„Objektorientierung als methodisch-operativer Ansatz zur Erstellung und Integritätssicherung von VLSI-Weltmodellen“

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

vom Fachbereich Elektrotechnik und Informatik der
Universität-Gesamthochschule Siegen
genehmigte

Dissertation

Erstgutachter: Prof. Dr.-Ing. Hans Wojtkowiak
Zweitgutachter: Prof. Dr. rer. nat. Wolfgang Merzenich

von
Dipl.-Ing. Klaus Quibeldey-Cirkel
geb. in Herten/Westfalen

Tag der mündlichen Prüfung: 14. September 1994

Vorwort

„I feel as if I have ventured into unknown territory, where few maps or roads yet exist. I definitely would have reached my destination more safely and comfortably if I had instead chosen a paved road to my doctor's degree; surely a new and improved algorithm for simulation or layout of integrated circuits would have been more easily recognizable and less difficult to ‚sell'. On the other hand, the adventure of seeing familiar landscapes from new perspectives offers more personal satisfaction, and more of the same thrill that Orville and Wilbur Wright must have felt when they first viewed the land from an airplane."

Karl KLINGSHEIM [Klingsheim, 1990]

Die Faszination der Informatik geht heute vom *Objekt* aus. Dem Reiz der Objektorientierung kann sich kaum jemand mehr entziehen, zu umfassend ist die neue Sicht der Dinge. Der Bogen spannt sich zwischen Wissenschaft und Technik, und die junge Informatik erhebt so ihren Anspruch, eine *Technik-Wissenschaft* zu sein [Luft, 1988]. Selten zuvor wurden Theorie und Praxis von einem einzigen *Paradigma* überbrückt, das heißt von einer gemeinschaftlichen Denk- und Handlungsweise. Der objektorientierte Ansatz löst die perspektivischen Widersprüche auf: Er vereint den wissenschaftlichen *Generalisten* mit dem professionellen *Spezialisten*. Dieser umfassende Ansatz trägt derzeit die Hoffnung auf eine „Wissenschaft des Entwerfens". Das Ziel ist der Entwurf *komplexer* Artefakte, sei es Soft- oder Hardware. „Creating the Artificial" nennt es der Nobelpreisträger Herbert SIMON [Simon, 1982] — „Informatik-Systemtechnik" ist die aktuelle Variation des Themas [Schweizer & Thomé, 1993].

Die vorliegende Studie plädiert keineswegs für einen „Alleingeltungsanspruch" der Objektorientierung. Das wäre vermessen und unrealistisch zugleich! Die Objektorientierung ist zwar omnipräsent, aber nicht omnipotent; sie ist eine probate Denk- und Handlungsweise, um besonders *datenintensive* und *komplex strukturierte* Probleme *durchgängig* zu lösen. Die konsistente VLSI-Datenmodellierung ist ein solches Problem.

Die Studie als Form wissenschaftlichen Arbeitens bot sich aus zwei Gründen an: Zum ersten konnte ich im BMFT-Forschungsprojekt DASSY[1] eigenständige Studien auf dem Gebiet der objektorientierten *Hardwaremodellierung* betreiben. Zum zweiten ließen sich die Projektergebnisse mit den Konzepten der objektorientierten *Softwaremodellierung* zu VLSI-*Weltmodellen* vereinen. Vorgehensweise und Ausarbeitung tragen folglich die Züge

[1] DASSY steht für „Datentransfer und Schnittstellen in offenen integrierten VLSI-Entwurfssystemen", Förderkennzeichen NT 2860 I6.

einer wissenschaftlichen Studie: Im Sinne von Umberto ECO ist sie eigenständige Forschung und kompilatorische Literaturarbeit zugleich [Eco, 1990].

Zwei Anmerkungen zur sprachlichen Gestaltung:

- Objektorientierung ist einerseits populärwissenschaftlich, andererseits Gegenstand der Forschung, somit ein techno-philosophisches Thema *par excellence*. Eine Abhandlung über dieses Thema richtet sich primär an das Kollegium, schließt aber auch andere Fachgruppen mit ein. Wie sollte man also in der Darlegung des Themas von sich selbst sprechen? Im bescheidenen Plural (*Pluralis modestiae*) oder aus der Sicht des Schreibers (*auktoriales Ich*)? Thomas KUHN hat den Begriff von der „wissenschaftlichen Gemeinschaft“ und dem „Paradigmenwechsel“ geprägt [Kuhn, 1989]. Die wissenschaftliche Gemeinschaft des Objekt-Paradigmas sprengt die herkömmlichen engen Grenzen: Der gegenwärtige Wechsel vom Struktur- zum Objektansatz betrifft Forscher, Praktiker, Lehrende und Studierende der Informatik. So habe ich mich für ein „auktoriales Wir“ entschieden.
- Die zweite Anmerkung bezieht sich auf das *Genus* der Akteure. Die Motive der Frauenbewegung, Einfluß auf die Sprache zu gewinnen, sind zweifellos ehrenwert, aber die Sprache fügt sich ihnen nur bedingt: Unsere Sprache mit aller Konsequenz emanzipieren zu wollen, würde sie „mit kaum erträglicher Umständlichkeit und oft lächerlichem Klang“ belasten, wie Wolf SCHNEIDER in seiner Stilkunde betont [Schneider, 1988]. „Das Wörtchen *man* ist als maskulin entlarvt und soll durch *man/frau* oder durch *mensch* ersetzt werden [...] Da der Mann sogar in *jemand* und *niemand* steckt — sollen wir *niefrau* oder *niemensch* sagen?“ Folglich bleibe ich bei den noch üblichen Wendungen und maskulinen Fürwörtern.

Ein Wort der Danksagung: Ich verdanke es Herrn Professor Dr.-Ing. Hans Wojtkowiak, daß ich nach mehreren Jahren universitärer Abstinenz an die Alma mater zurückkehren konnte, um als wissenschaftlicher Mitarbeiter zu promovieren (so blieb mir die Ochsentour einer Industrie-Promotion erspart). Seine offene Einstellung zu einem Thema, das sowohl auf interdisziplinäre *Breite* als auch auf intradisziplinäre *Tiefe* angelegt ist, war mir eine große Hilfe. Herrn Professor Dr. rer. nat. Wolfgang Merzenich danke ich für die spontane Übernahme des Korreferats. Seine konstruktive Kritik zu den mengentheoretischen Aspekten gab das rechte Maß zwischen inhaltlicher und formaler Betrachtung. Herrn Professor Dr.-Ing. Rainer Kammüller danke ich für die Übernahme des Vorsitzes in der Promotionskommission. Meinem Kollegen, Herrn Dipl.-Ing. Walter Lang, schließlich sei Dank für sein „unerschrockenes“ Probelesen und für zahlreiche Grundsatz-Diskussionen.

Marburg, im Sommer 1994 Klaus Quibeldey-Cirkel

Inhaltsverzeichnis

Vorwort iii

Exposé 1
Hintergründe . . . 2
Thema und Thesen . . . 6
Gang der Untersuchung . . . 8

I Das Problem: Komplexität versus Integrität 11

1 Die Natur der Komplexität 12
1.1 Deskriptiver Pluralismus . . . 12
1.2 Komplexität und ihre Dimensionen . . . 16
1.2.1 Begriffsrahmen . . . 17
1.2.2 Metriken . . . 20
1.3 Entwurfskomplexität . . . 25
1.3.1 Die Essenz der Software-Komplexität . . . 26
1.3.2 Die Essenz der Hardware-Komplexität . . . 28

2 Komplexitätsbewältigung I: Reduktionismus 33
2.1 Die „Magische Zahl Sieben“ . . . 34
2.1.1 „Chunking“ . . . 36
2.2 Die „Architektur des Komplexen“ . . . 38
2.2.1 „Near Decomposability“ . . . 38
2.2.2 Allgemeines Systemmodell . . . 40
2.3 Dekomposition: „Divide et Impera“ . . . 41
2.3.1 Randbedingungen . . . 41
2.3.2 Ökonomie . . . 44
2.3.3 „Information hiding“ . . . 46
2.3.4 Evolution: Vom Handentwurf zum „Silicon-Compiler“ . . . 47

3 Komplexitätsbewältigung II: Integration & Integrität 50
3.1 Bezugsrahmen: CAD-Infrastrukturen . . . 51
3.1.1 Schnittstellen . . . 53
3.1.2 Projektlandschaft . . . 56
3.2 Integrität von VLSI-Datenhaltungen . . . 59
3.2.1 Drei-Schema-Konzept . . . 60
3.2.2 Integrität = Realitätstreue + Vollständigkeit . . . 66
3.3 Referenzmodell der Datenintegrität . . . 70
3.3.1 Zustand und Veränderung . . . 70
3.3.2 Statische und dynamische Integrität . . . 71

4 Komplexitätsbewältigung III: Konzeptionelles Modellieren 74
4.1 Paradigmenwechsel im großen . . . 75
4.2 Kuhns These . . . 76
4.3 Paradigmenwechsel im kleinen . . . 78
4.3.1 Die methodenlose Zeit . . . 79
4.3.2 Programmierkunst versus Software-Engineering . . . 80
4.3.3 Der Faktor „Mensch“ . . . 81

4.3.4 SA/SD versus OOx . . . 82
4.4 Objektorientierte Weltmodelle . . . 86
4.4.1 Ausführbare Weltmodelle . . . 88
4.4.2 Szenario des objektorientierten Entwurfs . . . 88

II Objektorientierung als methodischer Ansatz 91

5 Von einer Wissenschaft des Entwerfens 92
5.1 „Creating the Artificial“ . . . 93
5.1.1 Curriculum einer Wissenschaft des Entwerfens . . . 94
5.1.2 Der Beitrag des Objekt-Paradigmas . . . 102
5.2 Der architektonische Entwurf . . . 103
5.2.1 Ideale einer verallgemeinerten Entwurfslehre . . . 104
5.2.2 Der gute Entwurf aus architektonischer Sicht . . . 105
5.2.3 Der Beitrag des Objekt-Paradigmas . . . 108
5.3 Die Ontologie des Entwerfens . . . 109
5.3.1 Tiefenstruktur: Zustände, Ereignisse, Gesetze . . . 110
5.3.2 Der gute Entwurf aus ontologischer Sicht . . . 112

6 Konventionelle Modelle des Entwerfens 115
6.1 Kybernetische Modelle . . . 116
6.1.1 Regelungstechnisches Modell nach Amkreutz . . . 117
6.2 Informationstheoretische Modelle . . . 120
6.2.1 Informationstheoretisches Modell nach Conant . . . 122
6.2.2 Informationstheoretisches Modell nach Koomen . . . 124
6.3 Verfahrensmodelle . . . 126
6.3.1 Y-Diagramm nach Gajski et al. . . . 126
6.4 Einordnung der Vorschläge . . . 128
6.4.1 Allgemeine Modelltheorie nach Stachowiak . . . 128
6.4.2 Gegenüberstellung . . . 130

7 Entwerfen als Problemlösen 132
7.1 Was ist ein Problem? . . . 135
7.1.1 Interpolationsprobleme . . . 135
7.1.2 Syntheseprobleme . . . 136
7.1.3 Dialektische Probleme . . . 137
7.2 Kognitive Strukturen . . . 137
7.2.1 Kognitive Abbildungen . . . 139
7.2.2 Kognitive Operationen . . . 141
7.3 Schema & Korrektur . . . 143
7.3.1 Was ist ein Schema? . . . 143
7.3.2 „Making-Matching-Remaking“ . . . 144
7.4 Der Beitrag des Objekt-Paradigmas . . . 146
7.4.1 Objektbegriff . . . 148
7.4.2 Vermeidung früher Fixierung . . . 151
7.4.3 Analogieschluß . . . 154

8 VLSI/CAD-Weltmodelle 158
8.1 Der objektorientierte VLSI/CAD-Entwurf . . . 159
8.1.1 VLSI/CAD- versus Objektmodell . . . 160
8.2 Informationsmodellierung . . . 167

8.3 Datenaustauschformate . . . 170
8.4 Prozedurale Schnittstellen . . . 172

III Objektorientierung als operativer Ansatz 177

9 Constraint-basiertes Entwerfen 178
9.1 „Synthesis of Form“ . . . 179
9.1.1 Reduktion der Vielfalt: Constraint-Propagierung . . . 180
9.1.2 Der VLSI-Entwurf als CSP: ein Beispiel . . . 183
9.2 Constraint-Aspekte . . . 186
9.2.1 Constraints versus Kreativität . . . 187
9.2.2 Constraints versus Integrität . . . 189
9.3 Variationen des Themas . . . 193
9.3.1 Constraints und Problemlösen . . . 193
9.3.2 Constraints und architektonisches Entwerfen . . . 197
9.3.3 Constraints und Softwaretechnik . . . 199

10 Das Tripel des Objekts 202
10.1 Synergie aus KI und Objektorientierung . . . 203
10.1.1 Ein frühes Beispiel: ThingLab . . . 204
10.2 Struktur . . . 206
10.2.1 Objektorientiertes Strukturieren . . . 207
10.2.2 Abstrahieren . . . 207
10.2.3 Teilen . . . 218
10.2.4 VLSI-Datenmodell nach Batory & Kim . . . 221
10.3 Verhalten . . . 224
10.3.1 Funktions- versus Klassenbibliothek . . . 225
10.3.2 Operator-Operanden-Modell versus Smalltalk-Modell . . . 227
10.3.3 Client-Server-Vertrag . . . 229
10.4 Rationale Beschränkung . . . 232
10.4.1 Beispiele zur rationalen Objektbeschränkung . . . 233
10.4.2 Constraint-basierte Integritätssicherung . . . 236
10.4.3 Objektorientierte Constraint-Modelle . . . 239

11 Resümee und Ausblick 245
11.1 Zusammenfassung der Ergebnisse . . . 246
11.2 Einordnung der Ergebnisse . . . 249
11.3 Weiterführende Forschung: „Informatik-Systemtechnik“ . . . 251

Anhang 255

A Metaphorik der Objektorientierung 256
A.1 Metapher: Sprechblume oder Bedeutungssprung? . . . 256
A.2 Metaphern der Objektorientierung . . . 258

B Ontologische Konzepte 260
B.1 Die Begriffswelt der Ontologie . . . 261
B.2 Ein ontologisches Objektmodell . . . 263

267
Abkürzungen und Literatur . . . 268

Exposé

„Versucht man heute, den Zustand der Informatik mit einem Wort zu charakterisieren, so stößt man unweigerlich auf das Wort ‚orientiert'. Denn viele Konzepte und Methoden der Informatik betreibt man anweisungsorientiert, funktionsorientiert, logikorientiert, systemorientiert, modellorientiert, sicherheitsorientiert, KI-orientiert, objektorientiert, netzorientiert, musterorientiert, datenorientiert, applikationsorientiert usw. Das Wort ‚orientiert' offenbart einen Wesenszug der heutigen Informatik: Sie ist keine in sich ruhende abgeschlossene Wissenschaft, sie befindet sich vielmehr in permanenter Weiterentwicklung, und sie experimentiert mit allen möglichen Entwicklungsrichtungen. Ursache dieses Suchens ist der Widerspruch zwischen Wunsch und Wirklichkeit, Ausdruck dieses Suchens sind ständig neue Methoden."

Volker Claus [Claus, 1989]

„Wunsch und Wirklichkeit" heißen „Spezifikation und Implementierung" in der Praktischen und Technischen Informatik. Auch dort versuchen wir, den Widerspruch methodisch zu lösen, auch dort wechselt die Orientierung in unserem Suchen nach neuen Methoden. Im Zug der Technologie verschärfen sich die klassischen Herausforderungen an die Architektur anspruchsvoller Hard- und Software: Komplexität, Veränderung und Zeit [Alexander, 1964]. Die *Ad-hoc*-Methode „Versuch und Irrtum", für überschaubare Entwürfe wirtschaftlich durchaus vertretbar, versagt jenseits einer gewissen Komplexitätsbarriere [Winograd, 1975a]: *System-on-Chip* in der VLSI-Technik [Rammig, 1989] oder *Programming-in-the-Large* in der Softwaretechnik [DeRemer & Kron, 1976]. Der technische Fortschritt einerseits und der Sog rasch wechselnder Marktbedürfnisse andererseits diktieren immer kürzere Entwicklungszyklen: *Time-to-Market* ist der Prüfstein neuer Methoden.

Der Entwurf komplexer Soft- und Hardware stellt eine Folge datenverarbeitender Handlungen dar. Daten werden in Daten transformiert: Begleitet von Testdaten gehen Spezifikationsdaten in Entwurfsdaten über und diese wiederum in Fertigungsdaten für den Compiler oder die VLSI-Prozeßlinie. Die Vielfalt der Transformationen hat uns einen „Methodenberg" beschert, der die Transparenz der Entwurfsphasen und ihrer Modelle belastet, die Schnittstellen vertieft und lange Entwicklungszeiten bedingt. Was bis vor wenigen Jahren unter diesem Methodenberg nicht zu finden war, ist ein umfassender Ansatz, der sich quasi als *Leitmotiv* durch den gesamten Entwurf zieht. Es mangelt an *durchgängigen* Analyse- und Designmethoden, die Konzepte und Ausdrucksmittel bereitstellen, um die Struktur eines Problems ohne Bruch auf die Struktur einer Lösung abzubilden. Das *Objekt-Paradigma* genügt diesem Anspruch: Es ist methodisch durchgängig und wirtschaftlich zugleich.

Die Fallbeispiele dieser Studie verdeutlichen, wie die Konzepte und Ausdrucksmittel der Objektorientierung die oft zitierten „semantischen Lücken“ zwischen den Modellierungsphasen im VLSI-Entwurf überbrücken helfen: von der Spezifikation bis hin zur Implementierung. Im Brennpunkt steht die methodisch-operative Integritätssicherung der VLSI-*Weltmodelle* für deskriptive und prozedurale Datenschnittstellen, also für Austauschformate und Werkzeuganbindungen. Zunächst aber seien als Standortbestimmung die Motive der Abhandlung, ihre Zielsetzung und Gliederung skizziert.

Hintergründe

Eine wissenschaftliche Studie sollte aus mehreren Quellen schöpfen: Das theoretische und empirische Material dieser Studie entstammt einerseits Forschungsprojekten, durchgeführt in der Fachgruppe Technische Informatik der Universität Siegen, zum anderen bezieht es sich auf Fallbeispiele aus der einschlägigen Literatur. Im einzelnen setzten die folgenden Faktoren Bezug und Ziel der Studie:

- Erfahrungen

  - BMFT-Projekt E.I.S.: „Entwurf integrierter Schaltungen“ [Quibeldey-Cirkel & Wojtkowiak, 1991]

    Das groß angelegte Verbundprojekt (Gesamtlaufzeit von 1983 bis 1992) machte den allgemeinen VLSI-Entwurf zum Hochschulthema und führte zur Konzeption, Entwicklung und Erprobung zahlreicher Entwurfs- und Testwerkzeuge. In den Prototypen kundenspezifischer Entwürfe als Gate-Arrays oder Standardzellen-ICs sowie im Fundus der Diplomarbeiten und Dissertationen gründet sich der akademische Erfahrungshorizont der Abhandlung.

  - Industriepraxis

    Zum einen stammen die Erfahrungen mit industriellen Forderungen an den VLSI-Entwurf aus Entwicklungsprojekten, die zusammen mit der Industrie durchgeführt wurden [Schneider, 1991; Schröder, 1992]. Thematischer Schwerpunkt: Konvertierung von Simulations- in Testdaten. Andererseits verfügt der Autor über Industrie-Erfahrung in der Einführung moderner CAD/CAE-Anlagen. Die Probleme liegen hier besonders in den inkompatiblen Daten- und Werkzeugschnittstellen, zum Beispiel in den Fehlanpassungen zwischen Schaltungs- und Leiterplattenentwurf, wenn also die Daten der Konstruktion zurück in die Entwurfsphasen geführt werden (*Back annotation*).

– BMFT-Projekt DASSY: „Datentransfer und Schnittstellen in offenen integrierten VLSI-Entwurfssystemen“ [Quibeldey-Cirkel, 1993a]

  Die in der EDA-Industrie allgemein beklagten Schnittstellen — inkompatible Werkzeuge und Datenaustauschformate auf der einen und heterogene Datenhaltungen auf der anderen Seite — haben maßgeblich das Verbundprojekt DASSY motiviert. CAD-Schnittstellen sind wirtschaftlich nur über den Weg breit akzeptierter Industriestandards überwindbar. Im Rahmen des Forschungsprojekts wurden hierfür international beachtete Vorschläge erarbeitet. Zum Teil mußte zuerst ein Problembewußtsein in den Normungsgremien von IEEE und ISO geschaffen werden: Allein die formale *Informationsmodellierung* dämmt die Flut der Erweiterungsvorschläge ein und schafft die Voraussetzung für eine Wertung unabhängig von der Syntax. Der Widerspruch zwischen inflationärer Syntax und vage formulierter Semantik wurde am Datenaustauschformat EDIF thematisiert. Die hier geleisteten Arbeiten [Quibeldey-Cirkel & Wahl, 1990b; Wilkes (Hrsg.), 1992] begründen den pragmatischen Anspruch der Studie.

• Erkenntnisse

– CFI: „CAD-Framework-Initiative“ [Barnes *et al.*, 1992; Rhyne, 1989]

  Bislang wurde die Standardisierung der VLSI-Datenformate und Werkzeugschnittstellen unkoordiniert betrieben; die CAD-Framework-Initiative hat sie erstmals kanalisiert. Auch die Standardisierungsarbeiten im DASSY-Projekt orientierten sich an den Zielen dieses internationalen Industriekonsortiums. Die DASSY-Ergebnisse fließen derzeit in die CFI-Richtlinien mit ein.

– Fallbeispiele

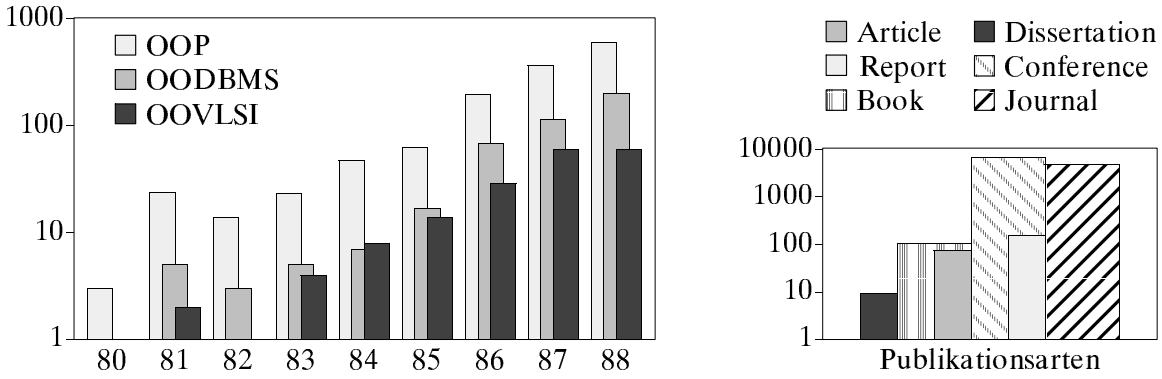


Bild 0.1: Objektorientierung: mehr als nur ein populärwissenschaftliches Thema

Eine Literaturrecherche, ursprünglich auf das Thema „Objektorientierte Datenbanken“ gerichtet, offenbarte die mannigfaltigen Querbezüge der Objektorientierung in der Informatik. Nicht nur in den klassischen Phasen der Systementwicklung, wie Analyse, Entwurf und Programmierung, wird ein objektorientierter Ansatz verfolgt, sondern auch in der Konzeption und Implementierung von Non-Standard-Datenmodellen, VLSI-Werkzeugen und CAD-Infrastrukturen. Die Balkendiagramme[2] in Bild 0.1 (vorige Seite) verdeutlichen den Trend der Objektorientierung zur wissenschaftlichen Breitenanwendung. Neben den Fallbeispielen aus der Literatur setzt sich die Studie hauptsächlich mit den Erkenntnissen aus dem DASSY-Projekt auseinander: theoretisch auf der Basis des DASSY-Datenmodells (DaDaMo) für eine standardisierte Werkzeugschnittstelle [DASSY, 1992a] und empirisch durch die Erprobung des DASSY-Prototyps [DASSY, 1992b].

– Paradigmenwechsel und interdisziplinärer Kontext

  Inspiriert durch die aufgedeckten Querbezüge des Objekt-Paradigmas und mit der Kenntnis der Wissenschaftstheorie nach Thomas KUHN, liegt die Vermutung eines *Paradigmenwechsels* nahe [Quibeldey-Cirkel, 1994a]. Die Grafik in Bild 0.2 zeigt anschaulich, wie die objektorientierten Themen (OOx) den vor zwanzig Jahren aus der Softwarekrise geborenen strukturierten Ansätzen für Analyse und Design (SA/SD) den Rang ablaufen.[3]

  Die techno-philosophische Breite des Objektansatzes unterstützt die These, daß die Suche nach einer interdisziplinären „Wissenschaft des Entwerfens“, wie sie Herbert SIMON, Heinz ZEMANEK und derzeit Christiane FLOYD fordern, ihren Ausgangspunkt im Objekt-Paradigma nimmt [Floyd, 1994; Simon, 1982; Zemanek, 1992]. Objektorientiertes *konzeptionelles Modellieren* ist interdisziplinär [Dillon & Tan, 1993; Fiadeiro & Sernadas, 1991]. Gerade die jungen Informatikzweige weisen synergetische Ziele auf: Wissensrepräsentation in der Künstlichen Intelligenz (KI), semantische Datenmodelle in der Datenbanktechnik und objektorientierte Analyse- und Entwurfsmethoden in der Softwaretechnik haben ihre Gemeinsamkeit in der Modellbildung auf konzeptionellen Ebenen. Sie teilen sich problemnahe Konzepte und Mechanismen, wie hierarchische Kategorien und Vererbung, um das Wissen über einen Weltausschnitt formal zu erfassen und für die informationstechnische Verarbeitung aufzubereiten.

– Weltmodelle

  CAD-Frameworks für datenintensive Anwendungen stehen in der Diskussion [Barnes *et al.*, 1992; Bhat & Taku, 1990; Quibeldey-Cirkel, 1993b; Quibeldey-

[2] Verlauf der Veröffentlichungen unterschieden nach objektorientierten Anwendungen: Programmierung (OOP), Datenbanksystemen (OODBMS), VLSI-Werkzeugen (OOVLSI)

[3] OOx steht als Sammelkürzel für objektorientierte Methoden und Techniken in den Entwurfsphasen des Software-Lebenszyklus: Analyse (OOA), Design (OOD), Programmierung (OOP) und Datenbanksysteme (OODBMS). In der Statistik wurden nur disjunkte Einträge (englischsprachige) gezählt, also entweder SA/SD oder OOx. Veröffentlichungen, die beide Ansätze verfolgen, fielen somit heraus.

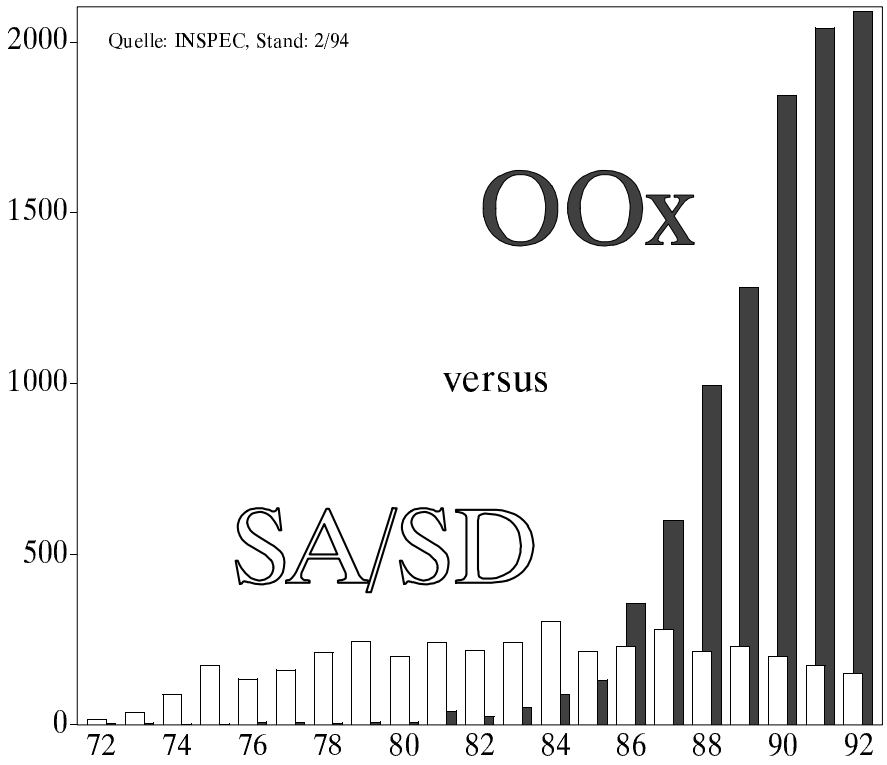


Bild 0.2: Statistisches Indiz für einen Trendwechsel

Cirkel, 1993c; Rammig & Steinmüller, 1992; Siepmann, 1991]. Sie rücken zwei Facetten der Informationsverarbeitung in den Mittelpunkt:

1. die Datenrepräsentation, das Modellieren und Schematisieren der Entwurfsinformation, und
2. die Datenverwaltung, das konsistente Fortschreiben und Abfragen des Datenbestands.

Damit gehen zwei Gegenstände der Modellierung einher: (a) deskriptive Datenaustauschformate und (b) prozedurale Werkzeugschnittstellen. Der deskriptive Aspekt umfaßt die statischen Strukturen einer Anwendung. Die Informationsmodellierung, aufgekommen in den letzten Jahren, bietet hier die konzeptionellen Mittel, um den Informationsgehalt formal zu beschreiben. Der prozedurale Aspekt zielt auf die effiziente Kommunikation zwischen heterogenen Entwurfswerkzeugen. Da dies nicht allein durch Import/Export-Dienste auf Dateibasis erreichbar ist, müssen die Werkzeuge integriert werden. Es bedarf vor allem einer schematisierten Modellvorstellung über das dynamische Datenaufkommen im Entwurf. Stetig mit Entwurfs- und Prozeßdaten angereichert, entsteht ein „Datenpool“, in dem die Entwicklung fortgeschrieben wird.

Zur Modellierung des Datenpools führen wir den Begriff des „Weltmodells“ ein, angelehnt an den Begriff des „Datenmodells“ in der Datenbanktechnik. Wie Datenmodelle, zum Beispiel das relationale von Edgar CODD [Codd, 1979], umfassen Weltmodelle die konzeptionellen Mittel zur Modellierung eines Weltausschnitts, *Miniwelt* genannt. Derartige Weltmodelle für den VLSI-Entwurf aufzustellen und ihre Integrität zu gewährleisten, ist das Thema zahlreicher Forschungsprojekte (EIS, ECIP, EVEREST, DASSY) und Ziel internationaler Standardisierungsgremien (CFI, EDIF, STEP).[4]

Thema und Thesen

Der Titel der Studie *„Objektorientierung als methodisch-operativer Ansatz zur Erstellung und Integritätssicherung von VLSI-Weltmodellen“* stellt die Pragmatik des Objekt-Paradigmas in den Vordergrund:

- methodischer Aspekt[5]

  Zum einen werden die Modelle für Daten- und Werkzeugschnittstellen objektorientiert gebildet: Die VLSI-Weltmodelle beruhen auf den Prinzipien der Kapselung von Struktur- und Verhaltensmustern, der Vererbung von Klassenmerkmalen und der Kommunikation zwischen Objekten. Zum anderen leitet das Objekt-Paradigma den Entwurfsprozeß in seinen konzeptionellen Phasen: es unterstützt methodisch durchgängig die Analyse- und Syntheseschritte.

- operativer Aspekt[6]

  Neben der methodischen Integritätssicherung greifen die objektorientierten Konzepte auch in der Implementierung: Programmierung, Datenhaltung, prozedurale Werkzeugschnittstellen und Konsistenzmechanismen beruhen auf gemeinsamen Grundsätzen. Eine operative Durchgängigkeit der Implementierungsphasen ist möglich und manifestiert sich zusammen mit der methodischen Leitlinie in den Ansätzen moderner VLSI/CAD-Infrastrukturen.

[4] Die Gremien und Projekte werden in Abschnitt 3.1.2 näher beschrieben.
[5] Im ursprünglichen Sinne von griechisch *methodos* meinen wir den „Weg zum Ziel“.
[6] Im Sinne konkreter Maßnahmen steht „operativ“ für „unmittelbar wirkend“.

Folgende Thesen sollen durch die Studie überprüft werden:

1. Objektorientierung ist *intuitiv*.

   Sie unterstützt das bewußte Erfassen von Wahrnehmungs- und Denkinhalten, die *Apperzeption* der realen Welt, abgegrenzt vom rein sinnlichen Wahrnehmen, der *Perzeption*, als erste Stufe der Erkenntnis. Sie stellt ein *ontologisches Grundprinzip*[7] dar, geeignet, auch technische Artefakte einfach und ganzheitlich zu modellieren. Sie kann überdies auf kognitionspsychologische Parallelen verweisen, die ihre Intuitivität plausibel machen.

   Objektorientierung ist zu einem übergeordneten Prinzip der Informatik aufgestiegen. Sie löst — im Sinne eines Paradigmenwechsels nach KUHN — das alte Paradigma der prozeduralen Strukturierung ab, das heißt: Sie hebt die willkürliche Trennung zwischen Daten und Operationen auf und schafft so *konzeptionelle Einheiten* aus Struktur und Verhalten, die von der Analyse bis zur Implementierung in Soft- oder Hardware fortbestehen. Die objektorientierte Datenkapselung geht also über den Modularisierungsbegriff der Programmiermethodik hinaus: sie steht dem Menschen näher als der Maschine.

2. Das Objekt-Paradigma eignet sich als *methodischer Ansatz* für die Entwurfsmodellierung.

   Es übernimmt die Funktion einer Handlungsanweisung und Orientierungshilfe. Das Entwurfsproblem wird methodisch verkürzt auf die Dekomposition in *beinahe* unabhängige Objekte und deren Interaktionen. Die objektorientierten Konzepte, wie Klassifikation, Kapselung, Vererbung und Nachrichtenaustausch, erleichtern die Modellierung komplexer Objektstrukturen und Verhaltensmuster. Sie helfen, die Weltmodelle für datenintensive CAD-Anwendungen aufzustellen und die EDA-Standardisierungsbemühungen voranzutreiben.

3. Die objektorientierte Datenmodellierung erschließt effektive Mechanismen zur Prüfung und Wahrung der Datenintegrität.

   Als *operativer Ansatz* in einer objektorientierten Datenhaltung mit prozeduraler Werkzeugschnittstelle ermöglicht sie die Rahmenkonzeption, um Entwurfsdaten und Werkzeuge einheitlich zu integrieren. Die objektorientierte *Constraint*-Modellierung faßt die konventionellen Anstoßtechniken, wie Event/Trigger-Mechanismus und „berechnete Attribute“, als einheitliches Konzept zusammen.

[7] Ontologie ist die wissenschaftliche Philosophie von der Modellierung der Objekte in der realen Welt [Bunge, 1979].

Gang der Untersuchung

Vorgehensweise und Ergebnisse der Studie spiegeln sich sowohl chronologisch als auch thematisch in den drei Teilen der Abhandlung:

I Das Problem: Komplexität versus Integrität

Ausgehend von der industriellen Entwurfssituation, dem „In-die-Welt-Gesetzt-Sein" des Entwerfers — er soll komplexe und zugleich integre technische Objekte mit wirtschaftlich vertretbarem Aufwand entwerfen —, erhellen wir die systemtheoretischen Grundlagen zur Komplexität von Soft- und Hardware (Kapitel 1) und skizzieren die klassischen Methoden und Techniken zur Bewältigung der Komplexität (Kapitel 2). Aspekte der Integritätssicherung in VLSI/CAD-Infrastrukturen werden aufgezeigt und die Standardisierungstrends, getragen von zahlreichen Framework-Initiativen, bewertet (Kapitel 3). Um die wissenschaftstheoretische Fundierung des Objektansatzes zu zeigen, untersuchen wir, ob sich die KUHNsche These des Paradigmenwechsels auf die Entwicklung der Informatikmethoden anwenden läßt (Kapitel 4).

II Objektorientierung als methodischer Ansatz zur Integritätssicherung

Zunächst rezipieren wir die Leitgedanken interdisziplinärer Forscher zu einer „Wissenschaft des Entwerfens" und eruieren den Beitrag, den das Objekt-Paradigma dazu leisten kann (Kapitel 5). Vom allgemeinen zum besonderen gehend, stellen wir konventionelle CAD-Modelle vor und ordnen deren Ansätze gemäß einer allgemeinen Modelltheorie (Kapitel 6). Als gemeinsames Manko der diskutierten Prozeßmodelle zeigt sich die Systemlastigkeit: *kognitive* Prozesse des Entwerfens bleiben unberücksichtigt. Wir vertiefen deshalb den kognitionspsychologischen Aspekt des Entwerfens (Kapitel 7). Motiviert durch die aufgedeckten Parallelen zu den objektorientierten Konzepten und Methoden, führen wir schließlich das Objekt-Paradigma in den Entwurfsprozeß ein, sammeln die aus aktuellen Forschungsprojekten gewonnenen Modellierungskonzepte für VLSI/CAD-Objekte und stellen sie zu einem universellen Objektmodell zusammen. Exemplarisch zeigen wir, wie die Objektorientierung als Leitmotiv die Weltmodelle moderner VLSI/CAD-Infrastrukturen und -Schnittstellen durchzieht. Wir gehen dabei ausschnittsweise auf die deskriptiven und prozeduralen Schnittstellen eines CAD-Frameworks ein (Kapitel 8).

III Objektorientierung als operativer Ansatz zur Integritätssicherung

Eine jede Theorie des Entwerfens wird *Constraints* berücksichtigen. Constraints sind operativer Ausdruck für Entwurfsprinzipien und Entwurfsregeln, für technische und wirtschaftliche Handlungsbedingungen, für Naturgesetze und Kulturnormen. Entwerfen ist das *Erforschen* dieser Constraints mit dem Ziel, eine *zufriedenstellende* Lösung zu finden. Sie sollte allen oder möglichst vielen Constraints genügen. Die

These lautet: „Entwerfen ist ein CSP — ein *Constraint-Satisfaction*-Problem (Kapitel 9). Constraints und Objekte sind ontologische und kognitionspsychologische Grundbegriffe. Beide zu einem gemeinsamen Konzept zu vereinen, ist nicht nur plausibel, sondern pragmatisch zugleich. Wir erweitern folglich den konventionellen Objektbegriff — den Dualismus aus Struktur und Verhalten — um eine dritte Komponente zum *Tripel des Objekts*: Struktur, Verhalten und *rationale Beschränkung* (Kapitel 10). Diese Erweiterung erlaubt uns, objektorientierte Constraint-Modelle zu entwickeln, auf deren Grundlage die operative Integritätssicherung vereinheitlicht werden kann. Im Gegensatz zur modellbezogenen Integrität, die das Datenmodell einer Datenhaltung *inhärent* gewährleistet, definiert sich die *semantische* Integrität der Daten allein aus der Sicht der Anwendung. Inwieweit objektorientierte Datenbanken und VLSI-Expertensysteme Constraint-Mechanismen zur Verfügung stellen, um die semantische Integrität zu sichern, untersuchen wir an Fallbeispielen.

Teil I

Das Problem: Komplexität versus Integrität

> *„Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.*
>
> *The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet — something to make software costs drop as rapidly as computer hardware costs do.*
>
> *But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.“*
>
> Frederick P. Brooks [Brooks, 1987]

Komplexität als reale und Produktivität als geforderte Größe — derlei Gegensätze lassen den Systementwurf nicht selten zum „monströsen“ Unterfangen ausarten. Der vielbesprochene Aufsatz von Frederick P. Brooks *No Silver Bullet: Essence and Accidents of Software Engineering* [Brooks, 1987] bringt den Konflikt auf den Punkt: Eindrucksvoll werden die in der Natur der Software liegenden Schwierigkeiten — die *Essenz* — und die lediglich mit der Entwicklung von Software einhergehenden, somit wesensfremden Schwierigkeiten — die *Akzidenzien*[8] — offengelegt. Wir wollen uns von der Brooksschen Metapher und den aristotelischen Begriffen von „Akzidens und Substanz“ leiten lassen, wenn wir im folgenden den Entwurf komplexer Artefakte aus Soft- und Hardware beleuchten. Dabei interessieren wir uns besonders für die Parallelen und Unterschiede in den Entwurfsdisziplinen der modernen Informatik: zwischen dem Software-Entwurf der Praktischen und dem VLSI-Entwurf der Technischen Informatik.

[8] Akzidens: „[lat.] *das*, das Zufällige, Unwesentliche; das Veränderliche an einem Gegenstand. Mz. *Akzidenzen, Akzidenzien*. Gegensatz: →Substanz“ [Brockhaus-Enzyklopädie, 1990].

1

Die Natur der Komplexität

Fragen wir nach der Natur der Komplexität und ihren Dimensionen, schicken wir uns an, das methodische Rüstzeug zu erkunden, um die Entwurfskomplexität technischer Objekte systematisch und wirtschaftlich zu bewältigen, dann begeben wir uns leicht in systemtheoretische Untiefen. Wir müssen aber diesen Weg beschreiten, wollen wir die Grenzen empirischer Vorgehensweisen überwinden, wo Ad-hoc-Strategien und das Prinzip von Versuch-und-Irrtum versagen. *Silver Bullets* gegen Soft- und Hardware-Entwürfe der BROOKSschen Qualität sind nicht in Sicht, aber *Hopes for the Silver*.

Übersicht: Abschnitt 1.1 stellt die Prinzipien der deskriptiven Komplexität heraus. Die Komplexität eines Systems liegt im Pluralismus der Systembeschreibungen begründet. Sie besitzt vor allem eine subjektive Komponente: Komplexität ist eine Frage der Betrachtung. Abschnitt 1.2 skizziert den Komplexitätsbegriff, wie er traditionell in der Systemtheorie verwendet wird. Gegenstand sind objektivierbare Kostenfaktoren. Die für Soft- und Hardware-Algorithmen typischen Kostenfaktoren werden motiviert und entsprechende Metriken am Beispiel erläutert. Abschnitt 1.3 schließlich nennt die *essentiellen* Faktoren, die den Entwurf komplizieren. Die Entwurfskomplexität bestimmt den Ressourcenbedarf an Man-power und Entwicklungszeit, das heißt, sie bestimmt die zeitkritische Markteinführung eines Produkts.

1.1 Deskriptiver Pluralismus

Als Einstieg schließen wir uns der Interpretation von Robert ROSEN an: Komplexität ist keine Objekteigenschaft *per se* oder die Eigenschaft einer einzelnen Objektbeschreibung. Die Essenz der Komplexität liegt vielmehr im *Pluralismus* der möglichen Beschreibungen, in der Vielfalt der Meßmethoden, -techniken und -werkzeuge, über die ein Betrachter verfügt [Rosen, 1977]. In diesem Sinne sind alle beobachtbaren Eigenschaften komplex, ist die Komplexität eine Frage der Betrachtung. William ASHBY nennt dazu ein plakatives Beispiel:

> „To the neurophysiologist the brain, as a feltwork of fibers and a soup of enzymes, is certainly complex; and equally the transmission of a detailed description of it would require much time. To a butcher the brain is simple, for he has to distinguish it from only about thirty other ‚meats', so not more than $\log_2(30)$, i.e. about 5 bits, are involved." (zitiert in [Klir, 1985, S. 132])

Der Gegenstand einer wissenschaftlichen Betrachtung — in unserem Fall der Entwurf anspruchsvoller Rechnerprogramme und integrierter Schaltungen — erscheint nicht aufgrund einer quantitativen Eigenschaft komplex, zum Beispiel aufgrund der Layoutdichte oder der Programmlänge. Vielmehr ist es die unüberschaubare Fülle möglicher Perspektiven und Modellvorstellungen. Darin unterscheidet sich ein potentiell komplexes System von einem einfachen. Komplexität ist also eng verwoben mit Information oder genauer: mit der Darstellung von Information.

Wir können zwei allgemeine Prinzipien *deskriptiver* Komplexität unterscheiden: Die Komplexität eines Systems ist proportional zur Informationsmenge, die erforderlich ist, um (a) das System hinreichend zu beschreiben oder um (b) eine Unbestimmtheit (Ungewißheit oder Ungenauigkeit) in der Systembeschreibung auszuschließen. Beide Prinzipien sind einander gegenläufig: Auf einer hohen Abstraktionsebene ist der Beschreibungsaufwand gering, die Unbestimmtheit aber groß und umgekehrt. Für den Chipentwerfer beispielsweise enthält ein Maskenlayout den höchsten Grad an Gewißheit: die Geometrie der Masken ist eindeutig festgelegt. Eine abstrakte Systembeschreibung auf der Registertransfer-Ebene hingegen hält die Entscheidungen auf der Maskenebene offen, birgt also für das Layout eine maximale Ungewißheit, was durchaus auch Entwurfs*freiheit* bedeuten kann. Das Beispiel verdeutlicht zugleich den Aufwand: Für eine *hinreichende Systembeschreibung* (erstes Prinzip deskriptiver Komplexität) genügen wenige Seiten Text oder Grafik. Die *Schaffung von Gewißheit* (zweites Prinzip) erfordert dagegen einiges mehr, in unserem Beispiel gewöhnlich Megabytes an Geometriedaten.

Der Begriff der „Information" wird in diesem Zusammenhang nachrichtentechnisch, das heißt *syntaktisch* verwendet. Semantische oder pragmatische Aspekte bleiben außen vor: Die Absicht des Senders oder die Betroffenheit des Empfängers spielen keine Rolle. Während sich das zweite Prinzip auf eine allgemeine Maßeinheit der *Unbestimmtheit* stützt, der *Entropie*[1], läßt sich für das erste Prinzip keine einheitliche Größe angeben. Fälschlicherweise dient hier oft die *Quantität* der Komponenten als Maß. So im VLSI-Entwurf, wo die Integrationsdichte gemeinhin für die „Komplexität" einer Schaltung steht: die Zahl der Speicherbits in der DRAM-Technik oder die verfügbaren Gatteräquivalente in der ASIC-Technik. Im Software-Entwurf ist es die LOC-Zahl: *lines of code*. Handelt es sich um *reguläre* Strukturen, dann ist die Quantität der Komponenten sicherlich kein Maß für die deskriptive Komplexität: So setzen sich höchstintegrierte Speicher *repetitiv* aus den

[1] Über die Entropie im Entwurf läßt sich zum Beispiel Cees KOOMEN in [Koomen, 1985] aus. Sie dient ihm als Maß für die Entwurfsfreiheit, für die Kreativität des Entwerfers (siehe Abschnitt 6.2.2).

gleichen 1-Bit-Speicherzellen zusammen; ihre Vernetzung ist streng regulär, das heißt symmetrisch und skalierbar. Der Entwurf regulärer Strukturen ist nicht komplex!

Das Maß für den Beschreibungsaufwand — für die deskriptive Komplexität — muß allgemeinen Forderungen genügen, die George KLIR formuliert [Klir, 1985, S. 134f.]:

> S sei die Menge aller Systeme eines bestimmten (epistemologischen) Typs, $P(S)$ die Potenzmenge von S und C_S ein Maß für die deskriptive Komplexität von S. Dann steht C_S für die Abbildung[2] $C_S : P(S) \rightarrow \mathcal{R}$, die folgende Axiome erfüllt:
>
> C1: $C_S(\emptyset) = 0$;
>
> C2: wenn $A \subset B$, dann gilt $C_S(A) \leq C_S(B)$;
>
> C3: wenn A ein homomorphes Abbild von B ist, dann gilt $C_S(A) \leq C_S(B)$;
>
> C4: wenn A und B isomorph sind, dann gilt $C_S(A) = C_S(B)$;
>
> C5: wenn (i) $A \cap B = \emptyset$, (ii) A und B nicht interagieren und (iii) weder A noch B ein homomorphes Abbild des jeweils anderen ist, dann gilt $C_S(A \cup B) = C_S(A) + C_S(B)$.

In Worten: Die Einschränkungen C1 und C2 stellen sicher, daß das Komplexitätsmaß C_S numerisch positiv ist. C2 und C3 implizieren monotone Eigenschaften: Die Komplexität sollte nicht zunehmen, wenn die Menge der betrachteten Systeme S abnimmt oder wenn weniger Details unterschieden werden. C4 wahrt die Unabhängigkeit des Maßes von der syntaktischen Bezeichnung: Nur die Strukturen, nicht die Länge ihrer Bezeichner, ob einsilbig oder episch, sind maßgebend. Die Forderung C5 schließlich führt auf wünschenswerte additive Eigenschaften: Werden zwei Systemmengen vereint, die in allen relevanten Aspekten disjunkt sind, also keine gemeinsamen Strukturen besitzen und nicht wechselwirken (interagieren), dann sollte sich die Gesamtkomplexität aus den Einzelbeiträgen summieren.

Die deskriptive Komplexität eines Systems läßt sich unabhängig von der gewählten Maßeinheit beträchtlich verringern, wenn das System „wohlstrukturiert“ vorliegt. Bild 1.1 verdeutlicht ein Beispiel [Klir, 1985, S. 137]: Der Speicherbedarf, um das Zustandsverhalten eines unstrukturierten Systems, bestehend aus n Variablen mit jeweils k Zuständen, vollständig zu beschreiben, beträgt nk^n Speicherzellen, wobei eine Zelle k Zustände speichern kann. Dagegen reduziert sich der Speicherbedarf erheblich, wenn das System „wohlstrukturiert“ vorliegt, wenn es sich zum Beispiel aus Teilsystemen mit jeweils zwei Variablen zusammensetzt. In diesem Fall beträgt die Zahl der Speicherbits $k^2n(n-1)$. Die grau unterlegte Fläche in Bild 1.1 macht die Einsparung für $k = 10$ anschaulich: Regelmäßige Strukturen verringern den Beschreibungsaufwand exponentiell.

Durch unsere Fähigkeit zu abstrahieren, können wir die Beschreibung eines komplexen Systems um weitere Größenordnungen vereinfachen: Betrachten wir nur die relevanten

[2] $\mathcal{R}$: Menge der reellen Zahlen

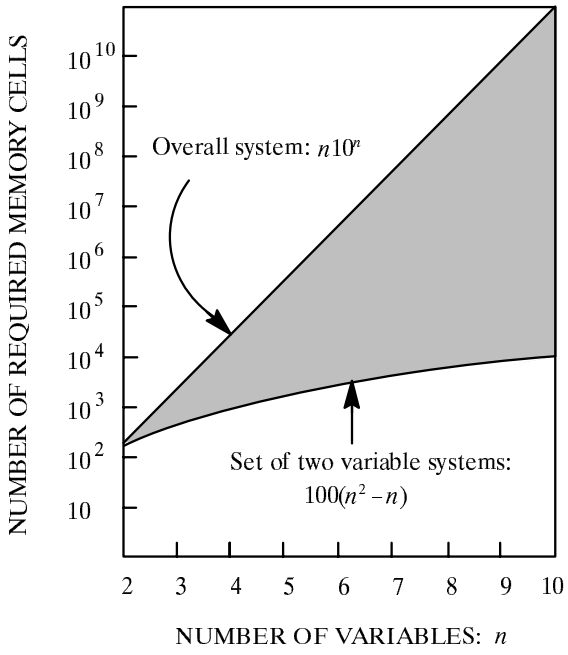


Bild 1.1: Komplexität versus Systemzerlegung

Eigenschaften, senken wir zugleich drastisch die Zahl potentieller Interaktionen mit dem System. Wir verzichten dabei auf Detailtreue und nehmen einen höheren Grad der Unsicherheit in Kauf. Die Technik setzt indes Grenzen: Das Medium der Implementierung beschränkt die Freiheitsgrade der Abstraktion, den besagten Pluralismus der Perspektiven. In der Softwaretechnik diktieren die VON-NEUMANN-Prinzipien der Rechnerorganisation die *prozedurale* Programmiermethode „von unten“: vom binären Maschinencode über Assembleranweisungen zu den Ablaufstrukturen in höheren Programmiersprachen. Linda WEISER FRIEDMAN resümiert:

> „The von NEUMANN-type of computer architecture, while serving as catalyst for a generation of programming languages may also, it appears, be guilty in large part for the degree of stagnation we have seen in the conceptual development of programming language technology. The stored program concept binds computer control directly and intimately with program variables representing memory locations and involves the use of a single counter that controls program flow by a sequence of instructions. Programming language paradigms relying on this underlying computer architecture tend to be statement-oriented, assuming the sequential execution of a limited number of operations. Programs written in these languages are composed of vast numbers of lines of code [...].“ [Weiser Friedman, 1992, S. 13]

In der VLSI-Technik lastet die Zweidimensionalität des Zielmediums „Maskenlayout“ auf den Freiheitsgraden der Abstraktionen: von der CIF-Notation über *Stick*-Diagramme und

Floorplan zu höheren Hardware-Beschreibungssprachen [Conway & Mead, 1980]. *Formale* Sprachen sind ohnehin dem kognitiven Prozeß des Problemlösens hinderlich. Bis heute ist die WHORF-Hypothese unwiderlegt: Erkenntnisfähigkeit ist sprachbedingt [Whorf, 1974]. Oder mit dem Sprachpragmatiker Ludwig WITTGENSTEIN: „Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt“ [Wittgenstein, 1973].

Als einen ersten Zugang zur deskriptiven Natur der Komplexität können wir somit festhalten: Komplexität ist ihrem Wesen nach *relativ*. Die Abstraktion bestimmt ihr Ausmaß. Was abstrahiert wird (lat. „ab-, wegziehen“) und folglich nicht zur Komplexität beiträgt, liegt im subjektiven Ermessen des Betrachters. KLIR faßt dies in poetische Worte: „the complexity of an object is in the eyes of the observer“ [Klir, 1985, S. 131].

1.2 Komplexität und ihre Dimensionen

Als Ausgangspunkt unserer Diskussion, wie sich die Komplexität im Soft- und Hardware-Entwurf konkret ausnimmt, folgen wir zunächst der Einteilung wissenschaftlicher Aufgaben nach Warren WEAVER. Sie erlaubt die Einordnung der Entwurfsdisziplin in den wissenschaftlichen Kontext [Weaver, 1948].

- *Problems of simplicity*

  Aufgaben dieser Art zeichnen sich durch einfache Modelle aus, die mit wenigen Variablen (meist zwei oder drei) und determiniertem Verhalten reale Phänomene hinreichend genau wiedergeben. Sie beschäftigten primär die Naturwissenschaften des 17. bis 19. Jahrhunderts. Ihr Lösungsansatz wird getragen von einem zentralen Motiv: der Entdeckung einer „verborgenen Einfachheit“ in einem scheinbar komplexen Phänomen. Vereinfachende Annahmen erlauben es, die wenigen bedeutsamen Faktoren eines Problems von der Vielzahl (vermeintlich) unbedeutsamer zu isolieren. Systeme mit diesen Eigenschaften sind gut verständlich und einer mathematisch analytischen Behandlung leicht zugänglich. Die NEWTONsche Physik steht exemplarisch für Aufgaben dieser Kategorie.

- *Problems of disorganized complexity*

  Am anderen Ende der WEAVERschen Komplexitätslinie stoßen wir ebenfalls auf leicht verständliche Aufgaben, aber mit diametralen Eigenschaften: Sie umfassen astronomisch viele Variable (Dutzende Größenordnungen) mit individuell zufälligem Verhalten. Das Interesse an derartigen Systemen kam Ende des 19. Jahrhunderts auf, als Gasmoleküle in geschlossenen Volumina untersucht wurden. Vorausgegangen war die Entwicklung statistischer Methoden der Analyse. Sie beruhen auf zwei vereinfachenden Annahmen: Erstens kann von der Vielzahl individueller Eigenschaften

abstrahiert werden, zum Beispiel den Bewegungen einzelner Gasmoleküle, und zweitens ist das Systemverhalten durch Aussagen über die idealisierten Eigenschaften weniger Systemkomponenten hinreichend beschreibbar: „Gesetz der großen Zahlen". Ungeordnete Komplexität besagt also: Das individuelle Verhalten der Variablen ist zufällig und nicht berechenbar, ihr Verhalten als Ganzes aber geordnet und statistisch erfaßbar. Exemplarisch für diese Aufgabenklasse steht hier die statistische Mechanik.

- *Problems of organized complexity*

  Das breite Mittelfeld des Komplexitätsspektrums — zwischen den beiden Extrema: „individuell determiniert" und „statistisch determiniert" — weist Systeme auf, deren Eigenschaften weder mathematisch noch statistisch bestimmbar sind (zumindest nicht mit vertretbarem Aufwand) und weder vernachlässigt noch idealisiert werden können. Die vereinfachenden Annahmen der beiden anderen Aufgabenklassen — Isolation und Abstraktion — sind hier unzulässig: Einerseits ist die Zahl der Variablen zu groß, um einer isolierten mathematischen Analyse zugänglich zu sein, andererseits zu gering, um statistische Aussagen zu erlauben. Mit Worten von WEAVER: „a sizable number of factors which are interrelated into an organic whole" [Weaver, 1948, S. 539]. Die Mehrheit wissenschaftlich interessanter Systeme fällt in diese Komplexitätsklasse! KLIR führt aus:

  > „Instances of systems with the characteristics of organized complexity are abundant, particularly in the life, behavioral, social and environmental sciences, as well as in applied fields such as modern technology and medicine." [Klir, 1985, S. 136]

  Analyse und Entwurf erfordern also ein interdisziplinäres methodisches Rüstzeug, das bislang kaum zur Verfügung steht und Gegenstand der modernen Systemtheorie schlechthin ist.[3]

Anspruchsvolle Soft- und Hardwaresysteme sind zweifellos im mittleren Komplexitätsbereich der WEAVERschen Linie einzuordnen. Wir gehen folglich von einem Problem *geordneter Komplexität* aus, die sich zwar linear aus der Anzahl einfacher und idealisierter Systemkomponenten, aber „kombinatorisch explosiv" aus deren mannigfaltiger Verbindungsstruktur ergibt. Hoffnungen auf eine „verborgene Einfachheit" oder auf ein „Gesetz der großen Zahlen" entbehren der Grundlage.

