $\wedge$ $\neg(ax_{int}^{neu}$[db]$)$

Soll diese Verschärfung der statischen Integritätsbedingungen mit dem SPECTRUM*-Realisierungsbegriff verträglich sein, so muß sich nachweisen lassen, daß die alte Axiomatisierung des* `OK`*-Prädikats logisch aus der neuen folgt. Mit Hilfe der oben angegebenen Eigenschaften (i) und (ii) muß also aus*

(iii) $\quad \forall$db:Db. OK(db) $=$ ax_{int}^{neu}[db]

die Formel

$\forall$db:Db. OK(db) $=$ ax_{int}^{alt}[db]

hergeleitet werden können. Einfache prädikatenlogische Umformungen zeigen, daß aus (iii) und (i) die Formel

$\forall$db:Db. OK(db) $\Rightarrow$ ax_{int}^{alt}[db]

folgt. Auf der anderen Seite folgt aus (iii) und (ii)

$\neg\forall$db:Db. ax_{int}^{alt}[db] $\Rightarrow$ OK(db)

Damit ist offensichtlich, daß $\forall$db:Db. OK(db) $=$ ax_{int}^{alt}[db] *nicht hergeleitet werden kann. Die Verstärkung der statischen Integritätsbedingungen etabliert also keine Realisierungsbeziehung im Sinne der formalen Spezifikationstechnik auf der Ebene der internalisierten Semantik.*

□

[5] Diese Formel enthält eine freie Variable `db` der Datenbanksorte `Db`. Diese Variable wird in der internalisierten Semantik in dem Axiom $\forall$db:Db.OK(db)$=ax_{int}$[db], das das `OK`-Prädikat spezifiziert, gebunden.

Wie oben gezeigt, ist die Verfeinerung der statischen Integritätsbedingungen auf der Ebene der internalisierten Semantik nicht mit dem SPECTRUM-Realisierungsbegriff verträglich. Dennoch können die Auswirkungen dieses Schritts auf eine Applikation beschrieben werden, die sich auf die internalisierte Semantik stützt: Die Transaktionen der Applikation wurden im Rahmen der formalen Entwicklung so spezifiziert und implementiert, daß sie die statische Integrität der Datenbank invariant lassen. Werden nun die statischen Integritätsbedingungen verschärft, so ist nicht zu erwarten, daß die Systemfunktionen auch diese stärkeren Bedingungen erhalten. Gelingt es jedoch nachzuweisen, daß die Systemfunktionen auch die neuen, stärkeren Integritätsbedingungen invariant lassen, ist die Verstärkung der statischen Integritätsbedingungen methodisch wieder gerechtfertigt.

Es ist deshalb möglich, in diesem Fall den SPECTRUM-Realisierungsbegriff in folgender Weise zu erweitern:

> Neben den in Abschnitt 2.2.2 beschriebenen Schritten der Verfeinerung und des Wechsels der Datenstruktur, ist es erlaubt, die im Datenschema gegebenen Integritätsbedingungen zu verschärfen, wenn alle aus dem `OK`-Prädikat generierten Verifikationsbedingungen für das neue, stärkere $\texttt{OK}^{neu}$-Prädikat neu nachgewiesen werden.

Mit Hilfe der zuletzt gegebenen Erweiterung des Realisierungsbegriffs ist es gelungen, die in Definition 3 gegebene Methodik zur Entwicklung von E/R-Schemata für den Ansatz aus Kapitel 4 zugänglich zu machen. Damit ist dieser Ansatz nicht nur methodisch untermauert, die gegebene Methodik ist auch vollständig in dem Sinn, daß jedes beliebige E/R-Schema mit seiner Hilfe erzeugt werden kann.

Wie oben geschildert, stellen die in Definition 3 gegebenen Entwicklungsschritte in der Terminologie von Batini, Ceri und Navathe sogenannte Bottom-Up-Schritte dar, das heißt sie erlauben das Hinzufügen von Information (strukturelle Information und Integritätsbedingungen) zu einem E/R-Schema. Sogenannte Top-Down-Entwicklungsschritte, die bereits im Schema vorhandene Konzepte durch verfeinerte Konzepte ersetzen, sind mit dem Begriff der Realisierung in SPECTRUM (auch in der oben angegebenen erweiterten Form) nicht verträglich, da sie Elemente aus der Signatur der statischen Semantik (Sorten beziehungsweise Funktionen) entfernen.

Es ist klar, daß die Beschränkung auf reine Bottom-Up-Schritte im Vergleich zu bei der Datenmodellierung herkömmlichen Vorgehensweisen eine erhebliche Einschränkung an Flexibilität bedeutet. Aus diesem Grund wird die abschließende Diskussion dieses Kapitels (Abschnitt 5.3) in einem Ausblick eine Möglichkeit schildern, wie Top-Down-Entwicklung, aufbauend auf in Abschnitt 5.2 entwickelte Techniken, im Rahmen des gegebenen Ansatzes ermöglicht werden kann.

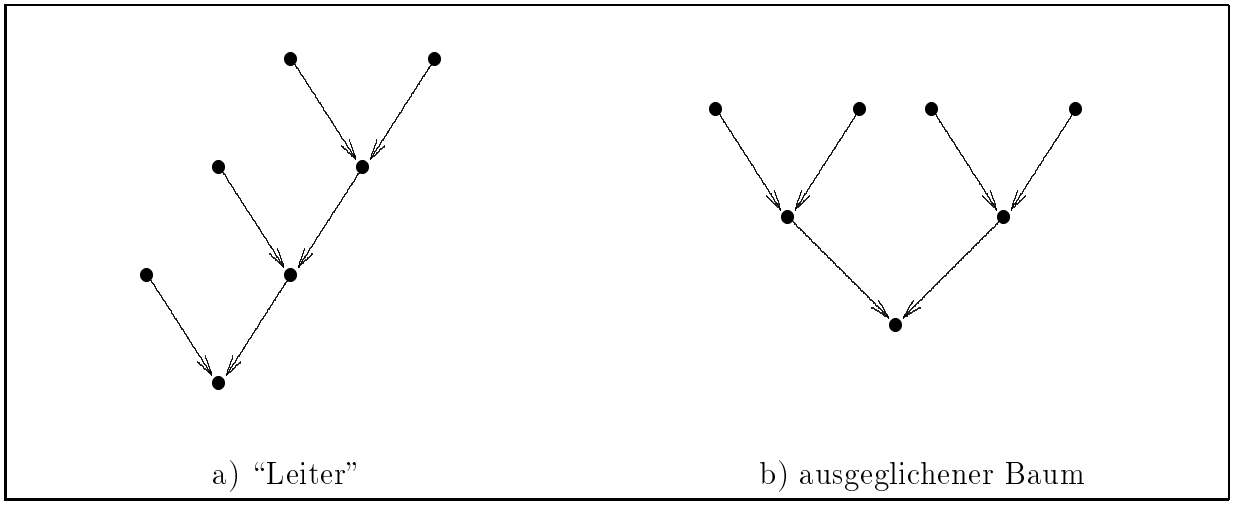


Abbildung 5.5: Verschiedene Integrations-Strategien

5.2 View Integration

Konzeptuelle Schemata entstehen nicht immer in der in Abbildung 5.2 angedeuteten linearen Weise, in der ein Schema in evolutionärer Weise weiterentwickelt wird, unabhängig davon, ob die angewendeten Entwicklungsschritte formal oder informell definiert sind. Oft werden mehrere verschiedene, unterschiedliche Aspekte des Anwendungsbereichs beleuchtende Datenschemata zu einem einzigen konzeptuellen Schema verschmolzen (Abbildung 5.3). Dies kann unterschiedliche Gründe haben, wie zum Beispiel die Notwendigkeit, bereits bestehende (Teil-)Datenschemata in das Schema eines größeren organisatorischen Kontextes einzubinden. Auch wenn diese Notwendigkeit nicht besteht, kann es methodisch vorteilhaft sein, mehrere Teil-Datenschemata des Anwendungsbereichs zu erstellen und diese dann zum konzeptuellen Schema zu integrieren, insbesondere wenn der Anwendungsbereich organisatorisch klar in Unter-Einheiten gegliedert ist, die ihre Daten weitgehend lokal verwalten und untereinander schmale Schnittstellen besitzen. Die Daten dieser Unter-Einheiten können auf diese Weise unabhängig voneinander im Team modelliert werden.

Die Aufgabe, unterschiedliche Schemata zu einem konsistenten Schema eines Anwendungsbereichs zu integrieren, ist im allgemeinen sehr komplex und nicht-trivial. Der Vorgang wird meist als *View Integration* bezeichnet (siehe zum Beispiel [BLN86]). Eine Methode zur Unterstützung der View Integration muß sich im wesentlichen mit zwei Problemen befassen:

Integrations-Strategie Es muß eine Vorgehensweise angegeben werden, nach der die Integration von n verschiedenen Schemata durchgeführt werden kann. Eine Möglichkeit ist es zum Beispiel, alle n Schemata in einem einzigen Schritt zu integrieren. Eine andere Möglichkeit besteht darin, schrittweise vorzugehen und in jedem Schritt nur jeweils zwei Schemata zu verschmelzen.

Abbildung 5.5 stellt zwei mögliche Varianten dieser Vorgehensweise dar. Selbstverständlich sind viele weitere Möglichkeiten zur Integration von n Datenschemata denkbar.

Integrations-Technik Die Integration verschiedener Datenschemata ist ein komplexer Vorgang, der sowohl das Identifizieren gemeinsamer Teile als auch das Erkennen und Auflösen von Konflikten und Inkompatibilitäten zwischen den Schemata erfordert. Es müssen Techniken bereitgestellt werden, die die Komplexität dieses Vorgangs beherrschbar machen und so die Anzahl der bei der Integration entstehenden Fehler gering halten.

Die vorliegende Arbeit beschäftigt sich im wesentlichen mit dem zweiten Punkt. Unter der Annahme, daß ein Integrationsschritt jeweils zwei unterschiedliche Schemata zusammenführt, wird gezeigt, wie sich die Integration mit der Definition der statischen Semantik verträglich durchführen läßt.

Integration zweier E/R-Schemata

Im folgenden wird eine Methode zur Integration zweier E/R-Schemata angegeben, die sich mit der statischen Semantik und insbesondere dem in Definition 3 gegebenen Realisierungsbegriff für E/R-Schemata verträgt. Sie besteht im wesentlichen aus drei Schritten:

1. **Herstellen der Verträglichkeit zwischen den Schemata**
   Die zu integrierenden Schemata werden durch Umbenennung miteinander verträglich gemacht. Der Begriff der Verträglichkeit wird weiter unten genauer erläutert. Dies ist Voraussetzung für die folgende Vereinigung der Schemata.

2. **Vereinigen der Schemata**
   Die Schemata werden zu einem gemeinsamen Schema vereinigt.

3. **Hinzufügen von Interschema-Eigenschaften**
   Beziehungen, die zwischen Elementen der unterschiedlichen Ausgangsschemata bestehen, werden zum vereinigten Schema hinzugefügt.

Diese drei Schritte kommen in der einen oder anderen Form in nahezu allen Methoden zur View Integration vor (siehe [BLN86]). Ihre spezielle Ausprägung im in dieser Arbeit gegebenen Ansatz wird im folgenden detailliert vorgestellt. Die einzelnen Schritte werden dabei fortlaufend anhand des folgenden Beispiels erläutert.

Beispiel 11 *Es soll mittels View Integration ein konzeptuelles Datenschema für die bereits bekannte Applikation RECHERCHE erstellt werden (siehe Beispiel 1). Die Daten des Anwendungsbereichs von RECHERCHE sind also zu analysieren und in Form eines E/R-Schemas festzuhalten.*

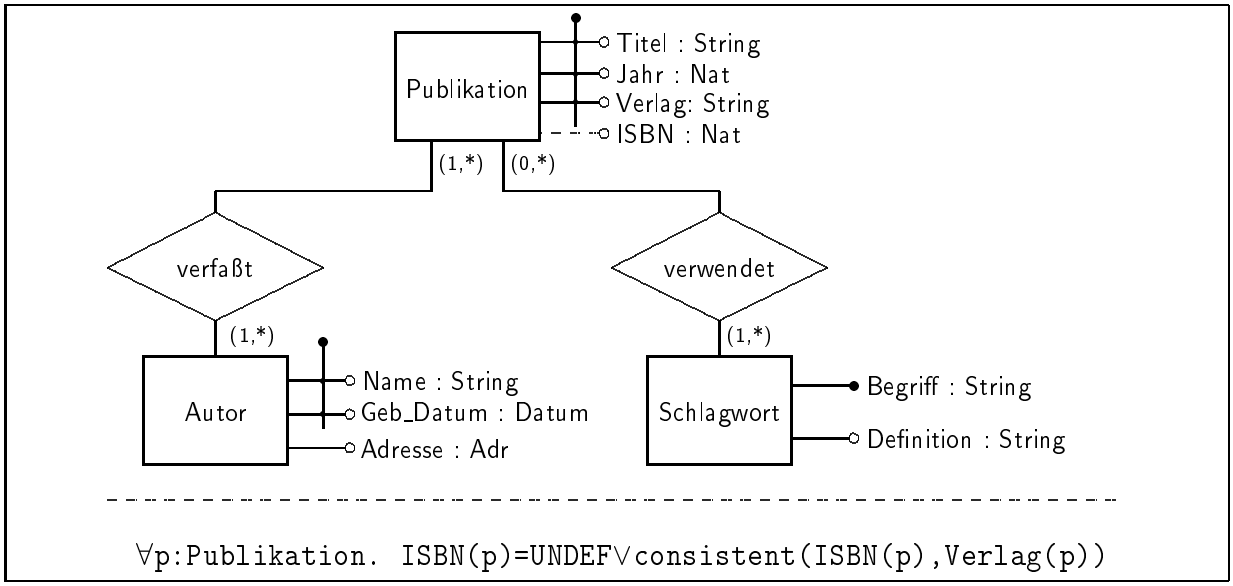


Abbildung 5.6: Erstes zu integrierendes Schema S_1

Die Abbildungen 5.6 und 5.7 zeigen zwei Schemata, die im Zuge der Analyse des Problembereichs entstanden sind. Wie man sieht, enthält jedes der beiden Schemata Beschreibungen von Daten des Problembereichs, die im jeweils anderen Schema nicht erfaßt sind. So enthält S_1 Informationen über die Autoren der gespeicherten Publikationen, während diese Information in das Schema S_2 nicht aufgenommen wurden. Andererseits erlaubt es Schema S_2 durch den Relationshiptyp zitiert*, Informationen darüber zu speichern, welche Publikationen sich gegenseitig zitieren. Diese Information kann in einer gemäß Schema S_1 entwickelten Datenbank nicht untergebracht werden. Ziel des Beispiels ist es daher, die beiden Schemata zu einem einzigen konzeptuellen Schema von RECHERCHE zu integrieren.*

Ebenso ist leicht zu erkennen, daß bei der Integration Konflikte aufzulösen sind. So ist zum Beispiel die Information über den Verlag, der eine Publikation herausgibt, in S_1 in Form eines Attributs enthalten, während S_2 diese Information mit Hilfe des Entitytyps Verlag *und des Relationshiptyps* gibt_heraus *modelliert wird. Auf diesen und alle weiteren zwischen den Schemata bestehenden Konflikte wird im Verlauf des Beispiels zu achten sein.* □

Schritt 1: Herstellen der Verträglichkeit

Die Vereinigung von E/R-Schemata wird in Schritt 2 durch die Vereinigung ihrer statischen Semantiken definiert werden. Die Vereinigung von (syntaktisch korrekten) Spezifikationen liefert in SPECTRUM nicht notwendigerweise wieder eine syntaktisch korrekte Spezifikation. Wird ein Bezeichner in zwei Spezifikationen verwendet, so werden die durch diese beiden Anwendungen des Bezeichners bezeichneten semantischen Objekte identifiziert. Hat der Bezeichner jedoch in den beiden Spezifikationen unterschiedliche Sorten, so entsteht bei

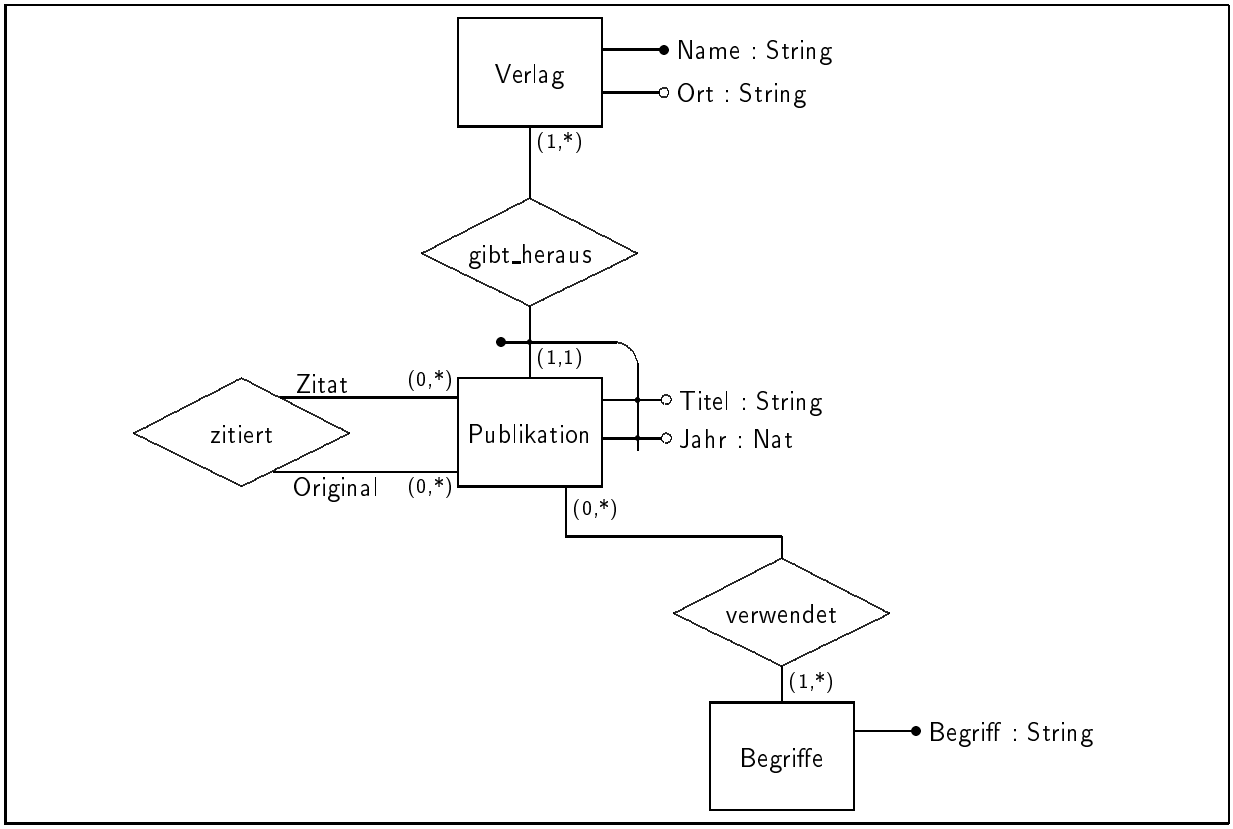


Abbildung 5.7: Zweites zu integrierendes Schema S_2

der Vereinigung ein Sortenkonflikt, der diese Identifizierung unmöglich macht. Da dem Bezeichner in der vereinigten Spezifikation keine eindeutige Sorte zugeordnet werden kann, ist diese Spezifikation syntaktisch nicht korrekt.

Vor der Vereinigung der beiden E/R-Schemata muß also sichergestellt werden, daß die zugehörigen statischen Semantiken syntaktisch verträglich sind, so daß sie ohne Probleme vereinigt werden können. Es ist also zum Beispiel nicht möglich, daß ein Bezeichner in einem Schema einen Entitytyp und im anderen einen Relationshiptyp bezeichnet. Es ist klar, daß diese Art der Verträglichkeit durch geeignete Umbenennung von Konstrukten der Schemata erreicht werden kann.

Neben dieser syntaktischen Form der Verträglichkeit zweier Schemata ist auch die Bedeutung der in den Schemata verwendeten Konstrukte abzugleichen. Enthalten die beiden Schemata gleich bezeichnete syntaktisch verträgliche Teile, die aber nicht die gleiche Information der realen Welt modellieren, so ist es nicht sinnvoll, diese Teile bei der Vereinigung der Schemata zu identifizieren. Hier ist also durch geeignetes Umbenennen der Konstrukte dafür zu sorgen, daß die Repräsentation unterschiedlicher Konzepte des Anwendungsgebiets in den Schemata auch unterschiedlich bezeichnet wird.

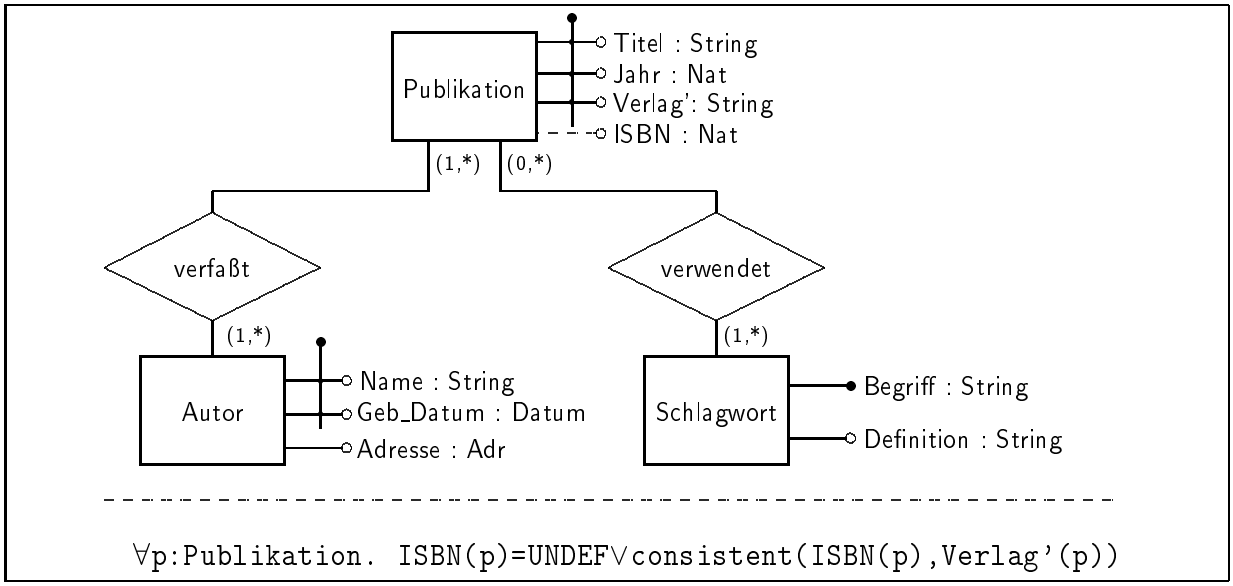


Abbildung 5.8: Schema S_1 nach Schritt 1

Umgekehrt ist es möglich, daß der gleiche Aspekt der realen Welt in beiden Schemata in syntaktisch verträglicher Weise modelliert, jedoch mit unterschiedlichen Bezeichnungen versehen wird. Hier sollte wiederum durch Umbenennen dafür gesorgt werden, daß die Bezeichner in diesem Fall vereinheitlicht werden.

Beispiel 12 *Die in Abb. 5.6 und 5.7 gegebenen Schemata sind syntaktisch unverträglich, da das Attribut* Name *in Schema S_1 ein Attribut des Entitytyps* Autor *ist, in S_2 jedoch Attribut des Entitytyps* Verlag. *Diese syntaktische Unverträglichkeit weist hier auf eine vorliegende semantische Unverträglichkeit hin, da* Name *in den beiden Schemata wirklich unterschiedliche Daten des Anwendungsgebiets modelliert. Der Konflikt wird gelöst, indem das Attribut* Name *des Entitytyps* Verlag *in S_2 in* V_name *umbenannt wird.*

Eine weitere syntaktische Unverträglichkeit besteht zwischen dem Entitytyp Verlag *des Schemas S_2 und dem in S_1 verwendeten Attribut* Verlag *des Entitytyps* Publikation. *Dieser Konflikt wird aufgelöst durch Umbenennen des Attributs in* Verlag'. *Dieser Schritt ist verglichen mit anderen Ansätzen zur View Integration ungewöhnlich, da der Bezeichner* Verlag *in beiden Schemata zur Modellierung des gleichen Konzepts der realen Welt verwendet wird. Techniken, wie sie in [BLN86] vorgestellt werden, würden in diesem Fall versuchen, das Attribut* Verlag *in Schema S_1 in einen Entitytyp weiterzuentwickeln und so die syntaktische Unverträglichkeit zu beseitigen. Auf diese Weise wäre der Entitytyp* Verlag *in beiden Schemata enthalten und könnte bei der Vereinigung der Schemata identifiziert werden. Mit dem in Definition 3 gegebenen Realisierungsbegriff für E/R-Schemata ist ein derartiger Top-Down Entwicklungsschritt jedoch nicht durchführbar. Aus diesem Grund wird die syntaktische Unverträglichkeit durch einfaches Umbenennen beseitigt. Die an dieser Stel-*

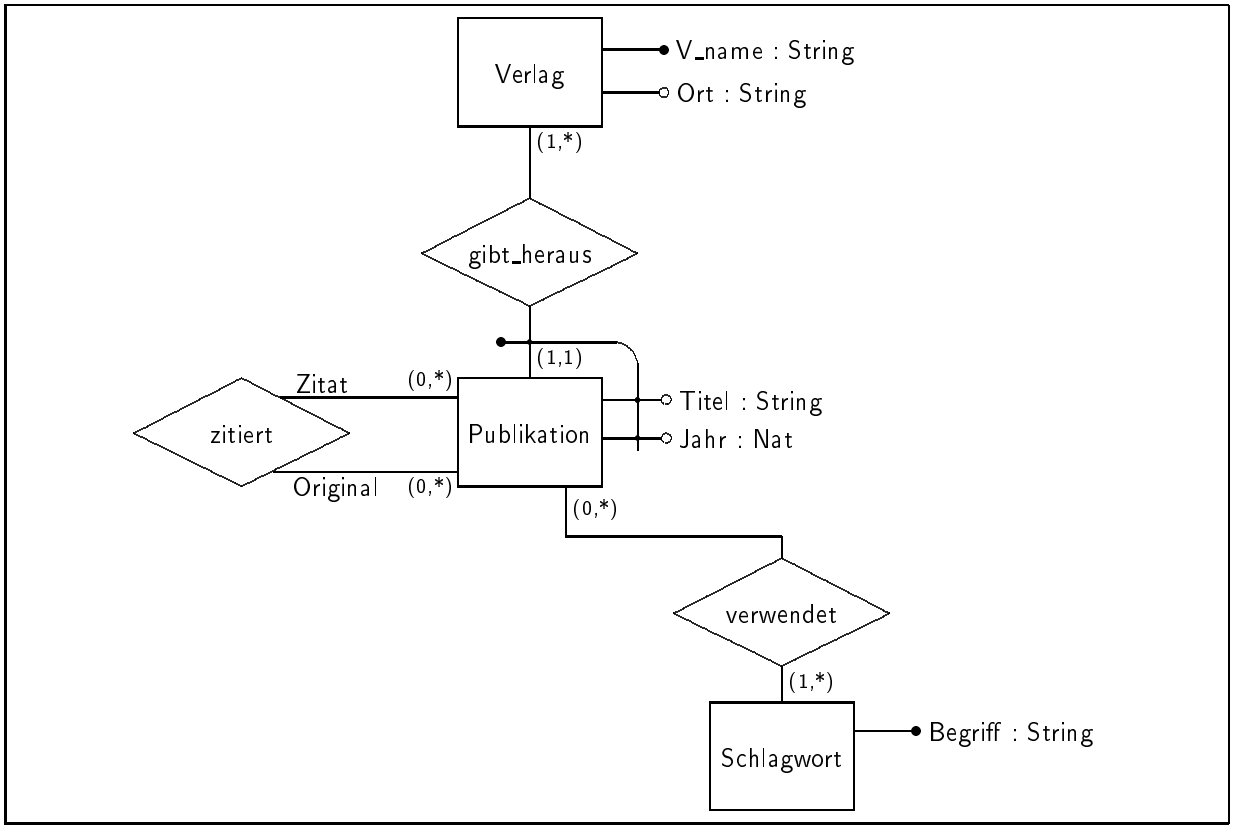


Abbildung 5.9: Schema S_2 nach Schritt 1

le ignorierte Information, daß die beiden Konstrukte denselben Aspekt der Wirklichkeit modellieren, wird in Schritt 3 in das integrierte Datenschema eingeführt.

Neben diesen syntaktischen Unverträglichkeiten kann festgestellt werden, daß die Entitytypen Schlagwort *und* Begriffe *das gleiche Konzept der realen Welt modellieren. Dabei entspricht das Attribut* Begriff *des Entitytyps* Schlagwort *dem Attribut* Begriff *des Entitytyps* Begriffe. *Um diese semantische Übereinstimmung zum Ausdruck zu bringen, wird* Begriffe *in S_2 in* Schlagwort *umbenannt.*

Nach diesen Umbenennungen sind die beiden Schemata verträglich. Damit ist Schritt 1 abgeschlossen. Das Ergebnis dieses Schritts ist in Abb. 5.8 und 5.9 dargestellt. □

Schritt 2: Vereinigen der Schemata

Nachdem die beiden Schemata verträglich gemacht wurden, können sie durch einfache Vereinigung zusammengeführt werden. Dabei werden gleich bezeichnete Teile überlagert, das heißt miteinander identifiziert. Diese Vereinigung der Schemata kann auf der Ebene der statischen Semantik offensichtlich als Vereinigung der statischen Semantiken der beiden

Schemata erklärt werden.

Da die Ausgangsschemata syntaktisch verträglich sind, ist die Vereinigung ihrer statischen Semantiken syntaktisch unproblematisch. Semantisch ist jedoch dabei darauf zu achten, daß die Vereinigung der statischen Integritätsbedingungen erfüllbar ist, das heißt daß die integrierte Spezifikation konsistent ist.

Enthalten die Schemata ausschließlich graphisch definierte Integritätsbedingungen, so ist es relativ leicht, sich über die Konsistenz des integrierten Schemas klarzuwerden:

Schlüsselbedingungen Die Vereinigung mehrerer Schlüsselbedingungen für einen Entitytyp kann keine Inkonsistenz hervorrufen. Man kann sich leicht überlegen, daß der Datenbankzustand, der genau eine Entity des Entitytyps enthält, niemals eine Schlüsselbedingung verletzen kann. Er stellt somit immer ein Modell dar.

Kardinalitäten Kardinalitäten beschreiben die Beteiligung von Entitytypen an Relationshiptypen in Form (min,max)-Intervallen. Werden nun Kardinalitäten für eine solche Beteiligung eines Entitytyps an einem Relationshiptyp vereinigt, so entsteht eine Inkonsistenz genau dann, wenn sich die durch die Kardinalitäten gegebenen Intervalle nicht überlappen. Verlangt zum Beispiel eine Kardinalität eine (1,1)-Beteiligung eines Entitytyps an einem Relationshiptyp und eine zweite Kardinalität eine (2,*)-Beteiligung, so ist dies selbstverständlich nicht erfüllbar, das heißt das integrierte Schema ist inkonsistent.

Enthalten die Schemata allgemeinere Integritätsbedingungen in Form von SPECTRUM-Formeln, so lassen sich keine so einfachen Kriterien zur Überprüfung der Konsistenz angeben. Hier muß wieder auf Standardtechniken wie die explizite Konstruktion eines Modells zurückgegriffen werden.

Beispiel 13 *Abbildung 5.10 zeigt das Ergebnis der Überlagerung der in Abb. 5.8 und 5.9 dargestellten Teilschemata.* □

Schritt 3: Hinzufügen von Interschema-Eigenschaften

Nach der Vereinigung enthält das integrierte Schema alle Konstrukte seiner Teilschemata. Somit können alle Beziehungen, die zwischen Konstrukten der Ausgangsschemata bestehen, als zusätzliche Integritätsbedingungen in dieses Schema mit aufgenommen werden. Dazu zählt insbesondere die Eigenschaft, daß zwei Konstrukte den gleichen Teil der Realität modellieren. Solche Eigenschaften, die Beziehungen beschreiben, die zwischen unterschiedlichen Ausgangsschemata bestehen, werden oft als *Interschema-Eigenschaften* oder *Zwischenschema-Eigenschaften*bezeichnet.