1.2.1 Begriffsrahmen

Der Begriff der Komplexität ist mehrdeutig: er weist sowohl objektiv meßbare als auch subjektiv wahrgenommene Aspekte auf. Um nun den Schwierigkeitsgrad im Entwurf komplexer Artefakte zu ermessen, soll zunächst ein übergeordneter Begriffsrahmen nach Robert

[3] KLIR und andere schlagen für die Analyse geordneter Komplexität die *Fuzzy*-Logik vor [Zadeh, 1965].

FLOOD vorgestellt werden [Flood, 1987]. Die zwei wesentlichen Bezüge sind hiernach das betrachtete System und der Betrachter. Ein System besitzt also objektivierbare Eigenschaften (Struktur und Verhalten) und perspektivische Auslegungen (wie im Zitat von ASHBY auf Seite 12). Das Begriffspaar — System versus Mensch — differenziert FLOOD auf weiteren Ebenen: siehe Bild 1.2.

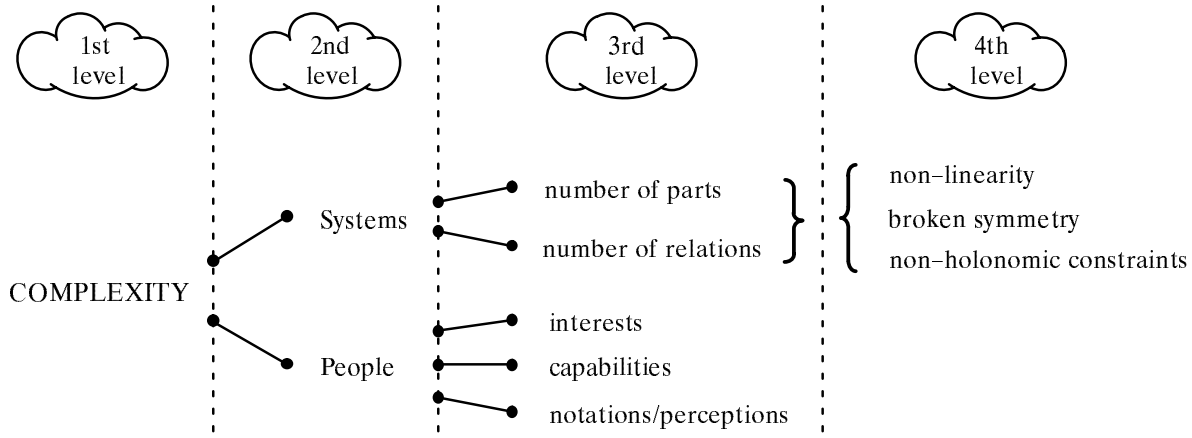


Bild 1.2: Zum Komplexitätsbegriff nach FLOOD

Die systembezogenen Differenzierungen auf der dritten Ebene decken sich mit der eingangs aufgezeigten deskriptiven Natur der Komplexität. Die Systemteile und ihre Beziehungen untereinander bergen eine derart *kombinatorische* Vielfalt an möglichen Wechselwirkungen, daß sie für den Betrachter insgesamt kontra-intuitiv wirken: Das Teile-Beziehungs-Geflecht verschließt sich seinem unmittelbaren Verständnis. Nur durch *selektives* Wahrnehmen (als Filter kommen hier die Interessen, Fähigkeiten und begrifflichen Vorstellungen des Betrachters in Frage) kann ein partielles Systemverständnis gewonnen werden. Ein Beispiel aus dem VLSI-Entwurf soll das „kombinatorische Potential" des Teile-Beziehungs-Schemas verdeutlichen:

> Ein Gate-Array-Master sei gekennzeichnet durch k Eingangs- und l Ausgangs-Padzellen sowie n Logikzellen mit jeweils zwei Eingängen[4] und einem Ausgang: Bild 1.3. Alle Freiheitsgrade vorausgesetzt, wächst die Vielfalt möglicher Wechselwirkungen im Verknüpfungsgeflecht aus Pad- und Logikzellen stärker als exponentiell. Für eine vollständige, nichtparallele Beschaltung der Padzellen gibt es $A_{in}(V_k^{2n})$ bzw. $A_{out}(V_l^n)$ *Variationen ohne Wiederholung*, für die interzelluäre Verknüpfung $A_{array}(\bar{V}_{2n-k}^n)$ *Variationen mit Wiederholung*, insgesamt also:
>
> $$A_{in} \times A_{array} \times A_{out} = \frac{(2n)!}{(2n-k)!} \times n^{2n-k} \times \frac{n!}{(n-l)!}$$

[4] Vereinfachend gehen wir von *asymmetrischen* Eingängen aus (zum Beispiel eines Komparators).

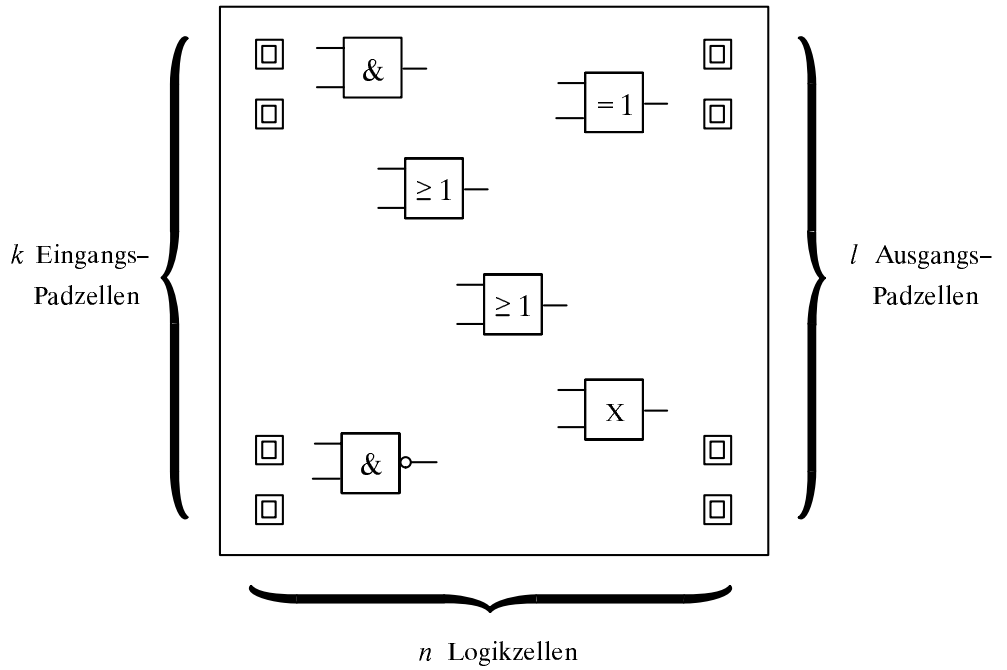


Bild 1.3: Zum kombinatorischen Potential eines Gate-Arrays

(zur Kombinatorik siehe [Bronstein, 1981]). Ein Zahlenbeispiel: Selbst für einen sehr niedrig integrierten Gate-Array-Master mit $n = 16, k = 4, l = 2$ ergeben sich schon $\approx 10^{42}$ mögliche, wenn auch nicht unbedingt sinnvolle Verknüpfungsmuster.

Auf einer vierten Ebene der Analyse, entlang dem „System-Ast“ in Bild 1.2, können weitere Faktoren unterschieden werden, die zur Komplexität maßgeblich beitragen:

- Nichtlinearität

  Die Beschreibung des dynamischen Verhaltens nichtlinearer Systeme ist bei weitem komplexer als im linearen Fall. Lineare Dynamik zeichnet sich im allgemeinen dadurch aus, daß verschiedene Anfangszustände auf denselben eingeschwungenen Endzustand übergehen, ohne instabiles Verhalten zu verursachen. Das dynamische Verhalten nichtlinearer Systeme dagegen ist schwerer zu erfassen: hier variiert der Endzustand mit dem Anfangszustand, und instabiles Verhalten ist prinzipiell möglich. Wirkungen hängen nicht geradlinig von den Ursachen ab, sie beeinflussen sogar rückkoppelnd die Ursachen und können zu drastisch sich selbst verstärkenden Prozessen führen. Die nichtlineare Dynamik durchkreuzt unsere intuitive Erwartung [Flood, 1987, S. 180] und nährt die populärwissenschaftlichen Chaostheorien [Brügge, 1993].

- Asymmetrie

  Asymmetrien, wie sie zum Beispiel in der Biologie durch differenziertes Zellwachstum auftreten und einen Organismus erst als solchen von einer bloßen Zellanhäufung unterscheiden, erschweren das Verständnis für die Gesamtstruktur. In der Technik potenzieren Asymmetrien den Entwurfs- und Konstruktionsaufwand, da einmalig auftretende Teilstrukturen ein planmäßiges Wiederverwenden vereiteln.

- „Lokale Anarchie“

  Nicht*holonome* (griech. *holos* = ganz, *nomos* = Gesetz) Systemeigenschaften schließlich bergen die Unwägbarkeit eines lokal unkontrollierbaren Verhaltens, beispielsweise instabiler Schwingungen durch positive Rückkopplung. Das Zustandsverhalten eines nichtholonomen Systems ist unberechenbar. In der VLSI-Technik überwiegen deswegen synchrone, einem zentralen Takt unterworfene Schaltungen, um auf diese Weise asynchrone Phänomene, wie *Hazards*, *Races* und *Glitches*, in den Griff zu bekommen [Seifart, 1988].

Die aufgezeigten Dimensionen des Komplexitätsbegriffs sind in erster Linie qualitativ und dienen dem Grundverständnis und der Einordnung komplexer Systeme. Die einzelnen Aspekte sind von grundsätzlicher Bedeutung sowohl für Software- als auch für Hardware-Entwürfe. Die vierte Zerlegungsebene nach FLOOD spielt die technische und wirtschaftliche Schlüsselrolle: So erlauben erst die Layoutsymmetrien und der arbeitsteilige, lokal isolierte Teilentwurf auf Bausteinebene die Höchstintegration elektronischer Schaltungen. Auch die Softwaretechnik erkennt im (objektorientierten) Bausteinentwurf das Potential der Wiederverwendung: *Megaprogramming* ist hier das Schlagwort [Boehm & Scherlis, 1987; Johnson *et al.*, 1992].

Was den Dimensionen des Komplexitätsbegriffs bislang fehlt, sind objektive Maßeinheiten, die wir nun für Systeme auf der dritten Ebene nach FLOOD, das heißt für die deskriptive Komplexität nachholen werden.

1.2.2 Metriken

Metriken für die Komplexität im Software- und VLSI-Entwurf orientieren sich an Kostenfaktoren, die maßgeblich die Wirtschaftlichkeit der Implementierung von Algorithmen bestimmen. Entsprechend sind die formalen Berechnungsmodelle auf technische Parameter ausgerichtet (Entwicklungs- und Wartungskosten — der Faktor „Mensch“ also — bleiben unberücksichtigt): In der Softwaretechnik (SW) sind dies Rechenzeit T_{SW} und Speicherplatz S einer TURING-Maschine, in der Hardwaretechnik (HW) Chipfläche A und Signallaufzeit T_{HW} eines Chips.

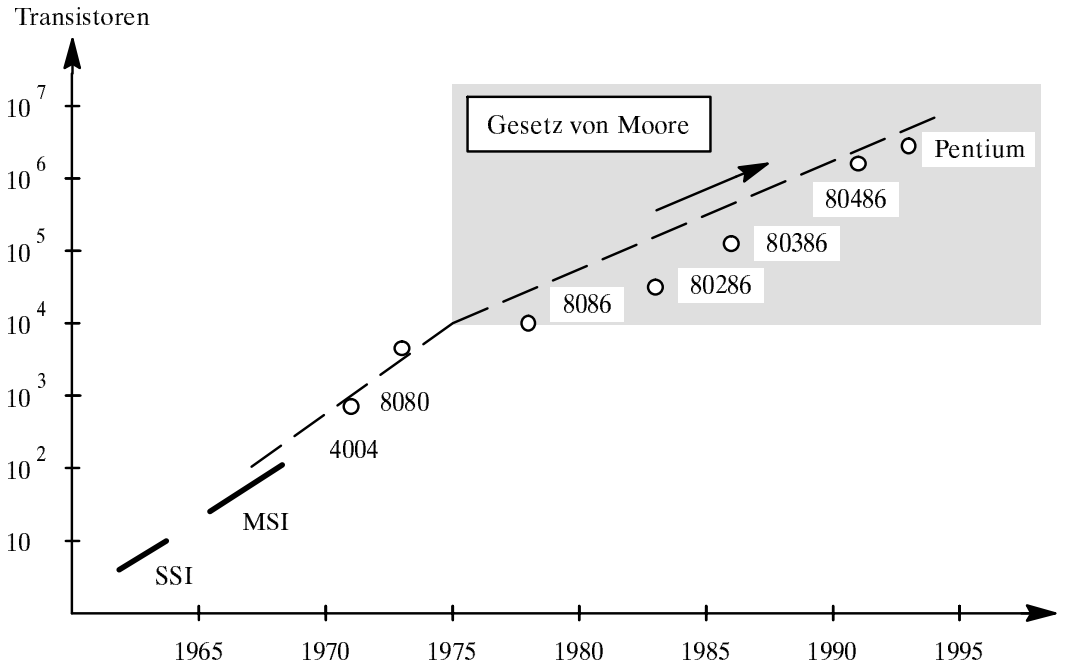


Bild 1.4: Integrationsdichte: Evolution der Intel-Prozessoren

Daß die Hardware-Komplexität, gemessen an der Zahl integrierter Transistoren oder binärer Speicherzellen, exponentiell wächst (Bild 1.4), verdankt sie den symmetrischen, sich wiederholenden Layoutstrukturen.[5] Mit gleichem kann der Software-Entwurf trotz der seit 25 Jahren geforderten Strukturierung und Modularisierung der Programme [Dijkstra, 1968; Parnas, 1972a; Wirth, 1971] nicht aufwarten. Gemessen an *quantitativen* Eigenschaften, zum Beispiel den Programmzeilen, bleibt die Softwaretechnik stets um Größenordnungen zurück. Skalierbare Struktureinheiten fehlen auf der Softwareseite (siehe den Aspekt der Asymmetrie bei FLOOD). Aus diesem Grund sind quantitative Komplexitätsvergleiche zwischen Soft- und Hardware-Entwürfen ungeeignet: Vergleiche der Art „SSI-MSI-LSI versus Programming-in-the-Small“ und „VLSI versus Programming-in-the-Large“ verbieten sich von selbst.

Ein mögliches *Tertium comparationis* der Soft- und Hardwarekosten läßt sich auf der Ebene der Betriebsmittel konstruieren: Da der Aufwand an Speicherplatz mit den Kosten für Halbleiterspeicher gekoppelt ist und diese wiederum von der Ausbeute $Y(A)$ (*yield at wafer sort*) der Chipfertigung abhängen, stellt die Chipfläche A einen gemeinsamen Kostenfaktor

[5] Gordon MOORE, Mitbegründer von Intel, stellte in [Moore, 1975] die Regel auf, nach der sich die Integrationsdichte der Halbleiterschaltungen alle 18 bis 24 Monate verdoppelt. Seine Voraussage von 1975 hat sich bislang bestätigt und wird wohl über das Jahr 2000 hinaus gültig bleiben.

dar. Sie bestimmt die Ausbeute nach der POISSON-Verteilung: $Y(A) \sim e^{-Ad}$, konstante Parameter der Fertigung, vor allem die mittlere Defektdichte d des Wafers, vorausgesetzt. Die Herstellungskosten wiederum verhalten sich umgekehrt proportional zur Chipausbeute.

Der Bedarf an Betriebsmitteln läßt sich auf der Grundlage formaler Berechnungsmodelle nur abschätzen. Dies ist Gegenstand der klassischen Komplexitätstheorie. Wir wollen im folgenden die Komplexität von Algorithmen an einigen Beispielen aus der Literatur aufzeigen, und zwar getrennt als Software-Implementierung (Programm) und als Hardware-Implementierung (VLSI-Layout). Die gewöhnungsbedürftige Notation für Komplexitätsfunktionen stellen wir voran.

Kostenfunktionen: obere und untere Schranken

Im allgemeinen ist man weniger an den exakten Werten einer Kostenfunktion[6] $f : \mathcal{N} \to \mathcal{R}$ interessiert, sondern vielmehr an ihrer qualitativen Wachstumsrate, der sogenannten *Ordnung*. Für die obere Komplexitätsschranke wird die Ordnung mit O (gesprochen: „groß O“) — dem LANDAUschen Symbol — bezeichnet und wie folgt definiert:

> $f(n) = O(g(n))$: „f hat *höchstens* die Wachstumsrate g“, wenn es positive Konstanten c und n_0 gibt, für die gilt $f(n) \leq cg(n)$ mit $n \geq n_0$. Für kleine n kann $f(n)$ beträchtlich größer als $g(n)$ sein. Mit Annäherung an n_0 aber geht der Unterschied zurück bis $f(n)$ höchstens proportional zu $g(n)$ wächst. Die Proportionalität wird durch c nach oben beschränkt.

Für die untere Schranke wird die Ordnung mit Ω (gesprochen: „groß Omega“) bezeichnet:

> $f(n) = \Omega(g(n))$: „f hat *mindestens* die Wachstumsrate g“, wenn es eine Konstante $c > 0$ gibt, so daß für alle Werte von n $f(n) \geq cg(n)$ gilt. Dadurch wird die Wachstumsrate von $f(n)$ nach unten begrenzt. Insbesondere gilt $f(n) = \Omega(g(n))$, wenn $g(n) = O(f(n))$.

Als *Problemgröße* dient im allgemeinen eine natürliche Zahl n, auf die sich die Bewertung der Güte eines Algorithmus bezieht, beispielsweise die Zahl der Elemente für ein Sortierverfahren oder allgemein die Vektorlänge einer Eingabe. Weiterhin unterscheidet man das Komplexitätsverhalten von Algorithmen nach *worst case* und *average case*: Das „Verhalten im schlechtesten Fall“ berücksichtigt diejenigen Eingabevektoren, für die der Algorithmus für einen Einzelfall der Problemgröße n den maximalen Aufwand an Betriebsmitteln erfordert. Letzter Fall, „Verhalten im Mittel“, geht von einer gegebenen Wahrscheinlichkeitsverteilung der Eingaben aus und führt auf gemittelte Aufwandskosten.

[6] $\mathcal{N}$, $\mathcal{R}$: Menge der natürlichen bzw. der reellen Zahlen

Software: Time/Space-Trade-off

Gegenstand der klassischen Komplexitätstheorie ist die Bestimmung der Ordnung *rechner*ausführbarer Algorithmen und deren Einteilung in Komplexitätsklassen. Demzufolge gehen vor allem die Betriebsmittel Rechenzeit und Speicherplatz in das Aufwandskalkül ein: Als Rechenzeit T_{SW} eines Algorithmus stehen die Rechenschritte, um eine Eingabe n zu bearbeiten. Der Speicherplatz S richtet sich nach dem Bedarf an Speicherzellen, um die Werte der Variablen aufzunehmen. Die Ordnung eines Algorithmus hat praktische Bedeutung: Sie erlaubt Aussagen über die Effizienz einer geplanten Implementierung, indem sie angibt, wie sich der Bedarf an Betriebsmitteln mit der Eingabelänge vergrößert. Als effizient gelten *polynomiale* Algorithmen der Ordnung $O(n^k)$ für ein geeignetes k. Unzugängliche, sogenannte *NP-schwierige* Probleme zeichnen sich durch eine exponentielle Ordnung $\Omega(a^n)$ mit $a > 1$ aus oder im Extremfall durch kombinatorisches Wachstum, siehe das Gate-Array-Beispiel im vorigen Abschnitt. Der Betriebsmittelbedarf verschiedener Algorithmen, die dasselbe Problem lösen, läßt sich unmittelbar vergleichen. Die Tabelle 1.1 nennt Beispiele [Claus & Schwill, 1993]:

Sortierverfahren	*Rechenzeitbedarf*	*Speicherplatzbedarf*
BubbleSort	$T_{SW}(n) = O(n^2)$	$S(n) = O(1)$
QuickSort	$T_{SW}(n) = O(n \log_2 n)$ (im Mittel) $T_{SW}(n) = O(n^2)$ (worst case)	$S(n) = O(\log_2 n)$
BucketSort	$T_{SW}(n) = O(n)$	$S(n) = O(n)$

Tabelle 1.1: Zur Vergleichbarkeit von Software-Komplexität

Die Gegenüberstellung der gängigen Sortieralgorithmen macht eins deutlich: Die Forderung nach minimalem Einsatz beider Betriebsmittel, Rechenzeit *und* Speicherplatz, ist selten zu erfüllen. In der Regel konkurrieren die Kostenfaktoren miteinander. Man sucht deshalb den *Kompromiß* in der Mittelverteilung, englisch: *trade-off*. Des weiteren verdeutlichen die Angaben die *relative* Aussagekraft der Ordnung einer Kostenfunktion: So führt eine Verdopplung der zu sortierenden Elemente auf eine Quadrierung der Rechenzeit beim *BubbleSort*-Verfahren, wohingegen der Rechenzeitbedarf beim *BucketSort*-Verfahren nur linear wächst (*BucketSort* ist allerdings nur für Spezialfälle anwendbar). Absolute Komplexitätsangaben haben in diesem Zusammenhang keinen Sinn.

VLSI: Area/Time-Trade-off

Die Kostenfaktoren in der VLSI-Technik adressieren „konkrete“ Physik: Chipfläche A und Signallaufzeit T_{HW} auf dem Chip. Da das VLSI-Problem im zweidimensionalen Entwurfsmedium *Layout* liegt, müssen entsprechende Berechnungsmodelle die geometrische Plazierung der aktiven Elemente und ihrer Verbindungen in der Chipebene berücksichtigen. Im Gegensatz zur Software-Komplexität geht somit neben der eigentlichen Verarbeitung von Information auch deren räumliche Übertragung in die Betrachtung mit ein. Die prinzipielle Schwierigkeit ist also vergleichbar mit der Aufwandsberechnung für die Kommunikation in verteilten Systemen.

Um vom Stand der VLSI-Technik unabhängig zu sein, wird für den Kostenfaktor Chipfläche die minimale Strukturgröße λ als Bezugsgröße eingeführt.[7] Nach dem THOMPSONschen VLSI-Berechnungsmodell gelten dann folgende Annahmen [Thompson, 1979]:

- Alle Leiterbahnen auf dem Chip verlaufen in einem rechtwinkligen Raster (*Gittermodell* oder *Manhattan-Architektur*) und besitzen die minimale Breite $\lambda > 0$. Höchstens zwei Bahnen können sich an einem Ort überlappen. Transistoren und Anschlüsse beanspruchen die minimale Fläche λ^2.
- Die Signallaufzeit auf dem Chip soll unabhängig von der Leiterbahnlänge zwischen zwei aktiven Elementen sein. Es liegt also ein synchrones Modell vor: nur ein Bit Information pro Zeiteinheit wird übertragen.[8] Als Rechenzeit T_{HW} für den implementierten Algorithmus gilt die Dauer der längsten Folge von Signallaufzeiten von der ersten Eingabe bis zur letzten Ausgabe.

Mit Hilfe des VLSI-Berechnungsmodells können asymptotische *untere* Schranken für die Komplexitätsfunktionen angegeben werden. Beispiele zeigt Tabelle 1.2. Die Kostenfunktion AT^2_{HW} beschreibt den Mittelbedarf realistischer als Kostenfunktionen, die getrennt für A und T_{HW} oder für das einfache Produkt AT_{HW} aufgestellt werden. Dies geht darauf zurück, daß die AT^2_{HW}-Berechnung nicht auf Speicherplatzanforderungen oder Eingangs/Ausgangs-Datenströmen, sondern auf dem Informationsfluß basiert.

Die THOMPSONsche Wachstumsfunktion $AT^2_{HW} = \Omega(n^2)$ ist wegen ihrer Allgemeingültigkeit für VLSI-Aufwandsbetrachtungen von besonderer Bedeutung. Ihre zugrundeliegende

[7] Die Strukturgröße λ und der gleichnamige Parameter λ' nach MEAD und CONWAY [Conway & Mead, 1980] unterscheiden sich: λ steht für die minimale Leiterbahnweite, während λ' hierfür 4 bis 6 λ je nach Maskenlayer definiert. Da es bei einer Abschätzung aber nur um die Größenordnung geht, können konstante Faktoren vernachlässigt werden.

[8] Während die Annahmen zur Chipfläche unstrittig sind, ist die einheitliche Laufzeit der Signale kritisch: Eine Leiterbahn weist parasitäre Eigenschaften auf, kapazitive und induktive, und diese wachsen jeweils linear mit der Leiterbahnlänge l. Die Signallaufzeit verhält sich also proportional zu l^2.

Algorithmus	*Problemgröße n*	*Kostenfunktion* AT^2_{HW}
FOURIER-Transformierte	n Stützpunkte	$AT^2_{HW} = \Omega(n^2)$
BOOLEsche Funktionen	n Eingänge	$AT^2_{HW} = \Omega(n^2)$
Integer-Multiplikationen	n-bit-Multiplikand	$AT^2_{HW} \geq \Omega(\frac{n^2}{64})$
Matrizen-Multiplikationen	$(n \times n)$-Matrix	$AT^2_{HW} = \Omega(n^4)$

Tabelle 1.2: Area/Time-Trade-off in Silizium implementierter Algorithmen

Beweistechnik sei deshalb kurz umrissen. Wir wählen hier die prägnante Darstellung von Randal BRYANT [Bryant, 1991], da sie den umfangreichen theoretischen Unterbau, in [Thompson, 1979; Ullman, 1984] nachzulesen, vermeidet:

> „The idea of these proofs is to show that if we consider any imaginary line drawn across the chip such that half of the inputs occur on either side, then αn bits of information must cross that line in order to correctly compute the function, for some constant $\alpha > 0$. Thus, if the chip is w units wide, we must have $wT \geq \alpha n$. This property holds across either dimension of a rectangular chip, and hence we can assume that $A \geq w^2$. Combining these yields an area-time lower bound $AT^2 = \Omega(n^2)$. These arguments hold even if some of the inputs are supplied to a single point on the chip sequentially. The only required assumption is that the circuit be „where-oblivious“, i.e., each input is supplied only once and at a location independent of its value. An imaginary line can then be used to partition the inputs into two sets, where inputs arriving at points intersected by the line can be placed in either set to equalize their sizes.“ [Bryant, 1991, S. 206]

1.3 Entwurfskomplexität

Die Metriken für den technischen Aufwand, einen Algorithmus in Soft- oder Hardware zu implementieren, berücksichtigen nur quantifizierbare Eigenschaften der *Endprodukte*. Der Aufwand für deren *Entwicklung* und *Wartung* läßt sich bestenfalls abschätzen, nicht aber berechnen — zu unterschiedlich sind die Herstellungsprozesse. Die Kostenfaktoren für die nichttechnischen Ressourcen — Personal und Entwicklungszeit — entziehen sich den formalen Berechnungsmodellen. Gerade aber der Zeitfaktor der Markteinführung entscheidet über den Erfolg eines Produkts (*Time-to-market*). Der Entwurf ist eben ein Problem der mittleren Kategorie nach WEAVER: seine „organisch“ gewachsene Komplexität verschließt sich einer exakten Bestimmung. Um dennoch die *qualitativen* Unterschiede zwischen Soft-

und Hardware-Entwürfen zu ermessen, wollen wir auf die BROOKSsche Entlehnung der aristotelischen Begriffe von „Akzidens und Substanz“ zurückkommen. Für den Akzidensteil gibt es, wie wir noch sehen werden, Ansätze und begründete Hoffnungen, die damit verbundene Komplexität zu bewältigen.[9] Die Essenz (Substanz) der Entwurfskomplexität hingegen bleibt. Wir wollen sie aber zumindest in ihren jeweiligen Ausprägungen überschauen.

1.3.1 Die Essenz der Software-Komplexität

Laut BROOKS liegt die Software-Essenz in der Vernetzung der Konzepte: Datenstrukturen, Algorithmen und Prozeduraufrufe. Nicht in der Darstellung, deren Verifikation und Wiedergabetreue verbirgt sich das Kernproblem — das macht lediglich den Akzidensteil der Komplexität aus —, sondern vielmehr in der Spezifikation, im Entwurf und Test der konzeptionellen Vernetzung. Zwar begehen Programmierer Syntaxfehler, diese sind aber von geringerer Bedeutung (da ihr Aufspüren automatisierbar ist) als die konzeptionellen Fehler, die sich in den Softwaresystemen verbergen. Aus welcher Sicht Software auch beschrieben wird, die Essenz bleibt. Ihre spezifische Ausprägung hat viele Gesichter. Die wesentlichen seien hier skizziert [Brooks, 1987]:

- Amorphe Strukturen

  Die Crux der Softwaretechnik sind die unregelmäßigen Programmstrukturen. Im Gegensatz zu den regelmäßigen Waferstrukturen in der VLSI-Technik lassen sich Programme nicht auf der Grundlage einer minimalen Strukturgröße und eines Satzes von Entwurfsregeln „skalieren“. Zumindest oberhalb der Anweisungsebene (*statements*) gibt es keine Struktureinheiten, die nach dem Bausteinprinzip etwa zu größeren Programmen erweitert werden könnten. Programmieren ist ein vernetzter Prozeß. Programmerweiterungen verlangen neue, auf das jeweilige Problem zugeschnittene Elemente, die wechselseitig unter sich und mit den bereits entworfenen Elementen interagieren. Der Entwurfsaufwand wächst also stärker als linear mit der Programmlänge. Nebenwirkungen durch Aktualisierungen (*Updates* und *Upgrades*) schleichen sich ein.

  In den Eigenschaften der amorphen Programmstrukturen liegen weitere Schwierigkeiten begründet: Sie beschränken die effektive Arbeitsteilung und Kommunikation zwischen Entwicklung, Test und Wartung, weil in jedem Ressort die Software als Ganzes verstanden sein muß. Sollte während der Entwicklung das verantwortliche Personal wechseln, bedingt die mangelnde Übersichtlichkeit der vernetzten Strukturen lange Einarbeitungszeiten mit flachen Lernkurven. Auch hier gilt die Sportdevise „never change a winning team“.

[9] Akzidenzien, die dem Software-Entwurf „zufällig“ anhaften, beschreibt eindrucksvoll die Essay-Sammlung von BROOKS über Software-Engineering *The Mythical Man-Month* [Brooks, 1975]: „Adding man power to a late project makes it later“.

- Konformität

  Software ist kein isoliertes Produkt und kann auch nicht isoliert erstellt werden. Nur im Anwendungskontext und in der vorgesehenen Laufzeitumgebung (Benutzungsoberflächen, Betriebssysteme, Datei- und Datenformate) vermag sie, Teilprobleme zu lösen. Die insularen Komponenten eines Softwaresystems werfen Schnittstellenprobleme auf, die den Software-Entwurf erheblich erschweren. Die Vielfalt der Anwendungen, die konzeptionell den gleichen Datenbestand bearbeiten, verlangt eine flexible Anpaßbarkeit der Komponenten. Der damit verbundene Entwurfsaufwand beschränkt sich nicht allein auf Entwurfskorrekturen (*redesigns*). Schnittstellen sind essentielle Bestandteile und Randbedingungen des Software-Entwurfs. Konformität ist ein ständiger Komplexitätsfaktor.

- Veränderbarkeit

  Software unterliegt einem permanenten Änderungsprozeß, da sie die Funktion eines Systems realisiert und die Systemfunktion für den Anwender die entscheidende Rolle spielt. Als abstraktes, eindimensionales „weiches" Produkt ist Software naturgemäß mit weniger Aufwand und Kosten zu modifizieren als konkrete, räumliche „harte" Konstruktionen aus Hardware. Wenn ein Softwareprodukt erfolgreich in den Markt eingeführt wurde und es sich bewährt hat, sind mittel- und langfristig zwei Änderungsprozesse am Werk: Zum einen werden neue Anwendungsgebiete erprobt, die am Rande oder jenseits der ursprünglich vorgesehenen Anwendung liegen. Das führt zu einem Änderungsdruck der Anwender auf die Entwicklungsabteilungen: Der Funktionsumfang soll modifiziert oder erweitert werden. Zum anderen muß sich eine in der Anwendung etablierte Software dem Stand der Hardwaretechnik anpassen (neue Prozessoren und Speichermedien, Bildschirme, Drucker).

  > „In short, the software product is embedded in a cultural matrix of applications, users, laws, and machine vehicles. These all change continually, and their changes inexorably force change upon the software product." [Brooks, 1987, S. 12]

- Unsichtbarkeit

  Software — die „weiche Ware" — ist ihrer Natur nach immateriell, ihre Realität räumlich nicht faßbar. Anschauliche geometrische Abstraktionen, wie sie für die Hardware typisch sind (Schaltdiagramme, Netzlisten, Floorplan, Stick- und Maskengeometrien), fehlen. Sobald wir versuchen, Softwarestrukturen zu skizzieren, geraten wir in das Netz gerichteter Graphen: Steuer- und Datenflüsse, Ablauf- und Abhängigkeitsgraphen, Namensräume überlagern sich. Wir verfügen nicht (oder nur sehr eingeschränkt) über das leistungsstarke Mittel der Visualisierung. Unser mentaler Prozeß der Mustererkennung findet nur wenig Unterstützung: Das Auffinden relevanter Information im Quelltext kann sich kaum an charakteristischen Mustern orientieren. Das Fehlen einer geometrischen Darstellung kompliziert aber nicht nur den Entwurfsprozeß des einzelnen Entwerfers, auch die Kommunikation im Entwurfsteam wird erschwert.

Die Essenz der Software-Komplexität liegt folglich in den *transitiven* Eigenschaften der Softwareprodukte: in ihrer amorphen Gestalt, der leichten Formbarkeit und ihrer fehlenden Gegenständlichkeit. Das Gesagte ist typisch für die Entwicklung von Industriesoftware, für das Programmieren im großen [DeRemer & Kron, 1976].

1.3.2 Die Essenz der Hardware-Komplexität

Auch wenn Hardware nicht isoliert von Software entworfen wird (die Werkzeuge des Entwerfers sind Software-Erzeugnisse, die Konfiguration der Hardware unterliegt der Systemsoftware), können wir auch hier Charakteristisches ausmachen. Zunächst die Randkomplexität: Nicht der Software-Entwurf, sondern die Anwendung verschiedener Software*werkzeuge* mit ihren Schnittstellen tragen zur akzidentiellen VLSI-Komplexität bei. Seitdem die Werkzeuge nicht mehr physikalischer Art sind (Layoutentwürfe am Reißbrett) und die vorgezogene Schnittstelle zwischen Chipentwerfer und Chiphersteller — die „simulierte Netzliste“ — den Einfluß der Fertigung auf den Entwurf unterbindet, zeigt sich die softwarebedingte Komplexität in Form inkompatibler Datenstrukturen und -austauschformate. Auf sie zielt die Forderung nach *CAD-Framework*-Lösungen (kompatible Werkzeuge, die auf einer gemeinsamen Datenbasis arbeiten [Barnes *et al.*, 1992; Bhat & Taku, 1990; Quibeldey-Cirkel, 1993b; Quibeldey-Cirkel, 1993c; Rammig & Steinmüller, 1992; Siepmann, 1991]). Setzt man die Implementierung und die Anbindung der Entwurfswerkzeuge einmal als fehlerfrei voraus, so läßt sich die Essenz der Hardware-Komplexität an folgenden Punkten aufzeigen (siehe auch [Séquin, 1983]):

- Räumlichkeit

  Die Mehrdimensionalität der VLSI-Technik (vertikal integrierte Transistoren sind Stand der Technik) schafft das hardwarespezifische Problem der Plazierung und Entflechtung in einem ursprünglich strukturlosen Bereich. Die damit verbundene topographische Komplexität, zum Beispiel bei der Integration eines *Single-Chip-PCs*, ist nur noch mit automatisierten Entwurfswerkzeugen (Layoutroutern) zu bewältigen. Es handelt sich aber nicht nur um einen rechenintensiven Entwurfsschritt auf der untersten Implementierungsebene: Die aus Effizienzgründen erforderliche physikalische Partitionierung beeinflußt auch die Entwurfsentscheidungen auf höheren Abstraktionsebenen, wie zum Beispiel die logische Partitionierung und die Festlegung der Modulschnittstellen.

- Bindungen

  Im Gegensatz zum Software-Pendant „jump to subroutine“ ist die Punkt-zu-Punkt-Kommunikation zwischen Modulen in einem physikalischen Medium sehr kostenintensiv. Die Bindungskraft eines Moduls auf einer bestimmten Abstraktionsebene weist zwei Aspekte auf: Kohäsion und Kopplung. Unter diesen Aspekten läßt sich die Art

und Weise, wie Komponenten zu Modulen kombiniert werden, auf der nächst höheren Beschreibungsebene charakterisieren. Kohäsion ist ein Maß für die modulinterne Bindung, zum Beispiel die *intra*zelluäre Verdrahtung einer Layoutzelle. Kopplung meint die Bindungen und Interaktionen zwischen Modulen, zum Beispiel die *inter*zelluäre Verdrahtung zwischen Layoutzellen. Die Tabellen 1.3 und 1.4, von Carlo SÉQUIN aus der Soft- in die Hardwaretechnik übertragen [Séquin, 1983], vermitteln einen Eindruck über die vielfältige Natur der Bindungen im VLSI-Entwurf.

Name	*Explanation*	*Example*
functional	all parts contribute to function	operational amplifier
sequential	portion of a data-flow diagram	filter cascade
communicational	data abstraction, same data used	LIFO stack
spatial	need to be physical close	sensor array
procedural	same procedure block	bus controller
temporal	used at a similar time	start-up circiutry
logical	a group of similar functions	I/O-pad library
coincidental	random collection	miscellaneous wiring

Tabelle 1.3: Kohäsionsaspekte in VLSI-Modulen

- Test

  Die Softwaremaxime „Der Rechner macht keine Fehler, er führt nur fehlerhafte Programme aus“ gilt in der Hardwaretechnik nur bedingt. Hier ist das Zielmedium der Implementierung nicht der Rechner oder eine *virtuelle Maschine*, sondern die Halbleiterfertigung. Diese ist bekanntlich fehleranfällig, was die hardwarespezifische Notwendigkeit bedingt, zwischen logischem Entwurf und physikalischer Konstruktion zu unterscheiden. Das Produkt muß nach dem verifizierten Entwurf zusätzlich getestet werden. Verifizierbare Entwürfe mit ausreichender Testbarkeit stellen hier die Probleme [Wojtkowiak, 1988]. Die Problemstellungen aufgrund transitiver Eigenschaften, wie Konformität und Veränderbarkeit, entfallen jedoch: Einmal entworfen und hergestellt, erlaubt die „harte Ware“ naturgemäß keine Änderungen. Während sich in der Softwaretechnik der Ausgangstest in der Regel auf *Redundancy Checks* der Programmkopien beschränken kann, nimmt das Testproblem in der Hardwaretechnik eine zentrale Stellung ein. Im Gegensatz zum Softwaretest ist die Zugänglichkeit der

Perspective	*Degree of Coupling*	*Example*
Topology:	linear flow-through	filter chain
	hierarchical tree	carry look-ahead
	lattice modularity	memory array
	irregular mosaic	processor control
Interaction:	continuous	light sensor
	periodic	timer
	occasional	reset
Bandwidth:	high	video signal
	burstly	disk head
	low	voice signal

Tabelle 1.4: Kopplungsaspekte zwischen VLSI-Modulen

zu testenden Hardwarestrukturen extrem eingeschränkt: Die feste Zahl der Chipanschlüsse begrenzt die Steuerbarkeit und Beobachtbarkeit interner Schaltungsknoten entscheidend und potenziert den Testaufwand [Parker & Williams, 1979].

- Sichten

  Kennzeichnend für VLSI-Entwürfe ist die perspektivische Strukturierung der Entwurfsschritte: Tabelle 1.5 zeigt die Abstraktionsebenen und Sichten nach Daniel GAJSKI [Gajski, 1988]. Diese Betrachtungsweise entflechtet einerseits das immense Datenaufkommen, birgt aber andererseits die Gefahr unverträglicher Daten, der *Inkonsistenz* zwischen den Sichten derselben Ebene und den Sichten verschiedener Ebenen. Ursächlich hierfür sind die Freiheitsgrade, die dem Entwerfer grundsätzlich offenstehen, um von einer Darstellung in die andere zu gelangen: *Expansion* in Richtung einer höheren Abstraktion und *Implementierung* in Richtung der Fertigungslinie (Layoutgeometrien).

- Datenarchivierung und -verwaltung

  Prinzipiell können wir drei Arten von VLSI-Daten unterscheiden: Entwurfs-, Fertigungs- und Meta-Daten. Die Entwurfsdaten dokumentieren die Vorgaben und Ergebnisse der Entwurfsphasen: Gesamt- und Teilspezifikationen, Grafik- und Layoutgeometrien, Simulationsdaten. Die Fertigungsdaten beinhalten die Vorgaben für die Chipfertigung: simulierte Netzlisten, Testdaten und Anschlußbelegungen. Die Meta-Daten schließlich sind Daten über Daten und dienen zum Beispiel der technischen Dokumentation, der Datenarchivierung und dem Projektmanagement. Das Mengenverhältnis der Datenaufkommen ist unbestimmt: Eine gegebene Menge an

Levels of Abstraction	Domains of Description		
	behavioral	*structural*	*physical*
system	performance specs.	CPUs, memories, controllers, busses	physical partitions
algorithmic	algorithms	hardware modules, data structures	clusters
micro-architectural	register transfers, state sequencing	ALUs, MUXs, registers, microsequencer	floorplans
logic	Boolean equations	gates, flip-flops, cells	cell, module plans
circuit	transfer functions, timing	transistors, wires, contacts	layout

Tabelle 1.5: VLSI-Schichtung versus Entwurfssichten nach GAJSKI

Fertigungsdaten, mit der letztlich die Systemspezifikation in Silizium realisiert werden soll, kann zu beliebig umfangreichen Mengen an Entwurfs- und Meta-Daten in Relation stehen. *A priori* läßt sich über diese Relation nichts aussagen, da sie von der angewandten Entwurfsmethode, den Richtlinien der Projektdurchführung und dem Datenhaltungskonzept abhängt. Nur soviel: Das Aufkommen an Entwurfs- und Meta-Daten ist im VLSI-Entwurf um ein Vielfaches höher als im Software-Entwurf, wo sich die Fertigungsdaten (Zielcode) direkt aus den Entwurfsdaten (Quellcode) erzeugen lassen.

Die Essenz der Hardware-Komplexität liegt folglich in der „Personalisierung“ der räumlich starren Eigenschaften eines Siliziumwafers. Die damit einhergehenden hohen Entwicklungskosten verlangen Strategien, die den verifizierbaren Entwurf (*correctness by verification*) und den automatisierten Entwurf (*correctness by construction*) zum Ziel haben, um so kostspielige Redesigns zu vermeiden und schon bei der ersten Implementierung funktionsfähige Muster zu erhalten.

Was mit Blick auf die Entwurfskomplexität als charakteristisch für Soft- und Hardware getrennt skizziert wurde, gilt nur eingeschränkt für die zukünftige Entwicklung. Es gibt Anzeichen dafür, daß sich die Charakteristiken von Hard- und Software immer mehr angleichen (Stichwort: *Co-Design*), je größer und vernetzter die zu entwerfenden Systeme werden. Dies gilt besonders für das Testproblem und die Bindungskomplexität.

Fazit

Es sollte deutlich geworden sein, daß die ingenieurmäßige Konstruktion anspruchsvoller Soft- und Hardware ein komplexes Unterfangen bedeutet. Komplexität, als deskriptiver Pluralismus nach ROSEN begriffen oder im konzeptionellen Rahmen nach FLOOD, ist keine Eigenschaft des zu entwickelnden Systems an sich, sondern eine zentrale Entwurfsgröße. Hierauf lassen sich auch keine quantitativen Metriken anwenden. Im Kontext kreativer Entwurfsprozesse geht der Faktor Mensch als unwägbares Betriebsmittel ein. Um so mehr, wenn der Umfang der Aufgabe ein arbeitsteiliges Vorgehen auferlegt, um ökonomische, das heißt überprüfbare, reproduzierbare und wartbare Produkte zu entwerfen. Die Entwurfsgröße Komplexität ist, um nochmals die BROOKSsche Metapher zu bemühen, essentiell, gegen sie gibt es kein methodisches Mittel: „no silver bullet“. Was aber nicht heißt, daß die Begleitformen der Entwurfskomplexität — die Akzidenzien — hingenommen werden müßten.

Die Hoffnungen sind durchaus begründet: althergebrachte reduktionistische Methoden, wie hierarchische Strukturierung und Dekomposition, stehen seit Jahrzehnten zur Verfügung. Innovative integrative Entwurfsumgebungen (Frameworks) werden in Kürze folgen. Ihr Ziel ist die größtmögliche Reduktion der babylonischen Vielfalt an Beschreibungsformaten und Schnittstellen im VLSI-Entwurf. Konzeptionelles Modellieren schließlich als weitere „hope for the silver“ vereint interdisziplinäre Ansätze, die — wie wir noch zeigen werden — das Leitmotiv der Abhandlung tragen. Die folgenden Kapitel skizzieren das methodische Rüstzeug des Entwerfers, um den Zufälligkeiten im Entwurf zu begegnen, die Entwurfskomplexität systematisch und wirtschaftlich zu bewältigen.

2

Komplexitätsbewältigung I: Reduktionismus

Auch wenn der *Ismus* abwertend mit „bloßer Theorie“ gleichgesetzt wird, ist der Reduktion*ismus* in der Theorie der Komplexitätsbewältigung keineswegs eine esoterische Lehrmeinung, eine „school of thought“. Der Reduktionismus liefert vielmehr das Repertoire an komplexitätsbewältigenden Methoden und Techniken für die meisten wissenschaftlichen Disziplinen. In der Regel verfügen wir über dieses Repertoire intuitiv, ohne darüber zu reflektieren. Die Systemtheorie hingegen setzt sich den Reduktionismus zum wissenschaftlichen Gegenstand. Gerald WEINBERG definiert sie sogar als Wissenschaft der „Simplifizierung“ und unterstreicht zugleich ihre Wichtigkeit:

> „NEWTON was a genius, but not because of superior computational power of his brain. NEWTON's genius was on the contrary his ability to simplify, idealize, and streamline the world so that it became, in some measure, tractable to the brains of perfectly ordinary men. By studying the methods of simplification which have succeeded and failed in the past, the general systems theorist hopes to make the progress of human knowledge a little less dependent on genius.“ (zitiert in [Klir, 1985, S. 135])

Allerdings sollten wir nicht alle Aussagen der Form „Komplexes läßt sich auf einfache Gesetze und Strukturen reduzieren“ arglos übernehmen. Bertrand RUSSEL weist auf mögliche Fallstricke einer Simplifizierung hin: Wären zum Beispiel die Asteroiden wesentlich häufiger in unserem Sonnensystem, hätte der überlagernde Einfluß ihrer Gravitation auf die Planetenbahnen die Verallgemeinerungen von KOPERNIKUS und KEPLER und schließlich auch die NEWTONsche Dynamik unmöglich gemacht: „thus we see how complexity can defeat science“ [Rosen, 1977, S. 227]. Auf eine weitere Gefahr eines „blinden Reduktionismus“ macht Robert ROSEN aufmerksam: Für die Analyse komplexer Systeme kann es kontraproduktiv sein, das Systemverhalten durch die Analyse einfacher und leicht zugänglicher Systemeinheiten erforschen und erklären zu wollen.

> „What is ‚natural‘ from the standpoint of easy measurability, and hence easy isolability of subsystems, can often be most unnatural from the standpoint of effective analysis of complex systems.

> [...] To insist on the cell as the ultimate unit of biological organisms, or on individual organisms as the ultimate unit of social structures, is often quite inappropriate for the analysis of many properties of the higher structures. This kind of ‚vested interest' is often called *reductionism*, and is of course most conspicuous in molecular biology. Its basic feature is *not* that it advocates analysis of complex systems into simpler subsystems (this is unavoidable in any case), but rather that it advocates, in advance, a set of abstract subsystems which it posits to be the only admissible units for analysis. But as we have seen, the relativity of system descriptions, together with the multiplicity of descriptions which provide the very definition of complexity, preclude any one class of subsystems, or any mode of analysis, from being universally valid." [Rosen, 1977, S. 230]

Trotz dieser Warnungen soll hier nicht das Wort für einen *Holismus* — für eine ganzheitliche Betrachtung der Komplexität — geredet werden. Das Ganze ist zwar im systemtheoretischen Sinne stets mehr als die Summe der Teile (wenn man so will, ist Komplexität gerade dieses „Mehr"), das Verständnis für ein komplexes System als Ganzes läßt sich aber nur über das Studium seiner Teile erschließen. Komplexe Formen sind im allgemeinen hierarchisch, und ein jeder Reduktionismus hat die Analyse der Hierarchie zum Gegenstand. Abseits dieser systemtheoretischen Diskussion wollen wir uns im folgenden auf die pragmatischen, für die Praxis des Entwurfs komplexer Systeme absolut wichtigen Grundlagen beschränken und das modelltheoretische Rüstzeug für den Systementwerfer zusammentragen. Zugleich sollen die meist unscharfen Begriffe „Modell", „Entwurfsaufgabe" und „Entwurfsprozeß" für die Studie präzisiert werden.

Übersicht: Abschnitt 2.1 skizziert die Begrenzung unserer kognitiven Fähigkeit, komplexe Sachverhalte gleichzeitig zu verarbeiten: Die Kanalkapazität unseres Kurzeitgedächtnisses ist auf eine „magische Zahl" beschränkt. Daß es sich hier dennoch um eine „weiche" Begrenzung handelt, wird am *Chunking*-Prozeß erläutert. Abschnitt 2.2 referiert die grundlegende Arbeit von Herbert SIMON zur inneren Struktur der Komplexität. Danach setzt sich Komplexes stets aus Einfachem zusammen. Das Grundprinzip der *beinahen Zerlegbarkeit* stellt den ersten und wichtigsten Ansatz zur Reduktion der Komplexität dar. Abschnitt 2.3 schließlich untersucht einige Aspekte des *Divide-et-Impera*-Prinzips für den modularen Soft- und Hardware-Entwurf.[1]

2.1 Die „Magische Zahl Sieben"

Bereits in den 50er Jahren, kurz nach der Veröffentlichung der Kommunikationstheorie von Claude SHANNON [Shannon & Weaver, 1949], übernahm die experimentelle Psychologie

[1] Der Abschnitt 2.1 und Teile der Abschnitte 2.2 und 2.3 wurden [Quibeldey-Cirkel & Wojtkowiak, 1991] entnommen.

dessen Begriffe und Lehrsätze, um ihre bislang verstreuten Ergebnisse zur Informationsverarbeitung einheitlich zu analysieren. In dem klassischen Aufsatz *The Magical Number Seven, Plus or Minus Two* [Miller, 1956] verdichtet George MILLER die wesentliche interdisziplinäre Erkenntnis:

> „And finally, what about the magical number seven? What about the seven wonders of the world, the seven seas, the seven deadly sins, the seven daughters of Atlas in the Pleiades, the seven ages of man, the seven levels of hell, the seven primary colors, the seven notes on the musical scale, and the seven days of the week? What about the seven-point rating scale, the seven categories for absolute judgement, the seven objects in the span of attention, and the seven digits in the span of immediate memory? For the present I propose to withhold judgement. Perhaps there is something deep and profound behind all these sevens, something just calling out for us to discover it. But I suspect that it is only a pernicious, Pythagorean coincidence.“ [Miller, 1956, S. 97]

Zwar läßt er die Frage nach dem Warum — dem „design behind it“ — unbeantwortet (um die Antwort bemühen sich die Neurophysiologie und die junge Kognitionswissenschaft [Gardner, 1985]), MILLER bringt aber das Phänomen der beschränkten Kanalkapazität des Menschen in die begriffliche Vorstellungswelt des Ingenieurs. Er verwirft die *Varianz*, die alte dimensionsbehaftete Größe für Information. An deren Stelle führt er das dimensionslose Maß *Bit* ein und interpretiert damit die alten empirischen Ergebnisse neu.[2] Auf diese Weise gewinnt er eine klare Aussage zur kognitiven Bewußtseinsbarriere des Menschen.

Die analysierten Experimente waren methodisch gleich strukturiert. Es wurde getestet, wie genau Probanden Zahlenwerte sensorischen Stimuli zuordnen können, wobei die Antwortzeit keine Rolle spielte: „experiments on absolute judgement“. Das VENN-Diagramm in Bild 2.1 verdeutlicht die beteiligten Informationsmengen: Stimuli S, Antworten A und übertragene Information T.

Die Anzahl der Stimuli-Alternativen wurde so weit erhöht, bis sich die übertragene Informationsmenge asymptotisch einem Grenzwert, der Kanalkapazität C, näherte. Im Fall eindimensionaler Stimuli, wie beispielsweise Tonhöhe, Geschmacksintensität von Salzlösungen oder Punktpositionen auf einer Linie, lag das Mittel der Kanalkapazität bei 2,6 bit mit einer Standardabweichung von 0,6 bit pro Zeiteinheit. Auch bei mehrdimensionaler sensorischer Erregung, wie Salz- und Zuckerkonzentrationen in gemeinsamer Lösung oder Punktpositionen in einer Fläche, kam MILLER auf das gleiche Ergebnis: Der Mensch vermag lediglich 7 ± 2 Kategorien gleichzeitig zu verarbeiten, das heißt selektiv mit Aufmerksamkeit zu belegen. MILLER nennt sie „chunks of information“: Informationsblöcke.

[2] Die SHANNONsche Kommunikationstheorie betrachtet nicht die Bedeutung, sondern ausschließlich den technischen Aspekt (das *Kanal*verhalten) einer Information und ihrer Übermittlung. Als Grad der Wahlfreiheit wird Information definiert über den Logarithmus dualis der Wahlmöglichkeiten.

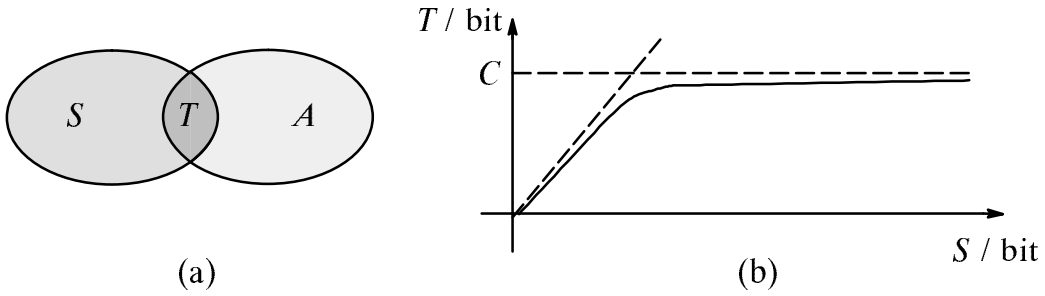


Bild 2.1: Zur menschlichen Kanalkapazität nach MILLER

Die *magische Zahl* wurde durch zahlreiche Merkspannen-Experimente als „Telefonbuchkonstante" bekannt: Normalerweise sind wir in der Lage, sieben Ziffern vom Ablesen im Telefonbuch bis zum Wählen zu behalten, sofern wir weder durch äußere Reize noch durch unsere eigenen Gedanken abgelenkt werden. Kennzeichnend für die kognitive Verarbeitungskapazität ist auch, daß nur die ersten zwei Blöcke aus einer Folge über eine Unterbrechung hinweg behalten werden, von den übrigen Blöcken bleiben nur Reste. Herbert SIMON verweist in diesem Zusammenhang auf die begrenzte kognitive Verarbeitungsrate: Im Mittel benötigt der Mensch fünf Sekunden, um einen „chunk" vom Kurzzeit- ins Langzeitgedächtnis zu übertragen und dort zu fixieren [Simon, 1982, S. 82].

2.1.1 „Chunking"

Die verblüffend geringe Kanalkapazität muß aber nicht unbedingt unsere kognitiven Fähigkeiten auf diese Größenordnung beschränken. Mit Hilfe der *Codierung* — Gruppieren und Organisieren der Informationsmenge — können wir semantisch beliebig große Blöcke erfassen und so den Informationsengpaß aufbrechen. Das unterstützt entscheidend den kognitiven Prozeß, zum Beispiel die Bewältigung einer VLSI-Entwurfsaufgabe: Durch den Vorgang des Modellierens wird der Informationsgehalt eines komplexen Realitätsbereichs derart codiert, das heißt auf wenige kognitiv gleichzeitig erfaß- und bearbeitbare Informationseinheiten und -strukturen abgebildet, daß die physio-psychologische Schwelle des Menschen (Kanalkapazität, Kurzzeitgedächtnis) scheinbar überschritten wird. Das *Chunking* muß hierbei MILLERs magischer Quantität genügen. In der VLSI-Praxis bedeutet dies beispielsweise, daß die Zahl der Abstraktionsebenen, das Grundsortiment an Funktions- und Strukturmodulen (Zellbibliotheken), die Alternativen in Bedienungsmenüs oder — allgemein gesprochen — die Menge der Klassenbildungen auf die MILLERsche Größenordnung beschränkt bleiben. Nur so kann ein Überschreiten der Bewußtseinskapazität des Modellbenutzers vermieden (der Übertrag ginge, informationstheoretisch gesehen,

in *Rauschen* über), seine maximale Aufnahmefähigkeit und damit die größtmögliche Modelleffizienz erzielt werden.

Die strukturierte Programmierung, um ein Softwarebeispiel zu nennen, überwindet gleichfalls die kognitive Begrenzung durch Blockbildung: Die Beschränkung auf wenige Ablaufstrukturen, wie Sequenz, Auswahl und Schleife, soll den Weg aus der Softwarekrise weisen. SHNEIDERMAN mutmaßt, daß der Chunking-Prozess für das Verständnis eines Programms unabdingbar sei [Shneiderman, 1980]. Der Leser eines Programms abstrahiert die Information des Quelltextes in einzelne Blöcke, die wiederum in eine mentale semantische Struktur eingebaut werden, die den Programmtext modelliert. Komplexe Programme lassen sich nicht schrittweise auf der Ebene der Anweisungen verstehen, es sei denn, daß eine Anweisung einen logischen Block darstellt. Die MILLERsche Zahl steht also auch motivierend hinter den *Struktogrammen* von NASSI und SHNEIDERMAN. Bild 2.2 zeigt ein Beispiel, das [Sommerville, 1990] entnommen wurde. Der BubbleSort-Algorithmus kann unabhängig von der gewählten Programmiersprache (der syntaktischen Notation) in nur wenigen semantischen Blöcken aufgenommen werden. Ist eine mentale semantische Struktur des Programms aufgebaut, kann sie dauerhaft ins Langzeitgedächtnis übernommen werden.

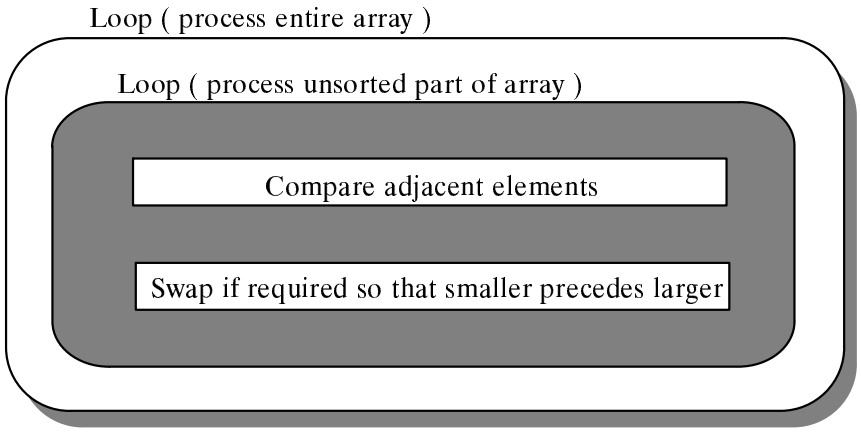


Bild 2.2: Chunking am Beispiel des BubbleSort-Algorithmus

Fazit zum Modellbegriff: Psychologisch stellt ein „Modell“ die Codierung eines komplexen Realitätsbereichs dar, um die kognitiven Möglichkeiten des Menschen besser zu unterstützen.

2.2 Die „Architektur des Komplexen“

Es ist hier nicht der Raum, um auf das breite Erkenntnisspektrum einer allgemeinen Systemtheorie einzugehen, obgleich systemtheoretische Begriffe und Methoden eine wichtige Rolle beim Entwerfen spielen. In diesem Abschnitt soll die wesentliche Gemeinsamkeit „erfolgreicher“ komplexer Systeme aus Natur, Gesellschaft und Technik — das *hierarchische Strukturierungsprinzip* nach Herbert SIMON — hervorgehoben und darauf aufbauend ein allgemeines Systemmodell der Entwurfsaufgabe vorgestellt werden.

2.2.1 „Near Decomposability“

Zunächst sei motivierend die klassische *Hora-Tempus*-Parabel von SIMON aus seinem Aufsatz *The Architecture of Complexity* [Simon, 1962] nacherzählt:

> Zwei renommierte Uhrmacher, Hora und Tempus, setzen ihre in der Komplexität von etwa 1000 Teilen vergleichbaren Uhrwerke auf unterschiedliche Weise zusammen. Während Tempus so vorgeht, daß er bei einer unvorhergesehenen Unterbrechung durch einen Kunden das bis dahin zusammengesetzte Werk gänzlich ablegen muß und dieses dadurch wieder in seine Bestandteile zerfällt, entwirft Hora seine Uhr in „ablegbare“ Einheiten. Jeweils zehn Teile bilden ein stabiles Gefüge, zehn solcher Gefüge wiederum das nächstgrößere und zehn dieser schließlich das Gesamtwerk. Wenn also Hora bei seiner Arbeit unterbrochen wird, so betrifft die destruktive Wirkung des Ablegens immer nur die letzte Teilkonstruktion. Auf diese Weise vollendet er sein Werk bei gleicher Unterbrechungsrate in einem Bruchteil der Arbeitszeit, die Tempus dafür benötigt. Beispiel: Eine Unterbrechungshäufigkeit von 1 zu 100 bewirkt im Mittel eine ca. 4000fach längere Herstellungszeit für Tempus-Uhren.

SIMON illustriert die empirische Anwendbarkeit seiner Parabel auf eine Vielzahl biologischer, soziologischer und physikalischer Systeme. Er unterstreicht die Häufigkeit, mit der Komplexität in der Natur hierarchisch strukturiert auftritt. Komplexe Systeme entwickeln sich evolutionär am schnellsten aus weniger komplexen Systemen, wenn diese jeweils *stabile Zwischenformen* repräsentieren: „complexity evolves from simplicity“. In hierarchischen Strukturen können Interaktionen zwischen (inter) und innerhalb (intra) von Teilsystemen unterschieden werden. Intra-Verbindungen sind dabei in der Regel stärker ausgeprägt als Inter-Komponentenverbindungen. Dies hat zur Folge, daß die hochfrequente Dynamik einer Hierarchie nur die innere Struktur der Komponenten betrifft, während sich die niederfrequente Dynamik auf die Interaktionen zwischen den Komponenten beschränkt. Die Unterscheidung zwischen hoch- und niederfrequenten bzw. starken und schwachen Interaktionen gilt für alle Ebenen einer Hierarchie. Die schwächsten Interaktionen ereignen sich zwischen Teilsystemen auf der höchsten Ebene und die stärksten innerhalb jener auf der untersten Ebene.

SIMON bezeichnet diese Grundeigenschaft als „near decomposability“. Sie erleichtert die Beschreibung und das Verständnis komplexer Systeme erheblich. Ein Schulbeispiel für Inter- und Intra-Bindungskräfte bietet die hierarchische Struktur der Materie. Die Komponenten der Materie, die SIMONs *stabilen Zwischenzuständen* entsprechen, sind zum einen hierarchisch organisiert: Atome binden Elementarteilchen, Moleküle binden Atome, und derAggregatzustand der Materie (fest, flüssig oder gasförmig) ist Ausdruck der Molekülbindung. Zum anderen sind die Bindungsenergien der Komponenten einer Ebene um Größenordnungen geringer als die Bindungsenergien der Komponenten der unteren Ebenen. Dies trifft besonders auf den Übergang von chemischen auf nukleare Strukturen zu. In der Tabelle 2.1 sind die energetischen Maßstäbe zwischen Inter- und Intra-Bindungen der Materie einander gegenübergestellt.

Komponenten der Materie	*Typische Bindungsenergien*	
molekulare Aggregate	thermische Energie	$kT \approx 0,04eV$
Moleküle	molekulare Energie	$0,25 - 5eV$
Atome	Ionisierungsenergie	$3 - 25eV$
Atomkerne	Kernenergie	$2 - 20MeV$
Elementarteilchen	?	$\gg 10^{11}eV$

Tabelle 2.1: Die Beinahe-Zerlegbarkeit am Beispiel

Die Unterscheidung zwischen schwachen und starken Interaktionen im Konzept der *beinahe zerlegbaren* Systeme sollte nicht die Illusion wecken, man könne komplexe Systeme in vollständig isolierte — disjunkte — Komponenten mit individuell unabhängigem Verhalten zerlegen. In diesem Fall verlöre das System seine Komplexität und reduzierte sich auf das Nebeneinanderstellen seiner Komponenten. Dies aber widerspräche dem Grundkonsens jeder systemtheoretischen Wissenschaft: „Das Ganze ist mehr als die Summe seiner Teile“, oder wie P.-J. COURTOIS feststellt:

> „It is somewhat paradoxical that it is precisely in these weak interactions that the raison d'être of a complex structure resides.“ [Courtois, 1985, S. 596]

Die Kriterien für die Zerlegbarkeit eines Systems in seine Teilsysteme werden wir im nächsten Abschnitt im Kontext des Software- und VLSI-Entwurfs erörtern. Wir wollen an dieser Stelle nur die Parallele zum Vorhergesagten ziehen: Im Sinne von MILLERs *chunking* (Seite 36) und ROSENs *vested interest* (Seite 34) trägt die hierarchische Zerlegbarkeit entscheidend zur kognitiven Bewältigung von Komplexität bei: *divide et impera*.

2.2.2 Allgemeines Systemmodell

Aufbauend auf dem Hierarchie-Postulat von SIMON soll abschließend ein Referenzmodell nach Günter ROPOHL beschrieben werden, das auf beliebige technische Objekte anwendbar ist und den Systembegriff für unsere weitere Diskussion klärt [Ropohl, 1980]. Die in der Systemtheorie gewonnenen Aspekte — funktional, strukturell und hierarchisch — verdeutlicht Bild 2.3.

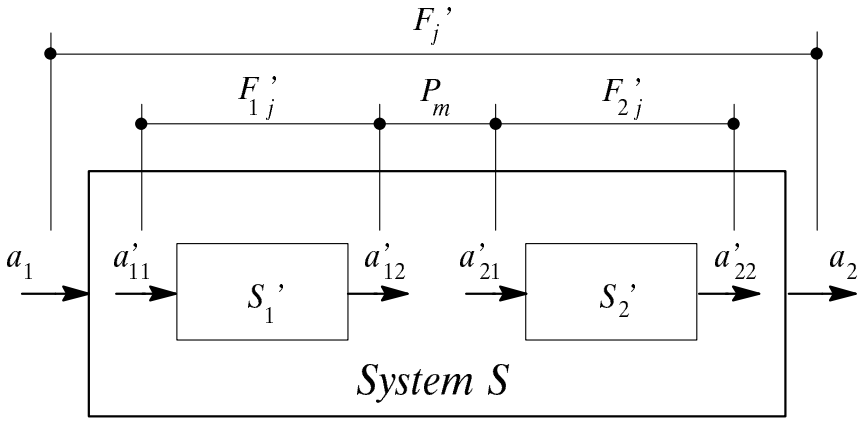


Bild 2.3: Didaktisches Modell für Sachsysteme nach ROPOHL

Der funktionale Systemaspekt geht auf den Primitivansatz der *Black box* als größtmögliche Abstraktion ihrer inneren Struktur zurück. Die Funktion F_j' des Systems gegenüber seiner Umgebung steht global für die Umsetzung einer Problembeschreibung (Spezifikation, Lasten- und Pflichtenhefte) in eine Problemlösung (Implementierung, Produkt). Die systematische Umsetzung der Ein- in die Ausgangseigenschaften ist Gegenstand der Informatik schlechthin: siehe zum Beispiel Ian SOMMERVILLEs „Wasserfall-Modell“ [Sommerville, 1990] oder Barry BOEHMs „Spiralmodell“ [Boehm, 1988] des Software-Lebenszyklus.

Der strukturelle Aspekt bezieht sich auf die Bindungen der Teilsysteme S_i', das heißt auf das Verbindungsgeflecht aus Serien-, Parallelschaltungen und Rückkopplungen. In diesem Aspekt liegt der schon zitierte Konsens der Systemtheorie: „Das Ganze ist ...“ Es ist nicht trivial, von den Eigenschaften der Teilsysteme auf die Eigenschaften des Gesamtsystems zu schließen. Die Struktur des Verbindungsgeflechts, gegeben durch die Menge aller Relationen P_m, bestimmt entscheidend das Verhalten des Systems.

Der hierarchische Aspekt schließlich fußt auf der von SIMON aufgezeigten Notwendigkeit, Komplexität geordnet zu zerlegen. Die Komponenten S_i' bilden eigenständige Teilsysteme mit möglicherweise weiterer Schachtelung bis hin zu Grundsystemen, die nicht weiter dif-

ferenziert werden. Es sei vermerkt, daß SIMON mit dem Begriff der Hierarchie nicht unbedingt eine pyramidenförmige Rangfolge verbindet.

Fazit zum Begriff „Entwurfsaufgabe“: Systemtheoretisch verlangt die Komplexität der „Entwurfsaufgabe“ deren Zerlegung in möglichst unabhängige strukturelle Einheiten (Entwurfsabschnitte) mit klar definierten Übergängen: hierarchische Strukturierung.

2.3 Dekomposition: „Divide et Impera“ oder „Separation of Concerns“

Die Technik der Dekomposition — des systematischen Zerlegens eines komplexen Systems in handhabbare Komponenten, der Analyse dieser Komponenten und ihrer Interaktionen — besitzt eine lange Tradition in den Ingenieurwissenschaften: Die modulare Programmierung und der zellbasierte VLSI-Entwurf sind nur zwei aktuelle Beispiele dieser bedeutenden Technik. Wenn auch Einigkeit über die Notwendigkeit zur Dekomposition herrscht, so gehen die Lehrmeinungen über die anzuwendenden Kriterien weit auseinander und werden mitunter zur Weltanschauung: funktionale kontra objektorientierte Dekomposition. In diesem Abschnitt wollen wir vor allem die pragmatischen Aspekte „erfolgversprechender“ Systemzerlegungen herausarbeiten, nämlich Randbedingungen und Ökonomie. Wir folgen hierbei im wesentlichen den Arbeiten von P.-J. COURTOIS *On Time and Space Decomposition of Complex Structures* [Courtois, 1985] und von David PARNAS *On the Criteria to be Used in Decomposing Systems into Modules* [Parnas, 1972a].

2.3.1 Randbedingungen

Das Grundprinzip der Dekomposition ist nicht gleichermaßen, im Sinne von Effizienz und Ökonomie, auf alle Klassen komplexer Systeme anwendbar. In der Regel liefert es nur dann hinreichend genaue Ergebnisse, wenn der zu analysierende Systemausschnitt isoliert — kontextfrei — betrachtet werden kann (Bild 2.4). Die vereinfachenden Annahmen sind zweifach. Zum einen wird vom Einfluß der umgebenden Prozesse auf die Eigenschaften des Teilsystems abstrahiert (äußerer Kontext): Die Umgebung sei konstant oder mit festen Werten parametrisierbar. Zum anderen wird vom Einfluß der Prozesse abgesehen, die den Komponenten innewohnen (innerer Kontext): Ihre Dynamik bleibt für die Analyse der Interaktionen zwischen den Komponenten unberücksichtigt. Es wird ein „Gleichgewicht“ der inneren Prozesse unterstellt, das Teilsystem als monolithischer Block aufgefaßt. Nur das statistische Verhalten der Komponenten geht mit wenigen Variablen ins Kalkül des Systemverhaltens ein.

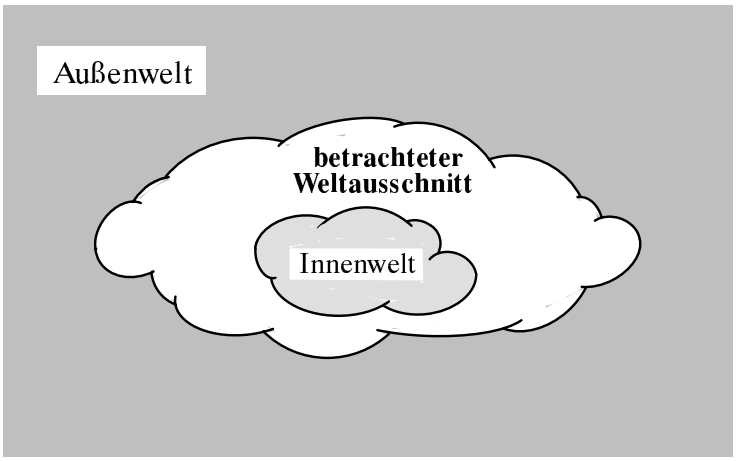


Bild 2.4: Zur „isolierten“ Analyse

Daß diese Vereinfachungen in der Praxis tragfähig sind, zeigt COURTOIS illustrativ am Beispiel des kinematischen Modells einer chemischen Reaktion.[3] Makroskopisch wird von einem „geschlossenen System“ ausgegangen: Die Wirkung des chemischen Prozesses auf die Umwelt wird vernachlässigt, die Parameter der Atmosphäre, wie Druck und Temperatur, werden als konstant angenommen. Mikroskopisch wird von der Struktur und der Dynamik der chemischen Reagenzien abstrahiert: Die Zahl der Moleküle je Volumeneinheit gilt als konstant. Das „Gesetz der großen Zahlen“ erlaubt diese Vereinfachung. Die Fluktuation der Moleküldichte liegt bei $1/\sqrt{2N}$, N steht hierbei für die Zahl der Moleküle. Beispielsweise beträgt N in einem Kubikmillimeter Luft unter Normalbedingungen $\approx 10^{16}$. Die Dichteschwankungen betragen somit $\approx 10^{-8}$.

Die „isolierte Analyse“ — Isolation nach außen: konstante Umwelt, Isolation nach innen: im Gleichgewicht befindliche Innenwelt — liefert nur dann brauchbare Erkenntnisse, wenn die zeitlichen und/oder räumlichen Maßstäbe („time and size scales“) der beteiligten Strukturen viele Größenordnungen auseinanderliegen [Courtois, 1985, S. 591]. Dies trifft ganz offensichtlich für die makroskopische Analyse einer chemischen Reaktion zu (24 Größenordnungen!)[4] oder auch für die hierarchische Struktur der Materie (siehe Tabelle 2.1 auf

[3] Er führt in seiner Übersichtsarbeit [Courtois, 1985] eine Fülle sehr überzeugender Beispiele aus der Physik, den Sozialwissenschaften und der Informatik an, von denen wir hier nur wenige aus unserem Fach aufnehmen können.

[4] COURTOIS weist aber auch darauf hin, daß die extrem geringen Dichteschwankungen bei Gasen für den Menschen dennoch *beobachtbar* sind. Bei kubischen Volumina, deren Kantenlänge der Wellenlänge des sichtbaren Lichts entspricht, beträgt $N \approx 10^6$ und die Fluktuation somit $\approx 10^{-3}$. Dieser Unterschied von neun Größenordnungen zwischen makroskopischem und mikroskopischem Verhalten ist einer der Gründe, warum unser Himmel vorwiegend blau erscheint.

Seite 39). Wie wir aber eingangs zu den WEAVERschen Klassen der Komplexität ausgeführt haben, fallen gerade die technisch interessanten Systeme in die Klasse der *organisierten Komplexität*, die sich in der Regel leider nicht durch astronomische Unterschiede in den zeitlichen und räumlichen Maßstäben ihrer Komponenten auszeichnen. Die für eine Dekomposition „heiklen“ Raum-Zeit-Maßstäbe dieser Komplexität lassen sich nach COURTOIS typisieren:

- *Multiscale Relations/Dissipative Structures*

  Strukturen dieser Eigenschaft können nicht isoliert untersucht werden. Die Komponenten hierarchisch unterschiedlicher Ebenen stehen in einer zu starken Wechselwirkung. Korrelationen können entlang der gesamten Skalierung der Systemeigenschaften — von makroskopisch bis mikroskopisch — die relevanten Phänomene bestimmen und eine Analyse unter den oben skizzierten Isolationsannahmen extrem erschweren. Derartige Probleme rangieren in der theoretischen Physik ganz oben: *dissipative* Strukturen. Hier resultiert das makroskopische Verhalten aus einem Verstärkungs- und Anhäufungseffekt der Interaktionen auf mikroskopischer Ebene. Zwei Beispiele: das paramagnetische Verhalten ferromagnetischer Stoffe oberhalb der CURIE-Temperatur oder die Supraleitfähigkeit keramischer Verbindungen.

  In der Informatik kennen wir das Phänomen des *thrashing*, das bei der Ressourcenverteilung und -auslastung eines Mehrbenutzer-Rechners auftritt: Die Ressourcen an Speicherplatz, Input/Output-Einheiten und CPU-Rechenzeit sollen den konkurrierenden Programmen unabhängig voneinander zugeteilt werden (schwache Interaktionen). Es zeigt sich nun, daß die Programme nur solange unabhängig voneinander ausgeführt werden, wie die Speicherauslastung einen kritischen Wert unterschreitet, wird er überschritten, kommt es zu einem exponentiell steilen Durchsatzeinbruch, *thrashing* genannt. Die Erklärung für dieses Phänomen liegt im Verstärkungseffekt starker Interaktionen, die bei der kritischen Speicherauslastung auftreten. Eine mathematische Analyse des Phänomens wird in [Courtois, 1977, S. 144ff.] erörtert, die intuitive Erklärung lautet: Der den Programmen zugeordnete Speicherplatz schrumpft. Als Konsequenz tendieren diese Programme dazu, ihren Speicherplatz länger besetzt zu halten, was wiederum dazu führt, daß die Zahl der Programme, die Speicherplatz anfordern, zunimmt. Die Schwelle, an der ein Leistungseinbruch einsetzt, ist schwierig zu bestimmen und zu beherrschen.

- *Poor Scale Separability*

  Viele Systeme weisen in ihren meßbaren Eigenschaften vergleichbare Größenordnungen auf. Das erschwert die Entscheidung, auf welcher hierarchischen Ebene die Analyse in größtmöglicher Isolation vom Einfluß anderer Ebenen durchzuführen ist. Hierfür wäre die detaillierte Kenntnis über Struktur und Verhalten der benachbarten Ebenen erforderlich. Ein Beispiel: Die Speicherkomponente eines Rechners ist im allgemeinen hierarchisch organisiert. Je nach Einsatz der Speichertechniken, wie

schnelle flüchtige Halbleiterspeicher oder langsame nichtflüchtige magneto-optische Speicher, reicht das Spektrum der Zugriffszeiten von Nano- bis Millisekunden und ist damit, verglichen mit den optimalen Skalierungsbedingungen einer chemischen Reaktion, bedeutend schmaler. Bei der Leistungsanalyse einer einzelnen Speicherebene (in großen Systemen können bis zu acht Ebenen vorliegen) üben die Datenströme zwischen benachbarten Ebenen einen bedeutenden Einfluß auf die Zuverlässigkeit der Ergebnisse aus [Courtois, 1977, S. 88ff.].

- *Kritische Ereignisse und schwache Interaktionen*

  Sollen Aussagen über das langfristige dynamische Verhalten einer Gesamtstruktur gewonnen werden, so sind es gerade die sonst vernachlässigbaren schwachen Interaktionen und seltenen Ereignisse, die den Ausschlag geben. Interessieren vor allem die geringen langfristigen Schwankungen im Systemverhalten, müssen die schwachen Interaktionen isoliert von den starken analysiert werden, was in der Praxis für die Modellbildung und -simulation mit einem erheblichen Rechenaufwand einhergeht. Wie wir weiter unten noch sehen werden, bestimmen die starken Interaktionen das kurzfristige, die schwachen das langfristige Systemverhalten, so daß insgesamt sehr unterschiedliche Zeit-Maßstäbe zu berücksichtigen sind. Dies gilt beispielsweise im VLSI-Entwurf für das Auffinden von *Hazards* in asynchronen Schaltungen oder — um einmal eine soziologische Parallele zu ziehen — in der Demoskopie, um das Wechselverhaltens von Stammwählern zu analysieren.

2.3.2 Ökonomie

Das Konzept der *near decomposability*, das wir im vorigen Abschnitt als architektonisches Prinzip komplexer Gebilde in Natur und Technik kennengelernt haben, ist gleichermaßen grundlegend für die Analyse der *dynamischen* Eigenschaften komplexer Strukturen: Die unterschiedlichen Kopplungsgrade in einem System bestimmen sein kurz- und langfristiges Verhalten. Herbert SIMON und Albert ANDO haben die zwei wichtigsten Theoreme für *beinahe zerlegbare* Systeme aufgestellt und mathematisch bewiesen [Ando & Simon, 1961] — Lehrsätze, die für die Natur- und Technikwissenschaften größte Tragweite besitzen. Ihre Theoreme in Prosa:

Theorem 2.1 (Simon-Ando) *Für jedes beinahe zerlegbare System läßt sich eine kurzfristige Zeitspanne angeben, in der es sich „annähernd“ wie ein vollständig zerlegbares verhält. Die kurzfristige Dynamik geht auf die starken inneren Bindungen zurück. Jedes Teilsystem strebt ein lokales Gleichgewicht an, das „annähernd“ unabhängig von dem der anderen Teilsysteme ist.*

Theorem 2.2 (Simon-Ando) *Die langfristige Dynamik resultiert aus den schwachen inneren Bindungen. Unter deren Einfluß strebt das Gesamtsystem auf ein globales Gleichgewicht zu, wobei die lokalen Gleichgewichte der Teilsysteme „annähernd“ und relativ zueinander erhalten bleiben.*

Die unscharfe Formulierung „annähernd“ stellt das eigentliche Analyseproblem dar. Der Fehler, der durch eine zu grobe Näherung entstehen kann, das heißt, wenn die obigen Isolationsannahmen nicht zutreffen, ist proportional zum maximalen Grad der Kopplung zwischen den Teilsystemen und umgekehrt proportional zur „Nicht-Zerlegbarkeit“ eines jeden Teilsystems [Courtois, 1985, S. 598]. Ein einfaches Beispiel für die Anwendbarkeit der Theoreme geben SIMON und ANDO:

> Ein Bürogebäude mit angenommener idealer Wärmeisolierung seiner Außenmauern läßt sich unterteilen in mehrere Abteilungen, deren Wände gute, aber nicht ideale Wärmeeigenschaften besitzen, und Großraumbüros mit schlecht isolierenden Trennwänden. Wir betrachten nun den Bürokomplex zu einem Zeitpunkt, zu dem große Unterschiede in der Temperaturverteilung zwischen den Abteilungen und deren Räumen herrschen. Das Gebäude befinde sich quasi in einem Zustand des thermischen Ungleichgewichts. Nach wenigen Stunden fänden wir in den einzelnen Büroräumen nur noch sehr geringe Temperaturunterschiede vor, zwischen den einzelnen Abteilungen wären diese aber noch gravierend. Nach mehreren Tagen wiese das gesamte Gebäude eine annähernd gleiche Temperatur auf.

Das zweite SIMON-ANDO-Theorem ist unserer Intuition nicht so leicht zugänglich. Während der längeren Phase, in der das Bürogebäude sein globales Gleichgewicht anstrebt, bleiben die Raumtemperaturen innerhalb der einzelnen Abteilungen — relativ zueinander — ungefähr im Gleichgewicht. Die Temperatur einer Abteilung als Ganzes kann aber währenddessen deutlich schwanken, bis sie schließlich den globalen Wert des Gebäudes annimmt. Mit anderen Worten: Die lokalen Gleichgewichtszustände der Teilsysteme bleiben langfristig erhalten, zwar nicht in ihren absoluten Werten, aber relativ zueinander.

Die beiden Theoreme haben für die Analyse schwach gekoppelter Teilsysteme größtes Gewicht. Das kurzfristige relative Verhalten und die relativen Gleichgewichtswerte können für die Teilsysteme individuell — in Isolation — ermittelt werden, als wäre das System vollständig zerlegbar. Da andererseits die Gleichgewichte der Teilsysteme langfristig annähernd erhalten bleiben, können wir auch das langfristige Verhalten der Gesamtstruktur modellieren. Dazu fassen wir das dynamische System als Aggregation weniger Variablen auf. Sie repräsentieren die Gleichgewichte der Teilsysteme (im Beispiel die Durchschnittstemperaturen der Abteilungen). Die Methode der *aggregierten Variablen* [Ando & Simon, 1961], die letztlich auf den SIMON-ANDO-Theoremen beruht, findet ihre Anwendung unter anderem in den Warteschlangen-Modellen für Client-Server-Konfigurationen. Sie wird für unsere Diskussion der objektorientierten Dekomposition noch von grundsätzlicher Bedeutung sein. Es verwundert nicht nur COURTOIS, daß die Universal-Theoreme von SIMON und ANDO in der einschlägigen Literatur selten erwähnt werden:

> „In retrospect, it is somewhat surprising that these theorems remain unknown to many authors who keep proposing empirical decomposition schemes in various application areas, more than 20 years after their first publication.“ [Courtois, 1985, S. 597]

Abschließend zur Diskussion der Kriterien für erfolgversprechende Systemzerlegungen wollen wir auf den vielzitierten Prüfstein von David PARNAS eingehen.

2.3.3 „Information hiding“

Als einer der ersten stellt David PARNAS den Begriff des *Moduls* als konzeptionelle logische Einheit im Software-Entwurf heraus [Parnas, 1972a; Parnas, 1972b]. Das konventionelle Verständnis eines Moduls als die syntaktische Zusammenfassung von mehreren Prozeduren, die unabhängig von anderen übersetzbar sind, erweist sich als ineffizient im Entwurf komplexer Software. PARNAS schlägt ein Konzept des *Information hiding* vor: Schwierige Entwurfsentscheidungen oder solche, die voraussichtlich im Laufe des Software-Lebenszyklus wesentlichen Veränderungen unterliegen, sollten in Modulen gekapselt und vor der Außenwelt — den Modulbenutzern — *verborgen* werden. Da Entwurfsentscheidungen meistens langfristiger Natur sind und die eigentliche Programmausführungszeit in aller Regel überdauern, verbietet sich die traditionelle algorithmus- oder prozeßorientierte Dekomposition von selbst (zum Beispiel als Flußdiagramm). Für eine in diesem Sinne modulare Dekomposition sprechen ökonomische Gründe:

- Effizientes Projektmanagement

  Große Softwareprojekte lassen sich nur noch arbeitsteilig im Team wirtschaftlich durchführen. Die Delegation der Entwurfsentscheidungen und der damit verbundenen Verantwortlichkeit auf getrennte Module, die von einzelnen oder einer Gruppe von Entwerfern entwickelt werden, verringert den Bedarf an Kommunikation: Wenn zuvor die Schnittstellen, das sind die Pflichtenhefte der Module, vom Projektmanagement eindeutig festgelegt wurden, entfällt ein Großteil der üblichen Rückfragen, Mißverständnisse und neuer Absprachen. Die Modularisierung der Entscheidungen verringert somit die Inter-Gruppenaktivität zugunsten einer verstärkten Intra-Gruppenarbeit, was wiederum zu kürzeren Entwicklungszeiten führt.

- Flexibilität

  Modularisierung im Sinne PARNAS' schließt das Prinzip der Lokalisierung mit ein. Module sind zwar untereinander „verbunden“, die Verbindungen sind aber von gänzlich anderer Art als im herkömmlichen Verständnis von Soft- und Hardware: weder prozedural (Zugriff auf gemeinsame Variable und Prozeduren) noch syntaktisch (physikalische Kopplung durch Leiterbahnen).

> „The connections between modules are the assumptions which the modules make about each other.“ [Parnas, 1972b, S. 339]

Als Schnittstelle bezeichnen wir also die Menge aller Annahmen — das Wissen — über ein Modul. Eine Modulschnittstelle abstrahiert von der tatsächlichen Modulimplementierung. Diese ist prinzipiell beliebig (*1-zu-n*-Abbildung). Das aber impliziert eine lokale Wartbarkeit: Solange die Modulimplementierung den Annahmen über sie genügt, haben modulinterne Veränderungen durch Wartung und Aktualisierung keine Nebenwirkungen auf andere Module. Ein modulares System erleichtert somit das Lokalisieren und Beheben von Fehlern. Die Korrektheit eines Moduls ist, ohne es zuvor in das System zu integrieren, in einem eigenen Testrahmen überprüfbar.

- Übersichtlichkeit

  Durch das Abstraktionskonzept des *Information hiding* wird MILLERs *Information chunking* unterstützt und der kognitiven Kapazität des Systementwerfers bzw. -benutzers Rechnung getragen. Für das Verständnis des Systemverhaltens und der Systemstruktur sind im allgemeinen nur wenige Modulspezifikationen pro Hierarchiestufe erforderlich. Struktur und Verhalten der Module können unabhängig voneinander studiert werden.

Des weiteren unterstützt eine modulare Entwurfsmethode die projektbegleitende Systemdokumentation, da man sich hier auf eine einheitliche Beschreibung der Schnittstellen festlegen kann, ohne die jeweils modulinternen, meist dedizierten Beschreibungsmittel in die laufende Gesamtdokumentation mit aufzunehmen. Ein weiterer, wirtschaftlich der bedeutendste Aspekt betrifft die Wiederverwendbarkeit. Sie ist im großen Stil (*large scale reuse*) nur durch eine standardisierte Modularisierung möglich. Soft- und Hardware-Ressourcen lassen sich als „konfektionierte“ Modul- und Zellbibliotheken beliebig oft wiederverwenden. Die Wirtschaftlichkeit modularer Komponenten ist ein entscheidender Vorzug des Objekt-Paradigmas. Wir werden hierauf noch ausführlich eingehen. Alle genannten Vorteile belegen die These von PARNAS, daß die Informationsverteilung im Entwurfsprozeß aus wirtschaftlichen Gründen restriktiv gehandhabt werden sollte: „information ‚broadcasting‘ is harmful“ [Parnas, 1972b, S. 339][5].

2.3.4 Evolution: Vom Handentwurf zum „Silicon-Compiler“

Nachdem wir nun Architektur, Dynamik und Dekomposition komplexer Artefakte näher untersucht haben, soll zum Abschluß das globale Wachstumsverhalten der Entwurfskomplexität entlang der technischen Zeitachse skizziert werden. Auch in der Evolution des

[5] Vergleiche mit Edsger DIJKSTRAs berühmtem Kommentar, der die Wende zur „Strukturierten Programmierung“ einleitete: *Go To Statement Considered Harmful* [Dijkstra, 1968].

Entwurfsprozesses beweist SIMONs Hierarchie-Postulat — Komplexes entwickelt sich aus Einfachem — seine Allgemeingültigkeit. Triebfeder jeglichen Entwerfens und Konstruierens ist der Drang des Menschen, seine Umwelt zu beherrschen oder seine Vorstellungen und Pläne in ihr zu realisieren. Als Werkzeugmacher und -benutzer zeichnet er sich vor allen anderen Lebewesen aus. Psychologisch gesehen resultieren Entwurf und Einsatz technischer Mittel zur Beeinflussung seiner Welt aus dem Streben nach Organverlängerung, -unterstützung und -ersatz.

Gleichviel, ob es sich um die Konstruktion von Soft- oder Hardware handelt, lassen sich mehrere Entwicklungsabschnitte[6] unterscheiden: Bild 2.5.

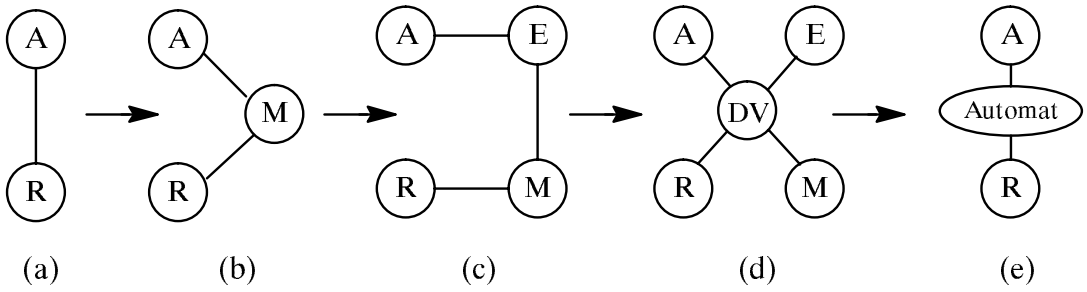


Bild 2.5: Zur technischen Evolution des Entwurfsprozesses

(a) Die Situation kennzeichnet ein Zeitintervall, in dem der Realitätsbereich R, den es zu manipulieren gilt, noch einfach und unmittelbar zu beeinflussen ist. Ein bezüglich des Anwenders A externer Modellierungsvorgang ist nicht erforderlich. Man spricht in diesem Zusammenhang auch vom „unbewußten" Modellierungsprozeß: Entwurf und Herstellung sind eins. Die Bildhauerei ohne Vorlage ist hierfür ein Beispiel.

(b) Mit zunehmender Komplexität der Entwurfsaufgabe stellt sich die Notwendigkeit, die zu verarbeitende Informationsmenge abschnittsweise geeignet zu modellieren. Der Modellierungsprozeß findet seine erste konkrete Ausprägung in einem externen Modell M, das die Komplexität des Entwurfsvorgangs durch Einführen von Abstraktionsebenen im MILLERschen Sinne *codiert*. Beispiele: NASSI-SHNEIDERMAN-Diagramme und Schaltpläne.

(c) Mit der Entwicklung leistungsfähiger Methoden und Techniken wachsen zugleich die Forderungen und Bedürfnisse und umgekehrt — nimmt die zu bewältigende Informationsmenge qualitativ und quantitativ zu. Das Modellierungsproblem überfordert den gelegentlichen Anwender. Es muß folglich an einen Experten oder an eine

[6] J. H. AMKREUTZ zum Beispiel skizziert in [Amkreutz, 1976] die technische Evolution für die Architektur.

Gruppe von Experten E delegiert werden, zum Beispiel Architekten, Programmierer und Chipentwerfer. Mit Einführung einer weiteren Instanz in den Entwurfsprozeß mit dem Ziel der Arbeitsteilung entsteht das Problem der Kommunikation. Die Distanz zwischen Anwender und Zielobjekt vergrößert sich. Nicht immer decken sich Wunsch und Wirklichkeit, Problembeschreibung und entworfene Lösung.

(d) Eine weitere Komplexitätszunahme in der algorithmischen Verarbeitung und in der Verwaltung des Datenaufkommens überfordert schließlich selbst den Experten. Die elektronische Datenverarbeitung (DV) — metaphorisch der *Rechner* — hat zwar die Effizienz der Datenverarbeitung revolutioniert, bleibt aber in der evolutionären Entwicklungslinie des gesamten Entwurfsprozesses. Die Situation spiegelt die heutige Konstellation der beteiligten Instanzen und deren Kommunikationspfade wider. Als Beispiel stehe hier das CAx-Szenario des VLSI-Entwurfs: Die Mensch-Maschine-Interaktion rückt in den Vordergrund. Schnittstellenprobleme treten auf [Quibeldey-Cirkel, 1993a].

(e) Eine derweilen noch futuristische Situation könnte schließlich die Erfahrungen und das Wissen des Experten sowie das zum Modell abstrahierte Entwurfsobjekt integrieren — in Form von wissensbasierten Automaten. In der VLSI-Technik wäre dies der ideale „Silicon-Compiler“, wie ihn Daniel GAJSKI definiert [Gajski, 1988, S. 39f.].[7] In der Softwaretechnik liefe dies auf die „Automatische Programmierung“ hinaus, deren Zukunftsperspektiven in [Balzer, 1985] aufgezeigt werden.

Wir können festhalten: Der Entwurfsprozeß verläuft evolutionär und ist im SIMONschen Sinne streng hierarchisch strukturiert. Die Modellbildung nimmt in allen Entwicklungsabschnitten eine Schlüsselstellung ein. Zur detaillierten Geschichte des Modellbegriffs und des Denkens in Modellen sei auf die Übersichtsarbeit von Roland MÜLLER verwiesen [Müller, 1983].

Fazit zum Begriff „Entwurfsprozeß“: Die evolutionär gewachsene Komplexität des Entwurfsprozesses als geordnete Abfolge von Entwurfsabschnitten mit isolierten Entwurfsentscheidungen erfordert die koordinierte Beteiligung mehrerer Instanzen: Anwender, Experte und CAx-Instrumentarium. Aus der Sichtweise des Benutzers zielt der technische Fortschritt auf eine Verkürzung der operativen Distanz zwischen Spezifikation und Implementierung.

[7] Als Utopie bezeichnet der Begriff die automatische Gewinnung aller Fertigungsdaten für eine integrierte Schaltung (Layoutgeometrien, Testmuster etc.). Als Eingabe soll eine anwendungsneutrale Hardware-Beschreibungssprache (VHDL) genügen, um eine „in Silizium zu gießende“ Problemlösung zu spezifizieren. Als Technikbegriff ist er wesentlich enger gefaßt: Nur wenige anwendungsspezifische Silicon-Compiler existieren derzeit (zum Beispiel für Digitalfilter). Der entscheidende Vorteil eines Silicon-Compilers ist sein Prinzip: *correctness by construction*. Der Compiler-Algorithmus garantiert stets die korrekte Implementierung.

3

Komplexitätsbewältigung II: Integration & Integrität

Die Frage der Integrität eines komplexen Systems ist vor allem eine Frage der Verträglichkeit der Entwurfskomponenten: Modelle, Methoden, Werkzeuge und Datenhaltung. Nur im integren Verbund aller können die BROOKSschen Postulate an den Entwurf erfüllt werden: *productivity*, *reliability*, *simplicity*. Während nun das Thema „Integration der Entwurfswerkzeuge" zum Gemeinplatz unter CAD-Anbietern und -Anwendern geworden ist, wird das Problem „Integrität der Daten" nur zögernd angefaßt. Der Tribut, der für die Wahrung eines korrekten Datenbestands zu zahlen ist — Einbußen an Flexibilität und Effizienz der Datenhaltung —, erscheint derzeit zu hoch. Mit wachsender Komplexität der Datenbankanwendungen aber werden sich Aufwand und Nutzen relativieren. Nur eine mit den Werkzeugen integrierbare Datenhaltung reduziert das *Akzidens* der Entwurfskomplexität: das redundante und inkonsistente Mehrfachspeichern in heterogenen Umgebungen. Integration und Integrität sind zu Schlüsselfragen der Komplexitätsbewältigung im Entwurfsprozeß avanciert.

Übersicht: Abschnitt 3.1 steckt den Bezugsrahmen der Arbeit schärfer ab: CAD-Frameworks.[1] Die Schnittstellen in der Infrastruktur offener integrierter Entwurfssysteme werden identifiziert und die aktuelle Projektlandschaft zur Standardisierung der CAD-Schnittstellen skizziert. Abschnitt 3.2 stellt eine mengentheoretische Definition des Integritätsbegriffs vor, zumal dieser in der technischen Literatur nicht eindeutig besetzt ist. Bislang fällt der Begriff in den Wirrwarr von mehr oder weniger exakten Datenbankbegriffen, wie Transaktion, Recovery und Datenkonsistenz, und interpretationsbedürftigen philosophischen Begriffen, wie logische Widerspruchsfreiheit und Plausibilität. Er besitzt überdies auch eine gesellschaftspolitische Komponente, den personenbezogenen Datenschutz. Abschnitt 3.3 schließlich faßt wichtige Integritätsaspekte technischer Entwurfsdatenbanken in einem Referenzmodell zusammen.

[1] Der Abschnitt erschien in gekürzter Form in [Quibeldey-Cirkel, 1993b].

3.1 Bezugsrahmen: CAD-Infrastrukturen

Das Chipzeitalter der Höchstintegration hat ein kaum zu überschauendes CAx-Szenario hervorgebracht, treffend umschrieben durch die Metapher *Islands of Automation* [Encarnação & Lockemann, 1990]. CA-Entwurfsmethoden für kundenspezifische Schaltungen und mannigfaltige CA-Techniken der Entwurfssynthese und -verifikation bilden ein komplexes Instrumentarium. Mit Einführung der *simulierten Netzliste* als formaler Schnittstelle zwischen Chipentwerfer und ASIC-Hersteller sind Entwurf und Test in die alleinige Verantwortung des Entwicklungsingenieurs übergegangen. Einerseits liegt die erweiterte Kompetenz in seinem Sinne: als Vorzüge der vorgezogenen Netzlistenschnittstelle seien hier nur die verkürzte *Turn-around*-Zeit und die höhere Entwurfssicherheit genannt. Andererseits werden hohe Forderungen an sein methodisches *Systemdenken* gestellt: „Welche Entwurfswerkzeuge von welchem Hersteller setze ich wann und in welchem Kontext ein? Wie sind die Schnittstellen definiert? Wie sind die Ein- und Ausgabedaten zu interpretieren, auf welchen Modellbildungen beruhen sie?“ An diese „W-Fragen“ knüpft sich weiter die Forderung nach der konsistenten Verwaltung der Daten von Entwurf und Test.

Was fehlt ist eine pragmatische, in einer standardisierten Umgebung eingebettete Entwurfsmethode, die seit der *Introduction to VLSI Systems* von Carver MEAD und Lynn CONWAY vor mehr als einem Jahrzehnt [Conway & Mead, 1980] erst in jüngster Zeit in neuen Ansätzen verfolgt wird. So hat sich zum Beispiel das Firmenkonsortium CFI (CAD Framework Initiative) zum Ziel gesetzt, bislang isolierte Werkzeuge herstellerübergreifend mit Hilfe eines prozeduralen Schnittstellenmodells zu integrieren [Goldfein *et al.*, 1990]; so steht die Modellierung in der EDIF-Welt unter dem Anspruch, den Informationsaustausch zwischen EDA-Werkzeugen und Entwurfsstätten zu standardisieren [EDIF, 1990]; und so zeichnet sich ein neuer Trend zum *design consulting* mittels wissensbasierter Systeme zur Modellierung von Fachkompetenz ab [Miller, 1989].

Integrationssysteme, neudeutsch *Frameworks* oder gehoben *Infrastrukturen*, haben nur dann einen wirtschaftlichen Sinn, wenn die Dienstleistungen und Schnittstellen Standards genügen, die von der Industrie aktiv unterstützt werden. Zwar haben die Schnittstellenprobleme einen neuen Markt für Softwarehäuser geschaffen, die maßgeschneiderte Einzellösungen offerieren, für den EDA-Markt insgesamt aber zeichnet sich ein Sättigungstrend ab: Anwender investieren zurückhaltender in neue Produkte, die mit eigenen *In-house*-Systemen oder den Werkzeugen von Drittherstellern unverträglich sind oder aber hohe Anpassungskosten mit sich bringen. Die Unterstützung darf sich also nicht nur auf entsprechende Konverter für existierende Produkte beschränken, sondern muß bereits bei der Entwicklung neuer Werkzeuge auf der Grundlage allgemein akzeptierter Industriestandards beginnen.

Die internationalen Anstrengungen für genormte Entwurfsumgebungen sollen also das heterogene Werkzeugspektrum der Dritthersteller kostengünstig erschließen. Das Schlüssel-

wort heißt: *Kompatibilität*. Die Entwicklung des EDA-Marktes ist geprägt von der engen Kopplung zwischen dem Integrationsgrad der Entwurfswerkzeuge und dem Grad der Standardisierung ihrer Schnittstellen. Der Generationswechsel in der Werkzeugkonfiguration, Tabelle 3.1, belegt diesen Trend.

Trend	*1. Generation* *70er Jahre*	*2. Generation* *80er Jahre*	*3. Generation* *90er Jahre*
Standardisierung	gering: firmeneigene Datenformate, erste singuläre, nur für bestimmte Werkzeugarten definierte Standards (CIF)	wachsend: herstellerneutrale Datenformate (EDIF), Schnittstelle zwischen ASIC-Entwickler und -Hersteller: simulierte Netzliste	dominierend: Datenzugriff und -austausch weitestgehend standardisiert (EDIF, VHDL, CFI)
Werkzeug-Integration	isolierte, inkompatible Werkzeuge (Insellösungen, *In-house*-Systeme)	geschlossene, herstellerspezifische *Turn-key*-Systemlösungen	*Frameworks*: offene Infrastrukturen für heterogene kooperierende Werkzeuge
treibende Kräfte	Werkzeugentwickler vorwiegend aus universitären Forschungseinrichtungen	CAD/CAE-Anwender, ASIC-Techniken	EDA-Industrie, Normungsgremien, Vielfalt an Hardware-Plattformen und Betriebssystemen

Tabelle 3.1: Generationswechsel der Hardware-Entwurfsumgebungen

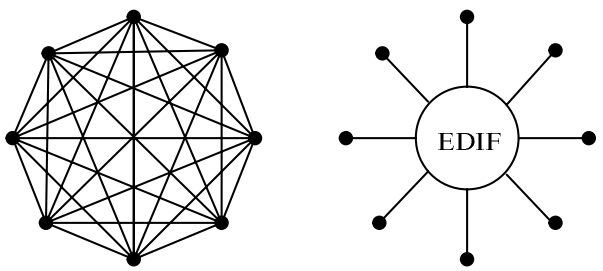


Bild 3.1: Reduktion der Schnittstellenkomplexität von $O(n^2)$ auf $O(n)$

Bild 3.1 verdeutlicht das Schnittstellenproblem in heterogenen CAD/CAE-Entwurfsumgebungen am Beispiel des elektronischen Datenaustauschformats EDIF [Quibeldey-Cirkel & Wahl, 1990a]. Die Daten- und Steuerflüsse zwischen den einzelnen Werkzeugen bzw.

Entwicklungsstätten (dargestellt als n Knoten) sind im allgemeinen bidirektional. Sie wirken in der Regel wechselseitig auf die Datenhaltungen aller am Entwurfsprozeß beteiligten Komponenten ein. Die gegenwärtig noch dominierenden Insellösungen lassen folglich den Aufwand für die Kommunikation (Datenkonverter und Werkzeuganbindungen) quadratisch mit der Vielfalt der Werkzeuge anwachsen: Situation (a). Allein standardisierte Schnittstellen können den Aufwand für kompatible Umgebungen drastisch reduzieren: Situation (b).

3.1.1 Schnittstellen

Mit ihrer massiven Unterstützung für normierte CAD-Infrastrukturen verfolgen die EDA-Anbieter und -Anwender eine von der Phono/Video-Industrie inspirierte *Plug-in-and-play*-Philosophie, die sich gemäß ihrem Kompatibilitätsanspruch zum Ziel gesetzt hat:

- Austausch und Wiederverwendung gemeinsamer Entwurfsdaten
- Einheitliche Benutzerführung und Methodenkontrolle
- Rechnerunterstütztes Prozeß- und Ressourcen-Management

Das Bild 3.2 auf der nächsten Seite zeigt das Schalenmodell einer „idealen“ VLSI/CAD-Infrastruktur. Die Schalen trennen die Entwurfskomponenten — Benutzerführung, Methoden, Werkzeuge und Daten — und legen die Übergänge fest. Zwei gegenläufige Achsen durchziehen die Schnittstellen: Mit zunehmender Abstrahierung nimmt der Detaillierungsgrad der betrachteten Entwurfsdaten ab, verringert sich also das Datenaufkommen. Umgekehrt geht die Konkretisierung der Entwurfsdaten mit einem größeren Aufwand für die Datenhaltung einher. Analog verhält es sich mit der Entwurfskomplexität: Je weiter „außen“ der Entwickler den Entwurfsprozeß steuert und seine Entscheidungen trifft, desto geringer ist seine kognitive Belastung durch die Details. Die Tabelle 3.2 auf der Seite 55 beschreibt die Schnittstellen im einzelnen.

Schnittstellen sollen „wohldefinierte“ und normative Übergänge schaffen und zugleich für übergreifende Interaktionen durchlässig sein. Keile, in das Modell getrieben, symbolisieren die übergeordneten Dienstleistungen:

- ▷ Integritätskontrolle: I–VI

  Die Überwachung der Datenintegrität erstreckt sich über alle internen Schnittstellen einer CAD-Infrastruktur und erfaßt somit die statischen, dynamischen und methodischen Entwurfsaspekte. Eine ausgewogene Kombination von zentralen und verteilten

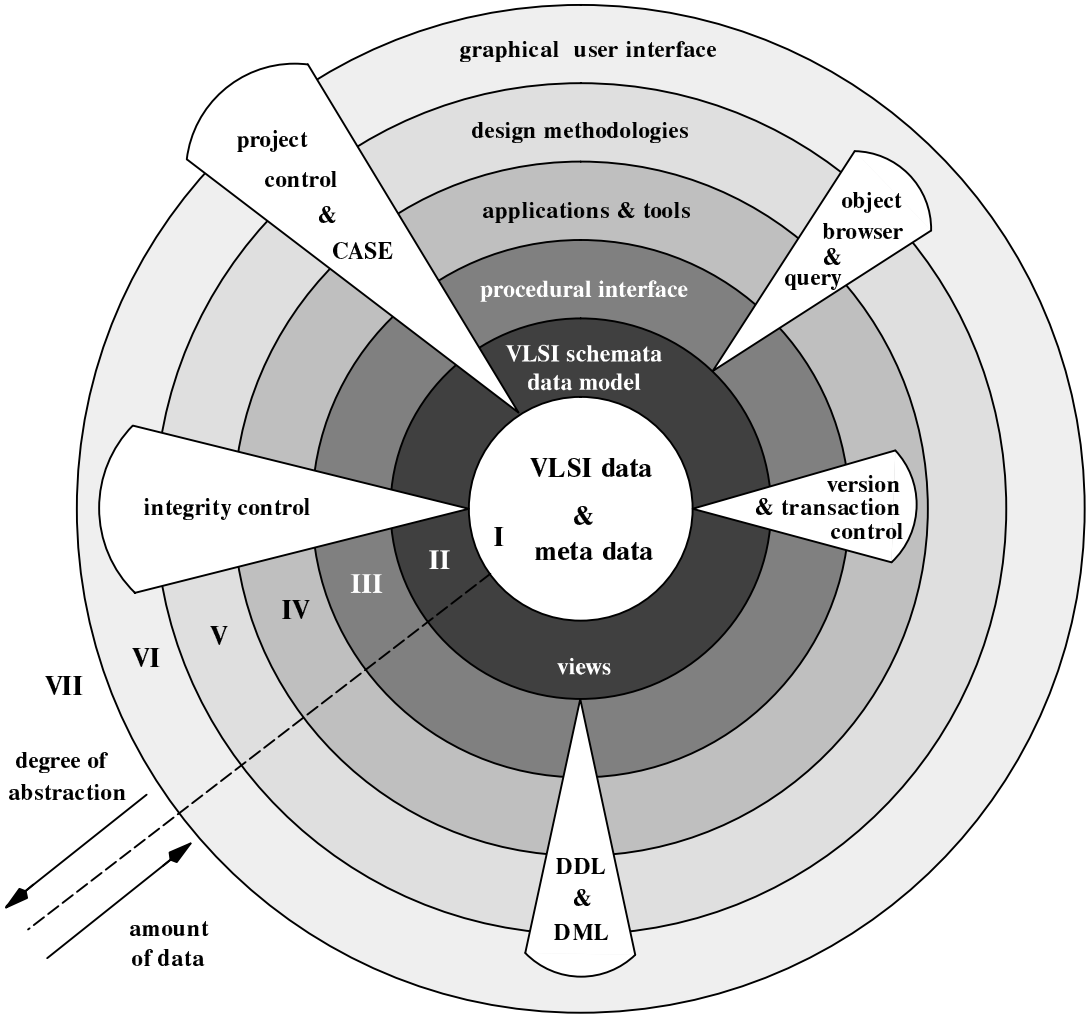


Bild 3.2: Schalenmodell einer VLSI/CAD-Infrastruktur

Kontrollmechanismen sorgt für die Einhaltung der Integritätsbedingungen. Formale Ansätze für einen Integritätsmonitor finden sich zum Beispiel in [Lipeck, 1989].

- Projektmanagement: I–VII

Dem Systementwerfer steht die Möglichkeit offen, jederzeit Status- und Prozeßinformation abzufragen und auszuwerten. Das Werkzeug- und Design-Management vergleicht die aktuellen Konstruktionsdaten mit der Projektspezifikation und schlägt korrigierende Entwurfsentscheidungen vor (Expertensysteme [Miller, 1989]) oder nimmt diese vom Entwerfer entgegen. Zur Verwaltung aller logistischen Daten stehen entsprechende CASE-Werkzeuge zur Verfügung.

Übergang	*Beschreibung*
I–II	CAD-Rohdaten und Meta-Daten (Daten über Rohdaten) werden gemäß dem Datenbankschema organisiert und zentral verwaltet. Direkte Zugriffsmöglichkeiten bestehen nicht.
II–III	Datenbankschemata und -teilschemata (*views*), erstellt auf der Grundlage eines objektorientierten Datenmodells, strukturieren die Objekt- und Beziehungstypen des abzubildenden Weltausschnitts und bilden in ihrer Gesamtheit das Weltmodell der unterstützten Anwendungen.
III–IV	Die prozedurale Werkzeugschnittstelle fungiert als Bindeglied zwischen der statischen Datenhaltung und den dynamischen Einwirkungen der Entwurfswerkzeuge auf den Datenbestand.
IV–V	Das Werkzeugmanagement verwaltet die Entwurfswerkzeuge gemäß der gewählten Entwurfsmethode: Auswahl der Werkzeuge, Generieren von Steuerdateien und Aufrufsequenzen usw.
V–VI	Das Design-Management überwacht den Entwurfsfluß, das heißt die zeitliche Abfolge synthetisierender und verifizierender Prozesse, und koordiniert den Input/Output-Datenfluß an den Schnittstellen der einzelnen Entwurfsabschnitte. Entwurfsentscheidungen können mit Hilfe von Expertensystemen unterstützt werden.
VI–VII	Die grafische Benutzungsoberfläche schließlich überbrückt die interaktive Schnittstelle Mensch-Maschine, die nach software-ergonomischen Gesichtspunkten gestaltet ist.

Tabelle 3.2: Schnittstellen in CAD-Infrastrukturen

▷ Versions- und Transaktionsmanagement: I–IV

Die für den VLSI-Entwurf charakteristischen Forderungen an ein Versions- und Transaktionskonzept [Katz, 1990; Kim *et al.*, 1983] — Protokollierung der zeitlichen und alternativen Entwurfsschritte und Teilergebnisse, Synchronisierung konkurrierender Datenzugriffe — resultieren aus der arbeitsteiligen Durchführung eines Projekts im Team. Transaktionen sind für den Benutzer transparent: die Kontrolle liegt beim Datenbanksystem. Auswahl und Zusammenstellung der für den Entwurf relevanten Versionen — die Konfiguration der Einzelentwürfe — koordiniert das Projektmanagement.

▷ Datenbeschreibung und -manipulation: II–VI

Für die Definition der Schemata und der Zugriffs- und Verwaltungsoperationen verfügt der Datenbankadministrator über die bekannten Sprachmittel konventioneller Datenbanken: DDL und DML (*Data Definition/Manipulation Language*). In

CAD-Infrastrukturen aber orientieren sich diese Sprachmittel nicht an einfachen Datenstrukturen, wie Datensätze und Tabellen, sondern an den semantisch mächtigeren Organisationsprinzipien des Objekt-Paradigmas: Klassen, Objekten, Beziehungen und Integritätsbedingungen (Constraints).