Beispiel 14 *In Schritt 1 wurde bereits festgestellt, daß der Entitytyp* Verlag *und das Attribut* Verlag' *zur Modellierung desselben Konzepts der Wirklichkeit eingesetzt werden. Der*

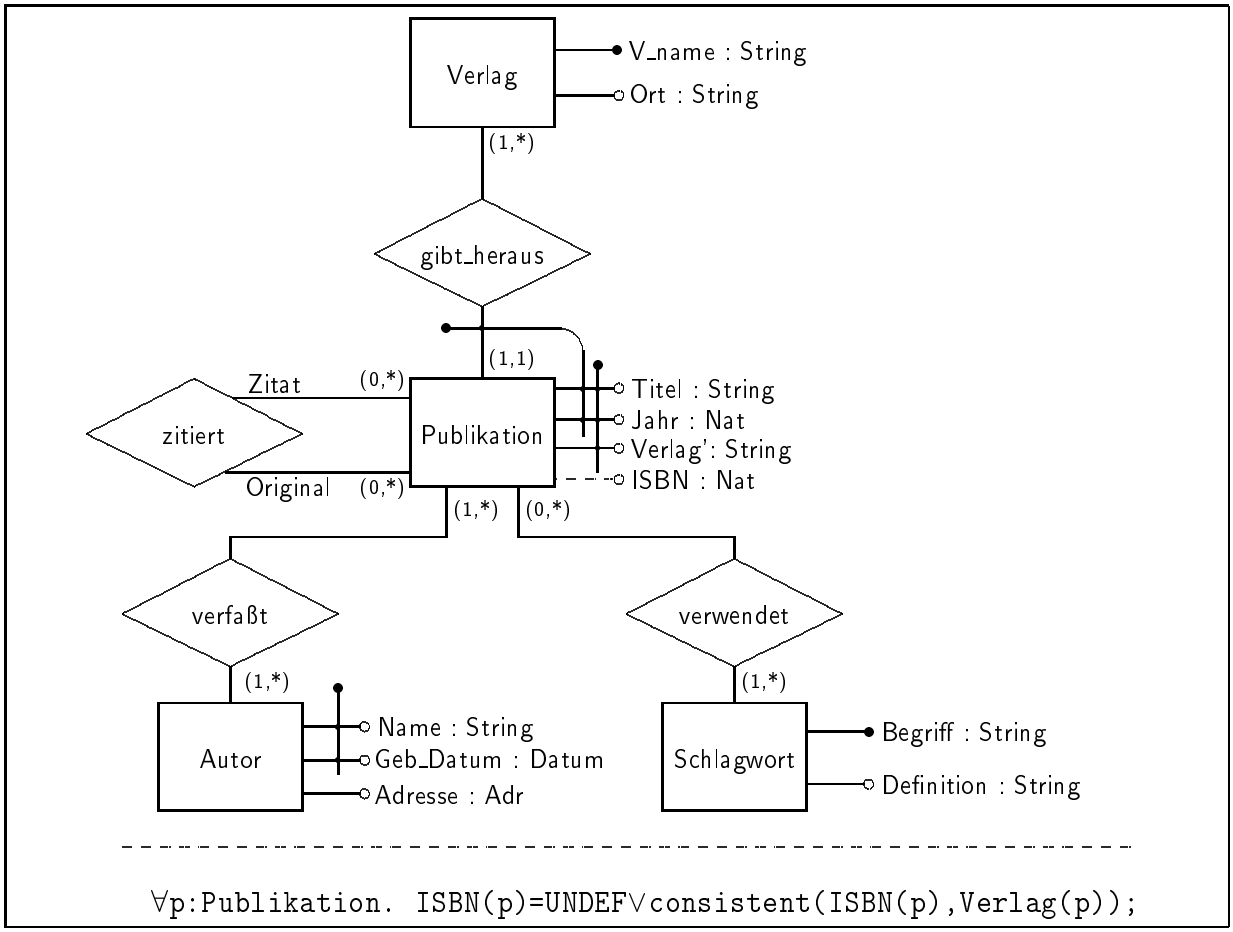


Abbildung 5.10: Integriertes Schema

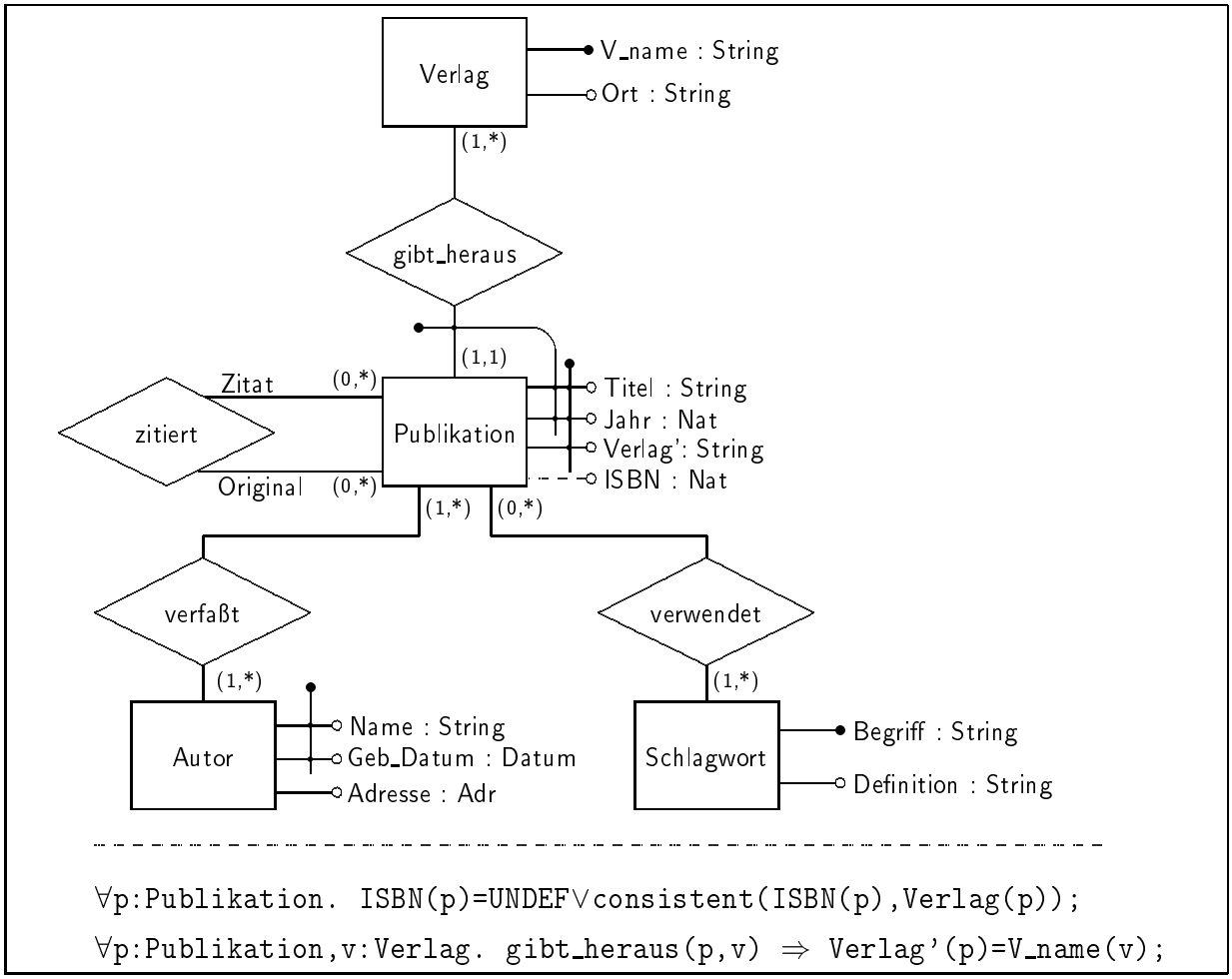


Abbildung 5.11: Integriertes Schema mit Interschema-Eigenschaften

genaue Zusammenhang ist, daß das Attribut Verlag' *und das Attribut* V_name *die gleiche Information, nämlich den Namen eines Verlags beinhalten. Dieser Zusammenhang kann in Form einer statischen Integritätsbedingung in das Schema aufgenommen werden:*

```
∀p:Publikation,v:Verlag. gibt_heraus(p,v) ⇒ Verlag'(p)=V_name(v)
```

Das fertige integrierte Schema ist in Abb. 5.11 dargestellt. Es unterscheidet sich von dem in Beispiel 2 (Seite 14) angegebenen Schema für RECHERCHE lediglich in zwei Punkten. Zum einen ist das zusätzliche Attribut Verlag' *enthalten, das aufgrund des oben angegebenen Zusammenhangs redundant ist. Zum anderen sind für den Entitytyp* Publikation *zwei Schlüssel angegeben. Wie man sich leicht überlegen kann, existiert aufgrund der (1,1)-Beteiligung von* Publikation *an* gibt_heraus *zu jeder Publikation genau eine Entity des Typs* Verlag *und damit genau ein Wert für das zugehörige Attribut* V_name. *Die beiden Schlüssel-*

definitionen sind damit aufgrund des Zusammenhangs zwischen Verlag' *und* V_name *äquivalent.* □

In der obigen Definition der drei zur View Integration erforderlichen Schritte wurde neben der Vereinigung von Spezifikationen lediglich die Umbenennung von Konstrukten des Schemas sowie das Anreichern um statische Integritätsbedingungen verwendet. Diese Schritte sind gemäß Definition 3 zur Weiterentwicklung von Schemata erlaubt und nach Satz 5 mit der Definition der statischen Semantik verträglich. Damit ist auch die hier gegebene Technik zur Integration von E/R-Schemata mit der statischen Semantik des E/R-Modells verträglich.

5.3 Diskussion

Das vorliegende Kapitel untersucht den Vorgang der konzeptuellen Datenmodellierung aus der Sicht der in Kapitel 3 gegebenen formalen Semantik des E/R-Modells. Es geht dabei von auf dem Gebiet der Datenmodellierung üblichen Vorgehensweisen aus, untersucht deren Verträglichkeit mit der statischen Semantik des E/R-Modells und gelangt so zu Vorgehensweisen, die zum in dieser Arbeit gegebenen Ansatz der formalen Datenmodellierung im E/R-Modell passen.

Realisierung

Abschnitt 5.1 untersucht typische Entwicklungsschritte für E/R-Datenschemata auf ihre Verträglichkeit mit der Definition der statischen Semantik. Es wird ausgegangen von der in [BCN92] vorgenommenen Klassifizierung solcher Schritte in Top-Down und Bottom-Up Entwicklungsschritte. Dabei wird festgestellt, daß die Klasse der Top-Down-Schritte sich nicht mit dem in der Sprache SPECTRUM definierten Realisierungsbegriff verträgt, da bei solchen Schritten im Datenschema vorhandene Konstrukte durch andere, verfeinerte Konstrukte ersetzt werden. Da auf diese Weise die Signaturen der statischen Semantik der an einem solchen Schritt beteiligten Schemata nicht in einer Inklusionsbeziehung stehen, besteht keine Realisierungsbeziehung zwischen den statischen Semantiken. Die Anwendung von Bottom-Up-Schritten, die das Anreichern von Datenschemata um neue Konstrukte und Bedingungen erlauben, etabliert eine SPECTRUM-Realisierungsbeziehung zwischen den beteiligten statischen Semantiken und ist deshalb mit der formalen Spezifikationstechnik verträglich.

Ausgehend von dieser Erkenntnis wird in Definition 3 ein formaler Bottom-Up Entwicklungsbegriff für E/R-Schemata definiert, der in Anlehnung an den für die Sprache SPECTRUM gegebenen Realisierungsbegriff als *Realisierung von E/R-Schemata* bezeichnet wird. Die Anwendung von Schritten dieses Entwicklungsbegriffs etabliert nicht nur eine SPECTRUM-Realisierungsbeziehung zwischen den zugehörigen statischen Semantiken (Satz 5). Eine kleine Erweiterung des Realisierungsbegriffs auf SPECTRUM-Spezifikationen garantiert

diese Beziehung auch auf der Ebene der Zugriffsspezifikationen. Auf diese Weise können Realisierungsschritte auch für E/R-Schemata eingesetzt werden, die im Rahmen der in Kapitel 4 gegebenen Technik zur formalen Entwicklung von Datenbankapplikationen mit Hilfe der Sprache SPECTRUM verwendet werden. Diese Technik ist damit auch methodisch untermauert.

Der Realisierungsbegriff für E/R-Schemata ist offensichtlich vollständig in der Hinsicht, daß mit seiner Hilfe jedes Schema erstellt werden kann. Er hat darüberhinaus den Vorteil, sich nahtlos in den in dieser Arbeit vorgestellten formalen Ansatz zur E/R-Datenmodellierung einzufügen. Dennoch bedeutet er methodisch eine starke Einschränkung gegenüber herkömmlichen Vorgehensweisen zur Datenmodellierung, in denen auch Top-Down Entwicklungen möglich sind. Am Ende dieser Diskussion wird deshalb in einem Ausblick eine Möglichkeit skizziert, wie sich Top-Down Entwicklungen in diesen formalen Rahmen integrieren lassen.

View Integration

In Abschnitt 5.2 wird eine mit der statischen Semantik verträgliche Technik zur Integration verschiedener Schemata zu einem konzeptuellen Datenschema angegeben. Die Technik ist gebräuchlichen Techniken zur View Integration in den drei grundlegenden Schritten sehr ähnlich: Zunächst werden die zu integrierenden Schemata durch Weiterentwicklung nach bestimmten Kriterien verträglich gemacht. Danach werden sie durch Überlagerung zu einem einzigen Schema vereinigt. Die zwischen den Schemata bestehenden Zusammenhänge werden dann in Form von Interschema-Eigenschaften in das integrierte Schema aufgenommen.

Der Unterschied der gegebenen Technik zu herkömmlichen Ansätzen besteht im wesentlichen aus zwei Punkten:

1. Durch die Möglichkeit, beliebige Integritätsbedingungen in Form von SPECTRUM-Formeln in ein Schema aufzunehmen, steht eine mächtige Sprache zur Formulierung von Interschema-Eigenschaften zur Verfügung. Da herkömmliche Ansätze zur E/R-Modellierung diese Möglichkeit nicht haben, können Interschema-Eigenschaften in ihnen nur indirekt durch Einführen neuer Konstrukte wie Relationshiptypen beschrieben werden. Während dies in manchen Fällen adäquat ist, ist es oft natürlicher, die Interschema-Eigenschaften direkt in Form von logischen Formeln als Integritätsbedingungen zu formulieren.

2. Der Fall, daß derselbe Aspekt der realen Welt in beiden Schemata auf syntaktisch unverträgliche Art modelliert wird, wird in der gegebenen Technik auf ungewöhnliche Art behandelt. Gebräuchliche Ansätze zur View Integration entwickeln die zu integrierenden Schemata weiter, bis der Aspekt von den beteiligten Schemata in syntaktisch verträglicher Weise beschrieben wird. Diese Entwicklung erfordert jedoch die

Anwendung von Top-Down-Schritten, da zumindest in einem Schema die Konstrukte, die die syntaktische Unverträglichkeit verursachten, durch syntaktisch verträgliche Konstrukte ersetzt werden müssen.

Da in der vorgestellten Technik Top-Down Entwicklungsschritte nicht möglich sind, wird die syntaktische Unverträglichkeit durch einfaches Umbenennen aufgelöst. Die Tatsache, daß die beiden unverträglichen Teilschemata denselben Teil der Realität modellieren, wird als Interschema-Eigenschaft in Form einer SPECTRUM-Formel in das integrierte Schema aufgenommen.

Die so gegebene Technik zur Integration von Datenschemata zeichnet sich vor allem durch die folgenden Punkte aus:

Redundanz Wie in Punkt 2 erläutert, enthält das integrierte Schema für jeden Aspekt der realen Welt alle in den Ausgangsschemata enthaltenen unterschiedlichen Modellierungen, wobei der Zusammenhang zwischen diesen Modellierungen in Form von allgemeinen statischen Integritätsbedingungen beschrieben wird. Die Modellierung der realen Welt des Anwendungsgebiets erfolgt im integrierten Schema also auf redundante Weise. Da das Schema im Gegensatz zu den meisten anderen Ansätzen die vollständige Information darüber enthält, welche Teile redundant sind, wirkt sich die Redundanz nicht nachteilig aus. Beim Übergang zur Implementierung des Schemas in einem Datenbanksystem ist es aufgrund dieser Information möglich, die Redundanzen zu eliminieren. Allerdings ist diese Aufgabe sicherlich sehr komplex, so daß Systemunterstützung für einen solchen Schritt als unabdingbar erscheint (siehe auch Abschnitt 6.2.1).

Erhalt der Ausgangsschemata Die im integrierten konzeptuellen Schema enthaltene Redundanz stellt gegenüber anderen Techniken zur View Integration in methodischer Hinsicht einen Vorteil dar, da die verschiedenen Ausgangsschemata im integrierten Schema vollständig enthalten sind. Oft stellen die Ausgangsschemata Sichten bestimmter Unterorganisationen auf die Daten der Gesamtorganisation dar. Bei der Definition des logischen Schemas (zum Beispiel im relationalen Modell) sind diese Sichten dann wieder in Form externer (relationaler) Schemata zu definieren. Dabei ist es natürlich von Vorteil, wenn die Ausgangsschemata noch unverändert im konzeptuellen Modell enthalten sind.