- Objekt/Schema-Browser und Datenabfrage: II–VI

  Zur Visualisierung des aktuellen Datenbestands (Objektebene) und der den Daten zugrundeliegenden Organisationsstrukturen (Schema-Ebene) werden von der grafischen Schnittstelle aus Objekt- und Schema-Browser eingesetzt. Weiterhin stehen die aus der Datenbanktechnik bekannten prädikatenlogischen Abfragetechniken zur Verfügung, um beispielsweise datensensitive Entwurfsentscheidungen abzusetzen oder Objektmengen nach gemeinsamen Eigenschaften zu qualifizieren.

Derzeit gibt es eine Vielfalt an VLSI-Frameworks. Sie weichen in einigen wichtigen Aspekten von der Idealvorstellung des Schalenmodells ab und stellen oft nur einen herstellerspezifischen Integrationsrahmen für die eigene Produktlinie dar. Um aber innovative Entwurfswerkzeuge in einen bestehenden Werkzeugverbund aufnehmen zu können (*technology push*) oder mit einem neuen Produkt schnell am Markt zu sein (*market pull*), bedarf es herstellerneutraler Schnittstellen.

3.1.2 Projektlandschaft

Daß Frameworks nicht nur ein marketing-wirksames Schlagwort[2] sind, sondern auch ein zentraler Gegenstand der Praktischen und Technischen Informatik, zeigt die aktuelle Forschungslandschaft. Schätzungen des amerikanischen Instituts für Verteidigungsanalysen (IDA) zufolge, verkürzen einheitliche Entwurfsumgebungen die Produkt-Entwicklungszeiten um 40 bis 60 Prozent. Das Bostoner Marktforschungsinstitut Technology Research Group (TRG) taxiert den Produktivitätsgewinn auf 30 Prozent [Bhat & Taku, 1990, S. 90]. Die aktuelle internationale CAD-Projektlandschaft wurde in erster Linie durch das Wissen um den ökonomischen Faktor „Framework“ motiviert.

Die wichtigsten Framework-Projekte und -Standardisierungsbemühungen lassen sich zwanglos den Schnittstellen im Schalenmodell zuordnen:

[2] Steven SCHULZ in *Electronic Design* [Schulz, 1991]: „One of the most fascinating aspects of the CAE industry is how quickly a new buzzword finds its way into vendor's glossy literature. The pamphlets unabashedly claim years of market leadership long before the rest of us have settled on a workable definition for the term.“

- *BMFT/DASSY*: I–IV

  Die Zielsetzung des Verbundprojekts DASSY (Datentransfer und Schnittstellen in offenen integrierten VLSI-Entwurfssystemen) umfaßt die einheitliche Modellierung aller Entwurfsdaten, die dem Zugriff der Entwurfs- und Testwerkzeuge unterliegen und von einem zentralen Datenhaltungssystem verwaltet werden [Quibeldey-Cirkel, 1993a]. Dies betrifft den externen Datenaustausch zwischen zwei EDA-Systemen verschiedener Anbieter oder unter EDA-Systemen, ATE-Komponenten und ASIC-Herstellern. Als Datenaustauschformat dient EDIF, in das Erweiterungen (*Test, Device Modeling & Verification*) und Verbesserungen hinsichtlich einer eindeutigen semantischen Hinterlegung (Informationsmodelle) in die entsprechenden EDIF-Gremien eingebracht werden [Blödel (Hrsg.), 1991; Wahl (Hrsg.), 1992; Wilkes (Hrsg.), 1992]. Zum anderen steht die Konzeption und prototypische Evaluierung einer standardisierten prozeduralen Werkzeugschnittstelle im Vordergrund, die unabhängig vom eingesetzten Datenhaltungssystem die Kommunikationsstrukturen (Objekte, Beziehungen, Constraints) und Datenzugriffe der Werkzeuge verwaltet [DASSY, 1990; DASSY, 1992a; DASSY, 1992b].

- *CFI*: II–VII

  Die CAD-Framework-Initiative (CFI) ist ein internationales Industriekonsortium aus EDA-Anbietern und -Anwendern mit dem Ziel:

  > „to develop industry acceptable guidelines for design automation frameworks which will enable the coexistence and cooperation of a variety of tools." [Rhyne, 1989]

  Das CFI-Konsortium unterhält eine Organisationsstruktur (Kooperationen zwischen den Entwicklungslabors namhafter EDA-Firmen), um neue Standards für CAD-Schnittstellen zu entwickeln oder bestehende zu bewerten. Anschließend sollen die CFI-Richtlinien als Industrienormen vorgeschlagen werden.

- *DoD/EIS*: II–VII

  Das EIS-Programm (Engineering Information Systems) adressiert alle Schnittstellen einer CAD-Infrastruktur [Naclerio, 1989]. Auch im Rahmen dieses groß angelegten Unterfangens sollen neue Industriestandards entwickelt und/oder existierende bewertet werden. Um die Brauchbarkeit und Effizienz der vorgeschlagenen Standards und Richtlinien zu demonstrieren, wurden auch hier Industriekooperationen vereinbart: In den Entwicklungszentren wichtiger CAE-Firmen soll parallel und unabhängig voneinander in unterschiedlichen CAD-Entwurfsumgebungen auf der Grundlage der EIS-Standards ein VHSIC-System entwickelt werden. Am Ende der öffentlichen Bewertungsphase werden dann die Ergebnisse in die Normungsgremien der Industrie eingebracht.

- *IEEE/DASS/WGDM*: II–VII

  Die Arbeitsgruppe Design Management (WGDM) des IEEE-Gremiums für Design Automation Standards (DASS) wurde gegründet, um Empfehlungen für den Einsatz

von Standards in CAD-Infrastrukturen auszusprechen. Als Grundlage für die IEEE-Richtlinien sollen die Vorschläge und Erkenntnisse aus CFI und EIS dienen.

- *ISO/STEP*: II–III

  Die ISO hat sich das ehrgeizige Ziel gesetzt, existierende nationale und internationale Industriestandards für mechanische, elektrische und elektronische Produktdaten durch einen einzigen Standard zu ersetzen: STEP — Standard for the Exchange of Product Model Data [McLaren, 1992]. Das bedeutet: STEP wird eine Vielzahl von Ingenieurdisziplinen umfassen, vom Entwurf integrierter Schaltungen bis zur Konstruktion von Ozeanriesen. Nicht nur die Fertigungsdaten, sondern auch sämtliche, während des Produkt-Lebenszyklus (Entwicklung, Projektierung, Wartung) anfallenden, logistischen Daten sollen durch STEP konzeptionell abgedeckt werden und maschinell austauschbar sein.

  Auch wenn der Bogen (zu) weit gespannt sein sollte, hat bereits ein Nebenprodukt der STEP-Aktivität in vielen Normungsgremien Einzug gehalten: Express, eine formale Beschreibungssprache — sowohl textuell als auch grafisch — zur Informationsmodellierung [Schenk, 1990]. Mit Hilfe dieses universellen Hilfsmittels läßt sich der statische Informationsgehalt beliebiger Diskursbereiche in Form von Referenzmodellen einheitlich darstellen und auf Konsistenzaspekte automatisch überprüfen. Express wird in den Arbeitsgremien von STEP, CFI, IEEE/VHDL, EDIF, EIS und im Forschungsprojekt DASSY eingesetzt. Wir werden später noch darauf zurückkommen.

- *EIA/EDIF*: III–IV

  Die EDIF-Gremien der Electronic Industries Association (EIA) entwickeln und pflegen einen gemeinsamen Standard für den Datenaustausch zwischen EDA-Anwendungen [Quibeldey-Cirkel & Wahl, 1990a]. Die derzeitige Version [EDIF-Manual, 1990] stellt den De-facto-Industriestandard für die Entwurfsbereiche Netzlisten und Schaltplaneingabe. Gegenwärtige Entwicklungen konzentrieren sich auf die Forderungen aus dem Bereich der gedruckten Leiterplatten (PCB) und der Datenkopplung zwischen Entwurf und Test. Darüber hinaus nehmen die Arbeiten zu einem grundlegenden EDIF-Informationsmodell einen breiten Raum ein [Quibeldey-Cirkel & Wahl, 1990b].

- *ESPRIT/ECIP*: III–IV

  Das European CAD Integration Project (ECIP) ist Teil der ESPRIT-Initiative [Chalmers & Meys, 1989]. Ziel ist die Entwicklung eines formalen konzeptionellen Referenzmodells für alle Arten elektronischer Produkte. Klassifikationsschemata und zugehörige Verzeichnisse der verwendeten Begriffe sollen anhand von effizienten *Benchmarks* auf Vollständigkeit und Widerspruchsfreiheit maschinell überprüfbar sein.

- *ESPRIT/EVEREST*: III–IV

  EVEREST (European Vanguard Effort on Research and Engineering of Systems for Testing) ist ebenfalls Teil der ESPRIT-Initiative [Schneider, 1991]. EVEREST zielt

auf den Test als eine Sicht des Entwurfsraums. In diesem Projekt wird die Spezifikation eines Informationsmodells für Testdaten betrieben. Dieses soll als Grundlage für eine EDIF-Erweiterung dienen [Wahl (Hrsg.), 1992].

Auf dem Weg zu industrieweit anerkannten CAD-Standards leisten die oben skizzierten Framework-Projekte und Normungsgremien die entscheidende Arbeit. Sie bieten zugleich die Foren für EDA-Produktanbieter und -anwender, um die Entwicklung und Etablierung der Framework-Technik aktiv voranzutreiben.

Der Framework-Technik wird eine maßgebliche Rolle im Wachstum des EDA-Marktes im allgemeinen und der ASIC-Industrie im besonderen zugesprochen [Bhat & Taku, 1990]. Um den Produkt-Entwicklungszyklus weiter zu straffen, werden Soft- und Hardware am Ende der 90er Jahre vorwiegend in standardisierten CAD-Infrastrukturen entworfen. Kürzere Markteinführungszeiten sind nicht mehr allein durch leistungsgesteigerte Werkzeuge möglich, sondern nur noch im standardisierten Verbund aller Entwurfskomponenten: Benutzerführung, Methoden, Werkzeuge und Datenmodelle.

3.2 Integrität von VLSI-Datenhaltungen

Mit der Einführung integrierter CAD-Umgebungen mit standardisierten Infrastrukturen hat die Datenhaltungskomponente die dominierende Rolle im Entwicklungszyklus übernommen. Vorbei die Zeiten, als insulare Werkzeuge mit eigenen, teils unüberschaubaren und vor allem inkonsistenten Dateisystemen den BROOKSschen Akzidensteil der Entwurfskomplexität in die Höhe trieben. Das primäre Ziel eines jeden Entwurfs, die Konstruktionsdaten für das Implementierungsmedium bereitzustellen, rückt mit der *Zentralisierung* der Datenhaltung in seine ursprüngliche Perspektive. Der Entwurf kann nunmehr datenzentriert ansetzen. Er reiht sich ein in das relativ gut verstandene Problem des *Datenbankentwurfs*. Mit dem Schwerpunkt auf dem Entwurf einer VLSI-Datenbank gelten die bekannten Grundsätze für Informationssysteme nun auch für unsere Disziplin:

- Schematisierung

  Der Datenbestand wird nach einheitlichen Organisationsprinzipien geordnet. Der innere Aufbau bleibt überschaubar. Die Redundanz durch unkontrolliertes Mehrfachspeichern entfällt.

- Datenunabhängigkeit

  Durch die Auslagerung der Daten und ihrer physikalischen Organisation aus den Anwendungen heraus werden die Wechselwirkungen aufgehoben, die aufgrund von

Aktualisierungen (*Updates* und *Upgrades*) in den Daten- oder Programmstrukturen auftreten und einen unwirtschaftlichen Aufwand nach sich ziehen. Die algorithmische Komponente eines Programms ist von der Datenhaltungskomponente unabhängig. Die organisatorische *Trennung* zwischen Programm und Daten gewährleistet somit ein hohes Maß an Änderungsflexibilität auf beiden Seiten.

- Datenintegrität

  Die Datenintegrität umfaßt die Bereiche: Konsistenzsicherung (Vermeidung widersprüchlicher Datenbankinhalte), Datensicherung (Schutz vor irrtümlichen, absichtlichen oder durch Systemfehler verursachten Verfälschungen) und Datenschutz (Schutz vor unberechtigtem oder mißbräuchlichem Zugriff).

Die beiden ersten Forderungen werden von herkömmlichen Datenbanksystemen (auf der Grundlage hierarchischer, netzartiger und relationaler Datenmodelle[3]) für betriebswirtschaftliche Aufgaben effizient erfüllt [Schlageter & Stucky, 1983]. Die Konsistenzsicherung beschränkt sich hier auf eine einfache Typ-Prüfung der Datenfelder. Für sogenannte „Non-Standard"-Anwendungen [Encarnação & Lockemann, 1990], wie komplexe Soft- und Hardware-Entwurfsumgebungen, sind vor allem die Punkte Schematisierung und Datenintegrität von größtem Interesse und Gegenstand der aktuellen Datenbankforschung. Gerade die Wahrung der Datenintegrität im Sinne von „logischer Korrektheit" der Datenbankinhalte (Übereinstimmung zwischen Realität und Modell) ist für ingenieurmäßige Anwendungen von ausschlaggebender Bedeutung. Bereits ein einzelnes verfälschtes Datum kann die Konstruktion eines komplexen Systems scheitern lassen oder, falls es unbemerkt bleibt, katastrophale Folgen in realen Laufzeitumgebungen verursachen. Trotz seiner Wichtigkeit fehlt es an einer klaren Definition des Integritätsbegriffs. Wir wollen in diesem Abschnitt eine anschauliche mengentheoretische Begriffsbestimmung einführen, die vor allem (aber nicht nur) im Bereich der VLSI-Datenhaltung zur Klärung beitragen kann. Um dies zu leisten, werden wir zunächst das klassische Drei-Schema-Konzept im Datenbankentwurf umreißen. Die Erkenntnis, daß sich der moderne Entwurf komplexer Systeme großenteils als eine datenverarbeitende Aufgabe stellt, legt diese Vorgehensweise nahe.

3.2.1 Drei-Schema-Konzept

Der Datenbankentwurf gliedert sich in mehrere Phasen, in denen die Teilabbildungen zwischen dem interessierenden Weltausschnitt und dem physikalischen Speichermedium fest-

[3] In der Datenbanktechnik bezeichnet ein *Datenmodell* die Menge aller Modellierungskonzepte, die ein Datenbanksystem für die Organisation und Verwaltung der Daten unterstützt. Peter LOCKEMANN sagt es anschaulich: „So wie ein Werkzeugkasten bestimmt, welche physikalischen Modelle gebaut werden können, so legt auch ein Datenmodell prinzipiell fest, wie immaterielle Modelle aussehen müssen" [Lockemann, 1986]

gelegt werden. Die einzelnen Phasen spiegeln dabei unterschiedliche Sichtweisen auf den Datenbestand und seine Organisation wider.

1. Auf der *konzeptionellen* Ebene steht die begriffliche Gesamtstruktur der Objekte im Vordergrund, deren Eigenschaften und Beziehungen untereinander. Um die Abbildungstreue zwischen Weltausschnitt und Modell von den Restriktionen von „unten", das heißt von der Implementierungsseite, zu entlasten, sollte der konzeptionelle Entwurf unabhängig von den Anwendungsprogrammen und vom Datenmodell — den Modellierungskonzepten — des Datenbankverwaltungssystems (DBMS) durchgeführt werden. Für die Strukturbeschreibung bieten sich die Konzepte der *semantischen Datenmodelle* an, die wir in einem späteren Kapitel noch eingehender behandeln werden (zum Beispiel *Entity-Relationship*-Modelle und deren Erweiterungen). Sie erlauben eine abstrakte, vom Diskursbereich unabhängige Beschreibung der Datenbankobjekte [Hull & King, 1987].

2. Im anschließenden *logischen* Entwurf wird das konzeptionelle Modell in ein logisches überführt, das den Modellierungsvorschriften des implementierten Datenmodells genügt. In dieser Meta-Phase des Datenbankentwurfs — zwischen Konzeption und Implementierung — finden zum einen anwendungsspezifische Forderungen Berücksichtigung, wie zum Beispiel die Repräsentation spezieller Sichten oder die Nutzung verteilter Ressourcen (Daten und Operationen) durch Vererbungsstrategien. Zum anderen können durch Ausnutzen der Eigenschaften des Datenmodells Optimierungen vorgenommen werden, beispielsweise *normalisierte Formen* im relationalen Modell [Schlageter & Stucky, 1983, S. 80ff.]. Auch diese Entwurfsebene liegt oberhalb des verwendeten Datenbanksystems, ist somit an keine physikalischen Speicherstrukturen gebunden.

3. Auf der untersten, der *physikalischen* Entwurfsebene wird schließlich das logische Modell in das Zielmedium des Datenbanksystems übersetzt und implementiert. Hier greifen die leistungssteigernden Maßnahmen der Datenverwaltung: Optimierung der Zugriffspfade und Auswahl effizienter Speicherstrukturen.

Der Formalisierung der Sichtweisen dienen *Schemata*, die das Ergebnis einer Modellierung unter einer bestimmten Sicht darstellen. Schemata definieren die in der Datenbank zugelassenen Datenstrukturen. Der gesamte Datenbankentwurf kann folglich als eine iterative Transformation von Datenbankschemata aufgefaßt werden: Bild 3.3.

Die Komponenten eines Datenbankschemas DB beschreibt das Tupel:[4]

[4] Wir übernehmen die Darstellung von Udo LIPECK, die besonders den dynamischen Integritätsaspekt einer Datenbank berücksichtigt [Lipeck, 1989, S. 4ff.]. Auf ihre Notation werden wir des öfteren zurückgreifen.

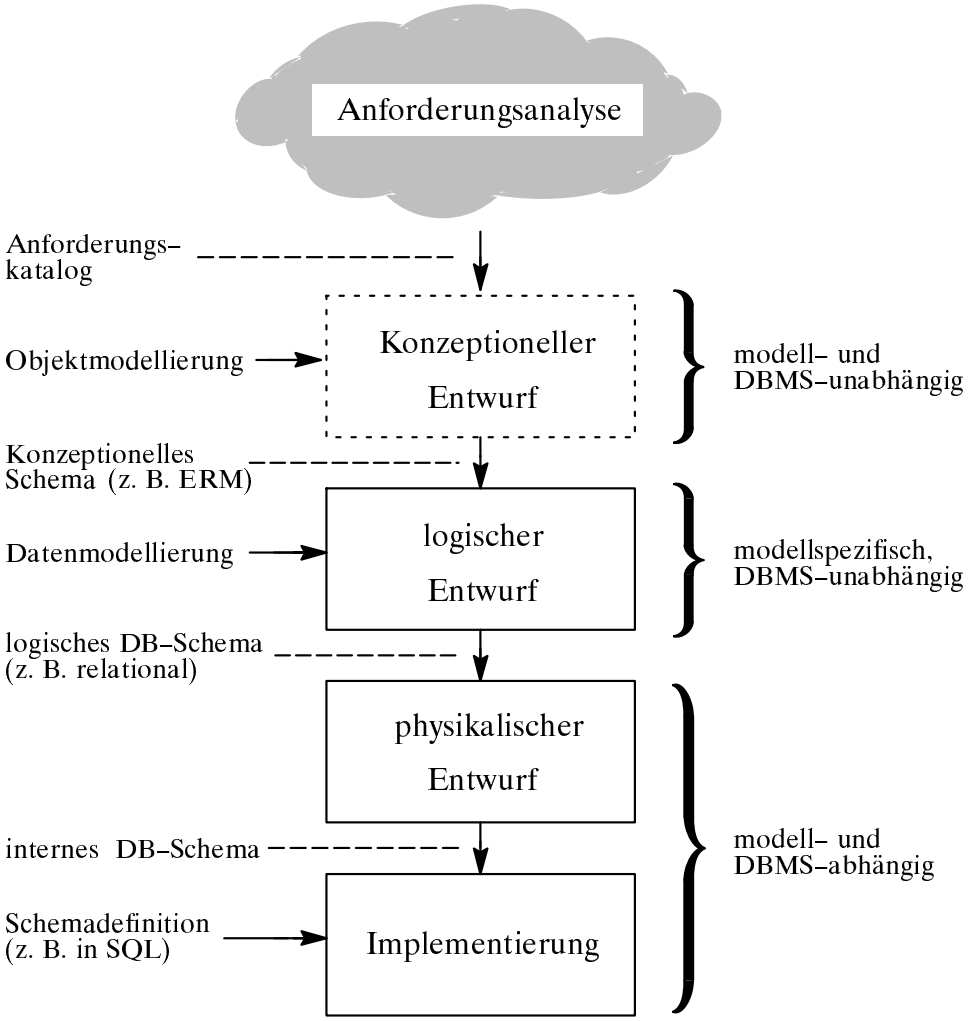


Bild 3.3: Datenbankentwurf als Schematransformation

$$DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$$

- Strukturen $\mathcal{S}$

  Typisierte Objekte und Beziehungstypen bilden die statischen Strukturelemente des modellierten Informationsbereichs (*universe of discourse* [Griethuysen, 1982]). Das gewählte Datenmodell legt dabei die begriffliche Repräsentation der Strukturen fest, zum Beispiel Typ-Hierarchien mit der Möglichkeit, Objekteigenschaften zu „vererben“, oder Typ-Aggregationen mit unterschiedlichen Kopplungsstärken. Die Definition der Strukturen enthält zugleich *modellinhärente* Integritätsbedingungen, die

also keiner ausdrücklichen Formulierung bedürfen, sondern im Datenmodell verankert sind: So impliziert beispielsweise das Löschen eines Aggregats das gleichzeitige Löschen der aggregierten Objekte.

Die Schemastrukturen $\mathcal{S}$ bestimmen die Menge Σ aller möglichen Datenbankzustände[5]. Damit ergeben sich auch die möglichen Zustandsübergänge $R \subseteq \Sigma \times \Sigma$ und Zustandsfolgen Σ^*.

- Operationen $\mathcal{O}$

  Auf den Strukturen sind (elementare) Operationen definiert, mit deren Hilfe Ausprägungen der Strukturelemente (Objekte und Beziehungen) manipuliert werden können: Erzeugen, Ändern, Löschen. Objektorientierte Datenmodelle, die das Prinzip der Datenabstraktion unterstützen, erlauben die Definition benutzereigener Operationen (auch *Methoden* genannt). Jede Operation *op* legt zusammen mit aktuellen Parametern einen elementaren Zustandsübergang fest:

  $$op : \Sigma \times < \text{Parameter} > \rightarrow \Sigma$$

  Die Abbildungsvorschrift berücksichtigt dabei die strukturellen Integritätsbedingungen.

- Constraints $\mathcal{C}$ (explizite Integritätsbedingungen)

  Integritätsbedingungen unterteilen sich stufenweise in statische $\mathcal{C}_{st}$, transitionale $\mathcal{C}_{tr}$ und dynamische Bedingungen $\mathcal{C}_{dyn}$:

  $$\mathcal{C}_{st} \subseteq \mathcal{C}_{tr} \subseteq \mathcal{C}_{dyn}$$

  Statische Constraints sollen die kombinatorisch *möglichen* auf die im Sinne der Anwendung *zulässigen* Zustände $\Sigma_{\mathcal{C}}$ einschränken. Das Attribut „zulässig“ definiert sich über die Einhaltung der Constraints. Transitionale Constraints spezifizieren entsprechend die zulässigen Zustandsübergänge $R_{\mathcal{C}} \subseteq \Sigma \times \Sigma$ und dynamische Constraints die zulässigen Zustandsfolgen $\Sigma^*_{\mathcal{C}}$.

- Transaktionen $\mathcal{T}$

  Jede Transaktion t setzt sich aus einer Folge von Laufzeitoperationen aus $\mathcal{O}$ zusammen, welche die Datenbank von einem zulässigen Zustand in einen anderen zulässigen Zustand überführen:

  $$t : \Sigma_{\mathcal{C}} \times < \text{Parameter} > \rightarrow \Sigma_{\mathcal{C}}$$

  Transaktionen müssen hierbei den GRAYschen *ACID*-Kriterien genügen [Gray, 1981]:

[5] Anschaulicher als Datenbank*zustand* ist die Bezeichnung Datenbank*inhalt*, die wir im folgenden synonym verwenden werden.

- *Atomicity A*

  Eine Transaktion entspricht genau einem zu übertragenden Vorgang aus der Datenbankanwendung. Sie ist logisch „geklammert", das heißt sie wird entweder als Einheit ausgeführt oder gar nicht.

- *Consistency C*

  Die Datenbankzustände, die durch eine Transaktion erreicht werden sollen, müssen zulässig sein, das heißt sowohl den inhärenten als auch den expliziten Integritätsbedingungen genügen.

- *Isolation I*

  Transaktionen laufen völlig unabhängig voneinander ab. Konkurrierende Datenbankzugriffe sind ausgeschlossen: *Ablaufintegrität*.

- *Durability D*

  Die Wirkung einer Transaktion ist dauerhaft (verbindlich), wenn sie ordnungsgemäß abgeschlossen wurde. Der eingestellte Zustand kann nur durch das Ausführen weiterer Transaktionen verändert und eventuell auf diese Weise wieder kompensiert werden.

Transaktionen bestimmen zusammen mit aktuellen Parametern die ausführbaren Zustandsübergänge $R_{\mathcal{T}} \subseteq \Sigma \times \Sigma$ und die ausführbaren Zustandsfolgen $\Sigma^*_{\mathcal{T}}$.

Zusammengefaßt definiert ein Schema $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$ sowohl die statische Struktur als auch das Laufzeitverhalten eines Datenbanksystems. Anwendungen (Programme und Anwender) können nur über die vorgesehenen Transaktionen $\mathcal{T}$ auf den Datenbestand zugreifen. Da das DBMS nur Operationen aus $\mathcal{O}$ für das Ausführen der Transaktionen verwendet, werden ausschließlich Zustände erreicht, die durch die Strukturen $\mathcal{S}$ vorgesehen sind. Die GRAYschen Ausführungskriterien für Transaktionen gewährleisten überdies, daß die erreichbaren Zustände die expliziten Integritätsbedingungen $\mathcal{C}$ erfüllen und damit auch zulässig sind. Die Zustandsfolgen, die eine Datenbank durchlaufen kann, sind somit insgesamt eingeschränkt auf die zulässigen und ausführbaren: $\Sigma^*_{\mathcal{C}} \cap \Sigma^*_{\mathcal{T}}$.

Beispielschema

Ein Beispiel aus der VLSI-Entwurfspraxis soll das Gesagte verdeutlichen: Bild 3.4 zeigt ausschnittsweise ein konzeptionelles Datenbankschema, das für den DASSY-Prototypen entworfenen wurde [DASSY, 1992b]. Es definiert die Semantik einer Netzliste. Beschrieben wird das Schema in der Notation der Modellierungssprache Express [Schenk, 1990].

Die Schemakomponenten unseres Beispiels seien hier zur Verdeutlichung ihrer Semantik stark verkürzt und informell wiedergegeben:

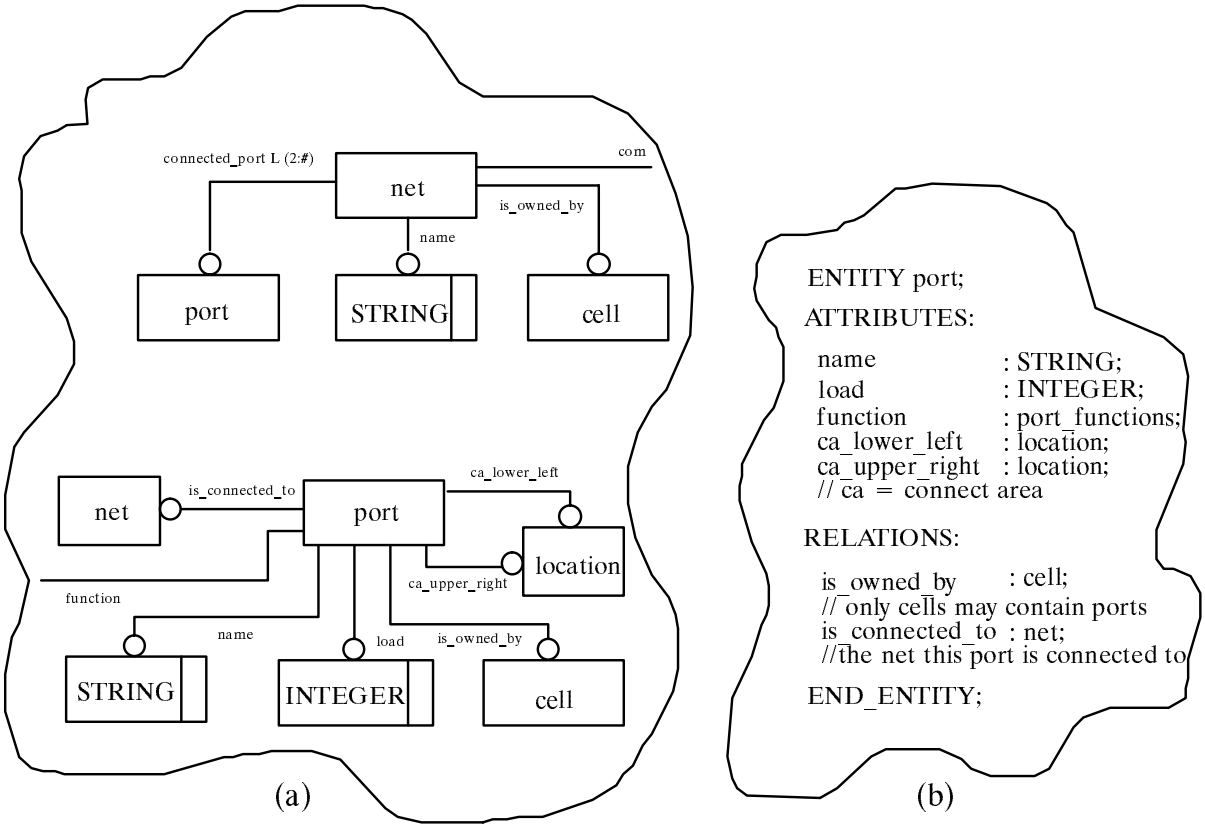


Bild 3.4: Beispielschema in Express: (a) grafisch, (b) textuell

- Strukturen $\mathcal{S}$

  Als wesentliche Strukturelemente treten Verbindungsnetze (*nets*), Zellen (*cells*) und Zellanschlüsse (*ports*) und deren Attribute auf. Die strukturelle Semantik wird durch die Beziehungen der Objekttypen modelliert: Ein Netz gehört zu einer Zelle und manifestiert sich in einer Liste von Zellanschlüssen.

- Operationen $\mathcal{O}$

  Neben den elementaren Manipulations- und Verwaltungsoperationen auf den Strukturen $\mathcal{S}$, wie Erzeugen, Verändern und Löschen von Datenbankobjekten, sind weitere denkbar: beispielsweise Operationen, um die aktuelle kapazitive Last eines Teilnetzes zu berechnen (*Fan-in/Fan-out*).

- Constraints $\mathcal{C}$

  Als inhärente Integritätsbedingung, auferlegt durch das Datenmodell, zeigt sich die Forderung, daß ein Netz aus mindestens zwei Zellanschlüssen besteht, was offene,

sogenannte *dangling nets* vermeidet. Dies ist ein Sonderfall semantischer Datenmodelle: eine explizite Integritätsbedingung wird modellinhärent formuliert (hier durch die Angabe der Kardinalität der Beziehung: *connected_port L[2:#]*).

- Transaktionen $\mathcal{T}$

  Für eine Klasse von Entwurfswerkzeugen können auf der Grundlage der Strukturen $\mathcal{S}$, der Operationen $\mathcal{O}$ und der Integritätsbedingungen $\mathcal{C}$ Transaktionen definiert werden, die den Datenzugriff auf die zulässigen Zustandsübergänge und -folgen einschränken. Dazu ein Beispiel: Der Zugriff eines Grafikeditors auf die Layoutpolygone der Datenbasis kann in der Auflösungstiefe parametrisiert sein (Inter- und/oder Intrazelluläre Verdrahtung).

Abschließend zum besseren Verständnis des Drei-Schema-Konzepts gehen wir kurz auf die grundsätzliche Unterscheidung zwischen Schema-Ebene und Objektebene ein: Die Schema-Ebene wird konzeptionell vollständig durch das Tupel $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$ beschrieben und hat keinen zeitlichen Bezug: Schemata sind *statisch*; sie dienen als formale Meta-Ebene für Objekte. Der Übergang von der Schema-Ebene zur Objektebene dagegen ist *dynamisch*; er geschieht zur Laufzeit der Datenbankanwendung und heißt *Ausprägung* oder Exemplarbildung des Datenbankschemas (weit verbreitet ist auch der Anglizismus „Instanziierung“ [Rechenberg, 1991]). Ein Analogon zu „Schema-Ebene versus Objektebene“ wäre „leeres versus ausgefülltes Formular“. So ist in unserem Beispiel aus Bild 3.4 die Menge aller Ausprägungen der als Objekttyp modellierten *port*-Eigenschaft *location* unendlich groß: potentiell umfaßt sie sämtliche Koordinaten des kartesischen Raumes. Die möglichen Ausprägungen der Eigenschaft *port_function*, um ein zweites Beispiel anzuführen, wäre die aufzählbare Menge $\{input, output, bidirectional, power, ground\}$. Auf diese Grundlagen können wir nun eine sehr intuitive mengentheoretische Definition für den meist unscharf benutzten Begriff der Integrität stellen.

3.2.2 Integrität = Realitätstreue + Vollständigkeit

Der Begriff der Datenbankintegrität weist eine qualitative und eine quantitative Komponente auf, die sich gegenseitig ergänzen: 1. Realitätstreue, die den Ausschluß unzulässiger Daten garantiert, und 2. Vollständigkeit, die den Einschluß aller relevanten Daten gewährleistet. Der umgangssprachliche Gebrauch des Attributs *integer* impliziert gleichfalls zwei duale Elemente: eine Wertung, *unbescholten* oder *unversehrt*, und eine Quantität, *ganzheitlich* oder *umfassend*. Amihai MOTRO hat die komplementären Sichten des Integritätsbegriffs für das relationale Datenmodell formalisiert und in konkrete Datenbankanwendungen eingeführt. Damit steht ein Bewertungsmaß zur Verfügung, das erlaubt, die Antworten, die ein Datenbanksystem auf Anfragen (*Queries*) erteilt, zu qualifizieren.

> „For example, when a prospective traveler requests a database to list all flights from Los Angeles to New York, he is concerned with the *validity* of the information he receives (does each listing indeed represent an actual flight from Los Angeles to New York?), but also with its *completeness* (are there any other flights from Los Angeles to New York that were not listed?).
>
> In other words, answers have integrity if they contain *the whole truth* (completeness) and *nothing but the truth* (validity)." [Motro, 1989, S. 480]

Für bestimmte Anwendungen kann die Qualifizierung einer Datenbankrecherche automatisiert werden, die Antwort bedarf also nicht mehr der Überwachung und Bewertung durch den Menschen. Dies gilt generell für Anwendungen, die auf einer *abgeschlossenen* Datenmenge operieren: *Closed World Assumption*. Unter dieser Annahme enthält eine Datenbank alle Datenobjekte, die sie durch ihr Schema zu modellieren versucht. Das bedeutet: Befindet sich ein Datum nicht in der Datenbank oder läßt es sich nicht aus einem Datenbankzustand ableiten, so ist es falsch. In der Realität trifft die *Closed-world*-Annahme nur auf Teilmengen einer Datenbank zu, nicht auf den gesamten Datenbestand. Für unsere mengentheoretische Diskussion des Integritätsbegriffs sind sowohl der hypothetische als auch der realistische Fall von Bedeutung. Zuvor aber notieren wir die Mengen der Ausprägungen des Datenbankschemas $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$, die für die Begriffsdefinition relevant sind:

$U := \{$ *alle zulässigen Ausprägungen* $\}$: Universum
$W := \{$ *alle aktuell interessierenden Ausprägungen* $\}$: Welt
$D := \{$ *alle aktuell gespeicherten Ausprägungen* $\}$: Datenbank

In Bild 3.5 sind die Mengen U, W und D als VENN-Diagramm für eine nicht weiter spezifizierte Relation dargestellt.

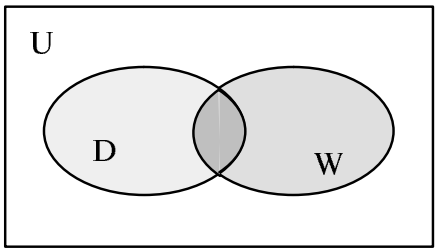


Bild 3.5: Für die Begriffsdefinition relevante Ausprägungen von $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$

Nehmen wir das Express-Schema aus Bild 3.4 und daraus die Ausprägungen des Strukturelements *port_function*, so könnten die hier betrachteten Mengen beispielsweise wie folgt aussehen:

U := { *input, output, bidirectional, power, ground* }
W := { *input, output* }
D_A := { *input* }
D_B := { *input, output, bidirectional* }
D_C := { *input, output* }

Wir betrachten nun Schnittmengen und Identitäten von U, W, und D für den hypothetischen Fall, daß das entworfene Schema $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$ unseren Diskursbereich W perfekt modelliert, und den realistischen Fall, daß nur ausschnittsweise, für selektive Sichten, die Welt W der Anwendung perfekt auf die Datenbank D abgebildet wird.

Hypothetischer Fall: absolute Integrität

Im hypothetischen Fall werden die Objekte (Inhalte) von $D_{A,B,C}$ und W als *Ganzes* miteinander verglichen. Wir erhalten in diesem Fall eine Definition für *absolute* oder auch *totale* Integrität.

(a) $D_A \subset W \hookrightarrow D_A$ ist *realitätstreu*, aber nicht *vollständig* bzgl. W.
(b) $W \subset D_B \hookrightarrow D_B$ ist *vollständig*, aber nicht *realitätstreu* bzgl. W.
(c) $D_C = W \hookrightarrow D_C$ ist *integer*, das heißt *realitätstreu* und *vollständig* bzgl. W.

Bild 3.6 zeigt die zugehörigen VENN-Diagramme.

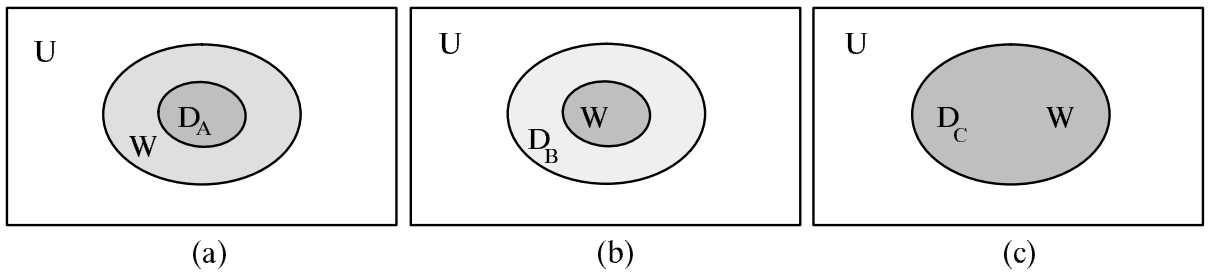


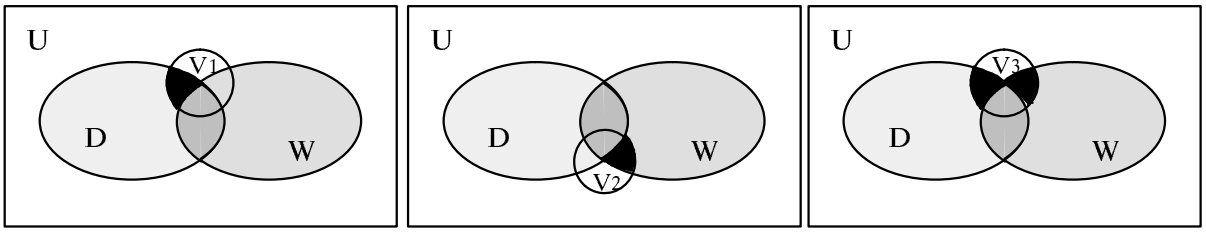
Bild 3.6: Schnittmengen und Identität im hypothetischen Fall

Selektive Sichten: partielle Integrität

Für den realistischen Fall gehen wir von eingeschränkten Sichten V_i (*views*) auf den aktuellen Datenbestand aus, für die eine *partielle* Integrität definiert werden kann. Eine Sicht

V_i kann gleichzeitig Teilmengen von U, W und D umfassen. In Bild 3.7 sind die entsprechenden Schnittmengen für realitätstreue (V_1), vollständige (V_2) und integre Sichten (V_3) dargestellt. Die Aussagen gelten unter der Voraussetzung, daß die schwarzen Flächen jeweils leer sind.

(a) $V_1(D) \subseteq V_1(W) \quad \hookrightarrow$ V_1 ist *realitätstreu* in D bzgl. W.
(b) $V_2(W) \subseteq V_2(D) \quad \hookrightarrow$ V_2 ist *vollständig* in D bzgl. W.
(c) $V_3(D) = V_3(W) \quad \hookrightarrow$ V_3 ist *integer* in D bzgl. W.

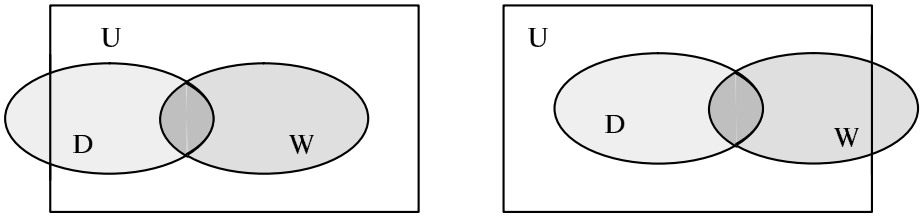


(a) (b) (c)

Bild 3.7: Schnittmengen und Identität im realistischen Fall

Zum Begriff der Inkonsistenz

Mit den oben eingeführten Objektmengen läßt sich nun auch der Begriff der Konsistenz bzw. Inkonsistenz von Datenbeständen veranschaulichen: Als statisch angenommener Bezug gälte die Menge aller zulässigen Objekte U von $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$, das heißt, dynamische Schema-Erweiterungen, sogenannte Schema-Evolutionen, seien ausgeschlossen.



(a) (b)

Bild 3.8: Zum Begriff der Inkonsistenz

Integritätsverletzungen können zum einen aufgrund von unzulässigen, durch Irrtum, Mißbrauch oder Systemfehler verursachten Veränderungen im Datenbestand D auftreten: $D \setminus U \neq \emptyset$, Bild 3.8 (a). Zum anderen können Veränderungen im Diskursbereich W, deren Struktur $\mathcal{S}$ nicht durch das Datenbankschema DB berücksichtigt wurde (unzureichende Modellierung), die Integrität des Datenbestands verletzen: $W \setminus U \neq \emptyset$, Bild 3.8 (b).

3.3 Referenzmodell der Datenintegrität

Die Definition „Integrität = Realitätstreue + Vollständigkeit“ kann nur dem Grundverständnis des Begriffs dienen. Die vielfältigen Integritätsaspekte moderner Non-Standard-Datenbanksysteme machen weitere Differenzierungen erforderlich. Dies soll abschließend in einem allgemeinen Begriffsrahmen — einem Referenzmodell der Integrität — geschehen, das für unsere weitere Diskussion den Umfang möglicher Integritätsverletzungen und Gegenmaßnahmen aufzeigt. Die Darstellung geht auf eine Klassifikation von Dieter STEINBAUER und Hartmut WEDEKIND zurück [Steinbauer & Wedekind, 1985], wurde aber hinsichtlich der pragmatischen Integrität um die Aspekte der Entwurfsmethodik erweitert. Ausgangspunkt der folgenden Ausführungen ist das Drei-Schema-Konzept des Datenbankentwurfs.

3.3.1 Zustand und Veränderung

Die Unterscheidung zwischen dem Zustand einer Datenbank und ihren Zustandsänderungen ist grundlegend für die Klassifikation der Integritätsforderungen. War der Begriff der Integrität ursprünglich überwiegend zustandsbezogen, kam mit der Nutzung der gespeicherten Daten durch konkurrierende Datenbankaktionen auch die Forderung der dynamischen Integrität hinzu, die zum Transaktionskonzept führte [Gray, 1981]. Besonders die Mehrbenutzerbelange der ingenieurmäßigen Non-Standard-Datenbanken verlangen die Korrektheit des Datenbankübergangs: die *Ablaufintegrität*. Wir unterscheiden also prinzipiell zwischen statischer und dynamischer Integrität. Die dynamische Integrität beinhaltet neben dem einzelnen Zustandsübergang, der einer singulären Operation in der Datenbankanwendung entspricht, auch *Folgen* von Zustandsübergängen. Die dynamische Integrität der Datenbankwelt erfaßt somit auch Operationssequenzen der Anwendung, schließt also die Forderungen der übergeordneten Entwurfsmethode mit ein. Ihrer Bedeutung wegen werden wir den methodischen Aspekt der Integritätssicherung aber getrennt betrachten.

Auf einen Punkt sei in unserer Darstellung der Integritätsaspekte besonders hingewiesen: Die Forderung nach einer zentralen Integritätssicherung steht in Konflikt mit zwei weiteren Hauptforderungen an das Datenbanksystem: *Flexibilität* in der Formulierung der Inte-

gritätsbedingungen einerseits und *Effizienz* im Laufzeitverhalten andererseits. Es scheint, als ließen sich in der Praxis immer nur zwei der Größen auf Kosten der dritten maximieren: Zum ersten: Effizienz gepaart mit Flexibilität verlagert die Integritätssicherung in die Anwendung, zum Beispiel *Design/Electrical Rule Checker*, obwohl deren Algorithmen und Parameter auch zentral vom Datenbanksystem verwaltet werden könnten. Zum zweiten: Effizienz zusammen mit Integrität ist derzeit nur für dedizierte, fest ausprogrammierte Standard-Datenbankanwendungen realisiert, die folglich wenig flexibel sind und deren Integritätsbedingungen sich in aller Regel auf die Typ-Prüfung der Objektattribute beschränken, zum Beispiel Kontoführungs- oder Reservierungssysteme. Zum dritten: Flexibilität und Integrität auf Kosten des Laufzeitverhaltens der Datenbank schließlich senkt die Akzeptanz für zeitkritische Anwendungen und ist zur Zeit das Manko vieler innovativer Datenbanksysteme. Das Referenzmodell laut Tabelle 3.3 nennt die Maximalforderungen an die Integrität eines Datenhaltungssystems für Non-Standard-Anwendungen, wohlwissend, daß in der heutigen Praxis die technischen Möglichkeiten beschränkt sind und Kompromisse eingegangen werden.

Aspekt	*Zustand*	*Zustandsübergänge und -folgen*
physikalisch/ technisch	Zuverlässigkeit des Speichermediums (Speicherkonsistenz)	Korrektheit der Implementierung und Befehlsausführung durch die Hardware (Recovery)
strukturell	datenmodellspezifisch, Typisierung der Attributwerte, Einhaltung der Objektidentität, referentielle Integrität	Ablaufintegrität, GRAYsche Transaktionskriterien (ACID)
semantisch	Einschränkung der syntaktisch möglichen Zustände (Schema) auf die „sinnvollen“ Zustände (konkrete Anwendung)	„Integrität des Übergangs“, Einschränkung auf „sinnvolle“ Übergänge (Vor- und Nachbedingungen)
pragmatisch/ methodisch	Normenlogiken vom *Sein-Sollen*-Typ	Normenlogiken vom *Tun-Sollen*-Typ

Tabelle 3.3: Referenzmodell zum Begriff der Integrität in Datenbanksystemen

3.3.2 Statische und dynamische Integrität

Statische Integritätsforderungen adressieren den korrekten *Ruhezustand* einer Datenbank, dynamische den korrekten Datenbank*ablauf*. Die einschlägige Literatur unterscheidet hier lediglich zwischen physikalischen und semantischen Integritätsbedingungen, wobei semantische in der Regel rein struktureller Art sind. Wir wollen in Anlehnung an STEINBAUER und

WEDEKIND feiner differenzieren und zwar in gleicher Weise für statische und dynamische Aspekte: 1. physikalisch-technische, 2. strukturelle, 3. semantische und 4. pragmatisch-methodische Integrität.

1. Die zustandsbezogene *physikalische Datenintegrität* umfaßt die korrekte Abbildung der logischen Datenstrukturen und Zugriffspfade über alle Zwischenstufen hinweg bis hinunter auf die technischen Speicherstrukturen. Sie impliziert die Zuverlässigkeit des Speichermediums. Es muß jederzeit die Gewähr bestehen, daß bei soft- und hardwarebedingten Systemausfällen ein *speicherkonsistenter* Datenbankzustand erhalten bleibt oder rekonstruierbar ist. Der dynamische Aspekt der physikalischen Integrität ist dagegen auf die fehlerfreie bzw. fehlertolerante Implementierung und Ausführung der Datenmanipulations-Operationen gerichtet. Entsprechende Kontrollmechanismen stützen sich auf Sicherungskopien, die es erlauben, die Datenbank im Fehlerfall zu restaurieren (*Recovery*).

2. Die *strukturelle Integrität* wird statisch durch das Datenmodell und dynamisch durch die GRAYschen Transaktionskriterien definiert. Das Datenmodell beinhaltet die konstitutiven Bedingungen für die Objekt- und Beziehungsstrukturen der Anwendungswelt. Die Nichteinhaltung der strukturellen Datenintegrität durch den aktuellen Datenbankzustand läuft auf eine Verletzung des konzeptionellen Schemas und damit der Begriffsstrukturen der Anwendung hinaus: Das realisierte Modell widerspricht der modellierten Realität. Bisweilen beschränken sich die Maßnahmen zur Sicherung korrekter Strukturen auf die Typ-Prüfung der Objektattribute. In relationalen Datenmodellen schließen sie auch *referentielle* Integritätsbedingungen mit ein [Codd, 1979]. Non-Standard-Datenbankanwendungen verlangen hier wesentlich umfangreichere Vorkehrungen, auf die wir in Teil III der Arbeit ausführlich eingehen werden. Insbesondere haben deren dynamische, das heißt operationale Forderungen an die strukturelle Integrität die Vielfalt derzeitiger Transaktionskonzepte hervorgebracht, beispielsweise das Konzept der *langen* und *geschachtelten* Transaktionen im VLSI-Entwurf [Kim *et al.*, 1983]. Der strukturelle Aspekt operationaler Integrität wird in der Literatur auch als *Ablaufintegrität* bezeichnet.

3. Die *semantische Integrität* ist gleichbedeutend mit der Forderung, die Zielsetzung der Anwendung durch den aktuellen Inhalt und durch den Ablauf der Datenbank korrekt zu repräsentieren. Mit anderen Worten: Die semantische Daten- und Ablaufintegrität fordert die Einschränkung der syntaktisch möglichen Zustände und Aktionen auf eine im Sinne der Anwendung geeignete Teilmenge: „sinnvolle“ Zustände und ausführbare Transaktionen. *Was* die Semantik einer Anwendung ausmacht und *wie* sie durch semantische Integritätsbedingungen oder durch Vor- und Nachbedingungen in der Spezifikation der Transaktionen formuliert werden muß, obliegt dem Verständnis des Datenbankentwerfers über die zu modellierende Anwendung und deren Restriktionen. Dieser Integritätsaspekt kann also nicht vollkommen gesichert, geschweige denn automatisiert werden. Er ist aber besonders für Plausibilitätsbetrachtungen relevant.

4. Die *pragmatisch-methodische Integrität* schließlich setzt modale, normenlogische Bedingungen voraus, mit deren Hilfe die „sinnvollen“ Zustände und Transaktionen weiter eingeschränkt werden: *gebotene*, *erlaubte*, *indifferente* und *verbotene* Datenbankinhalte und -aktualisierungen. Die Pragmatik der zustands- und transaktionsorientierten Integritätsbedingungen wird letztlich von der Entwurfsmethode der Anwendung diktiert: Aus den Detailspezifikationen der Entwurfsabschnitte leitet sich die Modalität — *Dürfen* und *Sollen* — der Zustände und Übergänge ab. Diese gilt es durch geeignete invariante Zustandsbedingungen auszudrücken. Das Erkennen nicht zulässiger Ereignisse und die erforderlichen Maßnahmen, diese wieder zu kompensieren, stellen hohe Ansprüche an das Datenbanksystem. Über mögliche Mechanismen, pragmatisch-methodische Integritätsbedingungen zu verwalten (*Event/Trigger*-Mechanismus, *Constraint-Propagierung* und Vererbungsstrategien in Objektstrukturen), werden wir im dritten Teil der Arbeit ausführlich diskutieren.

Fazit

CAD-Frameworks bilden den pragmatischen Rahmen unserer Untersuchung. Die Schnittstellen einer CAD-Infrastruktur müssen *De-facto*-Industriestandards genügen, sollen Entwurfswerkzeuge wirtschaftlich integrierbar sein. Die Integration wirkt zugleich der Diversifikation in den Datenhaltungen der Werkzeuge entgegen. Sie leistet somit einen entscheidenden Beitrag zur Integrität der Entwurfsdaten, die sonst durch interpretative Schnittstellenübergänge und unkontrollierbares Mehrfachspeichern gefährdet wäre. Eingebettet in einem CAD-Framework, kann eine zentrale Integritätsüberwachung mehr leisten als konventionelle Datenbanksysteme: *struktur-inhärente*, benutzer-definierbare *semantische* und systemgestützte *pragmatisch-methodische* Integritätsbedingungen sind formulierbar und überwachbar.

4

Komplexitätsbewältigung III: Konzeptionelles Modellieren

> *„New ideas go through stages of acceptance, both from within and without. From within, the sequence moves from ‚barely seeing' a pattern several times, then noting it but not perceiving its ‚cosmic' significance, then using it operationally in several areas, then comes a ‚grand rotation' in which the pattern becomes the center of a new way of thinking, and finally, it turns into the same kind of inflexible religion that it originally broke away from. From without, as Schopenhauer noted, the new idea is first denounced as the work of the insane, in a few years it is considered obvious and mundane, and finally the original denouncers will claim to have invented it."*
>
> Alan C. KAY [Kay, 1993]

Der Paradigmabegriff, von Thomas KUHN in die Wissenschaftstheorie eingebracht, findet vielerorts seinen Gemeinplatz — so auch in der Informatik. Neue Ideen und Strategien in der Softwaretechnik, wie „Megaprogramming" oder „CAD-Framework", tragen das marketingwirksame Etikett. Zuweilen und zur Rechtfertigung weiterer Programmiersprachen wird der Begriff zum bloßen Sprach-„stil" mißbraucht: „multi-paradigm languages" [Hailpern, 1986]. Dieses Kapitel klärt zunächst die wissenschaftliche Bedeutung und Verwendung des Paradigmabegriffs. Wir skizzieren die Methodenwechsel der Vergangenheit und kennzeichnen die gegenwärtige Hauptströmung: das Objekt-Paradigma. Den Hintergrund unserer Diskussion bildet die Methode des *konzeptionellen Modellierens*. Wir stellen sie als das Gemeinsame der Informatik-Fachgruppen heraus.[1]

Übersicht: Abschnitt 4.1 setzt den Rahmen der folgenden Betrachtungen: Paradigmenwechsel im großen. Im Abschnitt 4.2 klären wir zunächst die ursprüngliche Bedeutung des Paradigma-Begriffs. Wir beschreiben die typischen Reaktionsmuster auf neue Paradigmen und beantworten die Frage: Wie lange dauert es, bis sich ein „neues Denken" in Theorie und Praxis durchsetzt? Abschnitt 4.3 konstatiert die Paradigmenwechsel der Ver-

[1] Die folgenden Ausführungen wurden in der GI-Fachgruppe „Software-Engineering" zur Diskussion gestellt [Quibeldey-Cirkel, 1994a].

gangenheit und Gegenwart in der Softwaretechnik. Abschnitt 4.4 schließlich knüpft wieder an den interdisziplinären Kontext des aktuellen Paradigmenwechsels: Objektorientiertes *Konzeptionelles Modellieren* wird als fächerübergreifende Methode eingeführt.

4.1 Paradigmenwechsel im großen

Das Selbstverständnis der Informatik ist im Umbruch. Bisweilen bestimmten es zwei Elemente: (a) die Ausrichtung an formalen Strukturen nach dem Vorbild der Mathematik und Logik (Datentypen und Algorithmen) und (b) die Implementierung dieser Strukturen nach einer ingenieurmäßigen Pragmatik, die sich im Beiwort *Engineering* ausdrückt, wie Requirements- und Software-Engineering. Der duale Anspruch an Theorie und Praxis spiegelt sich in den heutigen Schuldefinitionen wider. Ein Beispiel aus dem Informatik-Duden:

> „Informatik (*computer science*): Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von ↑ Digitalrechnern (↑ Computer)." [Claus & Schwill, 1993]

Auf den Menschen als Gestalter und Betroffenen informationstechnischer Vorgänge wird, wie schon Rafael CAPURRO bemerkt, nicht verwiesen [Capurro, 1990]. In letzter Zeit aber hat sich das techno-zentrische Weltbild zu einem anthropozentrischen gewandelt: Der Mensch rückt in den Mittelpunkt der systematischen Informationsverarbeitung. Gegenüber der herkömmlichen Auffassung als Struktur- und Ingenieurwissenschaft wird die Informatik als eine *interdisziplinäre* Fachrichtung herausgestellt, „mit der Aufgabe der technischen Gestaltung menschlicher Interaktionen mit der Welt" [Capurro, 1990, S. 311]. Hierzu ein Definitionsvorschlag von Kristen NYGAARD, Mitschöpfer der Sprache Simula:

> „Informatics is the science that has as its domain information processes and related phenomena in artifacts, society and nature." (zitiert in [Capurro, 1990, S. 314])

Die Neuorientierung im Selbstverständnis der Informatik — vom Rohstoff „Information" zum Thema „Mensch" — wird in den Schlagworten deutlich, die derzeit in der Diskussion um die sozialen Aspekte hochgehalten werden: Ethik und Informatik, Informatik und Gesellschaft, verantwortliche Technikgestaltung und so fort. Im Kontext dieser philosophischen Neubesinnung soll im folgenden auf paradigmatische Wechsel im Kernbereich der Informatik, dem Software-Entwurf, näher eingegangen werden. Es zeigt sich, daß gegenwärtig das Objekt-Paradigma die Methodendiskussion in der Softwaretechnik bestimmt [Henderson-Sellers, 1992; Monarchi & Puhr, 1992; Wedekind, 1992; Winblad *et al.*, 1990].