Top-Down Entwicklung

Der in Abschnitt 5.1 für E/R-Schemata definierte Realisierungsbegriff beinhaltet gegenüber in der Datenmodellierung üblichen Entwicklungsbegriffen eine deutliche Beschränkung der Flexibilität, da auf seiner Basis nur Bottom-Up Entwicklung möglich ist. Obwohl es offensichtlich nicht möglich ist, Top-Down Entwicklungsschritte als Realisierung im Sinne von Definition 3 zu sehen, können sie jedoch mit Hilfe der in Abschnitt 5.2 definierten Technik der View Integration in der folgenden Weise repräsentiert werden.

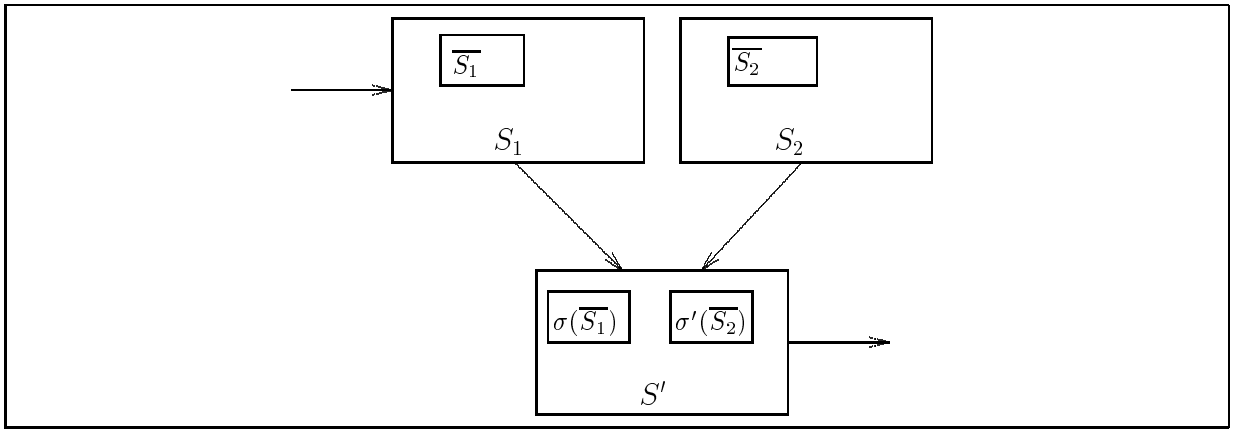


Abbildung 5.12: Top-Down Entwicklungsschritt mittels View Integration

Seien S_1 und S_2 E/R-Schemata, wobei S_2 mittels eines Top-Down Entwicklungsschritts aus S_1 entstanden sei. Da Top-Down-Schritte nicht Definition 3 genügen, besteht keine Realisierungsbeziehung zwischen den beiden Schemata.

Da S_2 mittels eines Top-Down-Schritts aus S_1 entstanden ist, enthält es ein Teilschema $\overline{S_2}$, das einen bestimmten Aspekt der Realität modelliert, der in S_1 durch das Teilschema $\overline{S_1}$ modelliert wird. Dabei ist $\overline{S_2}$ syntaktisch unverträglich mit $\overline{S_1}$. Abgesehen von $\overline{S_1}$ beziehungsweise $\overline{S_2}$ sind die beiden Schemata identisch. Mit Hilfe der in Abschnitt 5.2 Technik der View Integration können $\overline{S_1}$ und $\overline{S_2}$ zu einem Schema S' integriert werden, welches neben $\overline{S_2}$ redundanterweise auch $\overline{S_1}$ enthält (beide natürlich nach geeigneter Umbenennung, die die syntaktische Unverträglichkeit auflöst). Diese Situation ist in Abbildung 5.12 dargestellt. Der Zusammenhang zwischen $\overline{S_1}$ und $\overline{S_2}$ wird dabei in Form einer Interschema-Eigenschaft, das heißt als SPECTRUM-Formel in S' notiert. Da S' den angesprochenen Aspekt der Wirklichkeit mittels $\overline{S_2}$ beschreibt, ist der durch den Top-Down Schritt intendierte Zweck erreicht. Die Entwicklung des konzeptuellen Schemas kann nun von S' ausgehend fortgesetzt werden.

Beispiel 15 *Die Applikation RECHERCHE, die in der vorliegenden Arbeit als Beispiel dient, enthält ein primitives Schlagwortlexikon, in dem die in den gespeicherten Publikationen verwendeten Schlagwörter verwaltet werden. Die Schlagwörter werden dabei im Entitytyp* Schlagwort *verwaltet, wobei zu jedem Schlagwort eine textuelle Erklärung im Attribut* Definition *verwaltet wird (siehe Abbildung 5.11 auf Seite 93). Das Schlagwortlexikon soll nun dahingehend verfeinert*[6] *werden, daß die Erklärung zu einem Schlagwort aus mehreren Einzeldefinitionen bestehen kann, die neben einer textuellen Beschreibung jeweils auch eine illustrierende Abbildung enthalten können. Darüberhinaus können diese Erklärungen*

[6] Verfeinerung ist hier nicht im Sinne einer formalen SPECTRUM-Verfeinerung zu verstehen, sondern einfach als Ersetzen des Entitytyps Schlagwort durch ein detaillierteres E/R-Teildiagramm.

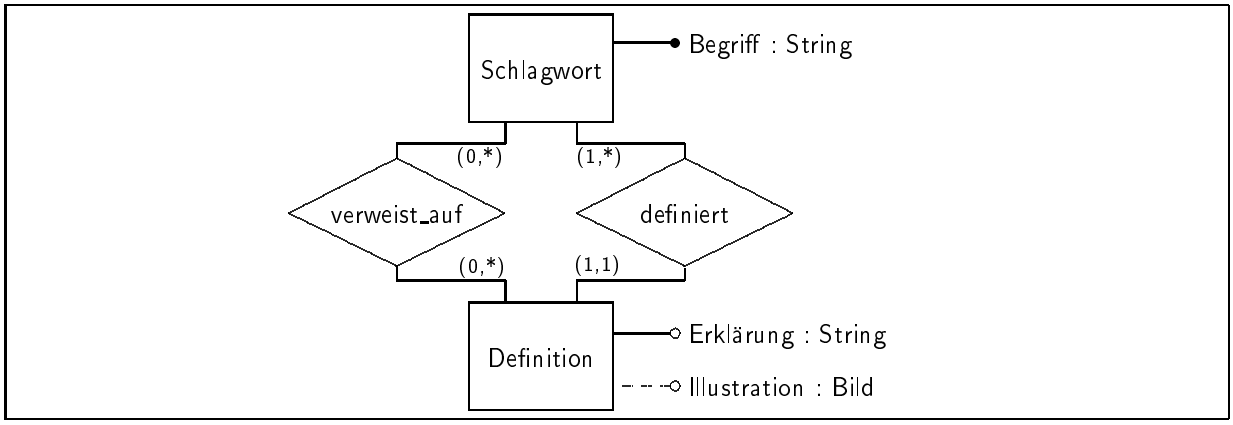


Abbildung 5.13: Schlagwortlexikon nach Top-Down Entwicklungsschritt

Querverweise auf andere Schlagwörter enthalten. Abbildung 5.13 zeigt ein E/R-Diagramm, das diese Sicht auf das Schlagwortlexikon widerspiegelt.

Die Weiterentwicklung des in Abbildung 5.11 gegebenen E/R-Schemas, die bewirkt, daß das Schlagwortlexikon wie in Abbildung 5.13 modelliert wird, ist ein Top-Down Entwicklungsschritt, weil dabei das Attribut Definition *des Entitytyps* Schlagwort *durch einen Entitytyp* Definition *ersetzt wird, der durch zwei Relationshiptypen mit* Schlagwort *verbunden ist. Im Sinne der in [BCN92] gegebenen Transformationsregeln stellt die hier angegebene Entwicklung übrigens keinen atomaren Entwicklungsschritt dar. Dazu sind vielmehr eine ganze Reihe von Anwendungen von Transformationsregeln nötig, sowohl Top-Down als auch Bottom-Up-Regeln. Dabei betrachten die Transformationsregeln aus [BCN92], wie bereits zu Beginn des Kapitels geschildert, lediglich die Struktur des E/R-Schemas und ignorieren die statischen Integritätsbedingungen.*

Mit dem in dieser Arbeit gegebenen Ansatz können die Schemata aus Abbildung 5.11 und Abbildung 5.13 zu einem Schema integriert werden. Die einzige syntaktische Unverträglichkeit zwischen diesen beiden Schemata wird dabei durch Umbenennen des Attributs Definition *in* Definition' *eliminiert. Die Tatsache, daß das Attribut* Definition' *redundant ist, weil es mit Hilfe des Entitytyps* Definition *und des Relationshiptyps* definiert *berechnet werden kann, wird als Interschema-Eigenschaft in Form einer* SPECTRUM*-Formel in das vereinigte Schema aufgenommen (siehe Abbildung 5.14).*

Im hier gegebenen Beispiel beschreibt die SPECTRUM*-Formel*

$$\forall \texttt{s:Schlagwort}.\exists \texttt{d:Definition .definiert(d,s)} \wedge \texttt{Definition'(s)} = \texttt{Erklärung(d)}$$

den Zusammenhang zwischen dem Attribut Definition' *und dem Entitytyp* Definition*. Sie besagt, daß das Attribut* Definition' *einer Entity des Typs* Schlagwort *immer dem Wert des Attributs* Erklärung einer *der Entities des Typs* Definition *entspricht, die mit der* Schlagwort*-Entity durch den Relationshiptyp* definiert *in Beziehung stehen. Es wäre an dieser Stelle*

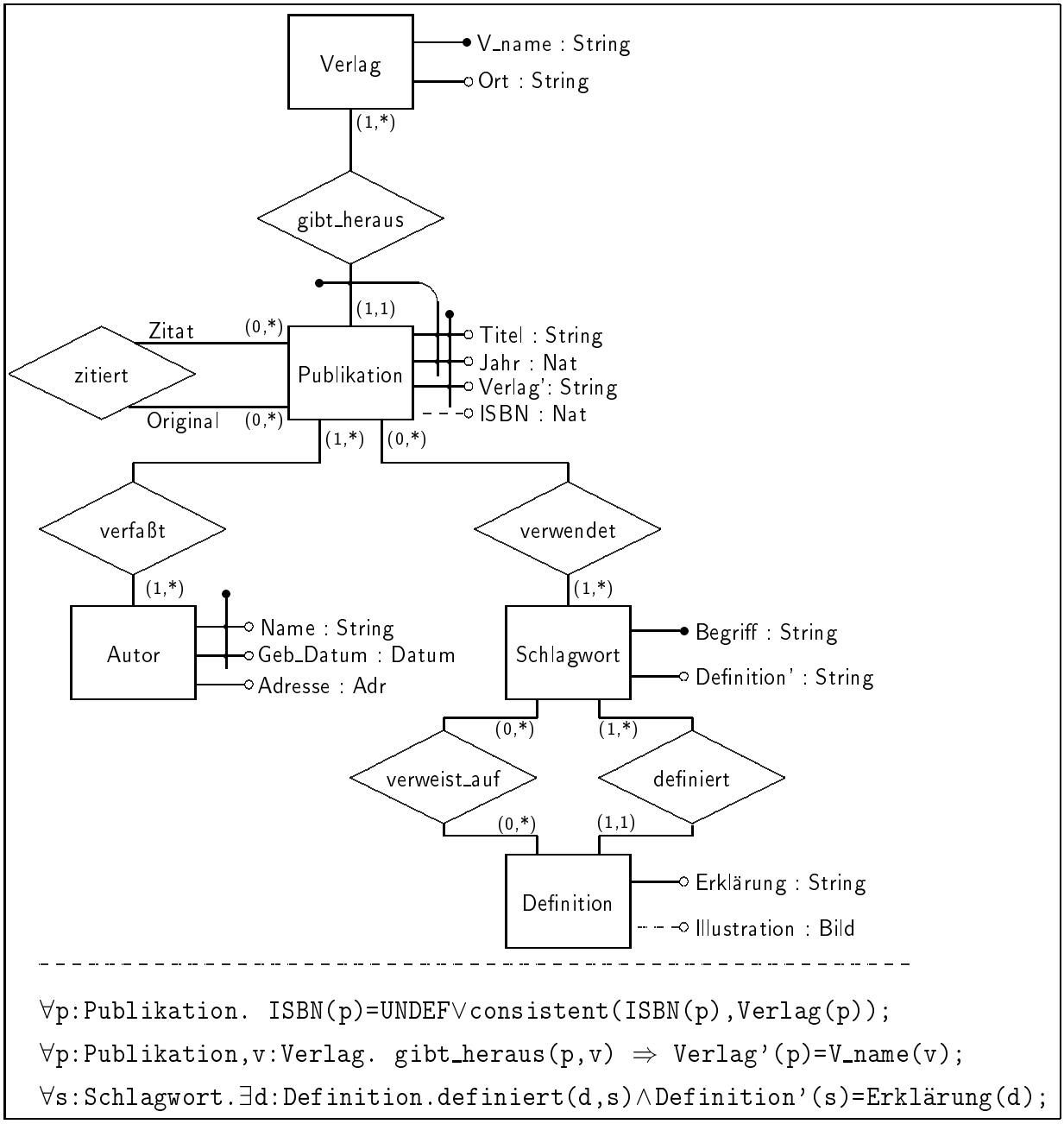


Abbildung 5.14: Datenmodell der Beispielapplikation nach Top-Down Entwicklungsschritt

auch möglich gewesen, die wesentlich abstraktere Forderung zu stellen, daß eine Funktion existiert, die den Wert des Attributs Definition' *einer* Schlagwort*-Entity s aus der Menge der Entities des Typs* Definition *berechnet, die mit s in einer Beziehung des Typs* definiert *stehen. Man sieht also, daß hier Anwendungswissen des Datenmodellierers in die Formulierung der Interschema-Eigenschaft eingeht.* □

Kapitel 6

Konklusion

6.1 Zusammenfassung der Ergebnisse

Die Arbeit bezieht ihre Motivation aus der These, daß eine geschickte Integration pragmatischer Software-Engineering Ansätze mit formalen Techniken, wie sie seit vielen Jahren bevorzugt im akademischen Bereich untersucht werden, Vorteile bei der Entwicklung qualitativ hochwertiger Software bringt. Sie beschränkt sich dabei auf einen Teilaspekt, der bei der Entwicklung von Softwaresystemen auftritt, nämlich auf den Aspekt der Modellierung der Daten eines Systems. Die Untersuchung dieses Aspekts wird durchgeführt, indem der Versuch der Integration zweier aus den beiden unterschiedlichen Welten der pragmatischen und der formalen Softwareentwicklung stammender Vertreter unternommen wird.

Aus der Welt des Software-Engineerings wurde als Vertreter das Entity/Relationship-Modell ausgewählt, weil es die mit Abstand am weitesten verbreitete und akzeptierte Technik zur konzeptuellen Datenmodellierung darstellt. Darüberhinaus ist die Bedeutung von E/R-Diagrammen sowohl informell als auch durch Übersetzung in das relationale Datenmodell eindeutig festgelegt, was die Formalisierung erheblich erleichtert.

Die Integration sollte mit einem Vertreter aus der Welt der algebraischen Spezifikationstechniken erfolgen. Der Grund für diese Wahl ist zum Einen, daß diese Techniken bereits sehr ausgereift sind und damit ein solides Fundament für die Integration bieten. Auf der anderen Seite haben gerade diese Techniken Probleme mit der Spezifikation stark datenorientierter Systeme und können so von der Integration mit einer Datenmodellierungstechnik erheblich profitieren. Mit der Sprache SPECTRUM fiel die Wahl auf einen der ausdrucksmächtigsten und modernsten Vertreter dieser Klasse der formalen Techniken.

Definition einer formalen Semantik

Durch Angabe einer Übersetzungsvorschrift, die jedem E/R-Schema eine Spezifikation in SPECTRUM, seine statische Semantik, zuordnet, wird dem E/R-Modell eine formale Se-

mantik in der Spezifikationssprache gegeben. Diese Semantik ist besser geeignet, die Eigenschaften des E/R-Modells zu beschreiben als die bekannte Abbildung in das relationale Modell oder ein anderes implementierungsnahes Datenmodell, da alle diese Modelle nicht in der Lage sind, die im E/R-Modell ausdrückbaren statischen Integritätsbedingungen zu beschreiben. In SPECTRUM dagegen können diese Bedingungen auf direkte Weise als logische Formeln ausgedrückt werden.

Im Vergleich zum Ansatz von [Hoh93] ist die Semantik nach Ansicht des Autors wesentlich leichter verständlich. Der Grund dafür liegt hauptsächlich in der größeren Reichhaltigkeit des in [Hoh93] verwendeten Datenmodells. Darüberhinaus wird mit der Sprache SPECTRUM eine vorgegebene Spezifikationssprache verwendet, während in [Hoh93] ein spezieller algebraischer Ansatz beschrieben wird, der ausschließlich dazu dient, die Semantik des erweiterten E/R-Modells zu definieren, der jedoch nicht dazu dienen soll, das E/R-Modell als Beschreibungstechnik zur Datenmodellierung in eine formale Spezifikationssprache zu integrieren.

Die Definition der statischen Semantik hat große Ähnlichkeit mit dem in [JRP91a, JRP91b] gegebenen Ansatz zur Formalisierung der Semantik des E/R-Modells in Z. Die Sprache Z ist eine Spezifikationssprache, die bereits in vielen Anwendungsbeispielen zur Entwicklung von Software eingesetzt wurde. Die zur Definition der Semantik des E/R-Modells eingesetzten Konzepte sind ähnlich einfach wie in der vorliegenden Arbeit. Damit kann auch diesem Ansatz ein hohes Maß an Verständlichkeit zugesprochen werden.

Allgemeine statische Integritätsbedingungen

Durch die Integration des E/R-Modells mit der formalen Spezifikationssprache SPECTRUM steht eine Möglichkeit zur Verfügung, nahezu beliebige statische Integritätsbedingungen zu formulieren, denen die spezifizierten Daten zu gehorchen haben. Die im E/R-Modell graphisch darstellbaren Integritätsbedingungen (Schlüsselbedingungen und Kardinalitäten) werden durch die gegebene Übersetzungsvorschrift ebenfalls in diese allgemeine prädikatenlogische Sprache transformiert.

Eine solche Möglichkeit, beliebige statische Integritätsbedingungen für die beschriebenen Daten formulieren zu können, ist methodisch von größter Bedeutung. Damit ist es möglich, *alle* Eigenschaften der von einem Informationssystem zu verwaltenden Daten in einem einzigen, homogenen Dokument zu beschreiben. Im Gegensatz dazu besteht bei Software-Engineering-Methoden, die das klassische E/R-Modell zur Datenmodellierung einsetzen, die Notwendigkeit, alle statischen Eigenschaften der Daten, die nicht als Schlüsselbedingungen oder Kardinalitäten beschrieben werden können, an anderer Stelle (zum Beispiel in Einträgen eines Data Dictionary) zu notieren. Oftmals werden diese Bedingungen in solchen Methoden in informeller Weise, zum Beispiel als natürlichsprachlicher Text, notiert. Sie entziehen sich damit der direkten Verwendung in der Entwicklung der Transaktionen des spezifizierten Informationssystems. Werden die statischen Integritätsbedingungen jedoch in Form prädikatenlogischer Formeln beschrieben, können automatisch Beweisver-

pflichtungen für die spezifizierten Transaktionen generiert werden, die die Integrität der verwalteten Datenbank als Invariante aller Transaktionen fordern.

Die Erweiterung des E/R-Modells um eine prädikatenlogische Sprache zur Beschreibung beliebiger statischer Integritäten erlaubt es insbesondere, zwei Klassen von Information über die zu beschreibenden Daten in das Datenschema aufzunehmen, die bei der Entwicklung der zugehörigen Datenbank von Bedeutung sind:

Funktionale Abhängigkeiten Einer der typischen Schritte, die auf dem Weg vom konzeptuellen Schema zur eigentlichen Datenbankdefinition erfolgen, ist die Normalisierung des Schemas. Normalisierung bedeutet Elimination von eventuell im Schema enthaltenen Redundanzen. Sie basiert im wesentlichen auf dem Wissen des Datenbankentwicklers über (im Schema nicht ausgedrückte) Zusammenhänge zwischen Teilen des Schemas (üblicherweise Attribute). Diese Zusammenhänge werden meist als *funktionale Abhängigkeiten* bezeichnet. Diese funktionalen Abhängigkeiten können nun als allgemeine statische Integritäten bereits in das konzeptuelle Schema aufgenommen werden. Die Normalisierung kann sich dann auf diese in formaler Form vorliegenden funktionalen Abhängigkeiten stützen. Damit besteht die Möglichkeit, den Vorgang der Normalisierung in einem deutlich höheren Maß maschinell zu unterstützen, als dies bisher der Fall ist.

Interschema-Eigenschaften Eine weitere Anwendung allgemeiner statischer Integritätsbedingungen stellt die Beschreibung von Interschema-Eigenschaften bei der View Integration dar. Die Spezifikation solcher Eigenschaften, die in einem integrierten Schema die Zusammenhänge zwischen Konstrukten unterschiedlicher Ausgangsschemata beschreiben, ist in Form logischer Formeln oft direkter und naheliegender als durch Einführung zusätzlicher Konstrukte (zum Beispiel Relationshiptypen), wie es in herkömmlichen E/R-Ansätzen möglich ist.

Methodik

Für die so semantisch formal fundierte und um die Möglichkeit zur Spezifikation beliebiger statischer Integritäten erweiterte Entity/Relationship-Technik wird eine Methodik zur konzeptuellen Datenmodellierung angegeben. Die Methodik ist mit der Semantikdefinition (das heißt der Übersetzung eines Schemas in seine statische Semantik) verträglich. Der für die Sprache SPECTRUM definierte Realisierungsbegriff, der sowohl Verfeinerung als auch einen Wechsel von Datenstrukturen ermöglicht, bleibt erhalten. Es werden im wesentlichen zwei Techniken definiert:

Realisierung von E/R-Schemata Der für E/R-Schemata definierte Entwicklungsbegriff wird in Anlehnung an den SPECTRUM-Realisierungsbegriff ebenfalls als Realisierung bezeichnet. Er erlaubt eine sogenannte Bottom-Up Entwicklung von E/R-Schemata durch Umbenennen von Konstrukten, Realisierung auf Ebene der Attribut-

sorten sowie Anreichern um Entitytypen, Relationshiptypen, Attribute und statische Integritätsbedingungen.

Ein interessanter Nebeneffekt ist, daß dieser Realisierungsbegriff auch auf Ebene der Zugriffsspezifikation eingesetzt werden kann, das heißt die hier festgelegten Entwicklungsschritte können auch dann noch angewendet werden, wenn bereits eine auf der spezifizierten Datenbank arbeitende Applikation in SPECTRUM entwickelt wurde.

Integration von Schemata Eine in der konzeptuellen Datenmodellierung häufig angewendete Technik ist die Integration verschiedener Schemata. In der vorliegenden Arbeit wird eine mit der Definition der formalen Semantik des E/R-Modells verträgliche Technik zur Integration von E/R-Schemata angegeben. Außergewöhnlich an dieser Technik ist, daß SPECTRUM-Formeln verwendet werden, um alle Interschema-Eigenschaften, zu denen auch Redundanzen in der Modellierung gehören, in Form von statischen Integritätsbedingungen im integrierten Schema zu spezifizieren.

Top-Down Entwicklungen, die vom gegebenen Realisierungsbegriff für E/R-Schemata nicht abgedeckt werden, sind im vorgestellten formalen Rahmen ebenfalls denkbar. Eine Möglichkeit, Top-Down Entwicklungsschritte mit Hilfe der Technik der View Integration zu realisieren, ist in der Diskussion zu Kapitel 5 skizziert.

Erweiterung von SPECTRUM um E/R-Datenmodellierung

Der vorgestellte Ansatz hat nicht nur Auswirkungen auf die Technik der konzeptuellen Datenmodellierung. Die formale Spezifikationstechnik SPECTRUM kann von der Integration mit dem E/R-Modell ebenfalls profitieren. Kapitel 4 zeigt, wie die E/R-Modellierung als Front-End zur Spezifikation der Daten eines Systems im Rahmen von SPECTRUM eingesetzt werden kann.

Die Spezifikation eines stark datenorientierten Systems besteht in diesem Ansatz aus einem E/R-Schema, eventuell mit zusätzlichen statischen Integritätsbedingungen in Form logischer Formeln, und einer darauf aufbauenden Spezifikation der Systemfunktionen in SPECTRUM. Diese Gesamtspezifikation kann aufgrund der Definition der formalen Semantik des E/R-Modells als homogenes formales SPECTRUM-Dokument betrachtet werden. Einer der großen Vorteile dieser Technik ist die Tatsache, daß die statischen Integritätsbedingungen im Rahmen der Spezifikationssprache als Beweisverpflichtungen für die entwickelten Transaktionen zur Verfügung stehen.

SPECTRUM ist wie alle algebraischen Spezifikationstechniken besser zur Spezifikation algorithmisch komplexer als stark datenorientierter Systeme geeignet. Die Bereitstellung des E/R-Modells als Front-End zur Beschreibung der relevanten Daten verbessert diese Situation wesentlich und stellt sicher eine wichtige Vorbedingung für die Anwendung dieser formalen Techniken auf Probleme realer Größe dar.

6.2 Ausblicke

Die vorliegende Arbeit hat (am Beispiel des Entity/Relationship-Modells und der axiomatischen Spezifikationssprache SPECTRUM) die Grundlagen für eine glatte Integration der Welt der konzeptuellen Datenmodellierung und algebraischer Spezifikationstechniken geschaffen. Es ist eine ganze Reihe von Arbeiten denkbar, die an diese Arbeit anschließen und auf die hier erzielten Ergebnisse aufbauen können.

6.2.1 Werkzeugunterstützung

Eine naheliegende Fortführung der Arbeit besteht in der Konzeption und Implementierung eines graphischen Werkzeugs zur Unterstützung der vorgestellten Techniken. Ein solches Werkzeug sollte idealerweise sowohl die konzeptuelle Datenmodellierung mit den in Kapitel 5 vorgestellten Techniken unterstützen als auch die Spezifikation von Systemfunktionen im Sinne von Kapitel 4 erlauben und die für solche Funktionen geltenden Beweisverpflichtungen aus den statischen Integritätsbedingungen erzeugen.

Für eine derartige Werkzeugunterstützung sind selbstverständlich viele unterschiedliche Konzeptionen und Architekturen denkbar. Eine mögliche Konzeption besteht in einem monolithischen, graphischen Werkzeug, das folgende Aufgaben erfüllt:

Verwaltung von Schema-Entwicklungen Die Entwicklung eines konzeptuellen E/R-Schemas, die mit Hilfe des in Kapitel 5 definierten Realisierungsbegriffs oder der gegebenen Technik zur View Integration erfolgt, kann als Graph gesehen werden, dessen Knoten E/R-Schemata bilden und dessen Kanten den angesprochenen Realisierungsbegriff symbolisieren. Das Werkzeug muß in der Lage sein, derartige Graphen zu verwalten, zum Beispiel in einer Datenbank.

Unterstützung der möglichen Entwicklungsschritte Die verwalteten Entwicklungsgraphen sollen ausschließlich mit Hilfe der in Kapitel 5 definierten Entwicklungsschritte (Realisierung von Attributsorten, Umbenennen und Hinzufügen von Attributen, Entitytypen und Relationshiptypen, Verstärken der statischen Integritätsbedingungen, View Integration) erstellt werden können.

Ein Werkzeug sollte also idealerweise in der Lage sein, die einzelnen im Entwicklungsgraphen enthaltenen Schemata mit Hilfe eines graphischen Editors darzustellen. Zur interaktiven Veränderung eines Schemas sollten die oben angesprochenen Entwicklungsschritte zur Verfügung stehen. Zur Unterstützung der Entwicklungsschritte gehört insbesondere, daß alle durch ihre Anwendung entstehenden Beweisverpflichtungen automatisch generiert und protokolliert werden. Zweckmäßigerweise sollte ein Anschluß an ein interaktives Beweissystem zum Nachweis dieser Beweisverpflichtungen zur Verfügung stehen. Für die in dieser Arbeit verwendete SPECTRUM-Logik bietet sich hier die SPECTRUM-Instanz des generischen Theorembeweisers ISABELLE [Pau94] an.

Unterstützung der Spezifikation und Verifikation von Transaktionen Die zu einem E/R-Schema gehörende Zugriffsspezifikation ermöglicht die Spezifikation von Transaktionen, die auf den durch das Schema beschriebenen Daten arbeiten, in SPECTRUM (siehe Kapitel 4).

Das Werkzeug sollte die Spezifikation solcher Transaktionen unterstützen und damit die Möglichkeit schaffen, die für die Transaktionen geltenden Verifikationsbedingungen (Invarianz der statischen Integrität) zu generieren und, falls ein Anschluß an ein Beweissystem zur Verfügung steht, diese auch nachzuweisen.

Da sich die Spezifikation der Transaktionen gemäß Kapitel 4 hierarchisch auf die Zugriffsspezifikation des E/R-Schemas stützt, muß das Werkzeug in der Lage sein, die angegebene Übersetzung des Schemas in diese Spezifikation durchzuführen.

Unterstützung des Übergangs zur logischen Modellierung Ein Aspekt, der über die Phase der konzeptuellen Modellierung hinausgeht, wäre die Unterstützung der Umsetzung des mit dem Werkzeug erstellten konzeptuellen E/R-Schemas in ein logisches Schema, also ein Schema, das bereits im Datenmodell des zur Implementierung eingesetzten Datenbanksystems formuliert ist. Hier ist natürlich vorzugsweise das relationale Datenmodell zu betrachten.

Durch die in den allgemeinen statischen Integritätsbedingungen enthaltenen funktionalen Abhängigkeiten und Redundanzen, die bei Schema-Integrations-Schritten entstanden sind, könnte hier sowohl die Normalisierung als auch die Definition externer (relationaler) Schemata effizient unterstützt werden.

Um den Übergang zu einem anderen Datenmodell wie zum Beispiel dem relationalen zu bewältigen, ist hier jedoch noch konzeptionelle Arbeit zu leisten, die über den Rahmen der vorliegenden Arbeit hinausgeht (siehe dazu Abschnitt 6.2.2).

Die Entwicklung eines derartigen Werkzeugs kann sich auf die Arbeit [Hub94] stützen. Im Rahmen dieser Diplomarbeit wurde das Werkzeug Genesys implementiert, das es erlaubt, E/R-Schemata mit Hilfe eines graphischen Editors zu erstellen und alle im vorliegenden Ansatz geforderten Textdarstellungen zu generieren. Es ist also in der Lage, das gezeichnete E/R-Diagramm sowohl in seine statische Semantik als auch in die zugehörige Zugriffsspezifikation zu übersetzen (siehe [Sal95]).

Genesys unterstützt dabei jedoch weder die Möglichkeit, E/R-Schemata mit logischen Formeln als statische Integritätsbedingungen zu annotieren noch die in Kapitel 5 angegebene Methodik. Es ist mit dem Werkzeug also nicht möglich, das zu bearbeitende Schema gemäß den in diesem Kapitel definierten Entwicklungsschritten weiterzubearbeiten.

6.2.2 Weitere Arbeiten

Neben der oben dargestellten Werkzeugunterstützung ist eine Reihe von theoretischen Arbeiten denkbar, die den vorgestellten Ansatz fortführen beziehungsweise erweitern.

Erweiterung des Datenmodells

In der vorliegenden Arbeit wurde, unter anderem um die Präsentation der vorgestellten Konzepte einfach zu halten, ein klassisches E/R-Modell betrachtet. Davon ausgehend kann der präsentierte Ansatz auf andere, reichhaltigere semantische Datenmodelle erweitert werden. So könnte das E/R-Modell beispielsweise um folgende Konzepte erweitert werden:

- Generalisierung (Vererbung),
- Aggregation,
- Zulassen von Entities als Attribute.

Die ersten beiden Erweiterungen (Generalisierung, Aggregation) stellen Schritte in Richtung objektorientierter Ansätze dar. Insgesamt entsteht durch die drei Erweiterungen genau das in [Hoh93] betrachtete erweiterte Entity/Relationship-Modell (EER-Modell).

Bei der Einführung derartiger Erweiterungen ist zu bedenken, daß die Formalisierung der neu hinzugekommenen Konzepte unter Umständen spezielle Anforderungen an die verwendete formale Spezifikationstechnik stellt. So ist eine Erweiterung des in dieser Arbeit vorgestellten Ansatzes um entitywertige Attribute trivial, da die Sprache SPECTRUM ohnehin keinen konzeptionellen Unterschied zwischen Entitysorten und Attributsorten macht. Für die Einführung des Konzepts der Generalisierung in das Datenmodell hingegen wäre eine Spezifikationstechnik wünschenswert, die im Gegensatz zu SPECTRUM das Konzept von Subsorten kennt [Car84]. Mit Hilfe von Subsorting ließe sich der durch die Generalisierung ausgedrückte Sachverhalt in natürlicher Weise im Sortensystem der Spezifikationssprache ausdrücken.