Die techno-philosophische Breite des Objektansatzes unterstützt die These, daß die Suche nach einer interdisziplinären „Wissenschaft des Entwerfens“, wie sie Herbert SIMON, Heinz ZEMANEK und derzeit Christiane FLOYD fordern, ihren Ausgangspunkt im Objekt-Paradigma nimmt [Floyd, 1994; Simon, 1982; Zemanek, 1992]. Objektorientiertes *konzeptionelles Modellieren* ist interdisziplinär [Dillon & Tan, 1993; Fiadeiro & Sernadas, 1991]. Gerade die jungen Informatikzweige weisen synergetische Ziele auf: Wissensrepräsentation in der Künstlichen Intelligenz (KI), semantische Datenmodelle in der Datenbanktechnik und objektorientierte Analyse- und Entwurfsmethoden in der Softwaretechnik haben ihre Gemeinsamkeit in der Modellbildung auf konzeptionellen Ebenen. Sie teilen sich problemnahe Konzepte und Mechanismen, wie hierarchische Kategorien und Vererbung, um das Wissen über einen Weltausschnitt formal zu erfassen und für die informationstechnische Verarbeitung aufzubereiten.

4.2 Kuhns These

Die bahnbrechende Arbeit von Thomas KUHN, *Die Struktur wissenschaftlicher Revolutionen* [Kuhn, 1989], führte zu der Erkenntnis, daß die Wissenschaftsentwicklung nicht geradlinig verläuft, es also keinen kontinuierlich kumulativen Fortschritt gibt. Am Beispiel der Naturwissenschaften verdeutlicht KUHN, wie die stationären Phasen *normaler* Wissenschaft von Perioden *revolutionärer* Wissenschaft abgelöst werden. Zwei Beispiele: die Welt als flache Scheibe versus die Kugel als Weltbild, NEWTONsche Mechanik versus relativistische nach EINSTEIN. Das gültige *Paradigma*[2] — die Konstellation von Meinungen, Werten und Methoden, die von den Mitgliedern einer wissenschaftlichen Gemeinschaft (*scientific community*) geteilt werden — wird in der revolutionären Phase durch ein innovatives Paradigma in Frage gestellt und schließlich überwunden.

Wissenschaft schreitet somit in zwei Modi voran, die sich zyklisch wiederholen. Erstens: Normale Wissenschaft verläuft empirisch und inkrementell. Beobachtungen und Fakten werden in das von der wissenschaftlichen Gemeinschaft aufgestellte Theoriengebäude eingefügt (KUHN nennt die Haupttätigkeit in dieser Phase *Rätsellösen* und *Aufräumen*). Zweitens: Das gültige Paradigma gerät in eine Krise. Wichtige Rätsel können nicht zufriedenstellend gelöst werden. Es werden Ad-hoc-Modifikationen ersonnen, um die „Anomalien“ zu lösen. Einzelne Wissenschaftler finden sich mit der Situation nicht ab und stellen neue Theorien auf. Die Krise des alten und der Drang des neuen Paradigmas führen schließlich zur wissenschaftlichen Revolution. Dem erfolgreichen Paradigmenwechsel schließt sich alsdann wieder eine Phase normaler Wissenschaft an.

[2] In seinem Postskriptum gibt KUHN zwei Definitionsansätze [Kuhn, 1989]: 1. Paradigma als *Musterbeispiel*: die einheitliche Überzeugung über den Forschungsgegenstand und über die Methodik seiner Erforschung und 2. Paradigma als *disziplinäre Matrix*: die Gesamtheit dessen, was eine Gemeinschaft von Wissenschaftlern an kognitiven und sozialen Elementen verbindet.

Die Reaktionsmuster auf einen Theorie-Anwärter sind deutlich sozio-psychologischer Art. Peter MOLZBERGER nennt drei typische Muster, die auf Konferenzen, an Universitäten und in der Industrie als Antwort auf innovative Softwaremethoden überwiegen [Molzberger, 1984]:

1. Emotionale Ablehnung

   Die Mitglieder der wissenschaftlichen Gemeinschaft, die das alte Paradigma vertritt, fürchten um ihre wissenschaftliche Reputation, um die Bedeutung ihres Wissens und Methodenbestands. Sie fühlen zwar, daß etwas Wichtiges im Gange ist, unterdrücken es aber und vermeiden jede bewußte Auseinandersetzung mit den Widersprüchen in ihren alten Überzeugungen. Psychologen wie Leon FESTINGER bezeichnen ein solches Reaktionsmuster mit „kognitiver Dissonanz" [Festinger, 1966].

2. Unreflektierte Euphorie

   Geboren aus der Frustration durch die methodischen Unzulänglichkeiten des vorherrschenden Paradigmas, werden innovative Ideen euphorisch aufgenommen, ohne deren Gültigkeit zu hinterfragen („Prinzip Hoffnung").

3. *Déjà-vu*-Erlebnis

   Die dritte Reaktion auf einen Paradigmenkonflikt ist die häufigste. Nach KUHN hätten die Beobachtungen, die das neue Paradigma unterstützen, auch zu Zeiten des alten gemacht werden können, aber diese waren damals nicht „sichtbar". Mit dem Aufzeigen des neuen Paradigmas kommt es bei vielen zum *Déjà-vu*-Erlebnis: Gegenwärtiges glaubt man, schon einmal in der Vergangenheit — meist unbewußt — erlebt zu haben. Der Neuigkeitswert wird angezweifelt.

Für die Informatik hat Volker CLAUS in empirischen Studien herausgefunden, wie lange es im allgemeinen dauert, bis neue Paradigmen in die Praxis umgesetzt sind [Claus, 1989]. Zwischen dem ersten Auftauchen in der Wissenschaft und der Einführung in die Praxis vergehen in der Regel:

- sieben bis zehn Jahre, wenn dadurch ein Engpaß in der wirtschaftlichen Machbarkeit eines Forschungsvorhabens beseitigt werden kann

  (Beispiel: Edsger DIJKSTRA argumentierte erstmals 1968 in seinem Brief an die ACM öffentlich gegen den Gebrauch von GOTO-Anweisungen: *Go To Statement Considered Harmful* [Dijkstra, 1968]; die GOTO-Kontroverse erreichte ihren Höhepunkt 1972 und wurde 1974 von Donald KNUTH endgültig beigelegt [Koepke, 1985]);

- 15 bis 20 Jahre bei breitbandiger Bedeutung für die Software-Industrie

  (Beispiel: Bereits 1965 wurden die wichtigsten Ideen und Konzepte der „strukturierten Programmierung" in DIJKSTRAs Aufsatz *Programming Considered as a Human*

Activity veröffentlicht [Dijkstra, 1965], zum Stand der Technik wurden sie erst in den 80er Jahren).

Die Praxisnähe der Methoden und die Verfügbarkeit effizienter Werkzeuge sind untrennbare Kriterien für einen erfolgreichen Paradigmenwechsel in der Industrie. Der Akzeptanzverzug in der industriellen Praxis ist im allgemeinen um den Faktor zwei größer als in der universitären Forschung. KUHN selbst geht von einem langwierigen Wandel in den Denkmustern einer wissenschaftlichen Gemeinschaft aus: mindestens 25 Jahre vergehen, bis ein Paradigmenwechsel vollzogen ist. Kraß formuliert es Max PLANCK, rückblickend auf seine wissenschaftliche Laufbahn:

> „Eine neue wissenschaftliche Wahrheit pflegt sich nicht in der Weise durchzusetzen, daß ihre Gegner überzeugt werden und sich als belehrt erklären, sondern vielmehr dadurch, daß die Gegner allmählich aussterben und daß die heranwachsende Generation von vornherein mit der Wahrheit vertraut gemacht ist." (zitiert in [Kuhn, 1989, S. 162])

4.3 Paradigmenwechsel im kleinen

Jede Disziplin hat ihre Lehrmeinungen, Methoden und Werte und durchläuft mehr oder minder abrupte Phasenwechsel im Erkenntnisprozeß. In der Informatik beschleunigt vor allem eine Triebfeder die Wissenschaftsentwicklung: die Suche nach übergeordneten Prinzipien und Methoden für Problemanalyse, Spezifikation, Entwurf und Implementierung. Ziel ist es, fehlerarme, wartbare, erweiterbare und wiederverwendbare Software mit wirtschaftlichem Aufwand herstellen zu können. Die wissenschaftlichen Anstrengungen, um dieses Ziel zu erreichen, streben eine Steigerung des „semantischen Ausdrucks" an. Vier zum Teil parallel verlaufende Leitlinien durchziehen das Streben nach höheren Software-Abstraktionen:

1. prozedurale Abstraktion: Funktionen, Prozeduren, Programmodule
2. syntaktische Abstraktion: Bitmuster, Mnemotechnik, Metaphern
3. Datenabstraktion: Datenstrukturen, abstrakte Datentypen, Klassen
4. Wissensabstraktion: Kategorien, Vererbung, Wissensrahmen

Entlang den chronologischen Abstraktionslinien, den Spuren eines 40-jährigen Lernprozesses, lassen sich die Paradigmenwechsel in der Softwaretechnik aufzeigen.[3] Sie spiegeln das

[3] Die geschichtlichen Daten wurden größtenteils [Koepke, 1985; Weiser Friedman, 1992] entnommen.

unruhige Wachstum der jungen Disziplin wider, ursprünglich bedingt durch den Zulauf von Quereinsteigern: Mathematikern und Physikern. Als Sammelbecken für euphorische Technikerwartungen (Stichwort: CASE) kann die Softwaretechnik nur langsam ihre Eigenständigkeit entwickeln.

4.3.1 Die methodenlose Zeit

Im Sinne von KUHN handelt es sich in der Zeit von 1940 bis etwa 1960 um eine *vorparadigmatische Phase*, in der noch nach einem ersten Paradigma gesucht wird und noch nicht von einer Gemeinschaft von Wissenschaftlern gesprochen werden kann.[4] Das Wort „Informatik" oder „Computer Science" wird erst Ende der 60er Jahre zum Begriff [Capurro, 1990, S. 313]. Forschung ist hier eher eine Einzelanstrengung. Diese später von KUHN als *Proto-Wissenschaft* bezeichnete frühe Phase der Wissenschaftsentwicklung ist geprägt durch das Bedürfnis und die Suche nach brauchbaren Methoden.

In den Anfängen der Rechnertechnik dominierten die geringen Hardware-Ressourcen an Rechenzeit und Speicherplatz die Forschung. Das Programmieren wurde nicht als eigenständige Aufgabe, sondern als pragmatische Nebentätigkeit des Hardware-Entwicklers gesehen. *One-man algorithmic programs* im Umfang von 100 bis 200 Programmzeilen in einer heutigen Hochsprache waren gerade noch machbar — mit verborgenen Tricks und Kniffen im binären Maschinencode (Programmierer hießen damals „Codierer"). So wie in der Automatentheorie nicht zwischen einer Maschine und ihrem Programm unterschieden wird, so bilden in der frühen Geschichte der Rechnertechnik Soft- und Hardware einen integrierten Forschungsgegenstand.

Die erste prozedurale Abstraktion, die *closed subroutine* (aufrufbares Code-Segment im Gegensatz zum Quelltext-Segment einer *open subroutine*), geht auf Konrad ZUSE zurück. Er führte sie 1945 in seinem *Plankalkül* ein, der ersten algorithmischen Sprache [Shaw, 1984; Zuse, 1993].[5] Unabhängig übersetzbare Unterprogramme wurden zuerst in Fortran-II, 1958, und geschachtelte in Algol-60 unterstützt. 1947 wurde erstmals die Verwendung von Flußdiagrammen, um den Steuerfluß zu visualisieren, durch Herman GOLDSTINE und John van NEUMANN vorgeschlagen. Mitte der 50er Jahre setzte die Entwicklung des Makro-Assemblers ein. Der breite Einsatz der prozeduralen Abstraktion in der Programmierung begann also erst mit den 60er Jahren.

[4] Nota bene: Zwar wurden gerade in dieser Zeit die wichtigsten formalen Ansätze der Programmierung entwickelt (das Maschinenmodell von TURING, das λ-Kalkül von CHURCH oder μ-rekursive Funktionen); jeder dieser Ansätze ist im KUHNschen Sinne aber noch kein Paradigma. Überdies sind Paradigmen nach KUHN erst für *reife* Wissenschaften typisch.

[5] Die ersten theoretischen Ansätze gehen allerdings auf die „Analytical Engine" von Charles BABBAGE und seiner Assistentin, Gräfin Ada Augusta von LOVELACE, zurück (ca. 1840).

4.3.2 Programmierkunst versus Software-Engineering

Der exponentiell wachsende Fortschritt in der Hardwaretechnik, der Ende der 60er Jahre einsetzte und bis heute andauert,[6] gab der Softwaretechnik eine neue Gewichtung. Waren es zunächst kleine Entwürfe, die es unter größter Hardware-Ausnutzung zu erstellen galt, so sind nun „gute" Entwürfe gefragt, die komplexe Aufgaben angehen. Da systematische Entwurfsmethoden erst zögernd aufkommen, stilisiert sich das Programmieren ohne Hardware-Auflagen schnell zu einer Kunst.[7]

Das Wort von der „Softwarekrise" geht um: Die mangelnde Lokalisierung und Robustheit der Entwurfsentscheidungen (Nebenwirkungen sind nicht auszuschließen) und, damit eng verbunden, die aufwendige Wartbarkeit der sich aufblähenden *Spaghetti*-Programme (zwei Drittel der Softwarekosten fallen allein auf die Wartung), werden als ursächlich erkannt. Auf der Garmisch-Konferenz, im Oktober 1968, mit dem epochemachenden Titel *Software Engineering* wird erstmals eine „ingenieurmäßige" Pragmatik für die Softwaretechnik gefordert [Naur & Randell (Hrsg.), 1969]. Punkte, um die sich bis heute die Methodendiskussion dreht:

- Einführung eines Phasenmodells des Software-Lebenszyklus (Wasserfall-Modell)
- systematisches Analysieren, Entwerfen, Programmieren, Testen und Dokumentieren
- Definition und Standardisierung der Programmierschnittstellen
- Rechnerunterstützung mit dem langfristigen Ziel, den Entwurfsprozeß zu mechanisieren („ausführbare Spezifikation")

Als Theorie-Anwärterin für einen Paradigmenwechsel tritt die „strukturierte Programmierung" auf den Plan (erste Monographie von DAHL, DIJKSTRA und HOARE: *Structured Programming* [Dahl *et al.*, 1972]). Unter den Sammelbegriff fallen die schrittweise Verfeinerung der Entwurfsentscheidungen (*Top-down*-Entwurf), die Beschränkung auf drei Ablaufstrukturen (Sequenz, Auswahl und Schleife: *GOTO-less programming*), der modulare Entwurf aus einzeln prüfbaren *single-entry/single-exit*-Modulen und die Einführung von Programmierrichtlinien (strukturierter Quellcode durch Einrückungen und *begin-end*-Blöcke, kleine Module).

Aus der Handwerkskunst des Programmierens wird langsam eine Wissenschaft. Die Etablierung des Fachs Informatik an deutschen Hochschulen datiert Alfred LUFT auf die späten 60er Jahre [Luft, 1988].

[6] Siehe Gordon MOORES Gesetz von der Verdopplung der Integrationsdichte digitaler Schaltungen alle zwei Jahre: Bild 1.4 auf Seite 21.

[7] Über das Programmieren, verstanden als eine der „schönen Künste", siehe Donald KNUTHS Turing-Award-Lecture [Knuth, 1974].

4.3.3 Der Faktor „Mensch“

Mit der *Human-Factors*-Bewegung in der Informatik der 80er Jahre — die Probleme sind nicht mehr ausschließlich algorithmischer Art, sondern zunehmend schlecht definiert und erfordern die Mitwirkung des Benutzers in der Definitionsphase der Software-Entwicklung — kommt auch die Forderung nach einer Rückbesinnung auf das schöpferische Potential des Programmierers auf. Gerald WEINBERGs legendäres Werk *The Psychology of Computer Programming* ist seiner Zeit weit voraus [Weinberg, 1971]: „People talked to me as if I was a crazy person“, sagt er nach der ersten Konferenz über *Human Factors in Computer Systems* in Gaithersburg — elf Jahre später [Molzberger, 1984].

Christiane FLOYD fordert die Wende von der produktorientierten (mechanistischen) Sicht hin zur prozeßorientierten, in der die Bedürfnisse und kreativen Fähigkeiten des Menschen als Entwerfer und Nutzer von Software im Mittelpunkt stehen [Floyd, 1988]. Die pragmatischen Aspekte in der Produktentwicklung, wie Qualitätssicherung, Methoden- und Dokumenteneinsatz, müssen den situativen Kontext und die kognitiven Beschränkungen des Menschen berücksichtigen. Das impliziert beiderseitige Lernprozesse durch Kommunikation und Kooperation zwischen den beteiligten Personenkreisen: *Reviewing* im Team als kritischer Diskurs über Entwurfsentscheidungen und *Prototyping* als exploratorischer *Trial-and-Error*-Prozeß zwischen Entwickler und Anwender.

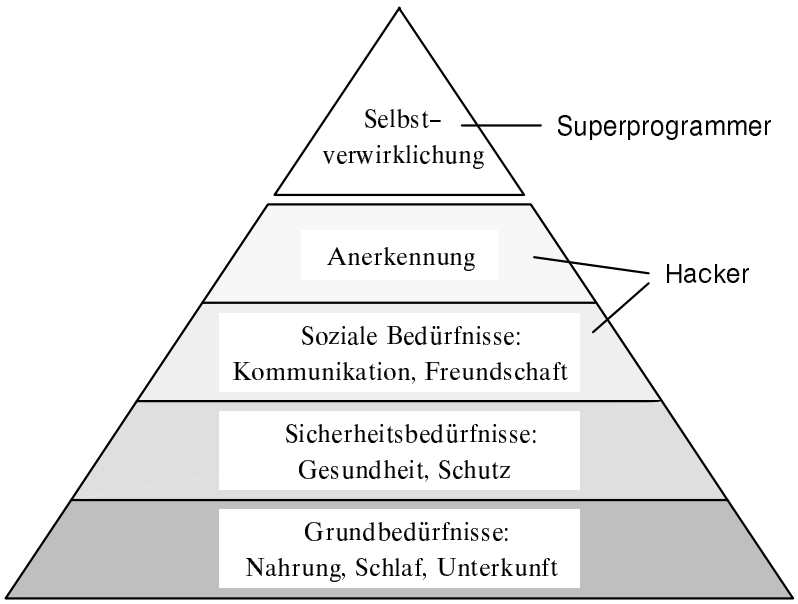


Bild 4.1: Superprogrammer versus Hacker

Peter MOLZBERGERs Suche nach dem kreativen und zugleich hocheffizienten *Superprogrammer* [Molzberger, 1984] und die von Frederick BROOKS erhoffte *Silver Bullet* gegen monströse Softwareprojekte, der *Great Designer* [Brooks, 1987], stehen exemplarisch für die Diskussion um das schöpferische Potential des Programmierers. Beide wissen die Erkenntnisse der modernen Motivationsforschung auf ihrer Seite: siehe Abraham MASLOWs Bedürfnispyramide in Bild 4.1. Es soll der *ganzheitliche* Mensch — seine analytischen-assoziativen Denkweisen und seine höheren Bedürfnisse nach Anerkennung (*ego needs*) und Selbstentfaltung — in den Entwurfsprozeß mit einbezogen werden.

„Ganzheitlichkeit" wird hier im Sinne der Neurophysiologie verstanden: In der linken Hemisphäre unseres Gehirns sind Sprache, Rationalität und Zielorientierung lokalisiert. Sie dominiert das Bildungswesen unseres Kulturkreises. Die rechte Hälfte beherbergt die assoziativen Erkenntnisprozesse, Kreativität und Intuition, und wird, zumindest was die Ausbildung der Programmierer betrifft, kaum gefördert. Das Phänomen des außergewöhnlichen Programmierers, wie BROOKS und MOLZBERGER es erkannt haben, ist eindeutig dem rechten Potential unseres Gehirns zuzuordnen. MOLZBERGER organisiert 1983 einen Workshop zum Thema: *Programming with the Right Hemisphere of the Brain.*

Auch in der Künstlichen Intelligenz wendet sich der Trend von der mechanistischen zur individuell menschlichen Orientierung. Glaubte man vorher noch, Denkprozesse auf Symbolmanipulationen reduzieren zu können [Simon, 1982], so beschränkt man sich heute auf die formale Wissensrepräsentation und versucht, aus dem Wissen eines einzelnen, des Experten, weiteres logisch abzuleiten [Brachman & Levesque (Hrsg.), 1985].[8]

4.3.4 SA/SD versus OOx

Wir kommen in unserer Chronik zum gegenwärtigen Paradigmenwechsel. Um zu zeigen, daß es sich hier in der Tat um einen Wechsel im KUHNschen Sinne handelt, wollen wir streng im klassischen Dreisatz vorgehen: 1. Rätsellösen in der Phase normaler Wissenschaft, 2. Auftreten einer Krise und 3. Überwindung der Krise durch einen neuen Theorie-Anwärter.

1. Rätsellösen

   Seit etwa Mitte bis Ende der 70er Jahre werden zahlreiche funktionale Dekompositionsmethoden[9] für das Struktur-Paradigma entwickelt und finden nach dem

[8] Kritische KI-Forscher, wie Terry WINOGRAD [Flores & Winograd, 1986] oder die Gebrüder DREYFUS [Dreyfus & Dreyfus, 1987], erfahren in ihrer *scientific community* das Reaktionsmuster erster Prägung, die „emotionale Ablehnung", und werden (vor)schnell als „Nestbeschmutzer" diffamiert [Luft, 1988, S. 71].

[9] Paul FREEMAN und Allen NEWELL modellieren die funktionale Dekomposition eines Artefakts als Strukturpaar: $< F, P >$. F steht für die funktionale Struktur des Artefakts. Sie repräsentiert die geforderte

„CLAUSschen Akzeptanzverzug“ zehn Jahre später Eingang in die industrielle Praxis. Die Analysemethode nach Tom DEMARCO und die SA/SD-Methode (*structured analysis/structured design*) nach Edward YOURDON gelten als etabliert [DeMarco, 1978; Yourdon, 1988].

2. Krise

Der Freiraum des Software-Entwurfs wird nicht länger durch die Unzulänglichkeiten der Hardware beschnitten. Immer komplexere Probleme werden angegangen, die nur noch arbeitsteilig im Team zu lösen sind. Frank DEREMER und Hans KRON prägen hierfür den Begriff des *Programming-in-the-Large* [DeRemer & Kron, 1976]. Peter SCHNUPP macht einen „Methodenberg“ des Struktur-Paradigmas aus [Schnupp, 1983], dessen Schwerpunkt über den technischen Phasen der Software-Entwicklung liegt: Bild 4.2. Die methodische Unterstützung in den problemnahen konzeptionellen Entwurfsphasen wird als unzureichend empfunden.

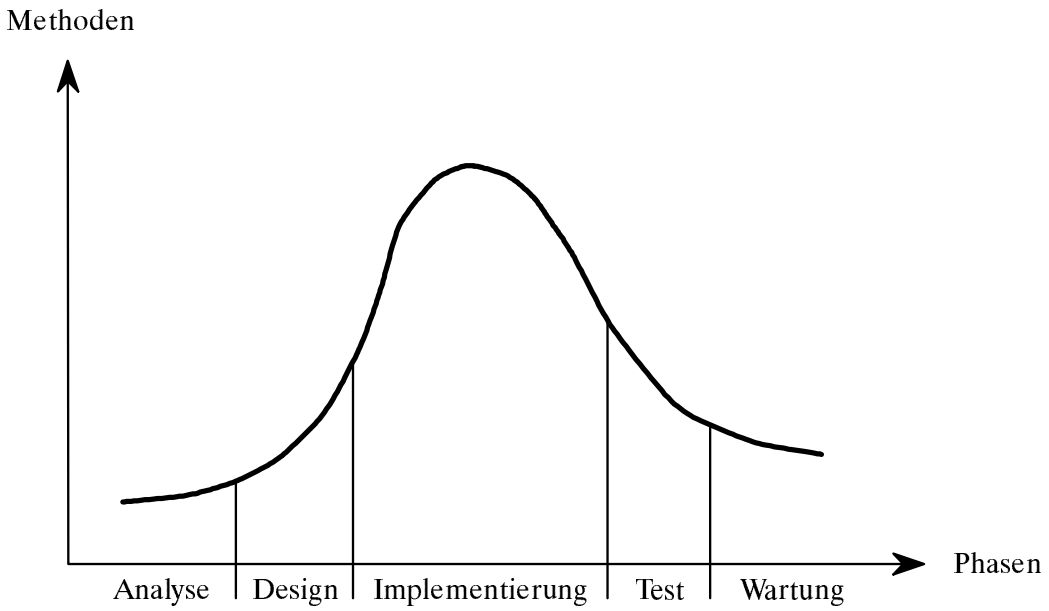


Bild 4.2: Methodenberg des Struktur-Paradigmas

Funktionalität (nicht die vage formulierten Kundenwünsche, sondern die aus der Analyse hervorgegangene Spezifikation). P steht für die physikalische Struktur des Artefakts. Sie repräsentiert die Komponenten (*bill of materials*) und deren Organisation. Nach dem Struktur-Paradigma verläuft der Entwurfsprozeß entlang einer Linie unabhängiger Prozeduren: 1. funktionale Dekomposition (F), 2. materielle Abbildung ($F \rightarrow P$), 3. physikalische Organisation (P) und 4. Verifikation der Funktionen in P, auf die sich F bezieht [Freeman & Newell, 1971].

Forderungen, die erkannten Defizite im Software-Lebenszyklus zu beheben, werden laut:

- Aufhebung der willkürlichen Trennung der Datenstrukturen von der Ablaufsteuerung oder mit anderen Worten: die Schaffung einer Einheit aus statischer Information und dynamischem Zugriff, der Arbeitsgegenstand wird ganzheitlich (Klassenkonzept in Simula [Dahl & Nygaard, 1966], abstrakter Datentyp [Liskov & Zilles, 1974])
- Verlagerung methodischer Ansätze vom Lösungs- in den Problembereich, somit eine größere Entfernung von den Vorgaben der Rechnerarchitektur eines John von NEUMANN
- Durchgängigkeit der Konzepte und Begriffe im Software-Lebenszyklus: die Ergebnisse der Phase i sollen sich ohne Konvertierungsaufwand als Grundlage für das weitere Vorgehen in der Phase $i + 1$ eignen

3. Lösung

   Die Indizien und Argumente für einen bevorstehenden Paradigmenwechsel häufen sich:

   - Neue Forschungsanstrengungen

     Bereits im Exposé dieser Abhandlung wurde ein statistisches Indiz für den aktuellen Methodenwechsel von „SA/SD nach OOx“ aufgezeigt (siehe Seite 5).

   - Außergewöhnliche Forscher

     (a) Alan KAYs Vision vom notizbuchgroßen *Personal Computer*, dem *Dynabook* [Kay, 1977], ist heute Realität und hat uns zahlreiche objektorientierte Errungenschaften beschert: neben der interaktiven Sprache Smalltalk vor allem hochflexible, an den kognitiven Fähigkeiten des Menschen orientierte Grafik-Rechner mit Fenstertechnik, Sinnbildern, Menüführung und der Maus als Eingabemedium [Kay, 1993]. Apples *Macintosh* mit all seinen ergonomischen Vorzügen ist das geistige Kind des Software-Visionärs Alan KAY. (b) Als ein weiteres Indiz für die Folgerichtigkeit der Objektorientierung in der Informatik mag Edward YOURDONs Sinneswandel gelten: vom Wegbereiter der strukturierten Systemanalyse (SA) und des strukturierten Entwurfs (SD) zum Verfechter des Objekt-Paradigmas in diesen Disziplinen: OOA [Coad & Yourdon, 1991a] und OOD [Coad & Yourdon, 1991b].

   - Bewahren erfolgreicher Lösungskonzepte und Versprechen neuer

     Das Objekt-Paradigma vereinigt drei Tendenzen in der Geschichte der Informatik: (a) die Entwicklung abstrakter Sprachmittel, die dem Problem näher sind als der Maschine, (b) die Entlastung des Systementwerfers von Entwurfsentscheidungen auf unteren Ebenen (*Information hiding* im Sinne von David

PARNAS [Parnas, 1972a]) und (c) die Suche nach interdisziplinären Prinzipien des Entwerfens.

Der Kürze wegen skizzieren wir das objektorientierte Lösungspotential — die Konzepte, Modelle, Methoden und Techniken — nach übergeordneten Gesichtspunkten: ETHOS [Thiele, 1991]. Zusammen betonen sie den ganzheitlichen Anspruch des Objekt-Paradigmas [Quibeldey-Cirkel, 1994c]:

E wie „economic": Das Klassenkonzept aus Simula und das Kapselungsprinzip des *Information hiding* erfüllen wichtige wirtschaftliche Forderungen: Software-Wiederverwendung, lokale Wartbarkeit und Erweiterbarkeit. Klassenbibliotheken unterstützen den ingenieurmäßigen „Bausteinentwurf", wie er im Maschinenbau, in der Bautechnik und Fließbandindustrie gang und gäbe ist. Dem *Time-to-Market*-Faktor wird Rechnung getragen: Die Entwicklungsproduktivität und die Produktqualität steigen [Endres, 1988; Meyer, 1990].

T wie „technical": Die Objektorientierung führt zu Modellierungsmethoden und -techniken, die durchgängig in den klassischen Phasen des Entwurfs Verwendung finden: Analyse- und Designmethoden [Blaha *et al.*, 1991; Booch, 1991; Coad & Yourdon, 1991a; Mellor & Shlaer, 1992; Wiener *et al.*, 1990]; inkrementelle Programmierung in Smalltalk [Goldberg, 1984] oder Eiffel [Meyer, 1988]; Datenarchivierung [Dittrich, 1989] und grafische Benutzungsoberflächen [Ege, 1991].

H wie „human": Die objektorientierten Metaphern in der Softwaretechnik (Beispiele: *Vererbung*, *Austausch von Botschaften* und *Client-Server*, siehe auch Anhang A) rufen Assoziationen und Analogien beim Benutzer hervor. Die rechte Hemisphäre des Gehirns wird angeregt und führt mit den algorithmischen Komponenten der linken zu einem *ganzheitlichen Denken*. Objektorientierung ist intuitiv. Sie unterstützt die Wahrnehmung und kognitive Verarbeitung eines Problems. Sie stellt zugleich ein *ontologisches* Grundprinzip dar, das geeignet ist, auch technische Artefakte einfach und ganzheitlich zu modellieren [Wand, 1989a]. Ontologische Referenzmodelle erlauben die vorurteilslose Bewertung von Analyse- und Designmethoden [Wand & Weber, 1989].

O wie „organizational": Die objektorientierten grafischen Oberflächen senken die Akzeptanzschwelle des Rechners in den betrieblichen Standardsituationen. Das konzeptionelle Potential des Objekt-Paradigmas wird zunehmend auch in der Unternehmensmodellierung und im Produktdatenmanagement ausgeschöpft. Objektorientierte Erweiterungen des *Entity-Relationship*-Modells von Peter CHEN setzen sich durch [Chen, 1976]. Produktdaten werden nach der ISO-Norm STEP in Express beschrieben, einer objektorientierten Modellierungssprache [Schenk, 1990]. Im Rahmen der *Programming-in-the-Large*-Projekte setzt die Projektführung verstärkt OO-Konzepte und -Methoden ein [Endres & Uhl, 1992; Gryczan & Züllighoven, 1992].

S wie „social“: Die Interdisziplinarität des Objektansatzes fördert die Kommunikation und Kooperation sowohl innerhalb der Informatik (Softwaretechnik, Datenbanken und KI) als auch zwischen den Vertragspartnern eines Softwareprojekts, wo Auftraggeber und Auftragnehmer aus unterschiedlichen Fachgebieten kommen. Die Schnittstellenprobleme kommunikativer Art werden durch das *Requirements-Engineering* und *Rapid Prototyping* als objektorientiertes dialogisches Bindeglied zwischen Benutzer und Systementwerfer entschärft [March & Umphress, 1991]. Auf einer höheren Ebene fungiert das Objekt-Paradigma als *disziplinäre Matrix* einer „Wissenschaft des Entwerfens“.

4.4 Objektorientierte Weltmodelle

Eingedenk des paradigmatischen Wechsels im philosophischen Selbstverständnis der Informatik und der aktuellen Methodenwende in der Softwaretechnik vom Struktur- zum Objekt-Paradigma läßt sich ein Bild des allgemeinen Entwurfsvorgangs skizzieren, für Soft- und Hardware gleichermaßen. Als Beispiel diene der VLSI-Entwurf, verstanden als konzeptionelles Co-Design von Soft- und Hardware.

Die zahlreichen Abstraktionen und Sichten im VLSI-Entwurf reichern einen „Datenpool“ an, der die Historie der Entwurfsobjekte fortschreibt. Zur Modellierung des Datenpools führen wir den Begriff des „Weltmodells“ ein, angelehnt an den Begriff des „Datenmodells“ in der Datenbanktechnik. Wie Datenmodelle, zum Beispiel das relationale von Edgar CODD [Codd, 1979], umfassen Weltmodelle die konzeptionellen Mittel zur Modellierung eines Weltausschnitts, *Miniwelt* genannt.

In den Vordergrund rücken also zwei Facetten der Informationsverarbeitung:

1. Datenrepräsentation: Modellieren und Schematisieren der Entwurfsinformation
2. Datenverwaltung: Konsistentes Fortschreiben und Abfragen des Datenbestands

Um Systeme zu modellieren, setzen wir Mittel der Abstraktion ein, Konzepte und Formalismen, die die Entwurfskomplexität reduzieren und beherrschbar halten [Quibeldey-Cirkel & Wojtkowiak, 1991]. Der Mensch mit seinen kognitiven Grenzen tritt in den Vordergrund [Norman, 1986]. Die Soft- und Hardware-Abstraktionen, wie sie beispielsweise von Carlo SÉQUIN für die Hardware [Séquin, 1983] und von Mary SHAW für die Software [Shaw, 1984] aufgezeigt wurden, stehen für die Eckpfeiler einer fortschreitenden methodischen „Disziplinierung“. Ob *Programming-in-the-Large* oder *System-on-Chip*, die Parallelen sind unverkennbar: Tabelle 4.1.

Abstraktion	*Hardware*	*Software*
Modularisierung	Bausteinprinzip: Zellbibliotheken, Steuer- und Operationswerke, Speichermodule, Boards	Blockstrukturen: Funktionen, Prozeduren, abstrakte Datentypen, Klassen
Hierarchie	Abstraktionsebenen (*system, algorithmic, micro-architectural, logic, circuit*), Sichten (*behavioral, structural, physical*) [Gajski & Kuhn, 1983]	*schrittweise Verfeinerung* [Wirth, 1971], Haupt- und Unterprogramme, Klassenhierarchien und Vererbung [Meyer, 1988]
Formalisierung	Datenaustauschformate, Hardware-Beschreibungssprachen, Silicon-Compiler [Gajski, 1988]	Algebraische Spezifikation [Horebeek & Lewi, 1989], Programmgeneratoren

Tabelle 4.1: Soft- und Hardware-Abstraktionen

Wir entnehmen der Methodenentwicklung eine übergreifende Gesamtausrichtung, eine Grundströmung, die sich durch die Praktische und Technische Informatik zieht: Wissensrepräsentation in der KI [Brachman & Levesque (Hrsg.), 1985], semantische Datenmodellierung in der Datenbanktechnik [Dittrich, 1989], Objektmodelle in der Programmiermethodik [Wand, 1989a] und Informationsmodellierung im Kontext von CAD-Frameworks [Quibeldey-Cirkel, 1993b]. Ohne es deutlich herauszustellen, betreiben die verschiedenen Informatikforschungen eine vergleichbare Methode der Komplexitätsbewältigung: die Modellbildung auf konzeptioneller Ebene. Auch wenn es sicherlich Unterschiede in der Zielsetzung gibt, so sind doch die Methoden vergleichbar. Die fachübergreifenden Konferenzen und Veröffentlichungen in der Informatik zeugen davon: [Brodie *et al.* (Hrsg.), 1984; Kim & Lochovsky (Hrsg.), 1989; Meersman & Sernadas (Hrsg.), 1988]. Es ist hier nicht der Raum, um die Parallelen im Detail aufzuzeigen, aber es genügt wohl ein allgemeines Methodenverständnis, um den Schluß nachzuvollziehen: Das augenfällig Gemeinsame ist die Suche nach einfachen und zugleich ausdrucksvollen Mitteln, um das Inventar und die Dynamik eines Weltausschnitts zu beschreiben.

Vor der Implementierung steht die Spezifikation und vor der Spezifikation das „Begreifen“ eines Problems, das kognitive Erfassen der Miniwelt. Bislang weist die Entwurfsschiene von der Idee zum Produkt *konzeptionelle* Fugen auf: Die Beschreibungsmittel der Problemanalyse, der Spezifikation und der Implementierung sind in der Regel verschieden, die „semantischen Lücken“ sprichwörtlich. Objektorientiertes *konzeptionelles Modellieren* will alle Phasen durchgängig halten: methodisch und begrifflich. BROOKS zählt sie zu den Hoffnungen auf eine *Silver Bullet* gegen monströse Softwareprojekte:

„Fashioning complex conceptual constructs is the *essence*; *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.“[10] [Brooks, 1987]

4.4.1 Ausführbare Weltmodelle

Wir stimmen überein mit David HARELs Antwort auf BROOKS' Pessimismus: *Biting the Silver Bullet: Toward a Brighter Future for System Development* [Brooks, 1987; Harel, 1992]. Die „Akzidenzien“ der Software-Entwicklung — die zufälligen Begleiterscheinungen und Schwierigkeiten — sind beherrschbar oder werden es in naher Zukunft sein. Grafik-Werkzeuge, wie Generatoren, Editoren und Browser, und Managementwerkzeuge für die Versionsverfolgung und Konfiguration von Objekten erleichtern dem Weltmodellierer die Arbeit. Seine Modelle konstruiert, analysiert und dokumentiert er bereits auf dem Rechner. Stets schwierig bleibt die „Essenz“, die in Struktur und Verhalten liegende Komplexität des Weltausschnitts.

Was wir brauchen, um die Essenz einer komplexen Entwurfsaufgabe in den Griff zu bekommen, sind Code erzeugende Werkzeuge, die das konzeptionelle Modell „zum Laufen bringen“, es sprachlich umformen in ein Programm oder in eine Hardware-Beschreibung (VHDL). Das sind zum Beispiel Schema-Übersetzer, die ein Weltmodell in ein Datenbankschema übersetzen oder Modell-Simulatoren, die eine Aussage über die Konsistenz der Objekte unter dynamischen Bedingungen erlauben. Das heißt also: Rechnerunterstützung nicht nur bei der textuellen und grafischen Darstellung von Weltmodellen, sondern auch bei der Inspektion, Integritätssicherung und Implementierung derselben.

4.4.2 Szenario des objektorientierten Entwurfs

Der objektorientierte Weg im Software-Lebenszyklus läßt sich durchgängig beschreiten: OO-Analyse und OO-Design führen zu den Weltmodellen der Anwendung. OO-Programmierung und OO-Datenbanken, wobei Entwurfswerkzeuge als Objekte verwaltet werden [Daniell, 1989], setzen die Weltmodelle und die aus ihnen gewonnenen Objekte in ausführbare Rechnermodelle dauerhaft um. Das VENN-Diagramm in Bild 4.3 soll die Schnittmengen an gemeinsamen Konzepten und Sprachmitteln verdeutlichen, die in einem voll objektorientierten Entwurfsprozeß auftreten und zur begrifflichen und methodischen Durchgängigkeit beitragen. Auf die Gemeinsamkeiten, wie Klassen, Objekte, Vererbung und Polymorphie, wird in [Snyder, 1993] näher eingegangen.

[10] Wie die *conceptual constructs* beispielsweise für den VLSI-Entwurf aussehen, wurde in [Quibeldey-Cirkel, 1993a] untersucht.

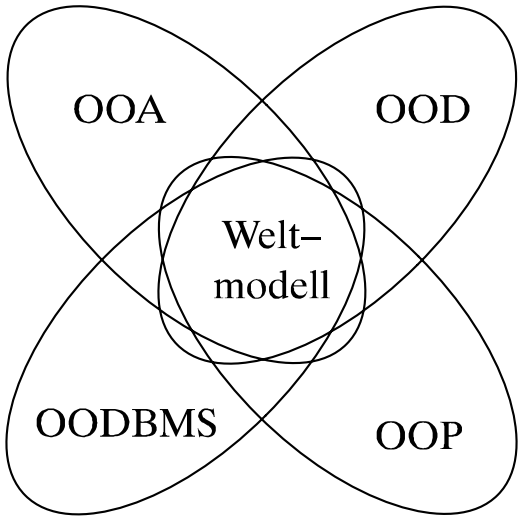


Bild 4.3: OO-Welt des Software-Entwurfs

CASE-Werkzeuge unterstützen bereits ausreichend die implementierungsnahen Entwurfsphasen: Objektorientierte Programmierumgebungen und Datenbanken sind Stand der Technik und werden erfolgreich eingesetzt [Endres & Uhl, 1992]. Die mehr der Problemanalyse zugewandten Entwurfsabschnitte hingegen befinden sich werkzeugtechnisch *in statu nascendi* und schicken sich erst an, die etablierten strukturierten Analyse- und Entwurfsmethoden in Frage zu stellen. Oder, um das Bild des Methodenbergs ein zweites Mal zu bemühen: Der Methodenbestand strebt erst noch die Form eines Tafelbergs an, der alle Phasen der Produktentwicklung gleichermaßen überdeckt.

Fazit

KUHNS Begriff des Paradigmenwechsels wird auch in der Informatik strapaziert. Einschneidende Phasenwechsel in einer Wissenschaft bleiben rar, sind dann aber bedeutungsvoll für den Erkenntnisprozeß. Die Voraussetzungen für die Verwendung des Begriffs sind im allgemeinen nicht gegeben oder werden ignoriert: Auftreten einer Krise im Rätselraten mit Hilfe des gültigen Paradigmas, Gelehrtenstreit mit gruppenpsychologischen Reaktionsmustern und Überwindung der Krise durch einen neuen erfolgversprechenden Theorie-Anwärter.

Die Paradigmenwechsel in der Softwaretechnik wurden skizziert. Der aktuelle Wechsel von SA/SD nach OOx wurde herausgestellt. Der Handlungsbedarf zielt auf die Problembeschreibung durch den Endbenutzer und auf die Problemanalyse durch den Systementwickler. Das hier angesiedelte moderne Requirements-Engineering sucht den einfachen Anschluß an die traditionelle Softwaretechnik (siehe zum Beispiel [Sommerville, 1990]). Die objektorientierte konzeptionelle Modellierung bietet eine tragfähige Bindung und universelle Sprachmittel, um das Wissen über einen Weltausschnitt ohne Strukturbruch zu repräsentieren und begrifflich durchgängig für die Verarbeitung aufzubereiten.

Ein neues Paradigma muß, will es erfolgreich sein, die Vorteile der alten Paradigmen in sich vereinen. Der Objektansatz, verbunden mit der konzeptionellen Modellierung, bewahrt die Errungenschaften seiner Vorgänger. Er überwindet zugleich die „Komplexitätsbarrieren“ [Winograd, 1975a] bisher ungelöster Rätsel. Die Ursprache der Objektorientierung, Simula-67, feierte schon ihr 25-jähriges Jubiläum. Das ist gerade die Mindestzeitspanne laut Thomas KUHN für einen erfolgreichen Paradigmenwechsel.

Teil II

Objektorientierung als methodischer Ansatz

„The concept of hierarchic order occupies a central place in this book, and lest the reader should think that I am riding a private hobby-horse, let me reassure him that this concept has a long and respectable ancestry. So much so, that defenders of orthodoxy are inclined to dismiss it as ‚old hat' — and often in the same breath to deny its validity. Yet I hope to show as we go along that this old hat, handled with some affection, can produce lively rabbits."

Arthur KOESTLER [Koestler, 1967]

Im Lichte der Objektorientierung rezipieren wir zunächst die Leitgedanken renommierter Forscher zu einer „Wissenschaft des Entwerfens". Gegenstand der nachfolgenden Kapitel ist dann die Modellierung komplexer Entwurfsprozesse. Rückschauend gehen wir auf konventionelle Modellierungsansätze ein und entwickeln ausblickend ein innovatives Objektmodell. Wir stellen zunächst Modellierungsvorschläge aus der einschlägigen Literatur vor und unterziehen sie einer vergleichenden Bewertung. Die Auswahl beschränkt sich auf Vorschläge, die Anleihen machen bei der Kybernetik, Informationstheorie und Verfahrenstechnik. Die Einordnung der untersuchten Ansätze erfolgt hierbei nach Erkenntnissen einer allgemeinen Modelltheorie. Die Studie legt offen, daß die Mehrheit der Entwurfsmodelle streng systemorientiert ist. Kognitive Aspekte des Modellentwerfers und -benutzers finden kaum Berücksichtigung.

Inspiriert durch den intuitiven Charakter objektorientierter Konzepte und Anwendungen [Lang & Quibeldey-Cirkel (Hrsg.), 1992; Quibeldey-Cirkel, 1994c] schlagen wir eine Entwurfsmethode vor, die kognitive Versäumnisse weitgehend vermeidet. Wir untersuchen die These der „vollständigen Durchgängigkeit" objektorientierter Konzepte aus der Sicht der Denkpsychologie: Entwerfen als Problemlösen [Quibeldey-Cirkel, 1994b]. Schließlich führen wir das Objekt-Paradigma in den Entwurfsprozeß ein, tragen die Modellierungskonzepte für VLSI/CAD-Entwurfsobjekte zusammen und stellen sie einem Objektmodell gegenüber. Anschließend zeigen wir, wie das Leitmotiv der Objektorientierung die Weltmodelle moderner VLSI/CAD-Entwurfsumgebungen und -Schnittstellen durchzieht.

5

Von einer Wissenschaft des Entwerfens

Der Paradigmabegriff hat durch KUHN selbst eine Doppelauslegung erfahren: Einerseits meint Paradigma die einheitliche Überzeugung über den Forschungsgegenstand und über die Methodik der Erforschung. Andererseits steht der Begriff auch für die Gesamtheit dessen, was eine Gemeinschaft von Wissenschaftlern an sozialen Elementen verbindet. KUHN bezeichnet die soziale Komponente als *disziplinäre Matrix*. Wir wollen uns seine Ansicht zu eigen machen und die disziplinäre Matrix des Objekt-Paradigmas untersuchen. Der Horizont der Erkenntnis geht dabei über die Grenzen unserer Disziplin hinaus: Daß sich die informationstechnischen Fachgruppen immer mehr durchdringen, haben wir schon betont. Die Verflechtung zwischen Künstlicher Intelligenz, Datenbanktechnik und objektorientierter Softwaretechnik sind bekannt und Thema *intra*-disziplinärer Zusammenkünfte. Der Weg zu den *objektorientierten Weltmodellen* als Ausdruck dieser Intra-Disziplinarität wurde bereits beschritten. Der soziale Aspekt eröffnet nun eine weitere, eine *inter*-disziplinäre Dimension: die wissenschaftliche Gemeinschaft der Entwerfer.

Übersicht: Abschnitt 5.1 skizziert die Vision Herbert SIMONs von einer universellen „Wissenschaft des Entwerfens". Die gemeinsame Grundlage aller Entwurfsdisziplinen ist die schöpferische Gestaltung der *zweckbestimmten* Schnittstelle zwischen der inneren und äußeren Umgebung eines Artefakts. Wir erläutern die wichtigsten Lehrinhalte eines Curriculums des Entwerfens nach SIMON. Die Logik des Entwerfens liegt in der Suche nach *zufriedenstellenden* Alternativen. Entwerfen ist kein Optimierungsproblem, sondern wird vom Kosten-Nutzen-Verhältnis geleitet. Im Abschnitt 5.2 gehen wir den curricularen Vorstellungen eines weiteren prominenten Wissenschaftlers nach: Heinz ZEMANEK. Er sieht den vereinenden Ansatz für eine Wissenschaft des Entwerfens vor allem in den architektonischen Parallelen, in den Architekturidealen von VITRUV und ALEXANDER. Wurden in den beiden vorangegangenen Abschnitten die Beiträge des Objekt-Paradigmas noch gesondert dargestellt, entfällt die textuelle Trennung im ontologischen Vergleich des Abschnitts 5.3. Denn Ontologie ist die wissenschaftliche Philosophie von den realen Dingen (Objekten) unserer Welt schlechthin. Das ontologisch formale Objektmodell nach Yair WAND, das wir hier skizzieren, steht also *implizite* für den Beitrag der Objektorientierung.

5.1 „Creating the Artificial“

Herbert SIMON prägte den Begriff schon 1969. Wir referieren im folgenden seine Grundüberlegungen [Simon, 1982]: Während die Naturwissenschaften natürliche Objekte und Erscheinungen untersuchen, stehen künstliche Objekte — speziell *künftige* künstliche Objekte mit *geplanten* Eigenschaften — im Mittelpunkt der „Sciences of the Artificial“. Die „Artificial Intelligence“ (KI) ist davon nur eine unter vielen. Ein künstliches Objekt, ein Artefakt, kann als Punkt der Begegnung, als „Schnittstelle“ betrachtet werden: zwischen einer „inneren“ Umgebung, der Substanz und Gliederung des Artefakts, und einer „äußeren“, in der es eingefügt ist. Wenn die innere Umgebung der äußeren angemessen ist oder umgekehrt, dann wird das Artefakt seinen Bestimmungszweck erfüllen. Zweckerfüllung vereint also drei Komponenten: den Zweck selbst, die Beschaffenheit des Artefakts und seine Umwelt: Bild 5.1. Die Naturwissenschaften betrachten lediglich die Wechselbeziehung zwischen den letzten beiden. Die erste Komponente aber, der *planerische Zweck*, der Entwurf eines Kunstprodukts, entbehrt bislang einer wissenschaftlichen Betrachtung. Ein Beispiel soll das Tripel verdeutlichen: Ob ein Messer schneidet (Zweck), hängt ab vom Material seiner Klinge (Beschaffenheit) und von der Härte der Substanz, an der es erprobt wird (Umwelt).

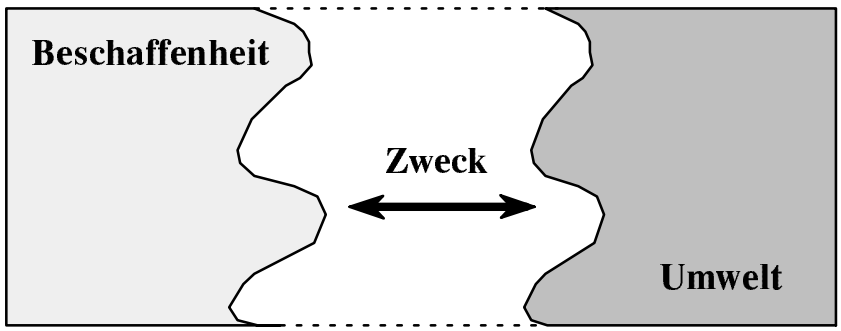


Bild 5.1: Das Tripel des Artefakts

Eine Trennung zwischen äußerer und innerer Umgebung hätte zwei Vorteile: Einerseits könnte das Verhalten eines adaptiven Systems vorausgesagt werden und das allein mit der Kenntnis seines Ziels und seiner äußeren Umgebung. Nur minimale Annahmen über seine innere Umgebung wären vorauszusetzen. Andererseits ergäbe sich ein entsprechender Vorteil aus der Perspektive der inneren Umgebung: Ob ein System sein Ziel oder seine Anpassung erreicht, entscheiden oft nur wenige Merkmale der äußeren Umgebung, nicht aber deren Einzelheiten. Biologen bezeichnen diese Eigenschaft adaptiver Systeme

als *Homöostasie*. Sie ist eine wichtige Eigenschaft aller guten Konstruktionen, seien sie biologisch oder künstlich. In der besten aller möglichen Welten des Entwerfers bestünde die Hoffnung, die Vorteile der Dreiteilung zu vereinen: Wir könnten die Haupteigenschaften eines Systems und sein Verhalten beschreiben, ohne die relevanten Naturgesetze in seiner inneren und äußeren Umgebung zu präzisieren: „We might look toward a science of the artificial that would depend on the relative simplicity of the interface as its primary source of abstraction and generality“ [Simon, 1982, S. 12].

Die Welt des Künstlichen hat ihre Achse genau in dieser Schnittstelle zwischen innerer und äußerer Umgebung. Die Mittel an die Umgebungen planend anzupassen, ist der Forschungsgegenstand des Künstlichen. Im Zentrum steht der Vorgang des Entwerfens. SIMON unterstreicht die Schlüsselrolle, die das Entwerfen in den Wissenschaften einnimmt:

> „Engineers are not the only professional designers. Everyone designs who devises courses of action aimed at changing existing situations into preferred ones. The intellectual activity that produces material artifacts is no different fundamentally from the one that prescribes remedies for a sick patient or the one that devises a new sales plan for a company or a social welfare policy for a state. Design, so construed, is the core of all professional training; it is the principal mark that distinguishes the professions from the sciences. Schools of engineering, as well as schools of architecture, business, education, law, and medicine, are all centrally concerned with the process of design.“ [Simon, 1982, S. 129]

Das Gemeinsame aller Wissenschaften vom Künstlichen findet sich also in einer *Wissenschaft des Entwerfens* wieder: „a body of intellectually tough, analytic, partly formalizable, partly empirical, teachable doctrine about the design process“ [Simon, 1982, S. 132]. Die neuzeitliche Gemeinsamkeit der verschiedenen Fachrichtungen liegt in der Tatsache begründet, daß alle Wissenschaftler, die den Rechner in komplexer Weise benutzen, ihn für den Entwurfsvorgang einsetzen. Der gemeinsame Wissenskern, die Konzepte und Modelle des Entwerfens, ist aber keineswegs augenfällig: er verbirgt sich in der Mannigfaltigkeit der künstlichen Phänomene. Da die objektorientierte Weltsicht auf den Entwurf beliebiger künstlicher Systeme anwendbar ist, liegt die Vermutung nahe, in einer Wissenschaft des Entwerfens auch objektorientierte Facetten zu entdecken. Wir werden bei der Erörterung der SIMONschen These danach ausschauen.

5.1.1 Curriculum einer Wissenschaft des Entwerfens

Künstlichkeit ist hauptsächlich dort interessant, wo sie komplexe Systeme betrifft, die in komplexen Umgebungen eingebunden sind. Künstlichkeit und Komplexität sind untrennbare Themen. SIMON stellt die These auf, derzufolge gewisse Phänomene in einem sehr spezifischen Sinne „künstlich“ sind: Sie sind nur deshalb wie sie sind, weil ein System durch Ziel und Zweck in die Umgebung eingepaßt ist, in der es „lebt“. Wenn natürliche

Erscheinungen — in ihrer Unterwürfigkeit gegenüber den Naturgesetzen — eine Aura von „Notwendigkeit“ um sich haben, so zeigen künstliche Erscheinungen — in ihrer Formbarkeit durch die Umwelt — eine Aura von „Zufälligkeit“ und „Möglichsein“.[1] Das hat stets Zweifel genährt, ob künstliche Phänomene überhaupt zum Gegenstand der Wissenschaft gehören: Wie kann man empirische Aussagen über Systeme machen, die unter anderen Umständen ganz anders wären, als sie gerade sind? SIMON zeigt, *daß* empirische Aussagen möglich sind. Beispiele hierfür entnimmt er den Wirtschaftswissenschaften (hier erhielt er 1978 den Nobelpreis), der Erkenntnispsychologie und dem Ingenieurwesen (hier bekam er zusammen mit Allen NEWELL den Turing-Award [Newell & Simon, 1975]). Er skizziert, wie eine Entwurfstheorie zu entwickeln wäre und wie sie aussähe.

Eine Wissenschaft des Entwerfens ist nach SIMONs These nicht nur möglich, sie ist im Entstehen. Ihr Einzug hat bereits begonnen: in die Ingenieurwissenschaften über die Lehrpläne der Informatik und der Systemtheorie und in die Wirtschaftswissenschaften über die Theorie der Verwaltung [Simon, 1976]. Bild 5.2 zeigt das Curriculum einer Wissenschaft des Entwerfens, wie es SIMON vorschlägt. Wir wollen auf die Lehrinhalte näher eingehen, um die Parallelen zum Objekt-Paradigma zu verdeutlichen.

1. Die Bewertung von Entwürfen
   - Theorie der Bewertung: Nutzentheorie, statistische Entscheidungstheorie
   - Methoden der Berechnung:
     - Algorithmen zur Auswahl *optimaler* Alternativen (Lineare Optimierung, Regelungstheorie, dynamische Programmierung)
     - Algorithmen und Heuristiken zur Auswahl *zufriedenstellender* Alternativen
2. Die formale Logik des Entwerfens: imperative Logik und Aussagenlogik
3. Die Suche nach Alternativen
   - Heuristische Suche: Aufgliederung und Mittel-Zweck-Analyse
   - Verteilung der Ressourcen für die Suche
4. Theorie der Struktur- und Entwurfsorganisation: hierarchische Systeme
5. Repräsentation der Entwurfsprobleme

Bild 5.2: SIMONs Curriculum einer „Wissenschaft des Entwerfens“

[1] SIMON spielt hier auf den philosophischen Gegensatz von „Kontingenz und Notwendigkeit“ an.