Erweiterung auf logische Modellierung

Das konzeptuelle Datenschema eines Informationssystems wird meist im E/R-Modell oder einem anderen semantischen Datenmodell beschrieben, da diese Modelle eine sehr problemnahe Sicht auf die betrachteten Daten erlauben. Da es nach dem heutigen Stand der Technik für diese Modelle nur sehr wenige, experimentelle (das heißt insbesondere ineffiziente) Implementierungen in Form von Datenbanksystemen gibt, wird das konzeptuelle Schema in einem nächsten Schritt in ein anderes, implementierungsnäheres Datenmodell transformiert. Das so entstandene Schema wird als logisches Schema bezeichnet (vergleiche Abbildung 5.1). Zur Formulierung des logischen Schemas kommt heute neben dem hierarchischen und dem Netzwerk-Datenmodell insbesondere das relationale Modell in Frage (für eine Definition dieser Modelle siehe zum Beispiel [SS83]). Es stellt eine interessante Arbeit dar, den vorgestellten Ansatz zur konzeptuellen Datenmodellierung so zu erweitern, daß auch die logische Modellierung mit eingeschlossen wird. Eine derartige Erweiterung soll im folgenden am Beispiel des relationalen Modells skizzenhaft diskutiert werden.

Um den Übergang zu einem relationalen logischen Schema modellieren zu können, ist es nötig, auch das relationale Modell im gleichen formalen Ansatz (im vorliegenden Fall SPECTRUM) zu formalisieren. Ist dies geschehen, sollte der Schritt zum logischen Schema idealerweise im Rahmen des normalen Entwicklungsbegriffs geschehen können. Wichtig ist auch, daß bei diesem Schritt die Möglichkeit allgemeiner statischer Integritätsbedingungen in Form von SPECTRUM-Formeln berücksichtigt wird. Gerade Bedingungen wie funktionale Abhängigkeiten müssen in das relationale Schema übernommen werden können, da sie erst hier (bei der Normalisierung des Schemas) ausgenutzt werden können. Auch die Formalisierung des relationalen Schemas muß also die Erweiterung um beliebige statische Integritäten enthalten.

In einem Datenmodell, das wie das relationale in Form von Datenbanksystemen implementiert ist, gewinnt auch die zur Manipulation der Daten eingesetzte Sprache an Bedeutung. Es genügt hier nicht mehr, eine einfache Zugriffsschicht anzugeben, wie dies in Kapitel 4 für das E/R-Modell geschehen ist. Vielmehr muß es in der formalen Fundierung des relationalen Modells möglich sein, die für dieses Modell tatsächlich eingesetzten Sprachen auszudrücken. Die Fundierung muß also eine Formalisierung der relationalen Operatoren Selektion, Projektion und Join enthalten. Darauf aufbauend muß vor allem die Sprache SQL eine formale Semantik im verwendeten Formalismus finden. Sehr gute Vorarbeit wurde in dieser Hinsicht in den Arbeiten von Hohenstein, Gogolla und Karge [KG90, GH91, Hoh93, Gog94] geleistet. Die dort vorgestellten Ansätze sollten deshalb in die Formalisierung des relationalen Modells eingearbeitet werden.

Anpassung des zugrundeliegenden Formalismus

Im Unterschied zum Ansatz von [Hoh93], wo ein spezieller algebraischer Formalismus zur formalen Fundierung des E/R-Modells definiert wurde, wurde in der vorliegenden Arbeit mit SPECTRUM eine vorgegebene Spezifikationssprache verwendet. Es hat sich gezeigt, daß SPECTRUM zur formalen Fundierung konzeptueller Datenmodellierung geeignet ist. Dennoch ist eine Reihe kleinerer Veränderungen denkbar, die SPECTRUM für diesen Verwendungszweck noch besser geeignet machen würden.

Am stärksten ist im Lauf der Arbeit der Wunsch nach einer Anpassung des Sortensystems an den vorliegenden Anwendungsfall entstanden. SPECTRUM besitzt ein starkes und statisches Sortensystem, das heißt jeder Term der Sprache hat eine eindeutige (im Sinne der Polymorphie allgemeinste) statisch bestimmte Sorte. Ein solches Sortensystem ist nach Ansicht des Autors bei der Spezifikation von Softwaresystemen unerläßlich, da mit seiner Hilfe eine große Klasse von möglichen Fehlern rein syntaktisch und damit sehr frühzeitig festgestellt werden können, was bei unsortierten Sprachen nicht möglich ist. Dennoch war dieses Sortensystem bei der Formalisierung des E/R-Modells in manchen Fällen nicht flexibel genug. Ein Indiz dafür ist die Tatsache, daß es in der vorgestellten Zugriffsschicht nötig war, für jede Entitysorte eigene Funktionen zum Speichern und Löschen von Entities dieser Sorte in der Datenbank zu definieren. Des weiteren ist es mit dem verwendeten Sortensystem nicht möglich, einen Operator wie den relationalen Join-Operator zu definieren, da

die entstehende Relation (Entitysorte) davon abhängt, welche Relationen dieser Operator verknüpft und in welcher Weise er die Verknüpfung vornimmt.

Es ist klar, daß das E/R-Modell wie auch das relationale Modell auch in SPECTRUM auf eine Art und Weise spezifiziert werden kann, die die Spezifikation von Operatoren wie dem Join möglich macht. Dies erfordert jedoch das Zusammenfassen aller Attribut- und Entitysorten in jeweils eine variante Sorte. Auf diese Weise wird für die betroffenen Sorten das SPECTRUM-Sortensystem ausgeschaltet, das heißt alle Sortenüberprüfungen müssen explizit (etwa in Form von Prädikaten) spezifiziert werden. Diese Technik wurde deshalb aus pragmatischen Gründen in der vorliegenden Arbeit bewußt nicht gewählt. Stattdessen wäre ein Sortensystem wünschenswert, das es erlaubt, für bestimmte Sorten wie Entitysorten die statische Überprüfbarkeit aufzugeben und so Operatoren wie den angesprochenen Join-Operator zu ermöglichen. Hier kann eventuell auf Erfahrungen aus der Typtheorie (dependent types, siehe zum Beispiel [LPT89]) zurückgegriffen werden.

Ein Sortensystem, das darüberhinaus die Spezifikation von Subsortenbeziehungen erlaubt, würde eine Erweiterung des betrachteten E/R-Modells um Generalisierung direkt unterstützen. Eine solche Erweiterung würde auch einen Schritt in Richtung objektorientierter Ansätze zur Modellierung bedeuten.

Anhang A

Beweise der verwendeten Hilfssätze

Dieser Anhang enthält die Beweise für die in Kapitel 4 verwendeten Hilfssätze.

Satz 1 *Die Abbildung* Φ *ist injektiv.*

Beweis *Seien* $f_1, f_2 \in W_{\Sigma^s}^{Bool}(\chi)$, $f_1 \neq f_2$. *Einfache Induktion über die Termstruktur (Abbildung 4.1) zeigt* $\varphi[f_1] \neq \varphi[f_2]$. *Damit gilt auch* $\Phi[f_1] \neq \Phi[f_2]$. □

Satz 2 *Die Axiome der statischen Semantik* Ax^s *können (nach Umbau mittels* Φ*) aus den Axiomen der internalisierten Semantik hergeleitet werden:*

$$\Phi[Ax^s] \subseteq \mathcal{TH}^{int}$$

Beweis *Sei* $ax \in Ax^s$. *Mit Hilfe des* SPECTRUM*-Kalküls ist* $\Phi[ax]$ *aus der Axiomenmenge* Ax^i *der internalisierten Semantik abzuleiten, das heißt es ist zu zeigen:*

$$Ax^i \blacktriangleright \Phi[ax] \equiv Ax^i \blacktriangleright \forall \texttt{db:Db. OK db} \Rightarrow \varphi[ax]$$

Aus Präsentationsgründen werden die im folgenden angegebenen Beweisbäume in ihre Teilbäume aufgespalten dargestellt.

Fall 1: $ax \in Ax^s_{Int} = \{ax_1, \ldots, ax_k\}$[1]

$$\dfrac{\dfrac{\dfrac{}{Ax^i \blacktriangleright \forall \texttt{db:Db.OK(db)} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k])}\,(hyp) \quad \dfrac{}{\emptyset \blacktriangleright (\forall x.p) = (\forall^{\perp} x.\delta(x) \Rightarrow p)}\,(ALL\_def)}{Ax^i \blacktriangleright \forall^{\perp} \texttt{db:Db. } \delta\texttt{(db)} \Rightarrow \texttt{OK(db)} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k])}\,(subst)}{Ax^i \blacktriangleright \delta\texttt{(db)} \Rightarrow \texttt{OK(db)} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k])}\,(ALLbE)$$

[1] Ax^s_{Int} bezeichnet die Menge der Axiome der statischen Semantik, die die statischen Integritätsbedingungen repräsentieren (siehe Abschnitt 4.1, Seite 43).

$$\dfrac{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k]) \quad \overline{\delta(\texttt{db}) \blacktriangleright \delta(\texttt{db})}\ (hyp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k])}\ (mp)$$

$$\dfrac{\dfrac{\dfrac{\overline{Ax^i \blacktriangleright \forall^{\perp} x.\delta(x) \Rightarrow \delta(\texttt{OK}(x))}\ (hyp)}{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \delta(\texttt{OK(db)})}\ (ALLbE) \quad \overline{\delta(\texttt{db}) \blacktriangleright \delta(\texttt{db})}\ (hyp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK(db)})}\ (mp)$$

$$\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} = (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k]) \quad Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK(db)})}{Ax^i, \delta(\texttt{db}) \blacktriangleright (\texttt{OK(db)} \Rightarrow \varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k]) \wedge (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k] \Rightarrow \texttt{OK(db)})}\ (eq\_to\_imp)$$

$$\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright (\texttt{OK(db)} \Rightarrow \varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k]) \wedge (\varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k] \Rightarrow \texttt{OK(db)})}{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k]}\ (conjunct1)$$

$$\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k] \quad \overline{\texttt{OK(db)} \blacktriangleright \texttt{OK(db)}}\ (hyp)}{Ax^i, \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[ax_1] \wedge \ldots \wedge \varphi[ax_k]}\ (mp)$$

Da $ax \in Ax^s_{Int}$, gibt es ein i ($1 \leq i \leq k$) mit $ax = ax_i$. Durch i-malige Anwendung der Regel (conjunct1) erhält man $Ax^i, \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[ax]$.

$$\dfrac{\dfrac{Ax^i, \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[ax] \quad \dfrac{\dfrac{\dfrac{\overline{Ax^i \blacktriangleright \forall^{\perp} x.\delta(x) \Rightarrow \delta(\texttt{OK}(x))}\ (hyp)}{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \delta(\texttt{OK(db)})}\ (ALLbE) \quad \overline{\delta(\texttt{db}) \blacktriangleright \delta(\texttt{db})}\ (hyp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK(db)})}\ (mp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax]}\ (impI)$$

$$\dfrac{\dfrac{\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax] \quad \overline{\emptyset \blacktriangleright \delta(\delta(\texttt{db}))}\ (strong\_DEF)}{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \varphi[ax]}\ (impI)}{Ax^i \blacktriangleright \forall^{\perp} \texttt{db:Db}.\, \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \varphi[ax]}\ (ALLbI) \quad \dfrac{}{\emptyset \blacktriangleright (\forall x.p) = (\forall^{\perp} x.\delta(x) \Rightarrow p)}\ (ALL\_def)}{Ax^i \blacktriangleright \forall \texttt{db:Db}.\, \texttt{OK(db)} \Rightarrow \varphi[ax]}\ (subst)$$

Fall 2: *$ax \in Ax^s \setminus Ax^s_{Int}$*

In diesem Fall gilt auch $ax \in Ax^i$. Da $ax \notin Ax^s_{Int}$, kommen in ax weder Sorten aus SC^s_E noch Funktionen aus F^s_R vor. Deshalb gilt offensichtlich $ax = \varphi[ax]$.

$$\dfrac{\overline{Ax^i \blacktriangleright \varphi[ax]}\ (hyp)}{Ax^i, \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[ax]}\ (weak)$$

$$\dfrac{\dfrac{Ax^i, \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[ax]}{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax]}\,(impI) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{Ax^i \blacktriangleright \forall^{\perp} x.\delta(x) \Rightarrow \delta(\texttt{OK(x)})}\,(hyp)}{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \delta(\texttt{OK(db)})}\,(ALLbE) \qquad \dfrac{}{\delta(\texttt{db}) \blacktriangleright \delta(\texttt{db})}\,(hyp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK(db)})}\,(mp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax]}\,(impI)$$

$$\dfrac{\dfrac{\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \varphi[ax] \qquad \dfrac{}{\emptyset \blacktriangleright \delta(\delta(\texttt{db}))}\,(strong\_DEF)}{Ax^i \blacktriangleright \delta\,(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \varphi[ax]}\,(impI)}{Ax^i \blacktriangleright \forall^{\perp} \texttt{db : Db}\,.\,\delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \varphi[ax]}\,(ALLbI) \qquad \dfrac{}{\emptyset \blacktriangleright (\forall x.p) = (\forall^{\perp} x.\delta(x) \Rightarrow p)}\,(ALL\_def)}{Ax^i \blacktriangleright \forall \texttt{db : Db}\,.\,\texttt{OK(db)} \Rightarrow \varphi[ax]}\,(subst)$$

□

Satz 3 *Sei $f \in W^{Bool}_{\Sigma^s_s}(\chi)$ eine Σ^s-Formel, $H \subseteq W^{Bool}_{\Sigma^s_s}(\chi)$ eine Menge von Σ^s-Formeln und B ein Beweisbaum für $H \blacktriangleright f$. Dann gibt es einen Beweisbaum $\Psi[B]$, der mit*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f] \blacktriangleright \Phi[f]$$

endet. Informell bedeutet diese Aussage, daß jeder Beweis, der über der Signatur der statischen Semantik geführt werden kann, nach Übersetzung aller beteiligten Formeln mittels Φ und unter Zuhilfenahme der Axiome der internalisierten Semantik über der Signatur der internalisierten Semantik "nachgespielt" werden kann. Die Abbildung Φ nimmt im wesentlichen eine Sortenrelativierung aller gebundenen Entityvariablen $x : \tau$ ($\tau \in SC^s_E$) mit dem Relativierungsprädikat

$$\rho[x] \stackrel{def}{=} (\delta(x) \Rightarrow x \in \texttt{ent'}\tau\texttt{(db)})$$

vor. Im Beweis $\Psi[B]$ müssen jedoch auch alle frei auftretenden Variablen für Entities mittels ρ eingeschränkt werden. Zu diesem Zweck wird in den Prämissen der auftretenden Sequenzen eine Formelmenge

$$\mathcal{ENV}[H, f] = \{\rho[x] | x : \tau \in (FV[H] \cup FV[f]) \wedge \tau \in SC^s_E\}$$

mitgeführt. Die Prämissenmenge $\mathcal{ENV}[H, f]$ enthält also Formeln, die genau für alle in H und f frei auftretenden Entityvariablen die Relativierung mittels ρ vornehmen.

Beweis *Der Beweis erfolgt durch Induktion über die Struktur des Beweisbaums B. Der Kalkül der Sprache* SPECTRUM *ist sehr umfangreich und besteht aus 68 Regeln. Deshalb wird an dieser Stelle nur auf einige typische Fälle eingegangen. Die restlichen Fälle lassen*

sich analog behandeln. Die einzelnen Fälle des folgenden Beweises werden mit den Namen der jeweils betrachteten Kalkülregeln bezeichnet.

Die folgende Konsequenz aus der Totalität des `OK`*-Prädikats wird im vorgestellten Beweis mehrfach verwendet werden:*

$$\dfrac{\dfrac{\dfrac{}{Ax^i \blacktriangleright \forall^{\perp} x.\delta(x) \Rightarrow \delta(\texttt{OK}(x))}\,(hyp)}{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \delta(\texttt{OK}(\texttt{db}))}\,(ALLbE) \quad \dfrac{}{\delta(\texttt{db}) \blacktriangleright \delta(\texttt{db})}\,(hyp)}{Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK}(\texttt{db}))}\,(mp)$$

Der Beweis behandelt 4 der insgesamt 68 Fälle. Es werden dabei je zwei Regeln ohne Prämissen und zwei Regeln mit Prämissen betrachtet. Als Beispiele für Regeln ohne Prämissen werden die Fälle (hyp) und (conj_ax2) ausgewählt.

(hyp) *Der Beweis B besteht aus der Anwendung des logischen Axioms (hyp)*

$$\dfrac{}{H \blacktriangleright f}\,\{f \in H$$

Da $f \in H$, gilt nach Definition von Φ auch $\Phi[f] \in \Phi[H]$ und somit auch $\Phi[H] \blacktriangleright \Phi[f]$. Durch Anwendung der Abschwächungsregel (weak) ergibt sich die Aussage des Satzes:

$$\dfrac{\Phi[H] \blacktriangleright \Phi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H, f] \blacktriangleright \Phi[f]}\,(weak)$$

(conj_ax2) *B besteht aus der Anwendung des logischen Axioms (conj_ax2):*

$$\dfrac{}{\emptyset \blacktriangleright (\texttt{true} \wedge y) = y}$$

Es gilt $H = \emptyset$, $f = ((\texttt{true} \wedge y) = y)$, $\Phi[f] = \forall \texttt{db:Db.OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y]$.

$$\dfrac{\dfrac{\dfrac{}{\emptyset \blacktriangleright (\texttt{true} \wedge \varphi[y]) = \varphi[y]}\,(conj\_ax2)}{\texttt{OK(db)} \blacktriangleright (\texttt{true} \wedge \varphi[y]) = \varphi[y]}\,(weak) \quad Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK(db)})}{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y]}\,(impI)$$

$$\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y] \quad \dfrac{}{\delta(\delta(\texttt{db}))}\,(strong\_DEF)}{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y]}\,(impI)$$

$$\dfrac{\dfrac{Ax^i \blacktriangleright \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y]}{Ax^i \blacktriangleright \forall^{\perp} \texttt{db:Db}.\delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y]}\,(ALLbI) \quad (ALL\_def)}{\dfrac{Ax^i \blacktriangleright \forall \texttt{db:Db.OK(db)} \Rightarrow (\texttt{true} \wedge \varphi[y]) = \varphi[y]}{Ax^i, \Phi[\emptyset], \mathcal{ENV}[H, f] \blacktriangleright \Phi[f]}\,(weak)}\,(subst)$$

Für die anderen Kalkülregeln ohne Prämissen erfolgt die Argumentation analog.

Da die Abbildung Φ im wesentlichen eine Sortenrelativierung der Entitysorten vornimmt, das heißt Quantifizierungen über Variablen von Entitysorten einschränkt, sind die Kalkülregeln, die Quantoren einführen oder eliminieren, die interessantesten Fälle. Deshalb werden im folgenden die Fälle (ALLbI) und (ALLbE), die Einführung beziehungsweise Elimination von Allquantoren vornehmen, behandelt. Im Fall des Existenzquantors verläuft der Beweis analog.

(ALLbI) *B endet mit einer Anwendung der Regel (ALLbI):*

$$\frac{H \blacktriangleright f}{H \blacktriangleright \forall^{\perp} x.f} \{x \notin FV[H]$$

Nach Induktionsannahme gibt es einen Beweis für:

$$Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall \mathtt{db}.\mathtt{OK(db)} \Rightarrow \varphi[f]$$

In einem ersten Schritt wird der Allquantor elimiert und die Prämisse `OK(db)` *auf die linke Seite der Sequenz gebracht:*

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall \mathtt{db}.\mathtt{OK(db)} \Rightarrow \varphi[f] \quad \overline{\emptyset \blacktriangleright (\forall x.p) = (\forall^{\perp} x.\delta(x) \Rightarrow p)}\ (ALL\_def)}{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall^{\perp} \mathtt{db}.\delta(\mathtt{db}) \Rightarrow \mathtt{OK(db)} \Rightarrow \varphi[f]}\ (subst)$$

$$\frac{\dfrac{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall^{\perp} \mathtt{db}.\delta(\mathtt{db}) \Rightarrow \mathtt{OK(db)} \Rightarrow \varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \delta(\mathtt{db}) \Rightarrow \mathtt{OK(db)} \Rightarrow \varphi[f]}\ (ALLbE) \quad \dfrac{}{\delta(\mathtt{db}) \blacktriangleright \delta(\mathtt{db})}\ (hyp)}{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\mathtt{db}) \blacktriangleright \mathtt{OK(db)} \Rightarrow \varphi[f]}\ (mp)$$

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\mathtt{db}) \blacktriangleright \mathtt{OK(db)} \Rightarrow \varphi[f] \quad \overline{\mathtt{OK(db)} \blacktriangleright \mathtt{OK(db)}}\ (hyp)}{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\mathtt{db}), \mathtt{OK(db)} \blacktriangleright \varphi[f]}\ (mp)$$

In welcher Weise nun der Allquantor eingeführt werden kann, hängt von der Sorte τ der Variable x ab.

Fall $\tau \notin SC^s_E$: *Die Variable x, über die der neue Quantor läuft, muß nicht eingeschränkt werden.*

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\mathtt{db}), \mathtt{OK(db)} \blacktriangleright \varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\mathtt{db}), \mathtt{OK(db)} \blacktriangleright \forall^{\perp} x.\varphi[f]}\ (ALLbI)$$

Diese Regelanwendung ist gerechtfertigt, da die Variable x nicht frei in den Formeln, die in der linken Seite der Sequenz auftreten, vorkommt. Nun kann die Prämisse

$\texttt{OK(db)}$ *wieder auf die linke Seite gebracht und* $\texttt{db}$ *mittels eines* $\forall$*-Quantors gebunden werden.*

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \forall^{\perp} x.\varphi[f] \quad Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\texttt{OK(db)})}{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]} \; (impI)$$

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f], \delta(\texttt{db}) \blacktriangleright \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f] \quad \overline{\emptyset \blacktriangleright \delta(\delta(\texttt{db}))} \; (strong\_DEF)}{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]} \; (impI)$$

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall^{\perp} \texttt{db}.\, \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]} \; (ALLbI)$$

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall^{\perp} \texttt{db}.\, \delta(\texttt{db}) \Rightarrow \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f] \quad (ALL\_def)}{Ax^i, \Phi[H], \mathcal{ENV}[H,f] \blacktriangleright \forall \texttt{db}.\, \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]} \; (subst)$$

Da $\Phi[\forall^{\perp} x.f] = \forall \texttt{db.OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]$ *und* $\mathcal{ENV}[H,f] = \mathcal{ENV}[H, \forall^{\perp} x.f]$*, gilt auch:*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f] \blacktriangleright \Phi[\forall^{\perp} x.f]$$

Damit ist die Aussage des Satzes bewiesen.

Fall $\boldsymbol{\tau \in SC^s_E}$***:*** *Da die Abbildung* φ *eine Relativierung von Entitysorten vornimmt, muß auch die Quantifizierung mit der entsprechenden Beschränkung eingeführt werden. Die Variable* x *wird durch die Eigenschaft* $\rho[x] = (\delta(x) \Rightarrow x \in \texttt{ent'}\tau\texttt{(db)})$ *eingeschränkt. Für den Beweis dieses Falles wird folgende Aussage über* ρ *benötigt:*

Hilfssatz *Das Sortenrelativierungsprädikat* $\rho[x]$ *ist für definierte Datenbankzustände immer definiert, das heißt es gilt:*

$$Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\rho[x])$$

Beweis *Es ist auf den ersten Blick überraschend, daß die Aussage des Satzes ohne Annahmen über die Definiertheit der Variablen* x *geführt werden kann. Die Formel* $\rho[x]$ *muß also auch für* $x = \perp$ *gelten. Der Beweis wird deshalb mittels Fallunterscheidung* $x = \perp \vee x \neq \perp$ *geführt:*

$\boldsymbol{x = \perp}$***:*** *Die Argumentation in diesem Fall ist, daß* $\delta(\perp) = \texttt{false}$ *und damit* $\rho[\perp] = (\delta(\perp) \Rightarrow \perp \in \texttt{ent'}\tau\texttt{(db)}) = \texttt{true}$*, unabhängig davon, ob die rechte Seite der Implikation definiert ist oder nicht. Formal läßt sich diese Aussage wie folgt zeigen:*

$$\dfrac{\dfrac{\dfrac{}{\emptyset\blacktriangleright \bot = \bot}\,(refl)}{\emptyset\blacktriangleright \neg(\neg(\bot = \bot))}\,(notnotI) \quad \dfrac{}{\emptyset\blacktriangleright \delta(\bot) = \neg(\bot = \bot)}\,(DEF\_def)}{\dfrac{\emptyset\blacktriangleright \neg(\delta(\bot))}{\emptyset\blacktriangleright \neg(\delta(\bot)) = \texttt{true}}\,(TT\_I)}\,(subst)$$

Einfache aussagenlogische Umformungen, die hier nicht formal gezeigt werden, liefern

$$\emptyset\blacktriangleright \texttt{false} = \delta(\bot)$$

Damit gilt dann:

$$\dfrac{\emptyset\blacktriangleright \texttt{false} = \delta(\bot) \quad \dfrac{\dfrac{}{\emptyset\blacktriangleright (\texttt{false} \Rightarrow \bot \in \texttt{ent'}\tau\,\texttt{(db)}) = \texttt{true}}\,(impl\_ax2)}{\emptyset\blacktriangleright (\delta(\bot) \Rightarrow \bot \in \texttt{ent'}\tau\,\texttt{(db)}) = \texttt{true}}\,(subst)}{\dfrac{\emptyset\blacktriangleright \texttt{true} = (\delta(\bot) \Rightarrow \bot \in \texttt{ent'}\tau\,\texttt{(db)})}{\emptyset\blacktriangleright \delta(\delta(\bot) \Rightarrow \bot \in \texttt{ent'}\tau\,\texttt{(db)})}\,(sym) \quad \dfrac{}{\emptyset\blacktriangleright \delta\,\texttt{(true)}}\,(DEF\_TT)}\,(subst)$$

***x* ≠ ⊥:** *Für definiertes x folgt die Aussage aus der Totalität der Funktionen* ent'τ *und* $.\in.$ *sowie des Junktors* $\Rightarrow$*. Auf eine formale Darstellung des Beweises wird an dieser Stelle verzichtet.* □

Nun kann mit dem Beweis der Aussage des Satzes fortgefahren werden. Dazu wird unterschieden, ob die Variable x, die mit $\forall^{\bot}$ *gebunden werden soll, in f frei vorkommt oder nicht.*

Fall x∈FV[f]: *In diesem Fall gilt* $\rho[x] \in \mathcal{ENV}[H, f]$.

$$\dfrac{Ax^i, \delta(\texttt{db})\blacktriangleright \delta(\rho[x]) \quad Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright \varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H, f] \setminus \rho[x], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright \rho[x] \Rightarrow \varphi[f]}\,(impI)$$

Wegen $x \notin FV[H]$ *gilt* $\mathcal{ENV}[H, f] \setminus \rho[x] = \mathcal{ENV}[H, \forall^{\bot} x.f]$ *und damit*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\bot} x.f], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright \rho[x] \Rightarrow \varphi[f]$$

Nun kann der $\forall^{\bot}$*-Quantor eingeführt werden:*

$$\dfrac{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\bot} x.f], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright \rho[x] \Rightarrow \varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\bot} x.f], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright \forall^{\bot} x.\rho[x] \Rightarrow \varphi[f]}\,(ALLbI)$$

Wird jetzt die Prämisse OK(db) *wieder auf die rechte Seite gebracht und* db *mittels* $\forall$ *gebunden, so ergibt sich*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\bot} x.f]\blacktriangleright \forall \texttt{db.OK(db)} \Rightarrow \forall^{\bot} x.\rho[x] \Rightarrow \varphi[f]$$

Damit ist die Aussage des Satzes bewiesen.

Fall $x \notin FV[f]$: *In diesem Fall gilt $\rho[x] \notin \mathcal{ENV}[H, f]$.*

$$\dfrac{Ax^i, \delta(\texttt{db}) \blacktriangleright \delta(\rho[x]) \quad \dfrac{Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)}, \rho[x] \blacktriangleright \varphi[f]}\ (weak)}{Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \rho[x] \Rightarrow \varphi[f]}\ (impI)$$

Wegen $\rho[x] \notin \mathcal{ENV}[H, f]$ ist $\mathcal{ENV}[H, f] = \mathcal{ENV}[H, \forall^{\perp} x.f]$ und damit

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \rho[x] \Rightarrow \varphi[f]$$

Wird nun der $\forall^{\perp}$-Quantor eingeführt, `OK(db)` *nach rechts gebracht und* `db` *mittels $\forall$ gebunden, ergibt sich wiederum die Aussage des Satzes:*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f] \blacktriangleright \forall \texttt{db}.\texttt{OK(db)} \Rightarrow \forall^{\perp} x.\rho[x] \Rightarrow \varphi[f]$$

(ALLbE) *B endet mit einer Anwendung der Regel (ALLbE):*

$$\dfrac{H \blacktriangleright \forall^{\perp} x.f}{H \blacktriangleright f[t/x]}$$

Dabei bezeichnet $f[t/x]$ die Substitution des Terms t für die Variable x in f. In dieser Regel wird angenommen, daß durch geeignete Umbenennung gebundener Variablen in f sichergestellt ist, daß durch die Substitution keine unerwünschten Bindungen von Variablen in t auftreten.

Wie schon im Fall ***(ALLbI)*** *hängt der Beweis der Aussage des Satzes von der Sorte τ der Variablen x ab.*

Fall $\tau \notin SC_E^s$: *Nach Induktionsannahme existiert ein Beweis für*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f] \blacktriangleright \forall \texttt{db}.\texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]$$

Wegen $\tau \notin SC_E^s$ gilt $\mathcal{ENV}[H, \forall^{\perp} x.f] = \mathcal{ENV}[H, f]$ und damit

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f] \blacktriangleright \forall \texttt{db}.\texttt{OK(db)} \Rightarrow \forall^{\perp} x.\varphi[f]$$

Analog zum Vorgehen im Fall ***(ALLbI)*** *kann nun der $\forall$-Quantor über* `db` *eliminiert und die Prämisse* `OK(db)` *auf die linke Seite der Sequenz gebracht werden. Damit existiert ein Beweis für*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \forall^{\perp} x.\varphi[f]$$

Nun wird der $\forall^{\perp}$-Quantor eliminiert:

$$\dfrac{Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \forall^{\perp} x.\varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H, f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[f[t/x]]}\ (ALLbE)$$

Wird nun `OK(db)` *wieder als Prämisse auf die rechte Seite gebracht und der* $\forall$*-Quantor über* `db` *eingeführt, entsteht ein Beweis für*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f] \blacktriangleright \forall \texttt{db} : \texttt{OK(db)} \Rightarrow \varphi[f[t/x]]$$

Da $\tau \notin SC^s_E$*, ist* $\mathcal{ENV}[H, f] \subseteq \mathcal{ENV}[H, f[t/x]]$*. Durch Anwendung der Abschwächungsregel (weak) entsteht die Aussage des Satzes:*

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H, f] \blacktriangleright \forall \texttt{db} : \texttt{OK(db)} \Rightarrow \varphi[f[t/x]]}{Ax^i, \Phi[H], \mathcal{ENV}[H, f[t/x]] \blacktriangleright \forall \texttt{db} : \texttt{OK(db)} \Rightarrow \varphi[f[t/x]]} \; (weak)$$

Fall $\boldsymbol{\tau \in SC^s_E}$***:*** *Ist* $\tau \in SC^s_E$*, so gilt folgende Beobachtung:*

Beobachtung (*) *Der Term* t*, der für* x *substituiert wird, ist selbst wieder eine Variable, da die Signatur* Σ^s *keine Funktionen enthält, die es erlauben würden, kompliziertere Terme der Entitysorte* τ *aufzuschichten.*

Nach Induktionsannahme gibt es einen Beweis für

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f] \blacktriangleright \forall \texttt{db} : \texttt{OK(db)} \Rightarrow \forall^{\perp} x.\rho[x] \Rightarrow \varphi[f]$$

Wird der $\forall$*-Quantor eliminiert und die Prämisse* `OK(db)` *nach links gebracht, ergibt sich*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \forall^{\perp} x.\rho[x] \Rightarrow \varphi[f]$$

Nun kann der $\forall^{\perp}$*-Quantor eliminiert werden:*

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \forall^{\perp} x.\rho[x] \Rightarrow \varphi[f]}{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright (\rho[x] \Rightarrow \varphi[f])[t/x]} \; (ALLbE)$$