Die Bewertung von Entwürfen: optimal versus zufriedenstellend

Die Naturwissenschaftler beschäftigen sich mit den Dingen, wie sie *sind*. Aussagen- und Prädikatenkalkül, gewöhnliche Logiksysteme also, leisten hier gute Dienste. Anders der Entwerfer: er interessiert sich dafür, wie die Dinge sein *sollten*, für das Erfinden von Artefakten, die Ziele erreichen *sollen*. „Sollen" impliziert eine andere Logik als die der Aussagen und Prädikate: eine *imperative* oder *modale Logik* mit Operatoren wie „erlaubt", „verboten" oder „geboten". SIMON zeigt nun, daß sich die modale Logik des Entwerfens wieder auf die deklarative Standardlogik reduzieren läßt. Für die Bewertung unserer Entwürfe können wir somit auf die klassischen Optimierungsmethoden zurückgreifen. Deren Logik skizziert SIMON wie folgt:

Die „innere" Umgebung des Entwurfs bildet die Menge der Handlungsalternativen. Die Alternativen können vollständig gegeben sein, gewöhnlich aber sind sie durch *Bestimmungsvariable* mit definierten Bereichen spezifiziert. Die „äußere" Umgebung bildet eine Menge von Parametern, die exakt oder nur als Wahrscheinlichkeitsverteilung bekannt sind. Die Ziele für die Anpassung der inneren an die äußere Umgebung definiert eine Nutzenfunktion: eine normalerweise skalare Funktion mit den Bestimmungsvariablen und Parametern als Argumenten, eventuell ergänzt durch Schranken in Form von Ungleichungen zwischen Funktionen der Argumente. Das Optimierungsproblem besteht nun darin, die Werte der Bestimmungsvariablen zu berechnen, die unter den gegebenen Einschränkungen und Parametern den Nutzen maximieren.

SIMON erläutert die Logik der Optimierungsmethode am Beispiel des „Diätenproblems": Es liegt eine Liste von Nahrungsmitteln vor. Bestimmungsvariable sind die Mengen verschiedener in der Diät vorgesehener Nahrungsmittel. Die Preise und Nährwerte (Kalorien, Vitamine, Mineralstoffe usw.) der einzelnen Nahrungsmittel stehen für die Umgebungsparameter. Die Nutzenfunktion ist durch die Kosten der jeweiligen Diätzusammenstellung gegeben. Als Charakterisierung der inneren Umgebung könnten zum Beispiel die folgenden Beschränkungen gelten: nicht mehr als 2000 kcal am Tag, Deckung des minimalen Bedarfs an Vitaminen und Spurenelementen sowie Höchstwerte für gewisse Schwermetalle. Das Problem besteht nun darin, jene Menge von Nahrungsmitteln zu bestimmen, die den Nährwertforderungen und Nebenbedingungen genügt und am wenigsten kostet.

Das Diätenproblem, einmal formalisiert, reiht sich ein in die mathematischen Standardprobleme der Linearen Optimierung: Maximierung einer Funktion unter Randbedingungen. Damit beruht auch die Logik des Problems auf der Standardlogik der Mathematik, dem Prädikatenkalkül. Wie vermeidet nun die Formalisierung des Optimierungsproblems den Gebrauch einer modalen Logik? Sie vermeidet ihn, indem sie sich mit Mengen von *möglichen Welten* befaßt: Ziehe zunächst alle möglichen Welten in Betracht, die den Beschränkungen der äußeren Umgebung genügen. Suche sodann in dieser Menge jene spezielle Welt, die den übrigen Beschränkungen des Ziels genügen und die Nutzenfunktion maxi-

mieren. Die Logik ist genau die gleiche, als setzten wir die durch das Ziel gegebenen Schranken und die Forderung nach Maximierung als neue „Naturgesetze“ den bestehenden, verkörpert in den Umgebungsbedingungen, an die Seite. Wir fragen einfach, welche Werte die Bestimmungsvariablen in einer Welt haben *würden*, die all diesen Bedingungen genügt, und folgern, daß dies die Werte sind, die die Bestimmungsvariablen haben *sollten*.

Nutzentheorie und statistische Entscheidungstheorie bilden also den logischen Rahmen für die rationale Auswahl unter gegebenen Möglichkeiten. Wir können somit auch den Technikbestand zur Bestimmung der optimalen unter den gegebenen Möglichkeiten für den Entwurf nutzen. Nun ist Entwerfen ein intellektueller Vorgang. Er unterliegt somit dem individuellen Wertesystem des Entwerfers und seiner *begrenzten Rationalität*. Ein vollkommen *rational* handelnder Entwerfer müßte über das Wissen aller Aspekte der Aufgabe verfügen. Er müßte die Güte aller möglichen Entwurfsalternativen bewerten und unter ihnen die *optimale* Alternative auswählen können. Seine physiologisch beschränkte „Kanalkapazität“ indes (siehe Abschnitt 2.1) schränkt seine Rationalität ein. Die begrenzte Rationalität und das Wissen um die endlichen Ressourcen regieren unser Entwurfsverhalten: Wir können nicht alle zulässigen Alternativen finden und ihre jeweiligen Vorzüge miteinander vergleichen. Noch könnten wir die beste Alternative sicher erkennen, selbst wenn wir sie früh fänden, da wir zum Erkennen des globalen Optimums alle lokalen Optima gesehen haben müßten.

In der klassischen Entscheidungstheorie geht es um die *Wahl* unter gegebenen Lösungen in einem kombinatorischen Entscheidungsraum. Das Ziel des Entwerfens dagegen liegt in der *Entdeckung* und Verfolgung von Alternativen. Die Lösungen zu einer Entwurfsaufgabe sind *a priori* unbekannt. Der Entwurf läßt sich folglich nicht als rationale Wahl unter gegebenen Lösungen modellieren. Das Entscheidungsproblem ist eine Frage der Ressourcen für Suche und Innovation. Eine erschöpfende Suche nach der optimalen Lösung ist wirklichkeitsfremd. Zwei kombinatorische Beispiele verdeutlichen dies: Wollten wir nach der Optimierungsmethode ein Schachprogramm entwerfen, müßten wir 10^{120} mögliche Partien simulieren [Simon, 1982, S. 137], was weder heute mit vorhandenen Rechnern noch morgen mit denkbaren zu schaffen ist. Die Suche nach dem „globalen Optimum“ im Falle des Handlungsreisenden wäre gleichsam aussichtslos: die Zahl der möglichen Rundreisen über n vorgegebene Orte ist gleich $n!$. Obwohl von immenser wirtschaftlichen Bedeutung, zum Beispiel bei der optimalen Standortbestimmung von Kraftwerken in Verbundnetzen, gibt es derzeit keinen Algorithmus mit vertretbaren Berechnungsaufwand für Wegenetze mit mehr als fünfzig Knoten.

Wegen unserer begrenzten Rationalität und der maximal verfügbaren Rechnerleistung müssen wir uns mit „guten“ oder „zufriedenstellenden“ Lösungen begnügen, optimale bleiben im Entwurf komplexer Artefakte ausgeschlossen. Die Natur des Entwerfens ist *nonoptimal*. Entwerfen heißt Suchen nach einer zufriedenstellenden Lösung. Bei unserer Suche im Entscheidungsraum leiten uns Faustregeln und Heuristiken. Auf diese Weise

können wir Lösungen entdecken, ohne zuvor den gesamten Entscheidungsraum durchsucht zu haben.

Logik des Entwerfens gleich Logik des Suchens

Jedes zielsuchende System ist mit seiner Umgebung durch zwei Arten von Kanälen verbunden: Durch *afferente*, das heißt sensorische Kanäle erhält es Information über die Umgebung, durch *efferente*, das heißt motorische Kanäle wirkt es auf die Umgebung ein. Das System muß über ein Gedächtnis verfügen, in dem die Information über Zustände der Welt speicherbar ist: sensorische wie auch motorische Information. Die Fähigkeit, Ziele zu erreichen, hängt ab von der Herstellung einfacher oder auch sehr komplexer Assoziationen zwischen bestimmten Zustandsänderungen der Welt und bestimmten Aktionen, die diese Änderungen herbeiführen.

Ein Beispiel: Abgesehen von wenigen angeborenen Reflexen, verfügt ein Neugeborenes über keine Grundlage, um seine Aktionen mit der sensorischen Information abzustimmen. Es muß erst lernen, daß bestimmte Aktionen oder Folgen von Aktionen bestimmte wahrnehmbare Zustandsänderungen der Welt verursachen. Bis zum Aufbau dieses Wissens sind Sinneswelt und motorische Welt zwei gänzlich verschiedene, unverbundene Welten. Erst wenn es Erfahrungen erworben hat, wie sich Elemente der einen auf Elemente der anderen beziehen, kann es zielgerichtet auf die Welt einwirken.

Auf dieses Modell des zielsuchenden Systems läßt sich das Suchproblem abbilden: Eine rational nicht zu erfassende Umgebung, der kombinatorische Suchraum aller theoretisch denkbaren Lösungspfade, wird *selektiv* durchsucht. Das Ziel: die Aktionsfolgen zu finden und zusammenzustellen, die von einer gegebenen zu einer erwünschten Situation führen. Welche Logik bestimmt eine derartige Suche? SIMON hält die Standardlogik für ausreichend: Der Entwerfer[2] bewegt sich in einem Labyrinth aus Knoten und Pfaden, wobei die Knoten die afferent beschriebenen Situationen darstellen und die verbindenden Pfade efferente Abfolgen, Aktionen, die eine vorgefundene Situation in eine andere überführen. Da nur ein zufriedenstellender, nicht aber optimaler Entwurf erreichbar ist, gilt die Suche Aktionen, die für das Entwurfsziel *hinreichend*, aber nicht unbedingt *notwendig* sind.

Es ist charakteristisch für die Suche nach Alternativen, daß sich der Suchpfad zum endgültigen Entwurf, die vollständige Aktionskette der Lösung, aus einer Folge von Teilpfaden zusammensetzt, den Teilaktionen. Die enorme Zahl der Alternativen geht auf die Zahl der möglichen Permutationen in den Handlungsfolgen zurück. Wir gewinnen viel, wenn wir statt der vollständigen Aktionskette die Teilhandlungen ins Auge fassen. Gewöhnlich gliedert sich die afferent betrachtete Situation in Komponenten, die sich zumindest annähernd

[2] SIMON bezieht die folgenden Ausführungen auf das KI-Programm „General Problem Solver“, mit dem er das Verhalten des problemlösenden Menschen zu simulieren versucht [Newell & Simon, 1972].

mit den Aktionskomponenten decken, die aus einer efferenten Aufgliederung der betrachteten Situation stammen. Wenn sich eine erwünschte Situation von der vorgefundenen durch die Differenzen $D_1, D_2, \ldots, D_n$ unterscheidet und wenn die Aktion A_i den Unterschied D_i beseitigt, dann kann die vorgefundene Situation in die erwünschte überführt werden durch die Aktionsfolge $A_1, A_2, \ldots, A_n$. Dieser Ansatz ist natürlich nur gültig, wenn die Aktionen unabhängig voneinander die Differenzen ausgleichen, wenn also die zu durchsuchende Welt *additiv* oder *aufgliederbar* ist. In diesem Fall lassen sich die Problemlösungstechniken der Mittel-Zweck-Analyse nutzbringend anwenden.

In der realen Welt des Entwerfens sind die Komponenten aber nur selten additiv: Aktionen haben Nebenwirkungen (können neue Unterschiede schaffen) und können zuweilen nur unternommen werden, wenn bestimmte Nebenbedingungen erfüllt sind (sie setzen die Beseitigung anderer Unterschiede voraus). Unter diesen Umständen ist es prinzipiell unbestimmt, ob eine Teilfolge von Aktionen, die gewisse Ziele erreichen, so fortgesetzt werden kann, daß alle Bedingungen erfüllt und alle Ziele erreicht werden, auch wenn es nur zufriedenstellende sind. Entwurfsverfahren, die lediglich Komponenten assemblieren, reichen nicht aus. Die passenden Zusammenstellungen der Teilaktionen müssen *gesucht* werden. SIMON nennt eine exploratorische Suchstrategie für nichtadditive Welten:

> „In carrying out such a search, it is often efficient to divide one's eggs among a number of baskets — that is, not to follow out one line until it succeeds completely or fails definitely but to begin to explore several tentative paths, continuing to pursue a few that look most promising at a given moment. If one of the active paths begins to look less promising, it may be replaced by another that had previously been assigned a lower priority.“ [Simon, 1982, S. 144]

Entwerfen kann schließlich auch als Verteilen von Ressourcen aufgefaßt werden: Ein Kriterium für zufriedenstellendes Entwerfen ist zum einen die Kostenminimierung der Entwurfsmittel, was immer schon ein Anliegen des Ingenieurs war, und zum anderen die Kostenminimierung des Entwurfsprozesses. Letztes ist von dringlicher Bedeutung. Es gilt, den Suchvorgang kostenminimal zu steuern. Unvollständige Suchwege müssen *bewertet* werden, wobei der Zweck der Bewertung als Anleitung zur Auswahl des nächsten Untersuchungsschritts zu verstehen ist. Erweist sich ein Weg als wenig aussichtsreich, so sind nur die Untersuchungskosten verloren. Das enttäuschende Ergebnis braucht nicht hingenommen, ein anderer Weg kann statt dessen eingeschlagen werden. Suchprozesse, so verstanden, sind Prozesse zur Beschaffung von Information über die Problemstruktur. Strukturinformation ist wertvoll für das effziente Entdecken einer Lösung.

Die Struktur komplexer Artefakte und die Entwurfsorganisation

Die Architektur des Komplexen ist die Hierarchie. Das Bild von der „Schachtel-in-der-Schachtel“ beschreibt den Grundgedanken: Die Komponenten des Systems führen verschiedene Teilfunktionen aus, die alle zur Systemfunktion beitragen. So wie die „innere Umgebung“ des ganzen Systems durch seine Funktionen beschreibbar ist, ohne auf die Mechanismen einzugehen, so kann auch die innere Umgebung eines jeden Teilsystems durch die Beschreibung der Funktionen dieses Teilsystems definiert werden, wiederum ohne deren Teilmechanismen zu detaillieren. Die Zerlegung hierarchischer Systeme in beinahe unabhängige Komponenten, die den Teilfunktionen entsprechen, ist eine leistungsstarke Technik im Entwurf komplexer Strukturen (siehe Abschnitt 2.2). Die einzelnen Komponenten lassen sich dann relativ unabhängig voneinander entwerfen. Jede wird auf die anderen Komponenten hauptsächlich über ihre Funktion einwirken, unabhängig von den Details der Mechanismen, die sie tragen.

Nun ist die Zerlegung eines vollständigen Entwurfs in seine funktionalen Bestandteile in der Regel mehrdeutig. Eine Möglichkeit, Zerlegungen zu untersuchen und zugleich die real immer vorhandenen schwachen Wechselwirkungen der Komponenten zu berücksichtigen, bietet der *Generator-Test-Zyklus*: Ihm liegt die kybernetische Vorstellung zugrunde, daß sich der Entwurfsvorgang zunächst mit der Erzeugung von Alternativen befaßt und anschließend diese unter den Auflagen und Einschränkungen der Aufgabe testet: Bild 5.3. Der Generator bestimmt indirekt die Zerlegung des Entwurfsproblems. Der Test garantiert, daß Nebenwirkungen bemerkt und gewichtet werden. Verschiedene Zerlegungen sind gleichbedeutend mit verschiedenen Aufteilungen der Verantwortung zwischen Generator und Test.

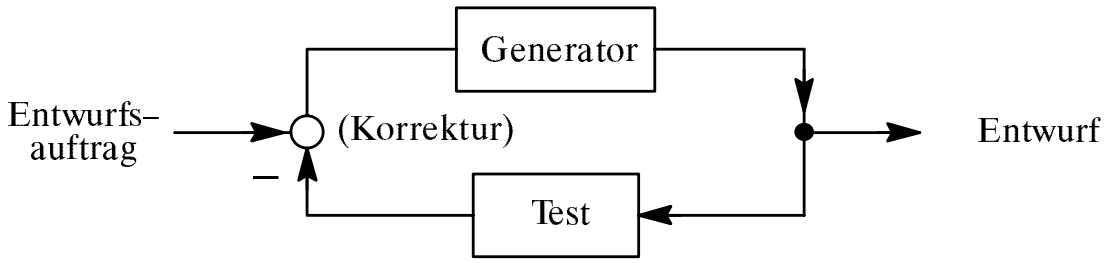


Bild 5.3: Entwerfen als Generator-Test-Zyklus

Weiterhin spielen Fragen des Vorrangs und der Abfolge im Entwurfsprozeß eine zentrale Rolle: Wie weit soll die Entwicklung möglicher Teilentwürfe vorangetrieben werden, bevor der koordinierende Gesamtentwurf detailliert entwickelt wird? Oder umgekehrt: Wie weit

muß der globale Entwurf gediehen sein, bevor die Komponenten entwickelt werden können? Mit diesen Alternativen des Entwerfens sind Architekten, Komponisten und Programmierer bestens vertraut. Die letzten zum Beispiel, wenn sie sich von der globalen Struktur eines Programms zu den Unterprogrammen hinunter-, oder von den Unterprogrammen zum koordinierenden Hauptprogramm hinaufarbeiten. Eine Theorie des Entwerfens wird Prinzipien zur Entscheidung organisatorischer Fragen im Entwurfsprozeß enthalten — nur wenige existieren bereits.

Organisation und Arbeitsteilung im Generator-Test-Zyklus beeinflussen nicht nur die Effizienz der Ressourcen, sondern auch die Form des endgültigen Entwurfs. Der Entwurfsstil geht nicht allein auf die Entscheidungen im Generator-Test-Zyklus zurück. Stilbildend sind auch die Gewichtungen in der Zielsetzung des zu entwerfenden Artefakts: Ein Architekt, der Bauten von außen nach innen entwirft, wird zu ganz anderen Gebäuden gelangen als einer, der von innen nach außen entwirft. Dies auch, wenn sich beide über die Eigenschaften einig sind, die ein zufriedenstellendes Gebäude aufweisen sollte. Sowohl die Struktur des zu entwerfenden Artefakts als auch Stil und Organisation des Entwerfens sind wesentliche Bestandteile einer Entwurfslehre.

Ein weiterer Bestandteil ist der Einfluß der *Repräsentation* auf das Entwerfen: Daß die Problemrepräsentation einen Unterschied macht, läßt sich zeigen: So sind wir zum Beispiel der Meinung, daß die Arithmetik durch die Einführung des Stellenwertsystems und der arabischen Ziffern anstelle der römischen einfacher geworden sei — eine theoretische Erklärung steht aber noch aus. Zum anderen lebt die Mathematik von der Repräsentation: In ihren Schlußfolgerungen zeigt sie nur das, was in den Prämissen bereits inbegriffen ist. Jede mathematische Ableitung kann einfach als Veränderung der Repräsentation betrachtet werden, die offenbart, was zuvor schon wahr, aber verborgen gewesen ist. Wir können diese Betrachtung auf alle Arten des Problemlösens verallgemeinern: Ein Problem zu lösen, bedeutet demnach, es so darzustellen, daß die Lösung transparent wird. Eine Theorie alternativer Repräsentationen liegt noch nicht vor. Sie wäre sicherlich ein wichtiger Bestandteil der Entwurfstheorie.

Abschließend zu den Ausführungen einer Wissenschaft des Entwerfens soll sein prominentester Verfechter noch einmal zu Wort kommen:

> „The proper study of mankind has been said to be man. But I have argued that man — or at least the intellective component of man — may be relatively simple, that most of the complexity of his behavior my be drawn from man's environment, from man's search for good designs. If I have made my case, then we can conclude that, in large part, the proper study of mankind is the science of design, not only as the professional component of a technical education but as a core discipline for every liberally educated person.“ [Simon, 1982, S. 159]

5.1.2 Der Beitrag des Objekt-Paradigmas

Was trägt das Objekt-Paradigma zu einer Wissenschaft des Entwerfens bei? Zunächst einmal das Modulkonzept. Wir können „Softwaremodul“ als Synonym für SIMONs „Artefakt“ setzen: Modul und Artefakt sind künstliche Objekte, Ergebnisse eines vom Entwerfer vollzogenen *Problemschnitts*. Die Schnittstelle trennt die innere von der äußeren Umgebung. Im Schnitt liegt sowohl die Bedeutung des Artefakts als auch die Bedeutung des Moduls. Das SIMONsche Tripel Umgebung-Zweck-Beschaffenheit spiegelt sich im Client-Server-Vertrag des Klassenkonzepts wider: Der Vertrag fixiert den *Zweck* der Dienstleistung. Der Client bestimmt die *Umgebung*, die von der Dienstleistung profitieren soll. Und der Server steht für die *Beschaffenheit* der Dienstleistung, das heißt für die Realisierung. Überdies vertieft das Kapselungsprinzip einer Klasse den Problemschnitt: Datenstrukturen und Operationen bilden eine semantisch und technisch autonome Einheit. Nach der These von SIMON konstituiert sich Komplexität in der äußeren Umgebung eines Artefakts. Von dessen Beschaffenheit, der inneren Umgebung, kann abstrahiert werden, wenn die Zweckbestimmung bekannt ist. Nichts anderes verfolgt das Klassenkonzept: Die tatsächliche Implementierung einer Klasse ist für deren Einsatz in komplexen Software-Umgebungen irrelevant (Laufzeitverhalten und Speicherausnutzung lassen sich der Zweckbestimmung unterordnen). Aus dem Blickwinkel des Entwerfers tragen die Klasseninterna nicht zur Entwurfskomplexität bei. Auf dieser Eigenschaft beruht letztlich das Konzept der Klassenbibliothek: die Wiederverwendbarkeit der Software-Artefakte definiert sich allein durch den Zweck und die Einsatzumgebung.

Der zweite wesentliche Beitrag der Objektorientierung ist die hypotaktische Ordnungsform: Vererbung. SIMONs „Architektur des Komplexen“ und sein Prinzip der „Beinahen-Zerlegbarkeit“ finden sich in der Struktur und den Eigenschaften der Vererbungsnetze wieder. So wie sich spezialisierte Klassen von generalisierten ableiten (in der Regel evolutionär von einer Wurzelklasse zu einer Blattklasse), so entwickelt sich nach SIMON Komplexes evolutionär aus Einfachem. Auf der Objektebene — bei SIMON ist das die Ebene der Komponenten, die „stabile Zwischenzustände“ verkörpern — können wir gleichermaßen zwischen schwachen Inter- und starken Intra-Bindungskräften unterscheiden. Auch im objektorientierten Entwurf streben wir eine lose Kopplung zwischen und eine starke Kohäsion in den Objekten an, um die ökonomische Forderung nach Erweiterbarkeit und Wartbarkeit zu erfüllen.

Eine dritte Parallele zwischen dem Objekt-Paradigma und einer allgemeinen Entwurfstheorie liegt in der Repräsentation. Fordert SIMON noch eine geordnete Vielfalt an Repräsentationsformen (eine *Taxonomie*), ohne diese im Detail zu spezifizieren, so verfügt die Objektorientierung bereits über eine Fülle unterschiedlicher Darstellungen, je nach dem Kontext ihrer Anwendung: (a) In der Softwaretechnik reicht die Palette der Darstellungen von der natürlichen Sprache des Kunden bis zur formalen des Programmierers, wobei für die Beschreibung der Objektschemata und der dynamischen Modelle zahlreiche grafische

Notationen zum Tragen kommen, wie Struktogramme und PETRI-Netze. (b) In der Hardwaretechnik lebt der Entwurf förmlich von seinen Repräsentationen: siehe die Sichten und Ebenen des Entwurfsraums nach GAJSKI auf Seite 31 und das entsprechende Y-Diagramm auf Seite 127. Hinzu kommt der rechnergestützte Entwurf (CAD), der komplexe räumliche Darstellungen ermöglicht.

Die aufgezeigten Gemeinsamkeiten waren technischer Art: Modularisierung, Hierarchie und Repräsentation. Es ließen sich sicherlich einige mehr aufzeigen, auch in den nichttechnischen Aspekten des Objekt-Paradigmas: so zum Beispiel der ökonomische Entwurf, die gemeinsame Begriffswelt der Entwerfer oder die „Logik" des Stöberns durch Klassenbibliotheken (*class browsing*) als Analogon zur Logik des Suchens. Wir wollen es aber hiermit bewenden lassen und die Thesen eines weiteren prominenten Verfechters einer allgemeinen Entwurfsdisziplin vorstellen: Heinz ZEMANEKs „Gedanken zu einer Theorie des Entwurfs" [Zemanek, 1992].

5.2 Der architektonische Entwurf

Zwei Ordnungsbegriffe prägen die Technik unseres Jahrhunderts: *System* und *Architektur*. Beide sind verwurzelt im heute dominierenden Begriff der *Information*: Das System wird zum Ganzen durch die Information; sie verknüpft das System mit der Umwelt. Die Architektur ist „systemgewordene Ordnung und Disziplin" [Zemanek, 1991, S. 311]. Eine Theorie der Systeme, über deren Strukturen und Verhaltensweisen, liegt in Ansätzen vor: in der Biologie durch Ludwig von BERTALANFFY und in der Nachrichtentechnik durch Karl KÜPFMÜLLER [Bertalanffy, 1973; Küpfmüller, 1990]. Die Informatik hat sie übernommen, wenn auch „ohne beeindruckende Systematik" [Zemanek, 1991, S. 313]. Eine Theorie der *verallgemeinerten Architektur* steht allerdings noch aus. Und es ist der Architekturbegriff, dem ZEMANEK den größten Stellenwert in einer zu schaffenden Entwurfstheorie beimißt. Wir wollen seine architektonische Leitidee skizzieren und sie dem interdisziplinären Anspruch der Objektorientierung gegenüberstellen.

Das Bauwesen ist die einzige technische Disziplin, die bisher *Entwurf* und *Ausführung* fachlich und ausbildungsmäßig trennt: in Architektur (Planungs- und Entwurfswesen) und Bauingenieurwesen (technische Konstruktion). In der Informatik ist der Begriff der Architektur zur Worthülse verkümmert. Er hat seine ursprüngliche Bedeutung eingebüßt — die architektonische *Gestaltung* eines Rechners. ZEMANEK zitiert die richtungsweisende Erstdefinition von Frederick P. BROOKS, einem der Architekten des IBM-Systems 360:[3]

[3] Es war das IBM-Konzept der „Rechnerfamilie", der Übergang vom Einzelentwurf zum Entwurf einer ganzen Rechnerpalette, aus dem sich der Architekturbegriff entwickelte: Rechner gleicher Architektur, aber unterschiedlicher Leistung. Das Familienkonzept erlaubte zum erstenmal den Entwurf hardwareverträglicher Software: Programme, auf einem Rechner der Familie entwickelt, liefen auch auf allen übrigen.

> „Computer Architecture, like any other architecture, is the art of determining the needs of the user of a structure and then designing it to meet those needs as effectively as possible within the economic and technological constraints.“ (zitiert in [Zemanek, 1992, S. 127])

Von dieser benutzerorientierten Definition ausgehend, verallgemeinert ZEMANEK den Architekturbegriff auf den Entwurf beliebiger technischer Systeme. Sein Hauptanliegen gilt der im klassischen Sinne des Wortes *universitären* Ausbildung zum „Systemarchitekten“, einem Schlüsselberuf der Informationstechnik von morgen.[4]

5.2.1 Ideale einer verallgemeinerten Entwurfslehre

Für den Brückenschlag von der Architektur der Gebäude zur Architektur von Soft- und Hardware gibt es auf der Seite der Gebäude-Architektur kaum ein theoretisches Fundament. Bis heute überwiegen in der architektonischen Literatur die Bilder(bücher). Grundsatzgedanken zu einer Theorie der Architektur sind äußerst spärlich formuliert. Zwei Ausnahmen hat ZEMANEK aufgetan: zum einen VITRUV, einen Baumeister, der in den Diensten von Julius CAESAR und AUGUSTUS stand und sich durch seine Lehre, in zehn Büchern beschrieben, als erster Theoretiker der Architektur auszeichnete. Und zum anderen Christopher ALEXANDER, der mit seinem Buch *Notes on the Synthesis of Form* [Alexander, 1964] die Linie von VITRUVs erstem Buch „in bescheidenem Umfang“, so ZEMANEK, weitergeführt hat. Die Lehre VITRUVs läßt sich wirksam auf die Rechnerarchitektur anwenden und ist die „Basis jeder Universalarchitektur“:

> „Und über die Verbindung zwischen VITRUV und der Computerarchitektur habe ich eine Vermutung. Gerrit BLAAUW, einer der drei Architekten des IBM-Systems — der dann den Beitrag für die *Elektronischen Rechenanlagen* [Blaauw, 1972] schrieb[5] — war ein Schüler von Professor van WIJNGAARDEN, der VITRUV gut kannte.“ [Zemanek, 1992, S. 130]

Als ersten Ansatz einer verallgemeinerten Architektur zitiert ZEMANEK die VITRUVschen Forderungen an die Ausbildung eines Architekten der Antike: Bild 5.4.

Natürlich ist dies ein gewaltiges und zugleich unrealistisches Programm, wie ZEMANEK selbst einschränkt: „Es kann mit Sicherheit angenommen werden, daß VITRUV ein Diplom in Informatik nicht als ausreichend angesehen hätte, um den Entwurf von Programmen anzugehen, um als Software-Architekt zu arbeiten“ [Zemanek, 1992, S. 131]. In unsere Gegenwartssprache übertragen, könnten die VITRUVschen Forderungen an ein neuzeitliches Curriculum so lauten, wie es Bild 5.5 zeigt. Wiederum ein Ideal, das man unter den heutigen Studienbedingungen nicht erreichen kann, aber anstreben sollte.

[4] Vergleiche mit der aktuellen Forderung nach einer „Informatik-Systemtechnik“ (siehe Abschnitt 11.3).
[5] Siehe auch die Anmerkung in der vorletzten Fußnote.

„Er muß umfassend gebildet sein,
er muß im Schriftlichen gewandt sein,
des Zeichenstiftes kundig,
in Geometrie, Arithmetik und Optik ausgebildet,
er muß viel Geschichte kennen,
fleißig die Philosophen studiert haben, so daß er
hohe Gesinnung hat und nicht anmaßend wird,
umgänglich, rechtschaffen und vor allem ohne Habgier ist,
er muß Musik verstehen und
darf nicht unbewandert sein in der Medizin,
auch die Rechtsvorschriften muß er kennen,
mit der Sternenkunde vertraut sein.“ [Zemanek, 1992, S. 131]

Bild 5.4: VITRUVs Ausbildungsideale der Architektur

Eine Ausbildung zum Generalarchitekten sollte von diesen Idealen geleitet sein. Unter den technischen und biologischen Systemen (Gebäuden und Organismen) ist es der Rechner, der die höchsten Ansprüche stellt. Die drei Eigenschaften, kennzeichnend für Systeme, — Dynamik, Komplikation und Abstraktion — finden sich geschlossen nur in der Rechnertechnik wieder: Tabelle 5.1. Eine verallgemeinerte abstrakte Architektur wird sich folglich an den Belangen des Rechner- und Programmentwurfs orientieren.

	Gebäude	*Rechner*	*Organismus*
dynamisch	nein	ja	ja
kompliziert	nein	ja	ja
formal/abstrakt	nein	ja	nein

Tabelle 5.1: Grundfelder der Architektur und ihre Eigenschaften

5.2.2 Der gute Entwurf aus architektonischer Sicht

Was macht einen Entwurf zum guten Entwurf, eine Architektur zu einer guten Architektur? VITRUV, der erste Architektur-Theoretiker, BLAAUW, einer der ersten Software-Architekten, und ZEMANEK, neben SIMON einer der ersten Theoretiker einer universellen Entwurfslehre, nennen die Hauptzüge:

„Der Architekt muß

- eine humanistische Erziehung haben, denn sein Entwurf muß auch in seinem nichttechnischen Umfeld bestehen,
- formale Methoden beherrschen, denn wenn andere seinen Entwurf ins Formale übersetzen, wird das kaum eine Verbesserung bringen,
- die notwendigen Berechnungen beherrschen, denn Daumenregeln werden nicht genügen — noch besser wäre, auch die Simulation zu beherrschen und sie beim Entwurfsprozeß anzuwenden,
- er muß Physik studiert haben, denn seine abstrakten Funktionen werden letzten Endes in der physischen Welt realisiert,
- eine Menge Geschichte kennen, weil jedes lebende Wesen und jedes technische Objekt auf seiner Entwicklung beruht und einen Teil seiner Geschichte mit sich herumträgt,
- er muß der Philosophie und der Philologie mit Aufmerksamkeit gefolgt sein, weil sowohl der Entwurf wie auch seine Beschreibung und seine Dokumentation ein Sprachproblem sind,
- er muß medizinische Kenntnisse haben, weil sein Entwurf physiologische und psychologische Folgen haben wird, vielleicht allen möglichen Leuten auf die Nerven geht,
- er muß die Sicht der Juristen kennen, weil Juristen Entscheidungen über Herstellung, Kauf, Einrichtung und Benützung treffen werden (außerdem werden Juristen als Benutzer auftreten und unangenehmer als andere werden, weil ihre Logik nicht die Logik der Architekten ist),
- er muß Philosophie — mindestens die Philosophie der Naturwissenschaften — studiert haben, so daß er die Grenzen seines Könnens und des Könnens des Computers abschätzen kann.“ [Zemanek, 1992, S. 131f.]

Bild 5.5: VITRUVs Ausbildungsideale in der Übertragung von ZEMANEK

- VITRUV über die Eigenschaften einer guten Gebäude-Architektur:
  - Ordnung: Maß und Proportion
  - Anordnung: Struktur
  - Eurhythmie: Ebenmäßigkeit und Schönheit
  - Symmetrie: Übereinstimmung und Zueinanderpassen
  - Angemessenheit: Grundsätze und Vorschriften
  - Ökonomie: verschwendungsfreier Umgang mit Material und Raum

- BLAAUW über die Eigenschaften einer guten Software-Architektur:

  - Vollständigkeit:
    Wenn ein Konzept aufgenommen wird, dann vollständig.
  - Allgemeinheit:
    Wenn ein Konzept aufgenommen wird, dann in seiner allgemeinsten Form.
  - Offenheit: Entwurfsfreiraum lassen.
  - Orthogonalität:
    Wenn ein Konzept aufgenommen wird, dann losgelöst von anderen.
  - Klarheit:
    Wenn ein Konzept aufgenommen wird, dann einsichtig und transparent.
  - Sicherheit: gegen natürliche und willkürliche Störungen.
  - Wirtschaftlichkeit: gutes Verhältnis zwischen Preis und Leistung.
  - Effizienz: gute Ausnützung der Mittel.
  - Umweltschonung: Vermeidung von Abfällen aller Art.

- ZEMANEK über das Wesen einer verallgemeinerten Architektur:

  - Gute Architektur ist konsistent:
    Sie wird nach allgemeinen Grundsätzen und festgelegten Vorschriften gestaltet; sie vermeidet Ausnahmen und mutwillige Ausschmückungen, einseitige Verbesserungen an einzelnen Stellen.
  - Gute Architektur hat einheitliche Ordnung:
    Es gibt Maße und Proportionen. Sie hat Struktur; sie ist wohlgegliedert und nichts erscheint an überraschender Stelle.
  - Gute Architektur ist redundant:
    Die Redundanz guter Architektur ist nützliche und nicht langweilige oder erbitternde Redundanz. Sie gibt dem Betrachter und dem Benutzer das Gefühl der Vertrautheit. Wenn er einen Teil der Architektur kennengelernt hat, ist er imstande, den Rest weitgehend vorwegzunehmen; er muß nur jeweils dazulernen, was für die bestimmte Stelle notwendig ist.

Entwerfen im Sinne von ZEMANEK ist ein ständiges „Ringen um Kompromisse aller Art“. Es ist damit auch im Sinne von SIMON ein Suchen nach der „zufriedenstellenden“ Lösung.

5.2.3 Der Beitrag des Objekt-Paradigmas

Was trägt das Objekt-Paradigma zur Verwirklichung der architektonischen Ideale bei? Zunächst einmal erfüllt es die Kriterien an eine gute Software-Architektur nach BLAAUW. In der 20-jährigen Tradition der Softwaretechnik stehend, sind es gerade die BLAAUWschen Forderungen, zu deren konsequenter Erfüllung das Objekt-Paradigma aus der Taufe gehoben wurde. Vollständig, allgemein, offen, orthogonal und klar sollen sie sein, die objektorientierten Klassen. Die anderen Eigenschaften, wie sicher, wirtschaftlich, effizient und umweltschonend, sind traditionelle Werte der Ingenieurkunst.

Was heißt nun „vollständig"? Nicht eine Auswahl von Diensten, sondern alle Dienste einer Klasse werden angeboten: Wenn eine Klasse in eine Klassenbibliothek aufgenommen wird, dann vollständig, das heißt für den allgemeinsten Fall ihrer Anwendung. Wenn zum Beispiel die Datenstruktur der mathematischen Menge aufgenommen werden soll, dann mit allen denkbaren Mengenoperatoren. Was heißt nun „allgemein"? Klassenbibliotheken sind Sammlungen verallgemeinerter Klassen. Auf das spezielle Problem zugeschnitten, werden allgemeine Bibliotheksklassen erst vom Kunden, bekanntlich durch Erweitern oder Überschreiben mit neuen Eigenschaften. Was heißt nun „offen"? Der Freiraum des Klassenbenutzers wird durch die Klassenspezifikation nur grob abgesteckt, nicht aber eingeschränkt: Die Klassen können individuell ausgestaltet werden. Was heißt nun „orthogonal"? Die Datenkapselung vermeidet unvorhergesehene Kopplungen zwischen den Objekten eines Programms (Nebenwirkungen). Sie approximiert das Ideal der lokalen Wartbarkeit. Was heißt schließlich „klar"? Wenn eine Klasse in eine Bibilothek aufgenommen wurde, so ist sie in einer wohldefinierten Vererbungsstruktur und damit in deren Semantik eingebettet. Einsichtigkeit und Transparenz motivieren das Konzept der abstrakten Klasse: Ihre Ausformulierung bleibt dem Benutzer verborgen. Analog zum „virtuellen Speicher", „der dem Benutzer unbegrenzte Speicherkraft vortäuscht" [Zemanek, 1992, S. 161], suggeriert ihm die abstrakte, auch als virtuell bezeichnete Klasse unbegrenzte Implementierungskraft.

Wie unterstützt nun das Objekt-Paradigma den guten Entwurf in der Definition nach VITRUV und ZEMANEK? Den interdisziplinären Anspruch teilen sich beide: Die Objektorientierung behauptet von sich, interdisziplinär zu sein, und der architektonische Entwurf setzt eine interdisziplinäre Ausbildung voraus. Im Abschnitt 5.3 nennen wir einige philosophische Argumente für die Güte des objektorientierten Entwurfs. An dieser Stelle beschränken wir uns auf die objektorientierten Parallelen zu den Wesenszügen einer guten Architektur nach ZEMANEK: Gute Architektur ist konsistent, hat einheitliche Ordnung und ist redundant.

Zur ersten Forderung „Gute Architektur ist konsistent": Die Grundsätze und Vorschriften, die einen konsistenten Entwurf definieren, spiegeln sich an den objektorientierten Konzepten, den Client-Server-Verträgen, Vererbungsregeln, Integritätsbedingungen und Zusicherungen. Überdies können sie auf eine ontologische Fundierung verweisen, das heißt, die

Rechtfertigung für Grundsätze und Vorschriften ist auch philosophisch einsichtig (siehe das ontologische Konzept des *Gesetzes* im Anhang B). „Ausnahmen und mutwillige Ausschmückungen, einseitige Verbesserungen an einzelnen Stellen“ waren kennzeichnend für die Softwarekrise Anfang der 70er-Jahre. Aus dieser entwickelte sich die strukturierte Vorgehensweise, welche gegenwärtig durch die objektorientierte weitergeführt wird.

Zur zweiten Forderung „Gute Architektur hat einheitliche Ordnung“: Mit dieser Forderung laufen wir offene Türen in der objektorientierten Welt ein. Das Maß aller Objekte ist dort die Klasse, jedes Objekt ist klassifiziert. Die Datenkapsel umschließt die WIRTHsche Gleichung „Programme = Datenstrukturen + Algorithmen“ [Wirth, 1983]. Die Summanden der Gleichung sind semantisch und technisch eins. „Nichts erscheint an überraschender Stelle“: Alle Objekte entstehen nur auf explizite Anforderung hin, sind wohlstrukturiert und werden individuell ausgestaltet. Und alle Objekte sind eingebunden in einer paratktischen und/oder hypotaktischen Ordnung. Die Aggregation ist dabei die Form der objektorientierten Parataxe (Nebenordnung), die Generalisierung/Spezialisierung die Form der Hypotaxe (Unterordnung). Klassenbibliotheken sind der Ort der geforderten „einheitlichen Ordnung“.

Zur dritten Forderung „Gute Architektur ist redundant“: Das Prinzip der Vertrautheit, verstanden als „nützliche und nicht langweilige oder erbitternde Redundanz“ spiegelt sich im Prinzip der objektorientierten Wiederverwendung. Das Ableiten neuer aus alten Eigenschaften, das Vererben, und das Verfügen über beliebig viele schablonierte Exemplare, über Objekte, die struktur- und verhaltenskonform aus ihren Klassen hervorgehen, wirken *vertrauensfördend*. Es sind Vorgehensweisen, die auf Bewährtes aufbauen und so der Unsicherheit im Entwurf entgegenwirken. Die Räder der Objektorientierung — die Klassen — werden nicht immer wieder neu erfunden: Das Expertenwissen erfahrener Programmierer steht dem gelegentlichen Programmierer übersichtlich geordnet (Klassenbibliotheken) und transparent zur Verfügung (das Verständnis der Klassenschnittstelle genügt im allgemeinen). Wenige, aber ausdrucksvolle Konzepte, wie Datenabstraktion und Vererbung, stehen repräsentativ für das Objekt-Paradigma (siehe zum Beispiel [Booch, 1991; Meyer, 1988]). Diese Konzepte sind *intuitiv* zugänglich, aber auch *formal* beschreibbar. Das führt uns auf die ontologische Darstellung des Entwurfsprozesses nach Yair WAND:

5.3 Die Ontologie des Entwerfens

Die ontologische Sicht ist die Sicht der Objektorientierung. Das folgende bedarf also nicht der expliziten Parallele, es *ist* die formalisierte Beschreibung des objektorientierten Entwurfs schlechthin. Yair WAND kommt das Verdienst zu, die Gemeinsamkeit der wissenschaftlichen Gemeinschaft der Entwerfer auf eine philosophische Grundlage gestellt zu haben [Wand, 1989a; Wand, 1989b; Wand & Weber, 1989]. Im Anhang B geben wir einen

Exkurs zur ontologischen Analogie objektorientierter Grundprinzipien. Hier skizzieren wir die Leitideen WANDs zu einer ontologisch begründeten Entwurfstheorie für Informationssysteme.

Entwurfsmethoden gibt es zuhauf: „It has been claimed that hundreds, if not thousands of methodologies for information systems development exist“ [Wand, 1989b, S. 201]. Dennoch gibt es keinen Konsens über die *Vergleichbarkeit* der Methoden. Was eignet sich besser für die Anforderungsanalyse: *Prozesse*, *Datenflüsse* oder *Objekte*? Welcher Maßstab sollte für den „Besser“-Vergleich angelegt werden? Den größten gemeinsamen Nenner finden wir in der ontologischen Betrachtung: Einerseits sollen Informationssysteme realexistierende Systeme modellieren, so wie der Mensch sie wahrnimmt. Andererseits ist es gerade die Ontologie, die sich mit der Modellierung der realen Dinge in der Welt befaßt. Um nun die Merkmale eines Weltausschnitts einzufangen, bedarf es eines Maßes für *Wiedergabetreue*: Das implementierte System soll die „Essenz“ (wieder im BROOKSschen Sinne) des wahrgenommenen Weltausschnitts *getreu* wiedergeben. In allen Transformationen während der Entwicklung sollte es unveränderliche Größen — *Invarianten* — geben, welche die Essenz des Weltausschnitts tragen und deren Erhalt die höchste Priorität im Entwurf zukommt. Ein wiedergabegetreues Informationssystem, wenn es seinem Anspruch gerecht wird, repräsentiert die Invarianten unabhängig von der gewählten Entwurfsmethode und unabhängig von seiner technischen Realisierung. Was sind nun die Invarianten eines „guten“ Entwurfs? Die Ontologie erlaubt ihre Formulierung.

5.3.1 Tiefenstruktur: Zustände, Ereignisse, Gesetze

Der Entwurf schreitet in drei transformatorischen Phasen voran: Analyse, Design und Implementierung. Die Transformationen seien wie folgt definiert:

1. Analyse

   Die menschliche Wahrnehmung (Perzeption) der Wirklichkeit wird in ein *formales* Modell der wahrgenommenen Wirklichkeit transformiert. Das resultierende Modell wird im allgemeinen als konzeptionell bezeichnet, im weiteren heiße es Weltmodell.

2. Design

   Das Weltmodell wird in ein Modell des Informationssystems transformiert. Dieses wird durch eine sprachliche Konstruktion beschrieben, die sich direkt auf die Implementierung abbilden läßt.

3. Implementierung

   Das Modell des Informationssystems wird in ein implementierbares (rechnerausführbares) Modell transformiert. Ein informationstechnisches Artefakt wird geschaffen.

Wir interessieren uns hier nur für die Transformationen in der Analyse- und Designphase. Die ontologische Modellierung dieser Phasen[6] soll uns normative Regeln für einen „guten" Entwurf an die Hand geben. Da wir den Einfluß der technischen Implementierung und der Zweckbestimmung auf den Entwurf ausklammern, können die Bedingungen an einen guten Entwurf nur notwendig sein. Mit Blick auf das Entwurfsprodukt sind sie sicherlich unzureichend. WAND bezeichnet den Aspekt der essenztragenden Entwurfsinvarianten als „Tiefenstruktur" des Informationssystems und den zweck- und implementierungsbedingten Schnittstellenaspekt als „Oberflächenstruktur". Das ontologische Modell des Entwerfens zielt ausschließlich auf den ersten Aspekt, akzidenstragende Benutzungsoberflächen und Betriebsmittel bleiben außen vor. Die ontologischen Begriffe von „Zustand", „Ereignis" und „Gesetz", beschrieben im Anhang B, genügen, um die Invarianten der Tiefenstruktur formal zu beschreiben.

Das Schema eines Systems, einerlei ob Weltmodell oder Artefakt, kann als ein Tripel beschieben werden: $< S, L, E >$. S steht für die Menge aller möglichen Zustände (*states*), die das System und seine Teilsysteme einnehmen kann, L für die Menge der Systemgesetze (*laws*), E für die Menge der relevanten externen Ereignisse (*events*). Das Adjektiv „relevant" schränkt die Menge aller möglichen Ereignisse so weit ein, daß die instabilen Zustände, hervorgerufenen durch externe Ereignisse, mit wenigen Systemgesetzen in stabile Zustände überführt werden können. „Stabil" heißen Zustände, wenn sie Fixpunkte eines Gesetzes sind: $L(s) = s$; für „instabile" Zustände gilt also: $L(s) \neq s$.

Auch die Dynamik eines Systems läßt sich mit dem Tripel $< S, L, E >$ formalisieren: Angenommen, das System befinde sich im stabilen Zustand s_1. Die Umgebung zwinge es nun, seinen Zustand zu wechseln. Diese Zustandsänderung stellt ein externes Ereignis dar. Es ist keine Zustandsänderung der Umgebung, sondern eine des Systems. Das externe Ereignis bedinge den neuen Zustand $\tilde{s}$. Angenommen, dieser neue Zustand sei instabil, dann werden die Gesetze des Systems es zwingen, einen stabilen Folgezustand s_2 einzunehmen.[7] Der Übergang $\tilde{s} \rightarrow s_2$ wird als Systemantwort auf das externe Ereignis interpretiert. Die Systemantwort ist ein Zustandswechsel, somit ein Ereignis. Ein solches soll internes Ereignis heißen, da es nur von den Systemgesetzen abhängt, nicht aber von der Umgebung. Die Dynamik eines Systems ist also insgesamt als Ereignisfolge beschreibbar: auf das externe Ereignis folgt die Systemantwort (*response*). Mit $e =< s_1, \tilde{s} >$ als externem und $r =< \tilde{s}, s_2 >$ als internem Ereignis läßt sich sich die Systemdynamik formalisieren:

$$\boxed{s_1 \Rightarrow e \Rightarrow \tilde{s} \Rightarrow r \Rightarrow s_2}$$

[6] Im folgenden setzen wir „Analyse" und „Design" mit „Entwurf" gleich. Wir folgen damit der Tendenz objektorientierter Methoden, zwischen Analyse und Design nicht mehr strikt zu unterscheiden (so zum Beispiel die Entwurfsmethode nach Grady BOOCH [Booch, 1991]).

[7] Ein Gesetz umfaßt zwei Arten der Information: Eine *Bedingung*, ob ein Zustand stabil ist, und eine *Regel*, wie sich ein instabiler Zustand in einen stabilen ändern soll. Ein Gesetz ist somit die kompakte Schreibweise für systeminterne Interaktionen.

Diese ontologischen Konzepte können gleichermaßen auf reale Systeme und deren Repräsentation als Informationssysteme angewandt werden. Insbesondere erlauben sie die formale Definition der notwendigen Bedingungen für einen „guten", das heißt „wiedergabegetreuen" Entwurf.

5.3.2 Der gute Entwurf aus ontologischer Sicht

Der gute Entwurf zielt auf die Konstruktion eines Artefakts, das die relevanten Eigenschaften des betrachteten Weltausschnitts — Struktur- und Verhaltensmuster — verzerrungsfrei wiedergibt. WAND definiert die notwendige Bedingung für Wiedergabetreue:

> „An information system is a *good representation* only if its sequence of states in time reflects the sequence of states the real system goes or may go through. We will say that *an information system is a state tracking mechanism.*" [Wand & Weber, 1989, S. 209]

Vier Bedingungen sind notwendig und hinreichend, um diese Definition zu erfüllen. Um die Bedingungen zu formalisieren, beschreiben wir Informationssysteme analog zum allgemeinen Fall: Der Unterscheidung wegen verwenden wir andere Bezeichner: anstelle des Tripels $< S, L, E >$ das Tripel $< M, P, T >$. M bezeichnet die Menge der möglichen Zustände (M wie „master", gemeint sind Dateien oder Datensätze), P die Gesetze[8] des Informationssystems (P wie „Prozesse") und T die relevanten externen Ereignisse im Informationssystem (T wie „Transaktionen"). Mit dieser Notation formuliert WAND die vier Bedingungen für Wiedergabetreue:

1. *Mapping Requirement*

   Es existiert eine surjektive Abbildung zwischen dem Zustandsraum S (der Menge aller Zustände) des realen Systems und dem Zustandsraum M des Informationssystems: $rep : S \rightarrow M$.

2. *Tracking Requirement*

   Für jeden Zustand des realen Systems $s \in S$ gilt: $\text{rep}(L(s)) = P(\text{rep}(s))$.

   Die beiden ersten Bedingungen implizieren eine Strukturähnlichkeit (Homomorphie) zwischen der Zustandsmenge des realen Systems und der des Informationssystems. Mit anderen Worten: Das Informationssystem „weiß", wie es die Struktur des realen

[8] Daten und Prozesse sind gewöhnlich die bestimmenden Größen in der Beschreibung von Informationssystemen. Um die Verbindung zur ontologischen Betrachtung zu knüpfen, können wir Daten als Implementierung von Zuständen und Ereignissen ansehen (ein Ereignis ist ein Zustandspaar: $e =< s_1, s_2 >$) und Prozesse als Implementierung von Gesetzen.

Systems nachbilden soll. Es weiß allerdings noch nicht, wie es das Verhalten des realen Systems, das heißt die Zustandswechsel aufgrund externer Ereignisse, repräsentieren soll. Dazu müssen zwei weitere Bedingungen erfüllt sein:

3. *Reporting Requirement*

   Sei $e \in E$ ein externes Ereignis des realen Systems: $e =< s, \tilde{s} >$. Der Zustand $m \in M$ des Informationssystems sei $m = \text{rep}(s)$. Das reale System wird ein externes Ereignis $t \in T$ für das Informationssystem erzeugen, so daß $t =< m, \tilde{m} >=< \text{rep}(s), \text{rep}(\tilde{s}) >$.

   In diesem Fall spiegelt das externe Ereignis im Informationssystem das externe Ereignis im realen System wider. Auch mit der dritten Bedingung ist noch nicht gewährleistet, daß das Informationssystem zeitgleich dem realen System folgt. Voraussetzung hierfür ist, daß $m = \text{rep}(s)$ zeitgleich mit dem externen Ereignis t gilt. Die vierte Bedingung spezifiziert den Zeitbezug:

4. *Sequencing Requirement*

   Seien $\{e_i\}$ die Menge der Ereignisse im realen System und $\{t_i\}$ die Menge der zugeordneten Ereignisse im Informationssystem. Dann müssen die Folgen $e_1, \ldots, e_n$ und $t_1, \ldots, t_n$ elementweise übereinstimmen.

Nehmen wir nun an, alle vier Bedingungen seien erfüllt. Das Informationssystem sei „zurückgesetzt“, um mit dem Zustand des realen Systems übereinzustimmen (*mapping*), bevor dieses begann, die Ereignisse im Informationssystem zu beeinflussen. Wenn nun Ereignisse im realen System auftreten, wird es Ereignisse für das Informationssystem erzeugen (*reporting*). Diese Ereignisse werden in der gleichen Folge auftreten wie die ursprünglichen Ereignisse im realen System (*sequencing*). Die Natur dieser Ereignisse ist derart, daß aufgrund der *Tracking*-Bedingung das Informationssystem eine Zustandsmenge durchqueren wird, die der Zustandsmenge entspricht, die auch das reale System durchquert. Damit garantieren die vier genannten Bedingungen die „Wiedergabetreue“ im Zustandsverhalten und damit eine „gute Repräsentation“ des realen Systems.

Zusammenfassend und ausblickend können wir die Vorteile einer ontologisch motivierten Vergleichsmethode wie folgt umreißen: Analyse und Design sind Modellierungsvorgänge. Damit ist die Ausdrucksstärke der Modellierung das Schlüsselkriterium für die Bewertung von Analyse- und Designmethoden. Zwei konträre Bezüge der Modellierung lassen sich unterscheiden: Sind die Mittel der Modellierung realitätsbezogen (*reality driven*) oder beziehen sie sich auf das zu entwerfende Informationssystem (*information system driven*)? Wie gut ist die Wiedergabe der statischen und dynamischen Eigenschaften (Struktur und Verhalten) des modellierten Weltausschnitts? Zweckfreie und implementierungsneutrale Antworten gibt die ontologische Bewertung. WAND hat diesen Ansatz bereits für zwei weitverbreitete Methoden durchgeführt: Datenflußpläne [DeMarco, 1978] und Entity-Relationship-Modelle [Chen, 1976].

Es wäre sicherlich sehr aufschlußreich, wenn zukünftige Arbeiten die objektorientierten Entwurfsmethoden mit einbezögen. Bislang wurden diese auf der Grundlage verallgemeinerter objektorientierter Begriffe bewertet — methoden-*immanent*, das heißt aus der objektorientierten Sicht [Arnold *et al.*, 1991; Champeaux & Faure, 1992; Monarchi & Puhr, 1992; Stein, 1993; Walker, 1992]. Der ontologische Vergleichsansatz indes erlaubt methoden-*transzendente* Aussagen über die Realitätstreue der Modellierung, über die Stärken und Schwächen einer Methode.

Fazit

Wie lautet der Tenor der drei Ansichten zu einer Theorie des Entwerfens — SIMONs Wissenschaft vom Künstlichen, ZEMANEKs Architekturideale und WANDs ontologische Analogie? Er lautet: weg von den Rezepten, von den technischen Regelwerken und hin zu einer interdisziplinären Wissenschaft des Entwerfens. Entwerfen ist eben nicht der alleinige Wirkungsbereich des Baumeisters, der seine Kompetenz durch nicht formalisierbare Eigenschaften belegt: durch handwerkliche Erfahrung und Geschick. Der Entwurf setzt sich nicht allein aus genormten Lösungen und vorgefertigten Bauteilen zusammen. *Kreativität* und *Ingenieurkunst* sind die zentralen Themen, die den Entwurfsprozeß vorantreiben. Das Objekt-Paradigma kann hier als konzeptionelles und technisches Substrat für eine Wissenschaft des Entwerfens dienen.

6

Konventionelle Modelle des Entwerfens

Angelpunkt einer jeden pragmatischen Methode ist ein Modell des Entwerfens. Es erlaubt dem Modellbenutzer, Entwurfsobjekte und -handlungen in den Kontext der Entwurfsaufgabe zu stellen. Zur Analyse und Optimierung der CAD-Einsatzmöglichkeiten, der Interaktion zwischen Mensch und Maschine und zur konsistenten Datenverwaltung ist die Kenntnis der modelltheoretischen Grundlagen unabdingbar. Nachdem wir den modelltheoretischen Unterbau eines allgemeinen Entwurfsprozesses im ersten Teil der Studie erläutert haben, wenden wir uns hier eingehender den konkreten Modellierungsansätzen zu.[1] Die gesichteten Vorschläge lassen sich grob nach dem formalen Hilfsmittel ordnen, das zur Beschreibung der Modellaspekte angewandt wird. Es zeigen sich vor allem Anleihen aus der Systemtheorie; besonders deren Teildisziplinen — die WIENERsche Kybernetik und die SHANNONsche Informationstheorie — spielen hier die Rolle von Hilfswissenschaften, ähnlich der Mathematik in den Naturwissenschaften. Als dritte Modellierungsklasse haben wir (mangels eines griffigeren Terminus) den Begriff der Verfahrenstechnik gewählt, der ursprünglich aus der Chemo-Produktionstechnik stammt. Er bietet sich zwanglos an, um die Verfahrensketten und -landschaften im Umfeld des VLSI-Instrumentariums zu beschreiben. Die so klassifizierten Modellierungsansätze sollen zunächst „wertfrei" dargestellt werden; eine vergleichende Bewertung mit Hilfe modelltheoretischer Kriterien schließt sich an.

Übersicht: Abschnitt 6.1 skizziert den kybernetischen Modellierungsansatz für den Entwurfsprozeß. Entwerfen wird als regelungstechnisches Problem analysiert. Abschnitt 6.2 erläutert den informationstheoretischen Ansatz und führt die *Entropie* als ein Maß für Kreativität ein. Abschnitt 6.3 greift das derzeit akzeptierte Vorgehensmodell für den VLSI-Entwurf auf: das *Y-Diagramm* nach Daniel GAJSKI. Der zugrundeliegende verfahrentechnische Ansatz wird am Beispiel erläutert. Im Abschnitt 6.4 stellen wir die modelltheoretischen Charakteristika der Ansätze einander gegenüber und ziehen das Fazit des Vergleichs:

[1] Grundzüge der Ausführungen — insbesondere die Rückschau und ein erster Ausblick auf das Objektmodell — wurden auf dem 5. E.I.S.-Workshop diskutiert [Quibeldey-Cirkel & Wojtkowiak, 1991].

die konventionellen Entwurfsmodelle sind überwiegend systemorientiert, kognitive Aspekte bleiben unberücksichtigt.

6.1 Kybernetische Modelle

Die Grundbegriffe, Methoden und Erkenntnisse der von Norbert WIENER begründeten Kybernetik zählen zweifellos zum Allgemeingut sowohl natur- als auch geisteswissenschaftlicher Disziplinen. Als interdisziplinäre „Wissenschaft von den Wirkungsgefügen“ [Sachsse, 1971] abstrahiert sie von der physikalischen Natur der Wirkungsträger: der Materie und Energie. Sie konzentriert sich auf die rein logisch-strukturelle Verknüpfung und erlaubt so Analogien in Struktur und Verhalten zwischen technischen und natürlichen Systemen. Der Untertitel in WIENERs klassischem Originalwerk weist bereits darauf hin: *Cybernetics: Or Control and Communication in the Animal and the Machine* [Wiener, 1948]. Der Universalcharakter, den die Kybernetik ihrem innewohnenden Abstraktionsprinzip verdankt — Informationsverarbeitung als das „Gemeinsame im Verschiedenen“ [Sachsse, 1971, S. 7] — prädestiniert sie als formale Meta-Sprache zur Modellierung komplexer Systeme. Was wunder, daß ein Großteil der Modellierungsansätze auf der kybernetischen Betrachtungsweise beruht!

Bevor wir auf die kybernetischen Modelle im Detail eingehen, sollen die wichtigsten Begriffe anhand einer allgemeinen Gleichung erläutert werden. Ausgehend vom vereinfachenden funktionalen Black-box-Ansatz:

$$\boxed{y_{t+k} = f(\delta_t) = f(x_t - \varphi_t)} \tag{6.1}$$

lassen sich die regelungstechnischen Aspekte der Kybernetik recht gut überschauen. Der *Beobachter* im Entwurfsprozeß — in unserer Diskussion der Mensch oder die Maschine als Modellbenutzer — ermittelt die Abweichung der Regelgröße x_t, des aktuellen Entwurfsergebnisses eines Entwurfsabschnitts, von der Führungsgröße φ_t, dem vorgegebenen Entwurfsauftrag. Die Differenz δ_t zwischen der Vorgabe und dem Ergebnis wird in geeigneter Form als Stellgröße auf den Eingang des Systems zurückgeführt: Prinzip der *Rückkopplung* und der *selektiven Information*. Am Systemausgang stellt sich die geregelte Größe y_t ein, verzögert um die Reaktionszeit k. Im „eingeschwungenen“ Zustand liegt schließlich ein hinsichtlich der Regelabweichung δ_t optimierter Entwurf vor.[2]

[2] Die Attribute „eingeschwungen“ und „optimiert“ definieren sich über die technischen und wirtschaftlichen Ressourcen des Entwurfs (siehe SIMONs Begriff der „zufriedenstellenden“ Lösung in Abschnitt 5.1).

Der Regelkreis heißt deterministisch, wenn die Reaktion des Systems durch die Eingangsinformation bestimmt wird. Er ist darüber hinaus dynamisch, wenn er über ein Gedächtnis für früheres Verhalten verfügt und wenn die Eingangsinformation und die Geschichte der zurückliegenden Einwirkungen gemeinsam die aktuelle Systemreaktion bestimmen. Die *Identifizierung* und *Beschreibung* der einzelnen regelungstechnischen Komponenten im deterministisch dynamischen Entwurfsprozeß sind Gegenstand der nun zu diskutierenden kybernetischen Modelle.

6.1.1 Regelungstechnisches Modell nach Amkreutz

Als einer der ersten, die den rechnerunterstützten Entwurf im kybernetischen Bezugsrahmen analysieren, unterstreicht J. H. AMKREUTZ den Universal-Charakter der Kybernetik als interdisziplinäre Meta-Sprache [Amkreutz, 1976]. Aus der Architektur kommend entwickelt er ein regelungstechnisches Modell des allgemeinen Entwurfsprozesses. Es erlaubt, „specific characteristics of man and machine“ für technische Problemlösungen zu untersuchen und zu integrieren. Vor dem Hintergrund der Evolution des Entwurfsprozesses (siehe Abschnitt 2.3.4), „the tension between more complex design requirements and objectives, and the tools offered to solve design problems“, verschafft uns AMKREUTZ Zugang zur inneren Struktur der Black-box „Entwurf“. Er begreift den Vorgang des Entwerfens als informationsverarbeitenden Prozeß. Seine Vorgehensweise der „model reticulation“ führt auf ein vernetztes Regelsystem, wie es Bild 6.1 zeigt.

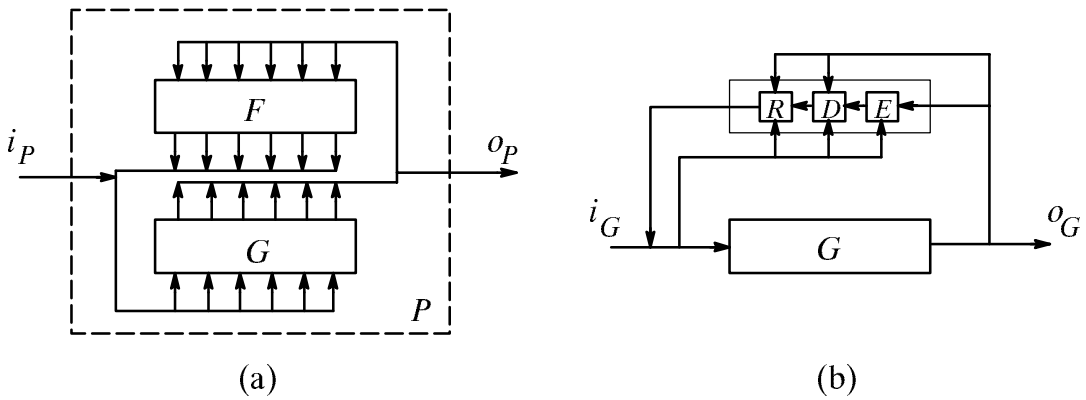


Bild 6.1: Der Entwurfsprozeß als Regelsystem nach AMKREUTZ

AMKREUTZ unterscheidet prinzipiell zwischen zwei informationsverarbeitenden Funktionsblöcken: „generation“ G und „feedback“ F. Die erzeugenden Funktionen setzen einen Entwurfsauftrag in eine Menge von Entwurfsalternativen um, die durch die rückkoppelnden

Funktionen unter dem Gesichtspunkt der Realisierbarkeit und vorgegebener Optimierungskriterien (Entwurfsgüte, Kostenfunktionen) auf ein Entwurfsergebnis eingeschränkt werden (vergleiche mit Bild 5.3 auf Seite 100 zum „Generator-Test-Zyklus“ nach SIMON). Da der Entwurf eine Vielzahl synthetisierender und verifizierender Aspekte beinhaltet, die abschnittsweise auf den Prozeß einwirken und wechselseitig alle Entwurfsabschnitte umfassen, setzt sich der gesamte Entwurfsprozeß letztlich aus einem System *gekoppelter* Regelkreise zusammen (Bild 6.1a). Die Bearbeitung der einzelnen Entwurfsaspekte erfolgt im allgemeinen nicht sequentiell, so daß in der Verbindungsstruktur zwischen den generierenden und rückführenden Funktionen kombinatorische Vielfalt herrscht.

Von zentraler Bedeutung für die Beurteilung des Einsatzes und der Entwicklung von CAD-Komponenten sind die „kybernetischen Basisfunktionen“. Funktionell teilt sich die Rückkopplung auf in: „evaluation“ E, „decision“ D und „regulation“ R (Bild 6.1b). Der Soll-Ist-Vergleich E liefert die Regelabweichung δ_t in der Grundgleichung 6.1. Die Entscheidungsinstanz D ermittelt für diese Regelabweichung eine geeignete Strategie und Stellgröße für den Steuerungsmechanismus. Die Steuerung[3] R führt die gewählte Entscheidungsstrategie aus, indem sie den Eingangsinformationsfluß entsprechend ändert. Zusammen konstituieren die kybernetischen Basisfunktionen den Regelungsanteil von $f(\delta_t)$.

Der Rückkopplungszweig dient somit der Anpassung an das globale Entwurfsziel, das erst im Gleichgewichtszustand des Regelsystems angenähert vorliegt. Da jeder kybernetische Prozeßschritt die Historie vorhergehender Schritte berücksichtigt (dynamisches System), stellt der Entwurf insgesamt einen *Lernprozeß* dar:

> „The cybernetic model developed is a dynamic decision model, in which the process is directed towards the production of an optimum design by making successive series of decisons on the basis of endogenous and exogenous information.“ [Amkreutz, 1976, S. 184]

Für die Analyse der Einsatzmöglichkeiten von Automaten ist die „Quantifizierbarkeit“ der kybernetischen Basisfunktionen maßgebend. Lassen sich Bewertungs-, Entscheidungs- und Steuerungsinstanzen auf formale Systeme (Algorithmen) abbilden, so sind sie auch automatisierbar. Wie die Aufgaben zwischen Anwender und CAD-System verteilt werden, bestimmt die Realisierbarkeit quantifizierbarer und nicht-quantifizierbarer (im allgemeinen heuristischer) Prozeßschritte. Um eine erfolgreiche Zusammenarbeit zu gewährleisten, ist eine sorgfältige Aufgabenteilung Voraussetzung. AMKREUTZ leitet hierfür drei Hauptforderungen ab:

- *Ergonomie*: Anpassung des CAD-Systems an die Arbeitsweise des Anwenders, um dessen Kreativität zu fördern

[3] AMKREUTZ unterscheidet hier nicht streng zwischen Steuern und Regeln, obwohl nur Regeln eine selbständige Verhaltensänderung durch Beobachten der Regelgröße bezeichnet.

- *Grafik*: geeignete Werkzeuge für die Kommunikation
- *Systemdenken*: Anpassung des Anwenders an die Systematik des CAD-Systems

Mehr-Ebenen-Erweiterung nach Rammig

Das abstrakte Grundmodell in Franz RAMMIGs „durchgängiger Systematik des Hardware-Entwurfs“ [Rammig, 1989, S. 5] stützt sich auf die kybernetischen Erkenntnisse von AMKREUTZ. Mit einem Jahrzehnt technischen Abstand verallgemeinert RAMMIG den Rückkopplungsprozeß „Entwurf“ über die Abstraktionsebenen hinweg. Sein „makroskopisches Modell des Entwurfsprozesses“ ist identisch mit dem Vorschlag von AMKREUTZ. Auch er unterscheidet zwei Klassen von Entwurfshandlungen: „generierende und überprüfende“ Handlungen; erhöht aber die Transparenz der Informationskanäle und des Aktivitätsflusses, indem er diese genauer spezifiziert.[4]

RAMMIGs „mikroskopische“ Verallgemeinerung des kybernetischen Ansatzes führt auf ein Klassifikations-Schema, das die erzeugenden und überprüfenden Enwurfshandlungen weiter differenziert in ebenen-variante — „Implementation“ und „Regulation“ (Bild 6.2) — und ebenen-invariante — „Modifizierung“ und „Optimierung“.

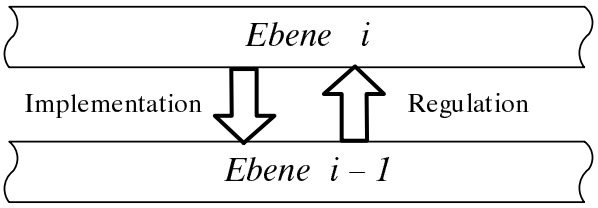


Bild 6.2: Zur Modell-Verfeinerung nach RAMMIG

Das so begründete „multi-level cybernetic model“ [Rammig, 1985] erlaubt es, Kompositionsregeln für die Entwurfsaktivität, wie Verkettung, Rekursion und Mehr-Ebenen-Rückkopplung, und klassische Entwurfsstrategien, wie *top-down*, *bottom-up* und Mischformen daraus, detaillierter zu modellieren.

Zusammengefaßt führt der kybernetische Ansatz auf ein regelungstechnisch formales Modell. Es dient der Grundanalyse des Entwerfens und als Grundlage für den Einsatz und die Entwicklung von CAD-Systemen. Es erlaubt die Klassifikation der Entwurfshandlungen

[4] Seine Systematik steht unter dem Anspruch, die höheren Abstraktionsebenen des Hardware-Entwurfs — System-, Verhaltens-, Registertransfer- und Gatterebene — einheitlich zu beschreiben: *Breitband-Hardware-Beschreibungssprache DACAPO III.*

mit Hilfe kybernetischer Prozeßelemente und eignet sich wegen seiner interdisziplinären Allgemeingültigkeit und intuitiven Zugänglichkeit besonders als didaktischer Angelpunkt einer VLSI-Entwurfsmethodik [Wojtkowiak, 1989].

6.2 Informationstheoretische Modelle

Gegenstand dieser Modellierungsklasse ist die Analyse des Informationsflusses im Entwurfsprozeß. Die von SHANNON begründete *Mathematical Theory of Communication* [Shannon & Weaver, 1949] dient hierbei als formales Medium, um die zwei wesentlichen Aspekte eines Entwurfs — Komplexität und Kreativität — quantitativ zu untersuchen. Besonders der zweite Aspekt ist für unsere Studie interessant; wird hier doch erstmalig versucht, die Kreativität des Entwerfers formal zu erfassen. Zunächst wird die allen realen Systemen innewohnende *Kausalität* informationstheoretisch interpretiert. Roger CONANT, der diesen Ansatz entwickelt hat, faßt die Auslegung des Kausalitätsprinzips zusammen:

> „If each variable in a system is viewed as a message source which sends information about its values to the other variables which are ‚listening', then what is conventionally seen as a network of causes and effects can be viewed as a network of transmitters, channels, and receivers. The strength of the causal relations can then be quantified by information rates T between variables or sets of variables. These rates are bounded by channel capacities. Information theory can therefore be used in the analysis of constraints or dependencies in multivariable systems, where it has particular advantages for dynamic systems and systems with hierarchical structure." [Conant, 1976, S. 254]

Obwohl in der Entwurfspraxis nicht von streng *stationären* (invariant gegenüber zeitlichen Verschiebungen), geschweige denn von *ergodischen*[5] Systemen ausgegangen werden kann und eine exakte Berechnung des Informationsflusses nur im Ausnahmefall möglich ist, besitzen die SHANNONschen Lehrsätze ihren allgemeinen „Praxiswert": *Erkenntnisgewinn* durch informelle Interpretation und Abschätzung der gefundenen Formeln und Identitäten. So gewann beispielsweise MILLER seine Erkenntnis von der Obergrenze der menschlichen Kanalkapazität durch informelle Anwendung des SHANNONschen Begriffsrahmens und der darin aufgestellten Lehrsätze (siehe Abschnitt 2.1). Im Kontext technischer Entwurfssysteme ergeben sich praktische Richtlinien und Grenzwerte vor allem für die Synthese. Bevor wir aber genauer auf informationstheoretische Modelle eingehen, stellen wir eine kurze Erläuterung der Grundbegriffe *Entropie*, *bedingte Entropie* und *Transmission* voran.

[5] WEAVER veranschaulicht den komplexen mathematischen Begriff der *Ergodizität*: „Es handelt sich um einen Vorgang, in dem eine Zeichenfolge erzeugt wird, die vielleicht der Traum eines Demoskopen wäre, weil jede einigermaßen große Probe danach strebt, repräsentativ für die Folge als Ganzes zu sein" [Shannon & Weaver, 1949, S. 21].

Grundbegriffe der Informationstheorie

Der auf BOLTZMANN zurückgehende Begriff der *Entropie H* erfaßt die *mittlere Information* einer diskreten Zufallsgröße x als Funktion der Wahrscheinlichkeiten p_i ihrer möglichen Ereignisse:[6]

$$H(x) = -K \sum_{i=1}^{n} p_i(x) \log p_i(x), \text{ mit } 0 \leq p_i(x) \leq 1 \wedge \sum_{i=1}^{n} p_i(x) = 1. \tag{6.2}$$

Übertragen steht Entropie für den Informationsgewinn, der in der Reduktion der Unbestimmtheit liegt (zum Beispiel in der Kenntnis eines Versuchsausgangs). Wenn n für die Anzahl disjunkter Ereigniswerte steht, die x annehmen kann, so fällt $H(x)$ in das Intervall $[0, \log_2 n]$. Eine niedrige Entropie weist auf eine konzentrierte Wahrscheinlichkeitsverteilung, eine hohe auf eine diffuse. Sie ist Null, wenn ein Ereignis die Wahrscheinlichkeit Eins besitzt: sicheres Ereignis; sie ist maximal bei Gleichverteilung aller Ereignisse.

Werden zwei Zufallsgrößen x und y betrachtet, so gilt für die Entropie bei gegenseitiger Abhängigkeit (Notation nach SHANNON):

$$H(x,y) = H(x) + H_x(y) = H(y) + H_y(x). \tag{6.3}$$

Die Gleichung besagt, daß der Informationsgehalt aller Ereignis-Kombinationen zweier Zufallsvariablen x und y gleich dem Informationsgehalt einer der beiden Variablen ist plus der verbleibenden *bedingten Entropie* der anderen Variablen, wenn der Ereigniswert der zweiten Variablen bekannt ist. (SHANNON bezeichnet $H_x(y)$ bzw. $H_y(x)$ auch als „equivocation" und unterstreicht damit die durchschnittliche Mehrdeutigkeit des Ergebnisses.)

Als Maß für den Zusammenhalt (Abhängigkeit oder Koordinierung) zweier Zufallsvariablen wird die *Transmission* $T(x:y)$ über die Entropien definiert:

$$T(x:y) = H(x) + H(y) - H(x,y). \tag{6.4}$$

Sie wird Null, wenn x und y statistisch unabhängig sind, das heißt $H(x,y) = H(x) + H(y)$; sie wird maximal, wenn eine Variable die andere bestimmt, das heißt $H_y(x)$ bzw. $H_x(y) = 0$. Im Zusammenhang mit Übertragungskanälen erklärt sich die Transmission als

[6] Aus historischen Gründen und wegen der weit verbreiteten Binär-Codierung wird der Logarithmus dualis gewählt. Damit entfällt die positive Konstante K, und die Einheit der Entropie ist das *Bit*. Für kontinuierliche Variablen wäre eine Integraldefinition anzusetzen.

die Information einer Nachrichtenquelle, vermindert um den Informationsverlust aufgrund einer gestörten Übertragung.

6.2.1 Informationstheoretisches Modell nach Conant

Roger CONANT betrachtet ein System S als eine geordnete Menge von Zufallsvariablen: $S = \{x_1, x_2, \ldots, x_n\}$ [Conant, 1976]. Von außen direkt beobachtbare Systemvariablen bilden die Menge S_o. S_{int} bezeichnet die verbleibenden internen Variablen, so daß $S = S_o \cup S_{int}$. E sei die Menge aller für das Systemverhalten relevanten Variablen der Umgebung (*environment*). CONANT formuliert nun ein Teilungsgesetz für die gesamte Informationsrate[7] F in S:

$$\boxed{F = F_t + F_b + F_c + F_n} \tag{6.5}$$

$F = \sum_{j=1}^{n} \bar{H}(x_j)$	total rate,
$F_t = \bar{T}(E : S_o)$	throughput rate,
$F_b = \bar{T}_{s_o}(E : S_{int})$	blockage rate,
$F_c = \bar{T}(x_1 : x_2 :, \ldots, : x_n)$	coordination rate,
$F_n = \bar{H}_E(S)$	noise rate.

Die Komponenten von F weisen hierbei folgende Eigenschaften auf:

F repräsentiert die Gesamtaktivität im System.

F_t mißt den System-Durchsatz und ist ein Maß für die Stärke der Bindung zwischen Umgebung und Systemausgang. Als Kommunikationskanal gesehen wird F_t durch die Kanalkapazität C begrenzt.

F_b stellt die Rate dar, mit der Information aus der Umgebung innerhalb von S unterdrückt wird, um den Ausgang nicht zu beeinflussen. Als aktive „Filter-Rate“ hat F_b große Bedeutung in informationsverarbeitenden Organismen. CONANT verweist hier auf Regelungsmechanismen in Natur und Technik, zum Beispiel auf das psychologische Phänomen der *Gewöhnung*: Als herabgesetzte Reaktion auf wiederholte Stimuli geht eine Gewöhnung hauptsächlich auf *Blockierung* zurück.

[7] Um die Historie der Ereignisse einer Zufallsvariablen x zu erfassen, führt CONANT Entropie-*Raten* ein (angedeutet durch Überstriche): „entropy of x conditional on all its prior values, which is the information carried per observation in a long sequence. The total uncertainty of a long sequence $x(t), x(t+1), \ldots, x(t+n)$ is then (approximately) this entropy rate times the sequence length“ [Conant, 1976, S. 244].

F_c steht für den erforderlichen Aufwand, um die Aktivität zwischen den Systemvariablen (Subsystemen) zu koordinieren. Sie kann als Maß für die interne Kommunikation in S angesehen werden, um ein Problem zu lösen. F_c ist das Charakteristikum eines Systems schlechthin: S agiert als Ganzes und nicht als Summe seiner Teile.

F_n schließlich repräsentiert den Rauschanteil am Gesamtfluß, die Informationsrate in S, wenn E vollständig bekannt ist. In der Kommunikationstechnik entspräche F_n dem thermischen Rauschen eines Kanals. Während CONANT diese Komponente zu Null setzt (deterministischer Fall), sieht Cees KOOMEN hier den *kreativen* Anteil des Entwerfers am Entwurfsprozeß. Wir werden im nächsten Abschnitt darauf zurückkommen.

Aus der additiven Eigenschaft des CONANTschen Teilungsgesetzes 6.5 ergeben sich bei vorgegebenem F (Man-power, Rechnerkapazität) konkurrierende Freiheitsgrade für die Komponenten F_t, F_b und F_c. Verfolgt man das *ökonomische Prinzip* — Maximierung des Ertrages bei Minimierung des Aufwands —, so lassen sich aus der Analyse von F Richtlinien für die Systemkonzeption ableiten:

1. „produce the minimum allowable output (do not do anything unnecessary),
2. perform as little blockage as possible (try to take in a minimum of irrelevant input),
3. reduce internal coordination to the minimum consistent with other requirements (maximize freedom of the components), [...]
4. as far as possible, match components to tasks so that each component is operated at capacity (let each component do what it does best, and work it as hard as you can)." [Conant, 1976, S. 248]

Diese Konstruktionsvorschriften liefern zugleich informationstheoretische Argumente für SIMONs hierarchisches Strukturierungs-Postulat:

$$\bar{H}(S) = \sum_{i=1}^{N} \bar{H}(S^i) - \bar{T}(S^1 : S^2 :, \ldots, : S^N). \tag{6.6}$$

In Worten: Die Entropie(rate) eines Systems S ist im allgemeinen geringer als die Summe der Entropien seiner Subsysteme S^i. Während SIMON seine These, daß die darwinistische Evolution Organismen mit isolierbaren Strukturen begünstige, auf Stabilitäts-Überlegungen stützt, erlaubt die Informationstheorie die gleiche Schlußfolgerung über die statistische Untersuchung des Informationsflusses: Höchste Ökonomie (ein Selektionskriterium der Evolution) bedingt eine möglichst geringe Transmissionsrate zwischen den Subsystemen. Dies entspricht SIMONs Theorem der „niederfrequenten Dynamik" auf den Inter-Komponentenverbindungen (siehe Abschnitt 2.1).

6.2.2 Informationstheoretisches Modell nach Koomen

Cees KOOMEN modelliert den Entwurfsprozeß als geordnete Transformationsfolge von Systemmodellen $< M_p, M_{p+1}, \ldots, M_q >$, wobei ein Modell M_i das Gesamtsystem auf dem Beschreibungsniveau i repräsentiert [Koomen, 1978; Koomen, 1985]. Als „modeling primitive“ definiert er die *Detailing-step*-Methode: Der Entwurfsprozeß verfolgt die sukzessive Detaillierung der Beschreibungsmodelle bis zum Grad des Zielmodells M_q. Ein Verfeinerungs-Schritt beinhaltet die Entwurfsaktivität, um von einer Spezifikation S der geforderten System-Eigenschaften (Ausgangsmodell M_i) zu einer Implementierung I (Entwurf von M_{i+1}) zu gelangen. Bild 6.3 verdeutlich diese Betrachtungsweise.

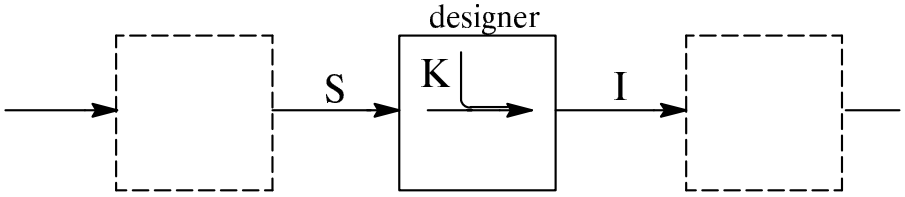


Bild 6.3: SKI-Modell eines Entwurfsabschnitts nach KOOMEN

Mit Einführung der Komponente K — des Wissens (*knowledge*), das vom Entwerfer geschaffen oder als Ressource eingesetzt wird, um I von S abzuleiten — verallgemeinert KOOMEN das informationstheoretische Modell von CONANT auf nichtdeterministische Systeme. Die Rauschrate F_n wird substituiert durch den *Einsatz* und die *Kreativität* des Entwerfers: $F_n \Leftarrow \bar{H}_S(I)$. „Hence, the designer's ‚noise rate' reflects the unexpected, the unpredictable, the creative act“ [Koomen, 1985, S. 21].

Auf der Grundlage der *Detailing-step*-Methode können informationstheoretische Aussagen über die Eigenschaften des Entwurfsprozesses gewonnen werden. Das Ausgangsrauschen einer Modell-Transformation $\bar{H}_S(I)$ ist größer als Null, wenn eine nichttriviale Implementation vorliegt. Da die Energie für den Implementierungsprozeß natürlich endlich ist, gilt insgesamt $0 < \bar{H}_S(I) < \infty$. Das Wissen K, um von einer Spezifikation S zu einer Implementation I zu gelangen, muß letztlich der Bedingung

$$\boxed{\bar{H}_{S,K}(I) = 0} \tag{6.7}$$

genügen. Wegen der Ähnlichkeit zum Energie-Erhaltungssatz der Thermodynamik bezeichnet KOOMEN diesen Ausdruck auch als „first information law for system design“. Die Gleichung impliziert, daß K durch eine Menge von Symbolen ausgedrückt werden kann, die

dem Beobachter bekannt sind. Falls K in einer formalen Sprache beschreibbar ist (vergleiche mit AMKREUTZ' Begriff der Quantifizierbarkeit), so kann die Modell-Transformation mit deterministischen Maschinen erfolgen.

Interpretiert man den Entwerfer als ein informationsverarbeitendes System (das heißt unter anderem, daß seine CONANTsche Informationsrate F begrenzt, seine kreativen und intellektuellen Fähigkeiten endlich sind), so läßt sich folgende Einschränkung formulieren:

$$\boxed{\bar{H}_S(I) \leq C_d(S, \tau_d)} \tag{6.8}$$

Hierbei steht $C_d(S,t)$ für die Informationsmenge, die der Entwerfer in einer Zeit t erzeugen kann, um eine Implementation von S zu finden. Bei vorgegebener Zeit τ_d für die Problemlösung setzt die Gleichnung 6.8 die zu erzeugende Information $\bar{H}_S(I)$ in Relation zur Kapazitätsfunktion C_d. Da durch die Relation eine obere Grenze für die Entropie-Änderung gegeben ist, bezeichnet KOOMEN dies auch als „second information law for system design" in Anlehnung an den Entropie-Richtungssatz der Thermodynamik. Für den Fall, daß $\bar{H}_S(I) > C_d(S, \tau_d)$, das heißt der Entwerfer ist nicht in der Lage, S zu implementieren, ergeben sich drei Lösungsmöglichkeiten:

1. Dem Entwerfer wird zur Lösungsfindung mehr Zeit gegeben (τ_d erhöhen).

2. Die Entwurfskomplexität wird erniedrigt, indem hierarchische Zwischenebenen eingeführt werden, so daß Implementierungen auf diesen Ebenen der obigen Bedingung genügen.

3. $C_d(S, \tau_d)$ wird erhöht, entweder durch mehr Personal oder durch mehr Rechnereinsatz.

KOOMEN untersucht des weiteren die informationsverarbeitenden Aspekte beim Einsatz von Maschinen und analysiert den Informationsfluß durch die Mensch-Maschine-Schnittstelle. Bezüglich Einzelheiten sei auf [Koomen, 1985, S. 26ff] verwiesen.

Zusammenfassend erlauben die informationstheoretischen Erkenntnisse von CONANT und KOOMEN die Formalisierung der Grundmechanismen im Entwurfsprozeß. Sie helfen, Begriffe, wie Kreativität, Komplexität und Modellinformation, quantitativ zu klären. Darüber hinaus präzisieren sie elementare Konstruktionsvorschriften für einen *ökonomischen* Systementwurf.

6.3 Verfahrensmodelle

In die dritte Kategorie für Entwurfsmodelle ordnen wir diejenigen ein, die den Entwurfsprozeß unter einem bestimmten Blickwinkel[8] — *sichtenorientiert* — betrachten. Da dies vor dem Hintergrund vielfältiger Verfahrenslandschaften geschieht (Methoden, Techniken und Werkzeuge), sprechen wir hier von handlungsorientierten Verfahrensmodellen. Die Sichtenorientierung erleichtert den Zugang zu hochkomplexen Entwurfsprozessen und wirkt zugleich einer Einengung der Modellierungsgenauigkeit entgegen. Repräsentativ befassen wir uns mit dem weit verbreiteten *Y-Diagramm*, das von Daniel GAJSKI und Robert KUHN entwickelt [Gajski & Kuhn, 1983], von Donald THOMAS und Robert WALKER verfeinert [Thomas & Walker, 1985] und von GAJSKI zuletzt aktualisiert wurde [Gajski, 1988].

6.3.1 Y-Diagramm nach Gajski et al.

Der Anspruch des Y-Diagramms — „a comprehensive view of VLSI CAD tools“ [Gajski & Kuhn, 1983, S. 12] — wurzelt in dem Bedürfnis, die Hauptströmungen in der Chipkonstruktion der 80er Jahre übersichtlich zu erfassen. Je nach dem Grad der Automatisierung und der Art der Wissenseinbringung unterscheiden GAJSKI und KUHN drei Verfahrenslandschaften:

1. *Entwurfsunterstützung*

   Alle Entwurfsentscheidungen obliegen dem Entwerfer. CAD-Werkzeuge wirken ausschließlich unterstützend: „efficient paper and pencil“.

2. *Expertensysteme*

   Verlagerung der Entwurfsunterstützung von den mechanischen zu den intelligenten Aspekten des Entwurfs — Expertenwissen als Quelle wird in Form von Entwurfsregeln in Wissensbanken abrufbar gespeichert (effizient für die Entwurfsprüfung, noch in den Anfängen bezüglich Entwurfssynthese und -planung).

3. *Entwurfsautomation*

   In der Zielsetzung konträr zum CAD-Ansatz: statt zu unterstützen, wird der teilweise oder gänzliche Verzicht auf den Entwerfer angestrebt (zum Beispiel teilweise durch das automatische Plazieren und Entflechten von Layoutzellen oder gänzlich durch den „Silicon-Compiler“).

[8] Als ein Beispiel sichtenorientierter Modellierung im VLSI-Entwurf sei auf das Datenaustauschformat EDIF verwiesen [Quibeldey-Cirkel & Wahl, 1990a]. Es beschreibt die Entwurfsdaten unter verschiedenen Sichten: *viewTypes* wie *schematic*, *netlist* und *maskLayout*.

Der Entwurfsprozeß wird schematisch als Achsen-Modell dargestellt: Die Achsen spannen den Entwurfsraum in drei komplementäre Entwurfsbereiche auf, als Sichten oder Domänen bezeichnet. Die *Verhaltenssicht* ist auf die Funktionalität, die *Struktursicht* auf die logische Struktur und die *physikalische Sicht* auf die physikalische Implementierung gerichtet. Andere Aspekte[9] können unter diesen Domänen subsumiert werden. Entlang der Raumachsen erfolgt von außen nach innen die Entwurfsverfeinerung auf Ebenen zunehmender Detaillierung.

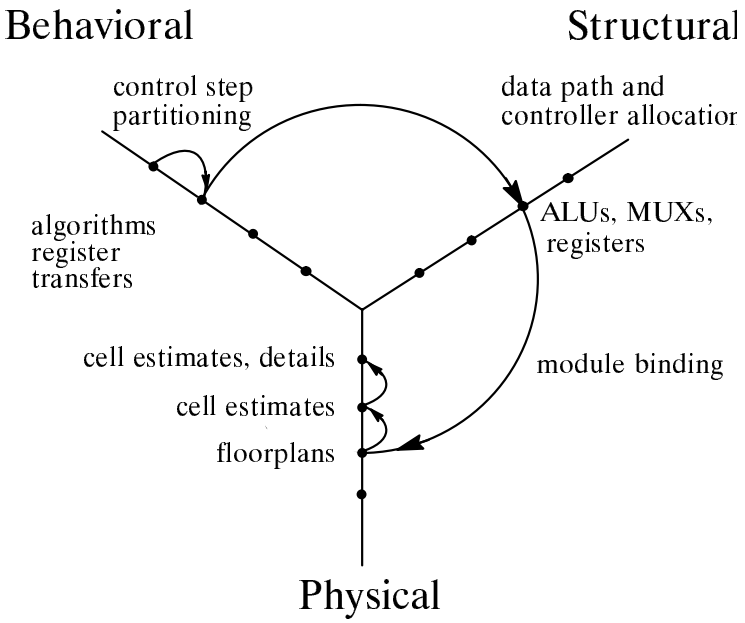


Bild 6.4: Das Y-Diagramm in der Anwendung nach THOMAS und WALKER

Das Y-Diagramm soll also eine Antwort geben auf die Frage: „Wie werden VLSI-Werkzeuge eingesetzt, um eine Entwurfsspezifikation auf höherer Ebene zu verifizieren und in eine Implementation auf niedriger Ebene zu überführen?“ Einerseits sollen die Verfahrensketten (Synthese, Analyse und Verifikation) bereits in der Entwurfsplanung veranschaulicht werden, und zwar im gesamten Entwurfsraum sichtenorientiert und unter Berücksichtigung der geltenden Entwurfsabstraktion. Andererseits soll das Y-Modell die Mechanismen und Interaktionen *realer* Entwurfssysteme transparent machen. Bild 6.4 illustriert dies für das von THOMAS und WALKER angegebene Beispiel der automatischen Datenpfad-Synthese [Thomas & Walker, 1985, S. 248].

Als „inter domain links“ verlaufen die Repräsentationsebenen konzentrisch um den Ursprung des Y-Diagramms. GAJSKI spezifiziert einen Standardsatz von fünf Abstraktions-

[9] RAMMIG regt zum Beispiel die Erweiterung des Y- zum X-Diagramm an, um testspezifische Aspekte im Entwurfsraum aufzuzeigen [Rammig, 1989, S. 14f].

ebenen[10] und nennt exemplarisch einige Entwurfsobjekte der jeweiligen Sicht (Tabelle 1.5 auf Seite 31). Mittels einer grafischen Notation lassen sich die Entwurfswerkzeuge nach Art und Wirkungsbereich im Entwurfsraum lokalisieren. *Synthetisierende* Werkzeuge werden als gerichtete Übergänge notiert: aus der Verhaltens- in die Struktur-Domäne (in Bild 6.4 „data path and controller allocation"), aus der Struktur- in die physikalische Domäne („module binding") oder von einer höheren Abstraktionsebene in eine niedrigere gleicher oder benachbarter Domäne („control step partitioning"). Umgekehrte Übergänge beziehen sich auf entsprechende *analysierende* Werkzeuge (zum Beispiel Schaltkreis-Extraktoren). Eine *Schleife* (Anfangs- und Endpunkt fallen zusammen) repräsentiert ein Analysewerkzeug, das ausschließlich auf einer Ebene arbeitet (Simulatoren).

Darüber hinaus erlauben Einstiegs- und Zielpunkt der Verfahrenskette das Identifizieren der ursprünglichen Entwurfsspezifikation (hier die algorithmische Verhaltensbeschreibung) und der erreichten Implementierungstiefe (Maskenlayout). Ein weiteres Charakteristikum des Y-Modells ist die „non-isomorphe" hierarchische Dekomposition der Entwurfskomponenten: *1-zu-n*-Abbildungen zwischen Verhaltens-, Struktur- und physikalischen Entwurfsobjekten. Im Beispiel von Bild 6.4: Für eine gegebene Menge von Register-Transfer-Beschreibungen existieren mehrere abstrakte Implementierungen in der Struktur-Domäne. „Choosing one of these alternatives is exploring the design space" [Thomas & Walker, 1985, S. 458]. Weiterhin erlaubt das Modell die Lokalisierung von Entwurfseinschränkungen, wie beispielsweise Flächenbedarf und Verlustleistung.

6.4 Einordnung der Vorschläge

Herbert STACHOWIAK kommt das Verdienst zu, als erster den Modellbegriff wissenschaftlich fundiert zu haben. Seine *Allgemeine Modelltheorie* [Stachowiak, 1973] erlaubt die Charakterisierung und Klassifizierung der unüberschaubaren Vielfalt wissenschaftlicher Modelle.

6.4.1 Allgemeine Modelltheorie nach Stachowiak

Um eine möglichst prägnante Gegenüberstellung der diskutierten Modellvorschläge zu erreichen, werden wir im folgenden die Erkenntnisse STACHOWIAKs schematisch anwenden. Zuvor aber sei die „Allgemeine Modelltheorie" kurz umrissen:

[10] Konrad WEBER schlägt für die Schichtung der Abstraktionsebenen den Begriff der „VLSI-Stratifikation" vor und spricht in diesem Zusammenhang von *stratifizierten* Entwürfen [Weber, 1985, S. 14f].

„Etwas schärfer gefaßt: Modelle sind zwar immer Modelle *von etwas*, Abbildungen, Repräsentationen natürlicher oder künstlicher Originale (die selbst wieder Modelle sein können). Aber sie erfassen im allgemeinen *nicht alle* Originalattribute, sondern stets nur solche, die für die Modellbildner und/oder Modellverwender relevant sind. Modelle sind mithin ihren Originalen nicht per se zugeordnet; sie erfüllen ihre Ersetzungsfunktion stets (a) für bestimmte Erkenntnis- und/oder Aktionssubjekte, (b) innerhalb bestimmter Zeitintervalle der Originalrepräsentation und (c) relativ zu bestimmten Zwecken und Zielen, denen die Modellbildung und die Modelloperationen unterliegen. Modellbildungen unterliegen hiernach dem Frageschema: Modell *wovon, für wen, wann* und *wozu*." [Stachowiak, 1983, S. 118]

Damit sind die drei Hauptmerkmale des allgemeinen Modellbegriffs genannt: *Abbildungs-*, *Verkürzungs-* und *pragmatisches Merkmal*. Das VENN-Diagramm in Bild 6.5 verdeutlicht die Zusammenhänge [Stachowiak, 1973].

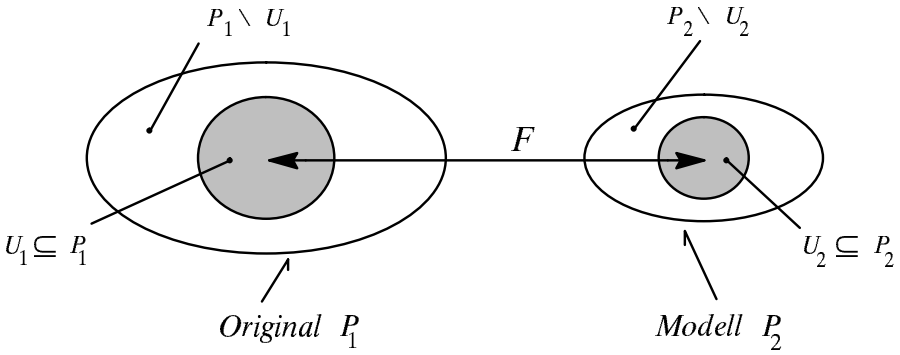


Bild 6.5: Zur Original-Modell-Abbildung nach STACHOWIAK

Die Abbildungsfunktion F ordnet Modell-Attribute Original-Attributen zu. Die Modelltheorie nach STACHOWIAK faßt *Original* und *Modell* als Prädikatenklassen P_1 und P_2 auf und stellt sie in eine mengentheoretische Relation zueinander. Für unsere Anwendung genügt es, unter dem Abbildungsmerkmal die Domäne — Umfang und Funktionalität — des modellierten Realitätsausschnitts zu verstehen.

Das Verkürzungsmerkmal bezieht sich auf die Menge der *präterierten*, das heißt auf die von F nicht erfaßten Attribute $P_1 \setminus U_1$, und der *abundanten*, das heißt im Original nicht vorkommenden Attribute $P_2 \setminus U_2$, wobei U_1 den *Abbildungsvorbereich* und U_2 den *-nachbereich* bezeichnen. Die Modell-Verkürzung ist also ein (negatives) Maß für die Realitätstreue, für die Ähnlichkeit zwischen Original und Modell. Sie gibt Aufschluß über die Bijektivität der Klassen P_1 und P_2. Für unsere Zwecke verbinden wir mit diesem Begriff die Verkürzung auf bestimmte Aspekte des Originals: system-, entwerfer- oder werkzeugorientierte Perspektiven. Weiterhin unterscheiden wir hier die Art der Modellaussagen, ob qualitativ, quantitativ oder phänomenologisch.

Das pragmatische Merkmal schließlich berücksichtigt: (a) die Zielsetzung (Vorgehens-, Erklärungs- oder Entscheidungsmodell), (b) das Zeitintervall (Gültigkeit, technische Annahmen) und (c) die Leistungen des Modells (zum Beispiel im Rahmen einer Entwurfsmethode).

6.4.2 Gegenüberstellung

Als Abschluß unserer Rückschau stellen wir mit Hilfe obiger Charakterisierung die diskutierten Modellierungsvorschläge einander gegenüber. Neben den drei Hauptmerkmalen Abbildung A, Verkürzung V und Pragmatik P kommt ein weiteres hinzu: das Konstruktionsmerkmal K, welches zur Klassifikation der Modelle dient. Kybernetische, informationstheoretische und verfahrenstechnische Modelle unterscheiden sich in ihrer Notation und ihrem Gegenstandsbereich. Tabelle 6.1 faßt die Charakteristika der Modelle vergleichend zusammen.

Konventionelle Modelle des VLSI-Entwurfs			
K	Kybernetik	Informationstheorie	Verfahrenstechnik
A	kybernetische Prozeßelemente: evaluation, decision, regulation (feedback)	Informationsfluß, Entropie, Transmission, Kanalkapazität	CAD/CAE-Instrumentarium, Verfahrensketten und -landschaften
V	systemorientiert	system- und entwerferorientiert	werkzeug- und praxisorientiert
	qualitativ: bewerten, entscheiden, steuern	quantitativ: Formeln, Identitäten, Eckwerte	phänomenologisch: VLSI-Objekte, Werkzeuge
P	Entscheidungsmodell	Erklärungsmodell	Vorgehensmodell
	allgemeingültig	allgemeingültig	technikbezogen
	Strukturieren der Entwurfsaufgabe, Identifizieren der Entwurfshandlungen und deren Wirkungsgefüge	Konstruktionsprinzipien, Analyse von Kreativität und Komplexität im Entwurfsprozeß, Grenzwertbetrachtungen	Identifizieren und Lokalisieren der Entwurfswerkzeuge und -handlungen, Orientierungs- und Führungshilfe

Tabelle 6.1: Gegenüberstellung der Modellvorschläge

Fazit

Als Quintessenz unserer Gegenüberstellung können wir festhalten: Systemtheoretische *und* kognitive, das heißt die begriffliche Vorstellungsweise des Modellbenutzers unterstützende Aspekte werden nicht von einem Modell allein reflektiert. Was fehlt ist eine *monistische Entwurfssicht* — ein Modell, das die Forderungen nach einer durchgängigen Entwurfsmethode, einem abstrahierenden „Systemdenken“ und nach globaler Datenintegrität *einheitlich* erfüllt. In den folgenden Kapiteln werden wir unsere Vorstellungen präzisieren.

7

Entwerfen als Problemlösen

Während der Begriff des „Problemlösens“ wissenschaftlich längst besetzt ist — die Denkpsychologie und Kognitionsforschung haben ihn zum Thema [Dörner, 1976; Gardner, 1985] — fehlt es bislang an einer allgemeinen „Theorie des Entwerfens“. Die Enzyklopädien schweigen sich aus über das Gemeinsame jenseits der phänomenologischen Unterschiede zwischen der Architektur, dem Ingenieurwesen und der Kunst. Wie wir in Kapitel 5 zeigen, ist es die Informatik, die hier auf den Plan tritt mit Curriculum-Vorschlägen für eine „Science of Design“ [Simon, 1982], mit „architektonischen Leitideen für Systeme“ [Zemanek, 1992] oder mit einer „künstlerischen“ Einstellung zum Software-Entwurf [Knuth, 1974]. Das nimmt nicht wunder, da gerade in der Informatik die fachübergreifenden Themen zwischen der Softwaretechnik, der Datenbanktechnik und der Künstlichen Intelligenz immer deutlicher werden und der *hybride* Systementwurf die informationstechnischen Ansätze zu vereinen sucht.

Als das Gemeinsame zwischen der Anforderungsanalyse in der Softwaretechnik [Wedekind, 1992], der Modellierung von „Miniwelten“ in der Datenbanktechnik [Encarnação & Lockemann, 1990] und der Wissensrepräsentation in der Künstlichen Intelligenz [Brachman & Levesque (Hrsg.), 1985] gilt die Methode des *konzeptionellen Modellierens* [Brodie *et al.* (Hrsg.), 1984]. In jüngster Zeit stellt man der Methode das Attribut „objektorientiert“ bei — in der Hoffnung, im Objekt-Paradigma die Synergie der Ansätze zu finden [Dillon & Tan, 1993; Fiadeiro & Sernadas, 1991; Quibeldey-Cirkel, 1994a]. Im folgenden soll der allgemeine Entwurf technischer Objekte als psychologisches, insbesondere als *kognitives* Problem untersucht werden. Wir wollen ermessen, was das Objekt-Paradigma zum Problemlösen beitragen kann. Zugleich beantworten wir die Fragen: Warum sind objektorientierte Konzepte und Methoden intuitiv? Warum erfahren sie eine so breite Akzeptanz in der Informatik?

Umgangssprachlich sind die Parallelen offenkundig: „Probleme tauchen auf, stellen sich ein — Lösungen werden *entworfen*“ [Duden, 1970]. Entwerfen ist also ein schöpferischer und planvoller Akt des Problemlösens, der neben der Intelligenz des Entwerfers vor allem seine Kreativität voraussetzt. Im Gegensatz zum schöpferischen *Problem*lösen steht das unschöpferische *Aufgaben*lösen, das sich beschränkt auf ein *reproduktives* Konstruieren mit Hilfe von Algorithmen, Regelwerken oder Rezepten. Lösungsweg und Lösungsmittel sind vorgegeben und müssen lediglich aus dem Gedächtnis abgerufen werden. Wie beides

— Aufgaben- und Problemlösen — als *kognitive Operationen* in der mentalen Informationsverarbeitung begriffen wird, lehrt die Denkpsychologie unserer Tage. Wir werden ihre Erkenntnisse skizzieren und im Sinne von Donald NORMANs *Cognitive Engineering* die kognitiven Parallelen zum ingenieurmäßigen Entwerfen ziehen [Norman, 1986]. Wir grenzen unsere Überlegungen und Beispiele auf den Software- und VLSI-Entwurf ein, was nicht zwingend ist, unser Fach und die Kürze der Darstellung aber gebieten. Im übrigen ist die Reduktion des Problembegriffs auf *technische* Probleme nicht nur in den Systemtheorien verbreitet [Athey, 1982], sondern auch in der Intelligenzforschung [Lindsay & Norman, 1972; Newell & Simon, 1972].

Übersicht: Zunächst skalieren wir die allgemeine „Zielentfernung“ zwischen der Problemerkenntnis und der technischen Problemlösung und deuten die Orientierung des Ingenieurs im Problemlöseprozeß. Abschnitt 7.1 definiert dann den Problembegriff und nennt die darauf bezogenen Problemtypen. Im Abschnitt 7.2 sammeln und ordnen wir die Erkenntnisse der Kognitionspsychologie, um die Parallelen zum Objekt-Paradigma aufzuzeigen. Der Abschnitt 7.3 setzt den Vorgang des Entwerfens, verstanden als Problemlösen, in das kybernetische Wechselspiel von „Schema und Korrektur“. Die wichtigste Bedeutung kommt schließlich dem Abschnitt 7.4 zu. Hier eruieren wir das Kreativpotential des Objekt-Paradigmas auf der Grundlage der aufgezeigten psychologischen Erkenntnisse.

Zielentfernung

Angelehnt an die Entwurfstheorie nach Dieter HERRIG können wir den Entwurf technischer Objekte als die prozedural-instrumentelle Überwindung einer „Zielentfernung“ auffassen [Herrig, 1982]. Danach besteht das Ziel jeder ingenieurmäßigen Handlung in der technischen Einwirkung auf die natürliche oder künstliche Umgebung. Die Zielentfernung e, die Mittelbarkeit der Einwirkung, läßt sich skalieren:

$\star$	$e = 0$:	unmittelbarer technischer Einfluß auf Mensch und Umgebung (Erfahrungsebene)
$\uparrow$	$e = 1$:	strategischer Entwurf (Verfahrenstechniken, um die Erfahrungsebene zu beeinflussen)
$\uparrow$	$e = 2$:	instrumenteller Entwurf (Werkzeuge, um die Verfahrenstechniken zu unterstützen)
$\uparrow$	$e = 3$:	konstruktiver Entwurf (Verfahrenstechniken, um Werkzeuge herzustellen)
$\vdots$		
$\uparrow$	$e \to \infty$:	Problemerkenntnis (Gesamtentfernung zwischen Wunsch und Wirklichkeit)

Ein technisches Problem wird seiner Lösung näher gebracht, indem man von einer Ebene des Entwerfens auf die nächsthöhere wechselt. Dabei lösen sich der Entwurf von Operationen (Verfahren) und der Entwurf von Operatoren (Werkzeugen) ab. Das Umsetzen eines Sandhaufens zum Beispiel setzt eine Strategie des Transportierens voraus (Einsatz von

Transportmitteln, Festlegung des Weges). Um die Strategie auszuführen, müssen Werkzeuge verfügbar sein (Schubkarre und Schaufel). Sind diese nicht verfügbar, müssen sie entworfen werden. Dazu sind wiederum Strategien erforderlich (Einsatz von Konstruktionswerkzeugen und Materialien zur Herstellung von Schubkarren und Schaufeln) und so weiter. Ganz zu Beginn steht natürlich der Wunsch oder das Bedürfnis, den Ort des Sandhaufens zu wechseln.

Problemlösen umfaßt also *intensionale* (zielgerichtete, zweckgebundene), *selektive* (auf einzelne Aspekte beschränkte) und *prozessuale* (mehrere Schritte umfassende) Denk- und Handlungsmuster für die aktive Suche nach Lösungen [Spinner, 1987]. An der erkenntnistheoretischen Stelle — den kognitiven Denkmustern — wollen wir ansetzen. Die Handlungsmuster des praktischen Software- und VLSI-Entwurfs sind hinlänglich bekannt [Rammig, 1989; Sommerville, 1990].

Auf den zeitgenössischen Ingenieur wirkt es im allgemeinen befremdend, die Erkenntnisse der Denkpsychologie auf das Problemlösen in seinem Metier zu übertragen; sind es doch die technischen Möglichkeiten, die ihn vornehmlich interessieren. Sein Blick ist auf die technische Verkürzung der Zielentfernung gerichtet: $e \to 0$. Im Idealfall ist dies der *automatisierte Entwurf*. Über seinen namengebenden *Genius* zerbricht er sich seltener den Kopf. Auch sind ihm die „weichen" Methoden der Denkpsychologen nicht unbedingt vertraut, mitunter sogar suspekt: „Selbstbeobachtung" und „lautes Denken" gelten ihm als vorwissenschaftlich. Zumindest zielen sie in die entgegengesetzte Richtung: $e \to \infty$. Vertrauter ist ihm da schon die statistische *Metrik* zur Auswertung empirischer Versuchs- und Kontrollreihen, eine Metrik, die auf George MILLER zurückgeht.

Es war der Psychologe MILLER, der die *informationstheoretische* Metrik des Mathematikers Claude SHANNON in die statistische Psychologie einführte [Miller, 1956; Shannon & Weaver, 1949]. Seiner „Magischen Zahl Sieben" fügte MILLER das *Bit* bei, das SHANNON-Maß für den Informationsgehalt. Das dimensionsbehaftete und somit unvergleichbare Maß der *Varianz* ist seitdem aus dem psychologischen Fachjargon verschwunden (siehe Abschnitt 2.1). Dadurch wurde der Weg frei für eine gegenseitige Befruchtung zwischen Psychologie und Informatik [Schauber & Tauber (Hrsg.), 1982], vor allem zwischen der Denkpsychologie (*Problemlösen als Informationsverarbeitung* [Dörner, 1976]) und der Künstlichen Intelligenz (*The Society of Mind* [Minsky, 1986]). Die Geschichte der Wissenschaften kennt viele Fälle interdisziplinärer Impulse. Dieser hier findet seinen Ausdruck in der modernen Kognitionswissenschaft (*The Mind's New Science: A History of the Cognitive Revolution* [Gardner, 1985]).

7.1 Was ist ein Problem?

Der Denkpsychologe Dietrich DÖRNER hat ein einfaches Konzept vorgeschlagen, das sich für die Klassifikation von Problemen eignet. Auch die Aspekte des Soft- und Hardware-Entwurfs fügen sich zwanglos in sein Konzept. Wir stellen es unserer erkenntnistheoretischen Betrachtung des Entwerfens voran.

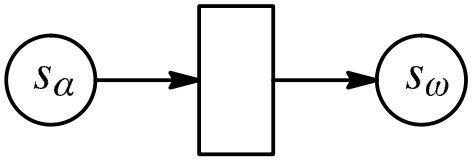


Bild 7.1: Problemkomponenten

Nach DÖRNER ist ein Problem gekennzeichnet durch drei Komponenten (Bild 7.1):

- unerwünschter Anfangszustand s_α
- erwünschter Endzustand s_ω
- Barriere in der Überführung von s_α in s_ω
  (kognitive und technische Hindernisse)

Eine *Aufgabe* unterscheidet sich von einem *Problem* durch das Fehlen der Barriere: Für die Bewältigung einer Aufgabe ist die Lösungsmethode bekannt; nur reproduktives Denken ist gefordert. Ob es sich also um ein Problem oder um eine Aufgabe handelt, hängt von der subjektiven Vorerfahrung des Entwerfers ab. Für einen erfahrenen CAD-Konstrukteur ist der räumliche Entwurf kein Problem, sondern eine Aufgabe. Für den Laien indes ist die 3D-Darstellung ein äußerst schwieriges Problem. An der Art der Barriere, die eine unmittelbare Transformation des Anfangs- in den Zielzustand im Moment verhindert, lassen sich die Probleme und deren Überwindung typisieren. DÖRNER nennt drei Problemtypen:

7.1.1 Interpolationsprobleme

Man weiß, was man will, und kennt die Mittel, um den wünschenswerten Zustand zu erreichen. Das Problem liegt in der richtigen *Kombination der Mittel*, in der Interpolation zwischen Anfangs- und Zielzustand. Die Anzahl der möglichen und jeweils zu überprüfenden

Transformationsketten ist im allgemeinen aber sehr groß: Bei anspruchsvollen Problemen, wie dem Schachspiel oder der kürzesten Rundreise des Handlungsreisenden, ist die Interpolationsbarriere selbst mit massivem Rechnereinsatz unüberwindbar (NP-vollständig, kombinatorische Explosion). Maschinelles Kombinieren und Ausprobieren der verfügbaren Mittel ist nur für eingeschränkte und wohlstrukturierte Anwendungen wirtschaftlich. Das gilt zum Beispiel für den *Silicon-Compiler*, eingeschränkt auf den automatischen Digitalfilter-Entwurf [Quibeldey-Cirkel, 1987], oder den von NEWELL, SHAW und SIMON programmierten *General Problem Solver* (GPS), eingeschränkt auf die Simulation mentaler Suchoperationen [Newell & Simon, 1972].

7.1.2 Syntheseprobleme

Man weiß von Beginn an, oder vermutet nach vergeblichen Lösungsanstrengungen, daß die bekannten Mittel nicht oder nicht allein hinreichend sind, um das Problem zu lösen. Problem des Alchimisten: Wie gewinne ich Gold aus Blei? Anfangs- und Endzustand sind bekannt, unbekannt oder unbewußt sind wichtige Einzeloperationen, kombiniert mit den bekannten. Das Problem liegt in der Zusammenstellung — der Synthese — eines geeigneten Inventars an Operatoren. Gelernte Einstellungen und Denkgewohnheiten können ursächlich für Synthesebarrieren sein. Denksportaufgaben enthalten in der Regel derartige Barrieren. Bild 7.2 zeigt ein Beispiel.