Wegen $((\rho[x] \Rightarrow \varphi[f])[t/x]) = (\rho[t] \Rightarrow \varphi[f[t/x]])$ *gilt*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \rho[t] \Rightarrow \varphi[f[t/x]]$$

Die nächsten Schritte hängen ab von der Menge der in f *enthaltenen freien Variablen:*

x $\boldsymbol{\in FV[f]}$***:***

$$\frac{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \rho[t] \Rightarrow \varphi[f[t/x]] \quad \overline{\rho[t] \blacktriangleright \rho[t]} \; (hyp)}{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\perp} x.f], \delta(\texttt{db}), \texttt{OK(db)}, \rho[t] \blacktriangleright \varphi[f[t/x]]} \; (mp)$$

Es gilt

$$\mathcal{ENV}[H, \forall^{\perp} x.f] = \mathcal{ENV}[H, f] \setminus \{\rho[x]\}$$

Wegen Beobachtung ***(*)*** *ist* t *eine Variable und es gilt*

$$\mathcal{ENV}[H, f[t/x]] = (\mathcal{ENV}[H, f] \setminus \{\rho[x]\}) \cup \{\rho[t]\}$$

Es gilt also auch

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f[t/x]], \delta(\texttt{db}), \texttt{OK(db)} \blacktriangleright \varphi[f[t/x]]$$

***x*∉*FV*[*f*]:** *Im obigen Beweis des Hilfssatzes wurde bereits gezeigt, daß*

$$\emptyset\blacktriangleright(\delta(\bot)\Rightarrow\bot\in\texttt{ent'}\tau\texttt{(db)}) = \texttt{true}$$

$$\dfrac{\dfrac{(\delta(\bot)\Rightarrow\bot\in\texttt{ent'}\tau\texttt{(db)}) = \texttt{true}}{\delta(\bot)\Rightarrow\bot\in\texttt{ent'}\tau\texttt{(db)}}\ (TT\_E)}{\exists^{\bot}x.\delta(t)\Rightarrow t\in\texttt{ent'}\tau\texttt{(db)}}\ (EXbI)$$

$$\dfrac{\exists^{\bot}x.\rho[t] \quad Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\bot}x.f], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright\rho[t]\Rightarrow\varphi[f[t/x]]}{Ax^i, \Phi[H], \mathcal{ENV}[H, \forall^{\bot}x.f], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright\varphi[f[t/x]]}\ (EXbE)$$

Diese Regelanwendung ist gerechtfertigt, weil $t\not\in FV[\varphi[f[t/x]]]$ *(wegen* $x\not\in FV[f]$*). Da* $x\not\in FV[f]$*, ist*

$$\mathcal{ENV}[H, f[t/x]] = \mathcal{ENV}[H, f] = \mathcal{ENV}[H, \forall^{\bot}x.f]$$

Es gilt also auch

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f[t/x]], \delta(\texttt{db}), \texttt{OK(db)}\blacktriangleright\varphi[f[t/x]]$$

Nun kann `OK(db)` *wieder nach rechts gebracht und* `db` *mittels* $\forall$ *gebunden werden. Damit ergibt sich der gesuchte Beweis für*

$$Ax^i, \Phi[H], \mathcal{ENV}[H, f[t/x]]\blacktriangleright\forall\texttt{db.OK(db)}\Rightarrow\varphi[f[t/x]]$$

Für die anderen Kalkülregeln (insbesondere die Regeln (EXbI) und (EXbE) zur Einführung beziehungsweise Elimination des Existenzquantors) ist die Argumentation analog. □

Literaturverzeichnis

[ABM88] M. P. Atkinson, P. Buneman und R. Morrison (Herausgeber). „Data Types and Persistence“, Topics in Information Systems. Springer (1988).

[ANS78] ANSI/X3/SPARC/Study Group — Database Management Systems. The ANSI/X3/SPARC DBMS Framework. *Information Systems* **3**(3), 173–191 (1978).

[Atr88] Shaku Atre. „Data Base: Structured Techniques for Design, Performance, and Management“. John Wiley & Sons (1988).

[BBS93] F. L. Bauer, W. Brauer und H. Schwichtenberg (Herausgeber). „Logic and Algebra of Specification“, Band 94 aus „NATO ASI Series F: Computer and Systems Sciences“. Springer (1993).

[BCN92] C. Batini, S. Ceri und S. B. Navathe. „Conceptual Database Design: an Entity-Relationship Approach“. Benjamin/Cummings (1992).

[BFG+93a] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch und K. Stølen. The Requirement and Design Specification Language SPECTRUM. An Informal Introduction. Version 1.0. Part I. Bericht TUM-I9311, Technische Universität München. Institut für Informatik (Mai 1993).

[BFG+93b] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch und K. Stølen. The Requirement and Design Specification Language SPECTRUM. An Informal Introduction. Version 1.0. Part II. Bericht TUM-I9312, Technische Universität München. Institut für Informatik (Mai 1993).

[BG84] F. L. Bauer und G. Goos. „Informatik. Eine einführende Übersicht“, Band 2. Springer (1984).

[BHS92] U. Bittner, W. Hesse und J. Schnath. Untersuchungen zum Methodeneinsatz in Software-Entwicklungsprojekten. *GI Softwaretechnik-Trends* **12**(3), 48–60 (August 1992).

[BJ95] M. Broy und St. Jähnichen (Herausgeber). „KORSO, Correct Software by Formal Methods“. (1995). Submitted to Lecture Notes in Computer Science.

[BLN86] C. Batini, M. Lenzerini und S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys* **18**(4), 323–364 (1986).

[Bro89] M. Broy (Herausgeber). „Constructive Methods in Computing Science“, Band 55 aus „NATO ASI Series F: Computer and Systems Sciences“. Springer (1989).

[BW82] F. L. Bauer und H. Wössner. „Algorithmic Language and Program Development“. Springer (1982).

[BW93] M. Broy und M. Wirsing. Korrekte Software: vom Experiment zur Anwendung. In Reichel [Rei93].

[Car84] L. Cardelli. A Semantics of Multiple Inheritance. In „Proc. Semantics of Data Types“, Seiten 51–68. Springer (1984).

[CCT90] CCTA (Herausgeber). „SSADM Version 4 Reference Manual“. NCC Blackwell (1990).

[Che76] P. Chen. The entity-relationship model — toward a unified view of data. *ACM Trans. on Database Systems* **1**(1), 9–36 (1976).

[CHL95] F. Cornelius, H. Hußmann und M. Löwe. The KORSO Case Study for Software Engineering with Formal Methods: A Medical Information System. In Broy und Jähnichen [BJ95]. Submitted to Lecture Notes in Computer Science.

[DCC92] Ed Downs, Peter Clare und Ian Coe. „Structured Systems Analysis and Design Method — Application and Context“. Prentice Hall (1992).

[DeM79] T. DeMarco. „Structured analysis and systems specification“. Prentice-Hall (1979).

[Dij76] E. W. Dijkstra. „A Discipline of Programming“. Prentice-Hall (1976).

[Dil94] A. Diller. „Z. An Introduction to Formal Methods“. John Wiley & Sons (1994).

[EGH+90] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake und H.-D. Ehrich. Conceptual Modelling of Database Applications Using an Extended ER Model. Bericht Informatik-Berichte 90-05, TU Braunschweig (1990).

[EKT93] R. A. Elmasri, V. Koumramajian und B. Thalheim (Herausgeber). „Entity-Relationship Approach — ER '93“, Band 823 aus „Lecture Notes in Computer Science“. Springer (1993).

[End72] H. B. Enderton. „A Mathematical Introduction to Logic“. Academic Press (1972).

[Eva92] M. Eva. „SSADM Version 4: A User's Guide“. McGraw-Hill (1992).

[Fuc94] Th. Fuchß. Translating E/R-diagrams into Consistent Database Specifications. Interner Bericht 2/94, Universität Karlsruhe (1994).

[Gen35] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift* **39**, 176–210,405–431 (1935).

[GH91] M. Gogolla und U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Transactions on Database Systems* **16**(3), 369–416 (1991).

[GHN+94] R. Grosu, R. Hettler, D. Nazareth, F. Regensburger und O. Slotosch. The Specification Language SPECTRUM — Language Report V 1.0. Bericht TUM-I9429, Technische Universität München (1994).

[Gin92] J. Ginbayashi. Analysis of business processes specified in Z against an E-R data model. Bericht PRG-103, Oxford University Computing Laboratory, Oxford (1992).

[Gog94] M. Gogolla. „An Extended Entity-Relationship Model. Fundamentals and Pragmatics“, Band 767 aus „Lecture Notes in Computer Science“. Springer (1994).

[GR94] R. Grosu und F. Regensburger. The Logical Framework of SPECTRUM. Bericht TUM-I9402, Technische Universität München (1994).

[Gri81] D. Gries. „The Science of Programming“. Springer (1981).

[GTWW75] J. A. Goguen, J. W. Thatcher, E. G. Wagner und J. G. Wright. Abstract Data-Types as Initial Algebras and Correctness of Data Representations. In „Proceedings Conference on Computer Graphics, Pattern Recognition and Data Structure“ (1975).

[Gur91] Y. Gurevich. Evolving Algebras — A Tutorial Introduction. *Bulletin of the EATCS* **43**, 264 – 284 (1991).

[Gut75] J. V. Guttag. „The Specification and Application to Programming of Abstract Data Types“. Dissertation, Department of Computer Science, Universtiy of Toronto (1975).

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* **8**, 231–274 (1987).

[Het93] R. Hettler. Zur Übersetzung von E/R-Schemata nach SPECTRUM. Bericht TUM-I9333, Technische Universität München (1993).

[Hoh93] U. Hohenstein. „Formale Semantik eines erweiterten Entity-Relationship-Modells“, Band 4 aus „Teubner-Texte zur Informatik“. Teubner (1993).

[HPW92] P. Hudak, S. Peyton Jones und P. Wadler (Herausgeber). „Report on the Programming Language Haskell, A Non-strict Purely Functional Language (Version 1.2)“. ACM SIGPLAN Notices (May 1992).

[Hub94] F. Huber. Ein generisches Werkzeug zur Bearbeitung von Graphen und deren Umsetzung in Textdarstellungen. Diplomarbeit, TU München (1994).

[Huß93a] H. Hußmann. Synergy between formal and pragmatic software engineering methods. Bericht TUM-I9323, Institut für Informatik, Technische-Universität München (September 1993).

[Huß93b] H. Hußmann. Zur formalen Beschreibung der funktionalen Anforderungen an ein Informationssystem. Bericht TUM-I9332, Technische Universität München (1993).

[Huß94] H. Hußmann. „Formal Foundations for SSADM“. Habilitationsschrift, TU München (1994).

[Inb83] M. Inbal. Application of the ISBN (International Standard Book Number) in a Computerized Library Acquisition System. In Keren und Perlmutter [KP83], Seiten 395–400.

[Ins86] American National Standard Institute. Database Language — SQL (1986).

[Jon86] C. B. Jones. „Systematic Software Develepment Using VDM“. Prentice-Hall (1986).

[JRP91a] M. B. Josephs und D. Redmond-Pyle. Entity-Relationship Models Expressed in Z: A Synthesis of Structured and Formal Methods. Bericht PRG-TR-20-91, Oxford University Programming Research Group (1991).

[JRP91b] M. B. Josephs und D. Redmond-Pyle. A Library of Z Schemas for use in Entity-Relationship Modelling. Bericht PRG-TR-21-91, Oxford University Programming Research Group (1991).

[KG90] U. Karge und M. Gogolla. Formal Semantics of SQL Queries. Bericht 90-01, TU Braunschweig (1990).

[KP83] C. Keren und L. Perlmutter (Herausgeber). „The Application of Mini- and Micro-Computers in Information, Documentation and Libraries“. Amsterdam, North Holland (1983).

[Lee90] J. Leeuwen (Herausgeber). „Handbook of Theoretical Computer Science. B Formal Models and Semantics“. Elsevier (1990).

[LPT89] Z. Luo, R. Pollack und P. Taylor. How to Use LEGO. *Departement of Computer Science, University of Edinburgh* (1989).

[LS87] J. Loeckx und K. Sieber. „The Foundations of Program Verification“. Wiley-Teubner (1987).

[Mai83] D. Maier. „The Theory of Relational Databases“. Computer Science Press (1983).

[Nic93] F. Nickl. Ablaufspezifikation durch Datenflußmodellierung und stromverarbeitende Funktionen. Bericht TUM-I9334, Technische Universität München (1993).

[Par90] H. A. Partsch. „Specification and Transformation of Programs. A Formal Approach to Software Development“. Springer (1990).

[Pau94] Lawrence C. Paulson. „Isabelle: A Generic Theorem Prover“, Band 828 aus „Lecture Notes in Computer Science“. Springer (1994).

[PBDD94] P. Pepper, R. Betschko, S. Dick und K. Didrich. Realizing Sets by Hash Tables: How to do it in KorSo. Bericht 94-30, TU Berlin (1994).

[PT92] G. Pernul und A M. Tjoa (Herausgeber). „Entity-Relationship Approach — ER '92“, Band 645 aus „Lecture Notes in Computer Science“. Springer (1992).

[PW95] P. Pepper und M. Wirsing. KORSO: a Method for the Development of Correct Software. In Broy und Jähnichen [BJ95]. Submitted to Lecture Notes in Computer Science.

[PWM93] F. Polack, M. Whiston und K. Mander. The SAZ Project: Integrating SSADM and Z. In Woodcock und Larsen [WL93], Seiten 541–557.

[PWM94] F. Polack, M. Whiston und K. Mander. „The SAZ Method, Version 1.1“. University of York (January 1994).

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy und W. Lorensen. „Object-oriented modelling and design“. Prentice-Hall (1991).

[Reg94] F. Regensburger. The Calculus of SPECTRUM. Bericht TUM-I9424, Technische Universität München (1994).

[Rei93] H. Reichel (Herausgeber). „Informatik — Wirtschaft — Gesellschaft. 23. GI-Jahrestagung, Dresden“. Springer (1993).

[Rum91] B. Rumpe. Spezifikation eines touristischen Buchungssystems: Eine SPECTRUM Fallstudie. Diplomarbeit, TU München (1991).

[Sal95] Mohsen Salhi. Anwendungsbeispiele für den generischen Graph-Editor GENESYS. Fortgeschrittenenpraktikum, TU München (1995).

[Sho67] J. R. Shoenfield. „Mathematical Logic“. Addison-Wesley (1967).

[Slo95] O. Slotosch. Implementing the Change of Data Structures with SPECTRUM in the Framework of KORSO Development Graphs. Bericht TUM-I9511, TU München (1995).

[SN90] F. Schönthaler und T. Németh. „Software-Entwicklungswerkzeuge: Methodische Grundlagen“. Teubner (1990).

[SNM+93] O. Slotosch, F. Nickl, S. Merz, H. Hußmann und R. Hettler. Die funktionale Essenz von HDMSA. Bericht TUM-I9335, Technische Universität München (1993).

[Spi92] J. M. Spivey. „The Z Notation: A Reference Manual“. Prentice-Hall (1992).

[SS83] G. Schlageter und W. Stucky. „Datenbanksysteme: Konzepte und Modelle“. Teubner, Stuttgart (1983).

[SW83] D. Sannella und M. Wirsing. A Kernel Language for Algebraic Specification and Implementation. Bericht CSR-131-83, University of Edinburgh (1983).

[Teo90] T. Teorey. „Database Modeling and Design: The Entity-Relationship Approach“. Morgan Kaufman (1990).

[TF82] T. J. Teorey und J. P. Fry. „Design of Database Structures“. Prentice-Hall (1982).

[TK81] D. Tsichritzis und A. Klug (Herausgeber). „The ANSI/X3/SPARC DBMS Framework — Report of the Study Group on Database Management Systems“. AFIPS PRESS, New Jersey (1981).

[TM87] W. M. Turski und T. S. E. Maibaum. „The Specification of Computer Programs“. Addison-Wesley (1987).

[Wed81] H. Wedekind. „Datenbanksysteme I“, Band 16 aus „Reihe Informatik“. B.I.-Wissenschaftsverlag (1981).

[Wir92] M. Wirsing. A Framework for Software Development in Korso. Bericht 9205, Ludwig-Maximilians-Universität München (1992).

[WL93] F. C. P. Woodcock und P. G. Larsen (Herausgeber). „FME '93“, Band 670 aus „Lecture Notes in Computer Science“. Springer (1993).

[Zam94a] A. Zamulin. The Database Specification Language RUSLAN (a preliminary communication). Bericht Preprint 28, Siberian Division of the Russian Academy of Sciences (1994).

[Zam94b] A. Zamulin. The Database Specification Language RUSLAN (specification examples). Bericht Preprint 29, Siberian Division of the Russian Academy of Sciences (1994).