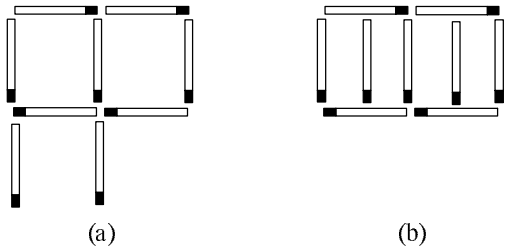


Bild 7.2: Beispiel eines Syntheseproblems

Die Streichholzkonfiguration (a) soll durch eine Lageveränderung zweier Hölzer in eine Konfguration aus drei gleich großen Quadraten überführt werden. Viele Versuchspersonen scheitern, weil sie die Möglichkeit überlappender Quadrate (b) von vornherein außer acht lassen. Die Synthesebarriere liegt hier in der mentalen Voreinstellung der Versuchsperson: Sie sucht die Lösung ausschließlich in der „überlappungsfreien“ planaren Geometrie. Der allgemeine VLSI-Entwurf, der sich nicht in einer technischen Umsetzung erschöpft (zum Beispiel die Umsetzung einer diskreten Schaltung in eine integrierte), ist hauptsächlich ein Syntheseproblem. Das Operatorinventar ist offen: für eine synthetische Lösung muß es um weitere Operatoren ergänzt werden.

7.1.3 Dialektische Probleme

Man weiß, daß die gegebene Situation verändert werden muß, kennt aber den Zielzustand nur vage oder gar nicht. Allenfalls sind globale Kriterien für den wünschenswerten Endzustand bekannt, im allgemeinen Komparative wie „schneller“ oder „benutzungsfreundlicher“. Die Lösung kann nur *dialektisch* erreicht werden: Der Entwurf wird auf Widersprüche überprüft. Je offener das Problem hinsichtlich seines Zielzustands ist, desto mehr muß man versuchen, durch zyklisches Erzeugen und Überprüfen von Alternativen die Zielvorstellung zu präzisieren. So werden in der Softwaretechnik die dialektischen Barrieren beispielsweise durch *exploratorisches Prototyping* überwunden.

Der Bekanntheitsgrad der Mittel und die Klarheit der Zielkriterien erlauben eine Kreuztabellierung der drei Barrieretypen. Tabelle 7.1 zeigt die Ausprägungen in beiden Dimensionen [Dörner, 1976, S. 14].

		Klarheit der Zielkriterien	
		hoch	*gering*
Bekanntheitsgrad der Mittel	*hoch*	Interpolationsbarriere	dialektische Barriere
	gering	Synthesebarriere	dialektische und Synthesebarrieren

Tabelle 7.1: Klassifikation der Barrieretypen

In der Praxis des Soft- und Hardware-Entwurfs treffen wir auf komplexe Probleme mit *heterogenen* Barrieren. Auch ist die Art der Barriere abhängig vom Erfahrungs- und Wissensstand des Problemlösers: Was für den Laien ein Syntheseproblem darstellt, wegen seiner geringen Kenntnis der Lösungsmittel, ist für den Experten möglicherweiser ein Problem der Interpolation, also der Kombination bekannter Mittel. Im kreativen und rationalen Prozeß des Problemlösens nimmt das verfügbare Wissen des Problemlösers die Schlüsselrolle ein. Die geistige Lösungskompetenz ist ausschlaggebend für die Überwindung der dialektischen, Interpolations- und Synthesebarrieren. Auf die kognitiven, das heißt erkenntnis- oder wissensmäßigen Aspekte gehen wir nun näher ein.

7.2 Kognitive Strukturen

Die Arbeitshypothese der Kognitionsforschung lautet: „Menschliches Denken ist Informationsverarbeitung.“ Daraus wurde die These der *Homomorphie* zwischen Mensch und

Maschine abgeleitet, das heißt der strukturellen Ähnlichkeit zwischen den elementaren kognitiven Prozessen, die der menschlichen Informationsverarbeitung zugrunde liegen, und den elementaren Lese-, Schreib-, Lösch-, Erzeuge-, Vergleichs- und Übertragungsprozessen in Rechnersystemen [Fischer, 1983]. Diese These findet ihre logische Konsequenz in den Arbeiten von Allen NEWELL und Herbert SIMON:

> „*The Physical Symbol System Hypothesis.* A physical symbol system has the necessary and sufficient means for general intelligent action."
>
> „*Heuristic Search Hypothesis.* The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search — that is, by generating and progressively modifying symbol structures until it produces a solution structure." [Newell & Simon, 1975]

Wir lassen hier den strittigen KI-Aspekt der Simulation menschlicher Intelligenz unbewertet und greifen nur den Struktur-Aspekt auf. DÖRNER faßt ihn als Konsens der kognitiven Psychologie zusammen. Danach sind zwei Gedächtnisstrukturen die Voraussetzung für das Lösen eines Problems:

> „Der Problemlöser braucht erstens ein Abbild des Realitätsbereichs, in dem das Problem lokalisiert ist, und zweitens muß er über heuristische Strategien verfügen. Er benötigt eine *epistemische* und eine *heuristische* Struktur." [Dörner, 1976, S. 116]

Es handelt sich um Wissensstrukturen, also um kognitive Strukturen, die miteinander interagieren: eine Wissensbasis (Fakten und Methoden) und eine Instanz zur Wissensverarbeitung. Die *epistemische* Struktur (griechisch episteme = Wissen) besteht aus einem *Afferenzteil*, dem Gedächtnis für Sachverhalte eines Realitätsbereichs, und einem *Efferenzteil*, dem Gedächtnis für Operatoren und Handlungen, mit deren Hilfe sich die Sachverhalte umwandeln lassen. Die *heuristische* Struktur der Wissensverarbeitung fungiert als Verfahrensbibliothek, die problem- und situationsabhängig Konstruktionsverfahren zur Verfügung stellt. Ihre Komponenten sind also Heuristiken (= Findeverfahren[1]). Ohne sie stünde dem Menschen als Konstruktionsverfahren nur das *ungeleitete* Probieren über der epistemischen Struktur zur Verfügung: Das Problemlösen reduzierte sich auf ein kombinatorisches Aufgabenlösen. „Träumen" und „Phantasieren" sind gleichsam ungeleitete Prozesse über der epistemischen Struktur (sie können aber heuristisch bedeutsam sein!).

Zwei Aspekte wollen wir für unsere Diskussion der psychologischen Parallelen zum Objekt-Paradigma hervorheben: zum einen den in der epistemischen Struktur abgebildeten *Realitätsbereich* eines Problems und zum anderen die elementaren *kognitiven Operationen*, aus denen sich alle Heuristiken zusammensetzen.

[1] Man assoziiere den Ausspruch ARCHIMEDES' im Badezuber, als ihm beim Überschwappen des Badewassers die Idee kam, über die Wasserverdrängung einer Goldkrone ihren Goldgehalt zu bestimmen: „*Heureka!*"

7.2.1 Kognitive Abbildungen

Ein Realitätsbereich ist durch zwei Mengen charakterisiert: *Sachverhalte* und *Operatoren* zur Umwandlung von Sachverhalten. Formal bilden sie die Komponenten eines *Netzes*. Die Sachverhalte bilden die Knoten im Netz, die Operatoren die Übergänge. Am Realitätsbereich „Schachspiel" erläutert, bilden die regelhaften Züge die Menge der Operatoren und die möglichen Konstellationen der Schachfiguren auf dem Feld die Menge der Sachverhalte. Problemlösen innerhalb eines so definierten Realitätsbereichs ist das Suchen nach dem optimalen oder im Sinne von SIMON „zufriedenstellenden" Weg im Labyrinth der möglichen Wege (die Logik des Suchens wurde in Abschnitt 5.1.1 vertieft). Die Suchstrategie richtet sich dabei nach den Eigenschaften des Realitätsbereichs. Diese werden wiederum durch die Eigenschaften seiner Sachverhalte und Operatoren bestimmt. Bei statischen Realitätsbereichen, also solchen, die sich nur durch den Eingriff des Problemlösers verändern, ist die kombinatorische Komplexität die bestimmende Größe, zum Beispiel die schon zitierten 10^{120} möglichen Partien beim Schachspiel.

Nehmen wir den Realitätsbereich der technischen Entwurfsdatenbanken [Encarnação & Lockemann, 1990], um das Gesagte weiter zu verdeutlichen. Er wird im allgemeinen „Miniwelt" oder „Weltausschnitt" genannt. Die möglichen Sachverhalte der Miniwelt werden durch das konzeptionelle Datenbankschema formal beschrieben (Entity-Relationship-Modelle). Die Menge der Operatoren, um einen Sachverhalt umzuwandeln, ist durch die erlaubten Datenbanktransaktionen bestimmt. Handelt es sich beispielsweise um eine VLSI-Entwurfsdatenbank, so besteht das Problemlösen, also das Entwerfen, in der Suche nach geeigneten Transaktionsfolgen, um die Spezifikationsdaten des zu entwerfenden Chips letztlich in geometrische Layoutdaten zu transformieren.

Sachverhalte können dabei CAD-Daten auf verschiedenen VLSI-Abstraktionsebenen und unter verschiedenen Sichten umfassen (siehe Tabelle 1.5 auf Seite 31 oder das Y-Diagramm auf Seite 127). Als Operatoren zur Umwandlung der Sachverhalte steht dem Entwerfer ein umfangreiches Instrumentarium zur Verfügung: generierende und verifizierende Werkzeuge, wie Layoutrouter und Layoutextraktoren. Elementare Datenbanktransaktionen wären zum einen das Erzeugen und Modifizieren von Datenbankobjekten, beispielsweise von Simulationsmodellen für Register-Transfer- (RT) oder Logikzellen, und zum anderen das Etablieren von Objektbeziehungen, den Netzlisten der RT- und Logikzellen. Komplexe Transaktionen (Makro-Operationen im Sinne von DÖRNER) wären zum Beispiel die grafische Darstellung des Datenbankinhalts durch Schaltungs- und Layouteditoren oder die Simulation des Datenbankinhalts auf bestimmten Abstraktionsebenen durch Schaltkreis-, Logik- und RT-Simulatoren.

Soviel zum Begriff des Realitätsbereichs. Die Informationsverarbeitung beim Problemlösen ist untrennbar mit dem Gedächtnis verknüpft, denn die erforderliche kognitive Abbildung des Realitätsbereichs ist eine Leistung unseres Gedächtnisses. Interessant am Zusammen-

hang zwischen Problemlösen und Gedächtnis ist besonders die Frage nach der Strukturähnlichkeit: Findet die Struktur eines Realitätsbereichs, zum Beipiel die datenbanktechnische Abbildung der VLSI-Miniwelt als netzartiges konzeptionelles Schema, ihr Pendant im kognitiven Bereich? In der Tat zeigen die Ergebnisse der Gedächtnisforschung eine frappante Ähnlichkeit: Nach Donald NORMAN ist die Grundstruktur unseres Langzeitgedächtnisses das „aktive semantische Netz“ (zitiert in [Dörner, 1976, S. 29]).

An dieser Stelle tritt eine Erkenntnis zu Tage, die in der Informatik in Vergessenheit geraten ist: Es war die kognitive Psychologie [Reitman, 1965], insbesondere die Gedächtnisforschung [Quillian, 1967], die erstmals den Begriff des „semantischen Netzes“ formulierte und ihn zur Arbeitshypothese über die Struktur unseres „kognitiven Apparats“ erhob. Die Brauchbarkeit dieser Hypothese wurde alsbald in der Künstlichen Intelligenz [Brachman & Levesque (Hrsg.), 1985] und der Datenbankforschung [Hull & King, 1987] erkannt, wo das semantische Netz als universelles (technisches) Schema übernommen wurde.

Das Gedächtnisbild eines Realitätsbereichs besteht aus verknüpften Gedächtnisinhalten. Drei allgemeine Formen der Verknüpfung lassen sich nach DÖRNER unterscheiden: die *Konkret-Abstrakt*-Relation (das Flipflop *ist ein* Speicherelement), die *Ganzes-Teil*-Relation (der Zentralprozessor *besteht aus* Rechen- und Leitwerk) und *raum-zeitliche* Relationen (die Simulation auf Transistorebene *folgt auf* die Extraktion des Layouts).

Ein Denkprozeß kann nun als „interpretativer Prozeß“ modelliert werden [Lindsay & Norman, 1972]. Das semantische Netz der Gedächtnisinhalte bildet die Datenbasis. Der interpretative Prozeß besteht darin, daß bestimmte Knoten im Netz ausgewählt und für die Informationsverarbeitung verfügbar gehalten werden. Die Knoten wirken dabei als „Ösen“, in die sogenannte „Informationshaken“ greifen können. Die Zahl der Haken ist durch die Kapazität unseres Kurzzeitgedächtnisses auf die MILLERsche Größenordnung von „7 ± 2“ begrenzt. Die an diesen Haken festgemachten Netzknoten bilden den Inhalt des Kurzzeitgedächtnisses. Der interpretative Prozeß sucht die Umfelder der ausgewählten Knoten ab und verarbeitet sie. Beispiele für einen interpretativen Prozeß sind die Prüfung, ob ein bestimmter Knoten im Umfeld eines anderen liegt, oder das Aufgreifen eines Knotens, der durch eine bestimmte Relation mit einem anderen verknüpft ist.

Heuristische Suchprozesse im netzartigen Labyrinth des Langzeitgedächtnisses laufen über die Informationshaken des Kurzzeitgedächtnisses ab, und zwar komplex und parallel. DÖRNER zeichnet für die Art der Verknüpfung zwischen der epistemischen (ES) und der heuristischen Struktur (HS) ein einprägsames Bild:

> „Die Beziehungen zwischen HS und ES sehen ungefähr so aus, wie die Beziehungen zwischen einem Tintenfisch und einem Fischernetz, welches dieser in seinen Tentakeln hält. Das Fischernetz entspricht der ES, und die Knoten des Netzes, die gerade von den Tentakeln gehalten werden, sind die Inhalte des Kurzzeitgedächtnisses (der Tintenfisch hat gerade acht Tentakeln, was der Kapazität des Kurzzeitgedächtnisses ganz gut entspricht). Wenn der

> Tintenfisch sich an dem Netz mit seinen Tentakeln entlangtastet, indem er jeweils von den Knotenpunkten ausgehend, die er festhält, andere Knotenpunkte aufnimmt, dann entspricht dieser Vorgang den heuristischen Prozessen." [Dörner, 1976, S. 37]

DÖRNER hat seine „Tintenfischhypothese" auch mathematisch formuliert. Wir wollen an dieser Stelle nur die zwei Grundannahmen der Gedächtnisforschung herausstellen, an die wir später wieder anknüpfen werden:

1. Die kognitive Struktur unseres Langzeitgedächtnisses ist das semantische Netz.
2. Die semantischen Relationen lassen sich klassifizieren in Konkret-Abstrakt-, Ganzes-Teil- und raum-zeitliche Relationen.

7.2.2 Kognitive Operationen

Welche Elementarprozesse konstituieren die Informationsverarbeitung beim Problemlösen? Bestehen Denkakte in der Verkettung bestimmter kognitiver Operationen? Die Beantwortung der Fragen zeigt Möglichkeiten auf, wie die Fähigkeit des Individuums, Probleme zu lösen, verbessert werden kann und welche Lösungshilfen hierfür geeignet sind.

Der Versuch, das Denken auf seine Grundelemente zurückzuführen, ist nicht neu: Auf ARISTOTELES geht der *Assoziationismus* zurück. Aus dessen Sicht bestehen kognitive Prozesse darin, daß Gedächtnisinhalte sich wechselseitig aufgrund ihrer Verknüpfungen aufrufen. Dabei unterscheidet man zwischen Assoziationen aufgrund der Gleichzeitigkeit kognitiver Vorgänge, der Ähnlichkeit (Ratte-Maus), des Kontrastes (heiß-kalt) und des räumlichen Zusammenseins (Stuhl-Tisch). Assoziationen breiten sich nach den genannten Prinzipien nicht ziellos aus, sondern unterliegen gewissen aktivierenden und hemmenden Konstellationen. Im Kontext „Das moderne Eingabemedium eines Rechners ist die Maus" gehen die Assoziationen zu „Maus" in eine andere Richtung als in jedem tierbezogenen Kontext.

Wenn es um die operativen Grundelemente des Problemlösens geht, erweist sich der undifferenzierte Standpunkt des Assoziationismus als wenig fruchtbar. Die Suche nach einer Grundmenge kognitiver Elementaroperationen, erschöpfend und disjunkt zugleich, ist bislang, wie DÖRNER eingesteht, erfolglos. Die bisher gewonnenen Erkenntnisse der kognitiven Informationsverarbeitung, die sich mehr phänomenologisch orientieren, sind dennoch für unsere Diskussion von großer Bedeutung. Sie machen uns die vielbeschworene *Intuitivität* der objektorientierten Konzepte plausibel. Nach H. LOMPSCHER können unsere geistigen Fähigkeiten zurückgeführt werden auf wenige elementare Operationen (zitiert in [Dörner, 1976, S. 112]):

1. *Zergliedern* eines Sachverhalts in seine Teile
2. *Erfassen der Eigenschaften* eines Sachverhalts
3. *Vergleichen* von Sachverhalten hinsichtlich der Unterschiede und Gemeinsamkeiten
4. *Ordnen* einer Reihe von Sachverhalten hinsichtlich eines oder mehrerer Merkmale
5. *Abstrahieren* als Erfassen der in einem bestimmten Kontext wesentlichen Merkmale eines Sachverhalts und Vernachlässigen der unwesentlichen Merkmale
6. *Verallgemeinern* als Erfassen der einer Reihe von Sachverhalten *gemeinsamen* und *wesentlichen* Eigenschaften
7. *Klassifizieren* als Einordnen eines Sachverhalts in eine Klasse
8. *Konkretisieren* als Übergang vom Allgemeinen zum Besonderen

Nach der Auffassung von LOMPSCHER sind diese Operationen grundlegend für jede geistige Tätigkeit. Sie treten aber nur selten isoliert auf, im allgemeinen sind sie die Komponenten komplexer Prozesse, die einander bedingen und durchdringen. Als Beispiel nennt LOMPSCHER das *Definieren*. Oberbegriff und wesentliches Unterscheidungsmerkmal — die klassischen Elemente einer Definition: „Eine Straßenbahn ist ein motorgetriebenes, schienengebundenes Verkehrsmittel“ — gehen hervor aus den Teilprozessen des Abstrahierens, Klassifizierens, Verallgemeinerns und weiterer Operationen. Allen gemeinsam ist das Erfassen von Eigenschaften und das Ausgliedern von Teilen. Unsere Umgangssprache kennt eine Vielzahl komplexer geistiger Operationen höherer Ordnung, wie „sich besinnen“ oder „eine Idee haben“. Sie lassen sich mehr oder weniger eindeutig auf eine Verknüpfung der LOMPSCHERschen Operationen zurückführen.

Heuristiken, verstanden als geistige Programme zur Lösungsfindung, setzen sich gleichsam aus Verknüpfungen kognitiver Operationen zusammen. Hier lassen sich zwei Prozeßgruppen unterscheiden: Veränderungs- und Prüfprozesse. Die Grundeinheit ihrer Organisation ist von zentraler Bedeutung, spiegelt sie sich doch auch außerhalb unseres „kognitiven Apparats“ in den Handlungsmustern des Entwerfens wider: *TOTE* (*Test-Operate-Test-Exit*). Das Flußdiagramm in Bild 7.3 beschreibt das Prinzip.

Die simple Heuristik für das Einschlagen eines Nagels zum Beispiel folgt dem TOTE-Bauplan. *Test*: Ist der Nagel tief genug eingeschlagen? Wenn nein, *Operate*: Erneuter Hammerschlag und *Test*. Wenn ja, *Exit*: Einschlagen beenden. Dieses Grundprinzip der Organisation von Denk- und Handlungseinheiten wurde bereits von Norbert WIENER als kybernetisches Universalschema erkannt [Wiener, 1948]. In den Wissenschaften hat es zahlreiche Namen erfahren, wir wollen es für unsere Diskussion des Entwerfens allgemein formulieren.

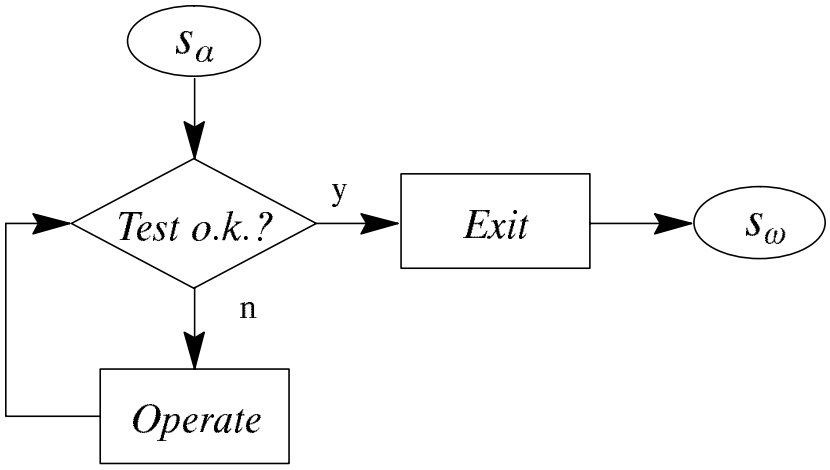


Bild 7.3: Grundeinheit einer Heuristik

7.3 Schema & Korrektur

In der Wissenschaftstheorie heißt das kybernetische Prinzip nach Karl POPPER „Vermutungen und Widerlegungen“ [Popper, 1963], in der Künstlichen Intelligenz nach Herbert SIMON „Generator-Test-Zyklus“ [Simon, 1982] oder im Software- und VLSI-Entwurf nach Karl KLINGSHEIM „ASE-Paradigma“ (Analyse-Synthese-Evaluation) [Klingsheim, 1990]. Weitere Begriffspaare lassen sich zwanglos anfügen, wie „Theorie und Kritik“, „Hypothese und Verwerfung“ oder „Versuch und Irrtum“. Allen kybernetischen Annäherungsschemata gemeinsam ist die TOTE-Heuristik: Ausgangsschema → erster Lösungsversuch → Kritik und Korrektur → korrigiertes Schema → zweiter Lösungsversuch und so fort bis zur akzeptierten Lösung. Im Zusammenspiel von „Schema & Korrektur“ wechselt das Problemlösen zwischen der Ebene der mentalen Operatoren und der pragmatischen Ebene der Entwurfswerkzeuge.

7.3.1 Was ist ein Schema?

Im allgemeinen handelt es sich um eine geistige, das heißt kognitive Vorlage, um die Information über ein Problem zu gestalten und zu gliedern. Ein Schema[2] kann somit als *Raster* mit bestimmten Ordnungsmustern aufgefaßt werden. In dieses Raster wird die Informa-

[2] Der Schemabegriff im psychologischen Verständnis geht auf die Gestalttheorie zurück [Bartlett, 1932] und meint dort *stereotype Erinnerungsmuster*. Marvin MINSKY hat ihn als *Frame* in die KI eingeführt [Minsky, 1981].

tion nebenordnend (parataktisch) oder über- und unterordnend (hypotaktisch) eingefügt. „Die wissenschaftliche Denk- und Darstellungsform ist ausgeprägt hypotaktisch“ [Spinner, 1987, S. xi]. Verglichen mit der Hypotaxe (Systeme und Hierarchien) stellt die Parataxe eine schwache Ordnungsform dar (Listen- und Mengen). Fehlt jegliches Schema, so liegt ein *Gemenge* vor, eine unstrukturierte zufällige Anhäufung aus Vermutungen, Fakten und Methoden aller Art. Eine solche Informations*ballung* wirkt auf den Menschen diffus und komplex. Für die Informationsverarbeitung ist sie ausgesprochen kontraproduktiv.

Der Schemabegriff ist zum Beispiel in der Datenbanktechnik allgegenwärtig: Hier hat man sich auf ein Drei-Schema-Konzept für den Datenbankentwurf geeinigt: konzeptionelles, externes und internes Schema [Schlageter & Stucky, 1983]. Der Entwurf reduziert sich so auf die Transformationen zwischen den verschiedenen Schemata (siehe Abschnitt 3.2.1).

„Schema & Fixierung“ wäre das zu „Schema & Korrektur“ gegenteilige Organisationsprinzip für Denk- und Handlungsmuster. Eine Fixierung auf vorgefertigte Entwurfsschemata unterdrückt jede Kreativität. Aber auch aus pragmatischer Sicht gilt es, Fixierungen zu vermeiden: Denn der erste Lösungsversuch ist mit Sicherheit falsch, sonst wäre er schon die vorweggenommene Lösung, und es läge kein Problem, sondern eine reproduktive Aufgabe vor. Die „tierische Prägung“ im Sinne der Verhaltensforschung nach Konrad LORENZ oder die menschliche „Dogmatisierung“ im Sinne autoritärer Ideologien sind Beispiele für Schemata *ohne* nachfolgende Korrektur: *Schablonen*. Eine Anpassung an abweichende Erfahrungen oder Erkenntnisse findet nicht statt.

7.3.2 „Making-Matching-Remaking“

Das Argumentationsverfahren von „Vermutungen und Widerlegungen“ wurde vom Kunsthistoriker Ernst GOMBRICH auf das Problem des bildlichen Entwerfens übertragen [Gombrich, 1978]. Jede Art, die Wirklichkeit darzustellen, ist ein aktiver rückgekoppelter Prozeß, der durch einen Zyklus aufeinanderfolgender Phasen gekennzeichnet ist (in Klammern stehen Beispiele für technische Bezüge):

1. *Making*: Vorkonstruktion eines kognitiven Schemas (fachliches Grobkonzept, konzeptionelles Modell, Blockschaltbild)

2. *Matching*: Interpretation der Wirklichkeit durch Einordnung der Information in das Schema (Prototyping, Simulation)

3. *Remaking*: Nachkonstruktion des Ausgangsschemas unter korrigierender Anpassung an die Wirklichkeit (Feinkonzept, Blockauflösung) oder, wenn eine Korrektur wirtschaftlich untragbar scheint, alternatives Ausgangsschema durch Ansatzwechsel

Zwei Akzente trägt dieses Phasenmodell: (a) Es betont die Priorität: *making* kommt vor *matching*. Erst in einem kognitiven Ordnungsschema, durch dessen Raster die Wirklichkeit (Weltinformation) aufgenommen und strukturiert wird, so grob und fehlerhaft dies auch im ersten Ansatz geschehen mag, entwickelt sich die Erkenntnis. (b) Das Phasenmodell betont den „*aktiven* Charakter aller Erkenntnis, die wir demnach immer ‚machen' und beständig ‚neumachen' müssen" [Spinner, 1987, S. xii].

Bild 7.4: „Wie man eine Katze zeichnet"

Der Annäherungsprozeß fortgesetzter Rekonstruktion (durch Variieren des Ausgangsschemas) dauert so lange, bis Darstellung und Dargestelltes in den artbildenden Eigenschaften übereinstimmen. Bild 7.4 zeigt den bildlichen Darstellungsvorgang im Zusammenspiel von „Schema & Korrektur" an einem Beispiel von GOMBRICH. Im Wechsel der Deutungen aufgrund neuer (Gegen-)Information zu den bisherigen Auslegungen entwickelt sich die beabsichtigte Darstellung als Folge zunehmend komplexerer Schemata, bis schließlich die Lösung, die eindeutige Erkennbarkeit des Objekts, erreicht ist.

Das *exploratorische Prototyping* folgt in gleicher Weise dem Vorgehensmodell *making-matching-remaking*, wie es GOMBRICH für den bildlichen Darstellungsvorgang analysiert hat. Das Wechselspiel von „Schema & Korrektur" bestimmt die Interaktion zwischen Kunden und Entwerfer. Das beabsichtigte Objekt ist hier die Benutzungsoberfläche für die gewünschte Anwendung. In der dialogischen Rückkopplung zwischen Entwerfer und Kun-

den werden die zuvor unscharfen Anforderungen des fachlichen Konzepts ausdifferenziert und zugleich am Prototypen erprobt.

Die Entstehung dieses Kapitels kann als ein weiteres Beispiel für das Wechselspiel von „Schema & Korrektur“ betrachtet werden: Der Text ist das Ergebnis eines iterativen Zyklus von *Textproduktion* (*making*: Formulierung der Gedanken: „Entwerfen = Problemlösen?“ „Die Objektorientierung setzt bei $e \rightarrow \infty$ an und reicht bis $e \rightarrow 0$“), *Textrezeption* (*matching*: Literaturauswertung, Kenntnisse und Erfahrungen aus Soft- und Hardwareprojekten) und *Textrevison* (*remaking*: erweitern, eingrenzen, korrigieren).

Die bis hier zusammengetragenen und im Lichte des Entwerfens geordneten Hypothesen und Erkenntnisse sollen die Intuitivität des Objekt-Paradigmas verdeutlichen. Viel ist bereits veröffentlicht worden zu den methodischen und pragmatischen Vorteilen objektorientierter Konzepte. Die Zahl der Publikationen steigt exponentiell (siehe Bild 0.2 auf Seite 5). Wenig wurde bisher geschrieben über die *missing link* zwischen den kognitiven Prozessen während des Entwerfens und dem konzeptionellen Modellieren als Einstieg in die OO-Linie. Deren Kettenglieder scheinen vollzählig zu sein: Analyse (OOA) → Design (OOD) → Programmierung (OOP) → Datenverwaltung (OODBMS). Die Frage „Warum wird die Objektorientierung als intuitive Leitidee des Entwerfens empfunden?“ läßt sich nun plausibel beantworten.

7.4 Der Beitrag des Objekt-Paradigmas

Seit Thomas KUHN wissen wir, daß sich Wissenschaften *revolutionär* entwickeln: Der Erkenntnisgewinn an Fakten und Methoden verläuft nicht kumulativ, sondern sprunghaft; die Sprungstellen sind durch *Paradigmenwechsel* ausgewiesen (siehe Abschnitt 4.2). Auch die Methodenentwicklung in der Informatik folgt langfristig einer Pendelbewegung: Mit leistungsfähigeren Methoden werden auch komplexere Probleme angegangen. Gelöste Probleme wecken neue Bedürfnisse. Und die wiederum spannen die Erwartung an das Potential der aktuellen Methode. Die Ausschläge des Pendels gehen also abwechselnd einher mit überzogenen Erwartungen und allgemein empfundenen Fortschrittskrisen K_i. Bild 7.5 zeigt den wellenförmigen Verlauf, den auch der Informatik-Duden konstatiert [Claus & Schwill, 1993].

Die Frage sei gestellt: Geht die Pendelschwingung tendentiell in Richtung einer *kognitiven* Unterstützung des Software-Entwerfens? Der mit den Paradigmenwechseln einhergehende Strukturwandel in der Programmiermethodik scheint die Frage zu bejahen:

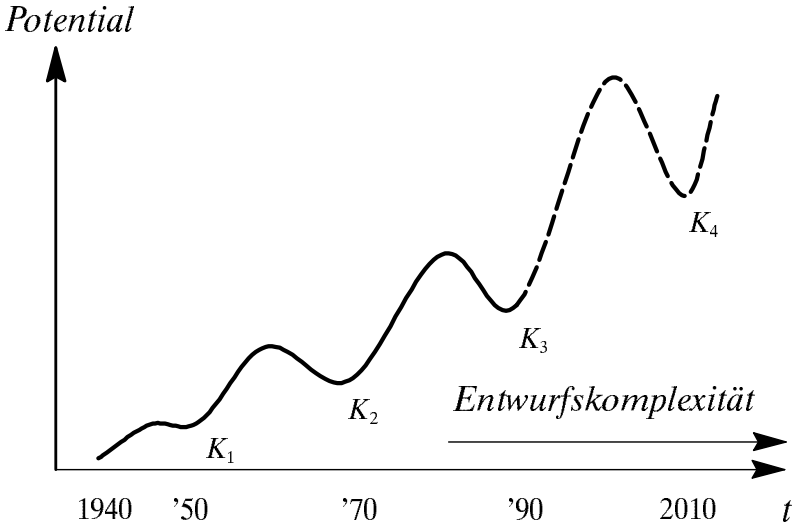


Bild 7.5: Methodenentwicklung in der Informatik

1. strukturlose Programmierung (methodenlose Zeit bis etwa 1950)

   Die *Methodenkrise* K_1 führt zur Entwicklung von Assemblern und Übersetzern für höhere Programmiersprachen.

2. künstlerische Strukturen (*Computer Programming as an Art* [Knuth, 1974]: 1950–1970)

   Die *Softwarekrise* K_2 führt zur Entwicklung strukturierter Programmiersprachen.

3. prozedurale Strukturen (strukturierte Programmierung: 1970–1990)

   Die *Wiederverwendungskrise* K_3 führt zur Entwicklung objektorientierter Programmiersprachen.

4. objektorientierte Strukturen (1990 –?)

   Die *Parallelitätskrise* K_4 führt ...

Mit der gegenwärtigen Loslösung von den prozeduralen VON-NEUMANN-Rechnerstrukturen durch die Konzepte objektorientierter Sprachen hat sich der Entwerfer weitestgehend von der Maschine entfernt. Mit der Unterstützung kognitiver Prozesse wird nun auch der Entwurfsprozeß über die gesamte Zielentfernung unterstützt: von $e \to \infty$ bis $e \to 0$ (siehe Seite 133). Die Durchgängigkeit der objektorientierten Methode — von der Idee zum Produkt — ist somit maximal. Im folgenden erläutern wir die These der „vollständigen Durchgängigkeit“. Wir ziehen dazu die Objekt-Parallelen zu den eingangs skizzierten Erkenntnissen und Hypothesen einer Psychologie des Problemlösens.

7.4.1 Objektbegriff

Das Objekt-Paradigma vereint in sich die drei positiv besetzten Technikattribute nach Kurt HÜBNER: Exaktheit, Rationalität und Fortschritt [Hübner, 1978].

Exaktheit ist durch die Formalisierbarkeit der Objektkonzepte gegeben: zum Beispiel ontologisch formal durch Yair WAND (siehe Anhang B) oder mathematisch formal durch David TOURETZKY [Touretzky, 1986]. Erst exakte Formen schaffen die technischen Voraussetzungen für die Massenproduktion. Für eine industrielle Softwareproduktion hat dies erstmals Doug MCILROY 1968 gefordert [McIlroy, 1969]. Im Wiederverwendungspotential der Klassenbibliotheken scheint die ökonomische Lösung zu liegen [Biggerstaff & Richter, 1989]. Klassen fungieren hier als exakte Schablonen, mit deren Hilfe struktur- und verhaltenskonforme Objekte beliebig oft „ausgestanzt" werden können.

Das Technikattribut *Fortschritt* ist durch die Allgegenwärtigkeit objektorientierter Konzepte und Anwendungen in der Software- und VLSI-Technik offensichtlich. Das Objekt-Paradigma gilt als favorisierter Theorie-Anwärter für den derzeitigen Paradigmenwechsel [Claus & Schwill, 1993]. Zum methodischen und pragmatischen Stellenwert des Objekt-Paradigmas sei auf die neueren Themenhefte einschlägiger Informatikzeitschriften verwiesen [CACM, 1990; Computer, 1992; Software, 1993; Informatik-Spektrum, 1992; Informatik-Spektrum, 1993].

Das dritte Attribut, die *Rationalität*, äußert sich sich in der Unterstützung kognitiver Prozesse während des objektorientierten Entwerfens. Den Rationalisierungsaspekt des Objektbegriffs erläutern wir hier an zwei Komponenten: (a) an der Modellbildung und (b) am analytischen Ansatz.

Modellbildung: In der darstellenden Kunst und auch in der Technik orientiert sich die Modellbildung seit eh und je am Objekt. Wir sprechen hier von konkreten *Gebilde-Modellen*, die ein Objekt ganzheitlich und im allgemeinen skaliert darstellen. Sie sind angefertigte Gegenstände, die dem Original strukturell oder funktional ähneln. Anders die abstrakten *Beschreibungsmodelle*. Diese sollen einen Realitätsbereich oder ein theoretisches System unter bestimmten Aspekten vereinfachend und idealisierend erklären. Die sprachliche und theoretische Erschließung eines wenig oder unbekannten Realitätsbereichs A erfolgt hier über Begriffe und Aussagen eines sprachlich und theoretisch bereits erschlossenen Realitätsbereichs B. So geschehen zwischen der NEWTONschen Himmelsmechanik und dem BOHRschen Atommodell.

In der Informatik sind ausnahmslos Beschreibungsmodelle bedeutsam, zumal der zu modellierende Rohstoff „Information" *per se* gestaltlos ist. Der Informations*gehalt* dagegen besitzt vielfache und vielfältige Strukturen, die es gilt, für die technische Darstellung und

Verarbeitung formal zu erfassen. Die *Informationsmodellierung* für datenintensive CAD-Anwendungen, wie den VLSI-Entwurf, hat derzeit einen hohen Stellenwert (siehe Abschnitt 8.2). Man hofft, durch den Entwurf von Informationsmodellen die Syntaxflut der inkompatiblen Datenaustauschformate einzudämmen und deren Standardisierung voranzutreiben.

Angelehnt an HÜBNER bilden Beschreibungsmodelle auf drei Stufen die Grundlage einer theoretischen Betrachtung mit zunehmender Abstraktion [Hübner, 1978, S. 368]:

1. Auf der ersten Stufe abstrahiert man von den „unmittelbaren Zwecken und besonderen Erscheinungen technischer Gegenstände", um „Einzelphänomene ableitbar und durch Einordnung in einen größeren Zusammenhang sowie durch Klassifizierung übersehbar zu machen".
2. Auf der zweiten abstrakteren Stufe werden die Modelle als kybernetische *Übertragungssysteme* betrachtet. Die Frage nach der Struktur steht im Vordergrund: „Wenn man beispielsweise zwischen einem technischen und einem natürlichen Übertragungssystem Isomorphie oder Homomorphie — also vollständig oder teilweise strukturelle Übereinstimmung — feststellt, dann wird, was das natürliche Übertragungssystem leistet, auch das technische vollständig oder teilweise hervorbringen."
3. Ausgehend vom gegebenen Übertragungssystem, konstruiert man schließlich auf der dritten Stufe der Modellbildung andere Übertragungssysteme durch „Kombination, Variation usw. nach mannigfaltigen Gesichtspunkten" und untersucht deren praktische Anwendbarkeit.

Die drei Stufen der Modellbildung finden wir nach Kurt HÜBNER und Alfred LUFT „nicht nur in der Schaltkreis- und Automatentheorie, sondern auch bei den Theoriebildungen der Künstlichen Intelligenz und der Kognitionswissenschaft" [Luft, 1988, S. 240]. Diese Stufen fortschreitender Abstraktion und Theoretisierung finden wir aber auch anderorts in der Informatik wieder: Das *semantische Netz*, in dem Knoten die Objekte und Kanten die Objektbeziehungen repräsentieren, ist die vorherrschende Graphenstruktur. Homomorphie herrscht unter den Analysemodellen des objektorientierten Requirements-Engineering, den Informationsmodellen in CAD-Frameworks, den konzeptionellen Schemata der Datenbanktechnik und der Wissensrepräsentation in der Künstlichen Intelligenz. Die strukturelle Übereinstimmung dieser Modelle wird heute als *object-oriented conceptual modeling* bezeichnet [Dillon & Tan, 1993; Fiadeiro & Sernadas, 1991; Quibeldey-Cirkel, 1994a].

Im gesamten Abbildungsprozeß während des objektorientierten Entwerfens ist die Homomorphie in Form semantischer Netze das Verbindende. Strukturell sind *1-zu-1*-Abbildungen zwischen der kognitiven Ebene der Ideenfindung über die konzeptionelle Ebene der Problemanalyse zu den Entwurfs- und Implementierungsmodellen möglich. Bild 7.6 verdeutlich die Strukturähnlichkeit zwischen den Abbildungsebenen: Während die ursprünglichen

Gedächtnisinhalte zunehmend konkretisiert und schließlich in Rechnerobjekte überführt werden, bleibt die Verknüpfungsstruktur erhalten.

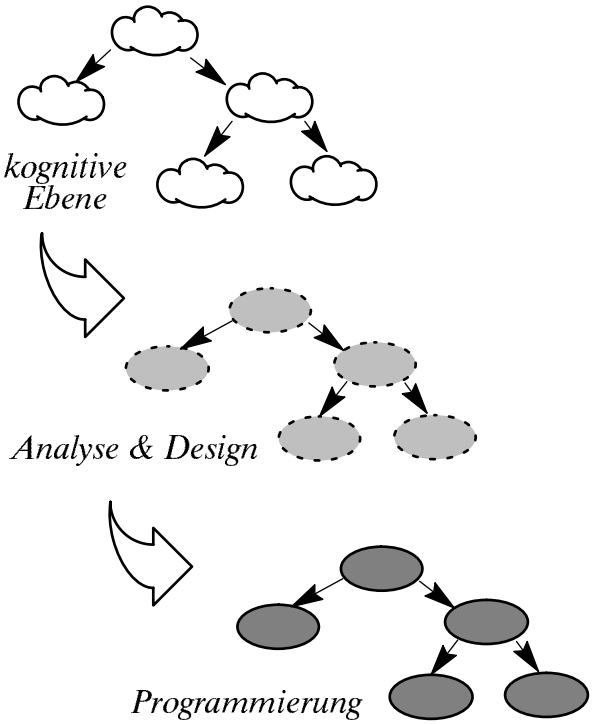


Bild 7.6: Homomorphie zwischen Idee und Produkt

Mit der bisherigen Diskussion über die Struktur unseres „kognitiven Apparats“ können wir nun also auf eine weitere, wohl die endgültige Übereinstimmung verweisen: auf die Homomorphie zwischen den kognitiven Gedächtnisstrukturen und dem Objekt-Paradigma schlechthin. Danach findet unser epistemisches Wissen sein Struktur-Pendant in den Informationsmodellen, Datenbankschemata und Wissensrepräsentationen in Form der *Klassenhierarchien*. Unser heuristisches Wissen spiegelt sich wider in den *Vererbungsstrategien* (einfache, multiple oder selektive Vererbung), in den *Kommunikationsprotokollen* auf der Objektebene (*Message passing*) und in der Integritätssicherung durch objektinterne *Constraints*.

Analytischer Ansatz: Die zweite Komponente der Rationalität des Objektbegriffs trägt entscheidend zur kognitiven Komplexitätsbewältigung bei. Der analytische Ansatz ist ein Hauptmerkmal der objektorientierten Denk- und Handlungsschemata: Abstraktion, Klassifikation, Hypotaxe, Parataxe und rationale Beschränkungen gehen aus der Analyse hervor.

Es ist offensichtlich, daß die im Abschnitt 7.2.2 eingeführten kognitiven Grundoperationen nach LOMPSCHER ihre Makro-Entsprechungen im Objekt-Paradigma haben: Tabelle 7.2.

LOMPSCHER-*Operatoren*	*objektorientierte Konzepte*
Zergliedern, Erfassen, Vergleichen	Objekt-Attribut-Beziehung
Ordnen	Aggregation
Abstrahieren	abstrakter Datentyp, virtuelle Operation
Verallgemeinern	Generalisierung
Klassifizieren	Klassenkonzept, Vererbung
Konkretisieren	Spezialisierung

Tabelle 7.2: Kognitive Parallelen

Die objektorientierten Konzepte bauen auf den kognitiven Operatoren auf. Sie erschließen somit das Potential der elementaren analytischen Denkleistungen — des Differenzierens, Strukturierens und Integrierens von Gedächtnisinhalten — für die entwurfstechnischen Ebenen. Mehr noch: Sie begünstigen das Modulkonzept der Informatik bereits in der für den Menschen denkbar frühesten Phase, der kognitiven Wahrnehmung. Aus der analytischen Sicht ist es dieser Zusammenhang zwischen den kognitiven Operatoren und den objektorientierten Konzepten, der die Intuitivität der Objektorientierung ausmacht. Die klassische ingenieurmäßige Entwurfsleistung, nämlich

1. Dekomposition der Wirklichkeit in Einzelheiten (Modularisierung) und
2. Rekonstruktion der Wirklichkeit in technischen Zusammenhängen (modularer Entwurf),

geht beim objektorientierten Entwerfen aus einem eng verzahnten Analyseprozeß hervor: aus dem Zusammenspiel kognitiver und pragmatischer Operationen, aus Denk- und Handlungsmustern also. Aus diesem Prozeß fallen die modularen Entwurfsleistungen, wie Klassen- und Objekthierarchien, „wie von selbst“ ab, weil sie eben auf denkpsychologischen Strukturen beruhen. Der analytische Ansatz des Objektbegriffs wird somit dem alten Modularisierungs-Postulat der Informatik, heute als Wiederverwendungs-Postulat neu formuliert, bereits auf einer Ebene gerecht, die der Softwaretechnik zeitlich und logisch vorgelagert ist.

7.4.2 Vermeidung früher Fixierung

Kreatives Problemlösen, und als solches verstehen wir das Entwerfen, verlangt *variierende* Schemata statt Fixierung. Die Denkpsychologie kennt seit langem die kognitiven Operationen, mit der man dem schablonierten Denken begegnen kann: DÖRNER bezeichnet das

Prinzip als „Hypothesenumbau". Die generelle Intention liegt im Aufbau von *Klassifikationssystemen*. Wir wollen auch hier die Parallelen ziehen.

Hypothesenumbau: Zunächst ist der Begriff der Hypothese in den technischen Kontext zu setzen. Nach Karl POPPER sind alle erfahrungswissenschaftlichen Aussagen, also auch die naturwissenschaftlichen oder technischen Gesetze und Theorien, nur Hypothesen über die Wirklichkeit. Der Wahrheitswert einer Hypothese ist grundsätzlich nicht beweisbar, er kann durch empirische Befunde nur widerlegt (*falsifiziert*) werden [Popper, 1963]. Die *Falsifikationsthese* von POPPER hat die Konsequenz, „daß auch wahre Gesetze und Theorien nur Hypothesencharakter haben und auf Dauer behalten" [Spinner, 1987, S. xi]. Demnach sind auch alle Aussagen über einen Realitätsbereich, wie wir ihn beispielsweise für den VLSI-Entwurf skizziert haben, hypothetisch: konzeptionelle Schemata und andere Wissensrepräsentationen sind Hypothesen. Diese Sichtweise hilft uns den Bezug zu den denkpsychologischen Fragestellungen herzustellen.

Fixierung verhindert die kognitive Variation. Ohne die Möglichkeit der kognitiven Variation legt jede Entwurfsmethode dem Entwerfer schablonenhafte Zwänge auf.[3] Die Möglichkeiten der objektorientierten Klassendifferenzierung[4] decken sich mit den von DÖRNER diskutierten kognitiven Operatoren für den Hypothesenumbau [Dörner, 1976, S. 127ff.]:

- *Abstrahieren* (Ersetzen eines Merkmals durch eine Unschärfestelle)

  Klasseneigenschaften (Attribute und Operationen) werden verallgemeinert, indem von den unbedeutenden Unterscheidungsmerkmalen abstrahiert und auf das Gemeinsame abgehoben wird. Dies führt auf Oberklassen, die sich von ihren Unterklassen durch eine größere „Unschärfe" auszeichnen.

- *Spezifizieren* (Ersetzen einer Unschärfestelle durch ein festes Merkmal)

  Unterklassen werden durch „artbildende" Eigenschaften von ihren Oberklassen differenziert.

- *Ersetzen* eines Merkmals durch ein anderes

  Für die konkrete Anwendung unbedeutende Klasseneigenschaften werden entweder ignoriert oder gezielt überschrieben (*overloading*).

[3] Das Zwanghafte unserer kognitiven Analyseschemata wird schon in Bild 7.4 deutlich: Man versuche einmal, im Schema „Katze" die „Geldbörse" wiederzuerkennen.

[4] Objektorientiertes Programmieren wird vielerorts als „programming the difference" bezeichnet, was heißen soll: Vordefinierte allgemeine Klassen, entnommen aus kommerziellen Klassenbibliotheken, werden soweit „ausdifferenziert", bis sie die spezielle Anwendung modellieren.

Angenommen, unsere Hypothese über das Inventar eines Realitätsbereichs sei die Objektmenge:

{ ♣ , ♡ , ♢ , ♠ , ♡ , ♢ , ♠ , ♣ }.

Durch Abstrahieren, Spezifizieren und Ersetzen können wir für die Objekte ein Ordnungsschema konstruieren. Wir abstrahieren zum Beispiel von den konkreten Objekten, indem wir ihre gemeinsamen Merkmale herausziehen und als Tupel darstellen: <*Größe*, *Form*>. Das führt uns auf das Schema (in der objektorientierten Terminologie Klasse genannt) der „kleinen und großen Kartensymbole".

Interessieren wir uns nur für die Symbolformen, wollen wir also unsere ursprüngliche Hypothese verallgemeinern, indem wir von allen anderen Merkmalen absehen, so fügen wir anstelle des Merkmals *Größe* eine Unschärfestelle *U* ein. Unsere so definierte Oberklasse ist durch das Tupel <*U*, *Form*> charakterisiert und heiße nun allgemeiner „Kartensymbole". Die Klasse „kleine und große Kartensymbole" ist dann deren Unterklasse, die durch den Ersatz der Unschärfestelle *U* durch das Merkmal *Größe* spezifiziert ist.

Wollten wir unsere Hypothese über die Welt der Kartensymbole revidieren, um anstelle eines früheren Aspekts einen neuen einzubringen, steht uns das Mittel des Merkmalersatzes zur Verfügung. Zum Beispiel könnten wir die erste Stelle im Tupel <*Größe*, *Form*> durch das Merkmal *Farbe* ersetzen. Wir hätten dann als revidierte Unterklasse „rote und schwarze Kartensymbole". Diese Modifikation unseres Ausgangsschemas entspricht in der objektorientierten Datenbankterminologie der Schema-Evolution.

Der kognitive Hypothesenumbau wird also auch vom Objekt-Paradigma konsequent unterstützt. Der Grad der Schemavariation bestimmt den Freiheitsgrad des Entwerfens. Die Variationsvielfalt objektorientierter Schemata reicht dabei von linearen Strukturen durch parataktisches Ordnen (aggregierte Objekte als Listen- oder Mengen) bis zu hierarchischen Strukturen durch hypotaktisches Ordnen (Mehrebenen-Analyse, Klassenhierarchien). Das Paradigma der strukturierten Programmierung dagegen unterstützt die Schemavariation nur in der *prozeduralen* Analyse. Hier dominieren die Ablaufstrukturen (Sequenz, Auswahl und Schleife) die Analyseschemata.

Das Repertoire des Objekt-Paradigmas umfaßt beides: analytische Schemata für Struktur *und* Verhalten. Mit ihnen lassen sich die von DÖRNER herausgestellten Relationen im semantischen Netz unserer epistemischen Gedächtnisstruktur hinreichend modellieren. Neben den globalen Analyseschemata — wie „Außen-Innen", modelliert durch *Objektkapseln*, „Abstrakt-Konkret", modelliert durch den Mechanismus der *Objektausprägung*, und „Ganzes-Teil", modelliert durch die *Aggregation* von Objekten — kann letztlich jedes anwendungsspezifische Schema modelliert werden. So geschehen im DASSY-Projekt für den

Realitätsbereich „VLSI-Entwurf“, wo dedizierte Beziehungstypen und objektinterne Constraints VLSI-spezifische Relationen beschreiben [Quibeldey-Cirkel, 1993c].

7.4.3 Analogieschluß

Ein weiterer, in der Informatikliteratur kaum untersuchter psychologischer Wirkfaktor des kreativen Entwerfens ist die *Metaphorik*.[5] Objektorientierte Metaphern, wie *Vererbung* und *Nachrichtenaustausch*, fördern den Analogieschluß. Abstrakt formuliert besteht ein Analogieschluß darin, „daß angenommen wird, die Relationen zwischen den Sachverhalten eines Bereichs A bestünden in gleicher oder ähnlicher Form auch zwischen den Sachverhalten eines Bereichs B“ [Dörner, 1976, S. 81]. Der Analogieschluß hat sich in der Wissenschaftsgeschichte als „Königsweg“ der Suchraumerweiterung erwiesen, als ein wichtiges Verfahren der Hypothesenbildung:

> „KÉKULÉ soll die ringförmige Kopplung der Kohlenstoffatome im Benzolmolekül dadurch gefunden haben, daß die aus dem Kaminfeuer in bogenförmigen Bahnen geschleuderten Funken in ihm das Bild einer sich in den Schwanz beißenden Schlange aufriefen.“ [Dörner, 1976, S. 81]

Auch MENDELEJEW soll ein Analogiesprung auf das Periodensystem der Elemente gebracht haben: Die zweidimensionale Ordnung der Karten im Kartenspiel diente ihm als Analogon. *Gruppen* und *Perioden* chemischer Elemente entsprechen der Einteilung eines Kartenspiels nach Bild (As, König usw.) und Farbe (Kreuz, Pik, Herz, Karo). ARCHIMEDES, um ein letztes Beispiel zu zitieren, analogisierte sich selbst mit der Goldkrone und fand durch diesen Schluß die Lösung (siehe Fußnote 1 auf Seite 138). Die Beispiele belegen die wichtige Rolle des Analogieschlusses als Mittel zur hypothetischen Vervollständigung eines nur teilweise bekannten Realitätsbereichs. Denken in Analogien erweist sich als überaus kreativ. Metaphern beflügeln die Phantasie. Als *symbolische Analogien* (im Sinne der *Synektik*-Lehre [Gordon, 1961]) stellen sie ein sehr wirkungsvolles Mittel für ein produktives Problemlösen und somit auch produktives Entwerfen dar.

Vertragsgebundenes Entwerfen

Auf eine Leitmetapher der Objektorientierung sei im folgenden näher eingegangen. Sie symbolisiert den Analogieschluß zwischen den marktwirtschaftlichen Reglements des Dienstleistungsgewerbes und der methodischen „Disziplinierung“ des Software-Entwerfens: die *Client-Server-Metapher* in Bild 7.7.

[5] Im Anhang A werden die objektorientierten Metaphern herausgestellt und ihre Bedeutung und Funktion erläutert.

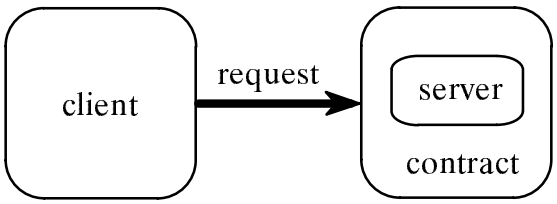


Bild 7.7: Die Metapher des Client-Server-Vertrags

Rebecca WIRFS-BROCK hat das Client-Server-Modell konzeptionell erweitert: Um die Kommunikation zwischen Objekten verläßlich zu halten, führt sie das Pflichten-Konzept ein [Johnson & Wirfs-Brock, 1990]. Die Pflichten (*responsibilities*) eines Objekts umfassen alle öffentlich zugänglichen Dienste, über die es verfügt. Das Objekt erfüllt seine Pflicht, indem es die angeforderten Dienste entweder selbst oder in Zusammenarbeit mit anderen Objekten ausführt. Pflichten und Kooperationen werden in einem „Geschäftsvertrag" fixiert. Er beschreibt die möglichen „Geschäftsvorfälle" zwischen Client- und Server-Objekten. Beide sind verpflichtet, den Vertrag einzuhalten: Das Client-Objekt darf nur Dienste anfordern, die auch im Klassen-Vertrag vereinbart wurden, das Server-Objekt muß die Anfragen durch geeignete Dienstleistungen beantworten.

Ein Vertrag legt also Pflichten und Kooperationen fest, auf die sich ein Client-Objekt ohne Kenntnis der konkreten Vertragserfüllung verlassen kann. Mit der Metapher des Vertrags wird das Prinzip des *Information hiding* nach PARNAS (siehe Abschnitt 2.3) und der objektorientierten Datenkapselung (beides wiederum Metaphern) nicht mehr allein als ein Belang der Implementierung begriffen, sondern als ein wichtiger Aspekt im Klassenentwurf: *responsibility-driven design* [Wilkerson & Wirfs-Brock, 1989]. Der Vertrag enthält nur Aussagen darüber, welche Handlungen unter welchen Bedingungen durch welche Objekte ausgeführt werden. Das *Wie* einer Handlung tritt nach außen nicht in Erscheinung, bleibt in der Klasse lokalisiert. Die Realisierung wird auf die Phase der Implementierung verschoben.[6]

Als symbolische Analogie erschließt die Metapher vom „Vertrag zwischen Kunden und Dienstleister" einen traditionell gut verstandenen Realitätsbereich. Die dort bewährten Reglementierungen lassen sich methodisch und pragmatisch auf den Realitätsbereich des Programmentwurfs übertragen. Unsere Alltagssprache lebt von den Metaphern, ohne daß wir diese immer bewußt als solche empfinden. Wir sind es gewohnt, in Analogien zu den-

[6] Bertrand MEYER hat das Vertragskonzept bereits durch entsprechende Ausdrucksmittel in seiner objektorientierten Sprache Eiffel verankert: Zusicherungen (*assertions*). Seine Programmiermethode bezeichnet er als *Programming by Contract* [Meyer, 1988].

ken, um Probleme zu lösen. Deshalb erscheint uns das Entwerfen und Programmieren mit und in Metaphern als besonders intuitiv. Bedeutung und Funktion der objektorientierten Metaphorik sind bislang wenig beachtet worden. Hier steht sicherlich Untersuchungsbedarf an. Denn im systematischen Einsatz metaphorischer Methoden und Sprachen liegt ein ungeahntes Kreativpotential. „Kreativitätstechniken" in der Softwaretechnik werden in Zukunft an Bedeutung gewinnen, wenn die dialektischen und synthetischen Problembarrieren immer höher werden.

Fazit

Die Aussage „Objektorientierte Konzepte sind intuitiv" ist zum Allgemeinplatz geworden; Enthusiasten wie Skeptiker sind sich in diesem Punkt einig. Die ureigenen Denk- und Verhaltensweisen des Menschen orientieren sich am Objekt. Um in der täglichen Auseinandersetzung mit einer komplexen Welt zu bestehen, abstrahieren wir vom Unwesentlichen, klassifizieren das Wesentliche und leiten neue Eigenschaften aus alten ab. Intuition kann nicht gelehrt werden. Sie entspringt dem unreflektierten Fundus an Wissen und Erfahrung. Intuition kann auch nicht exakt begründet werden. Sie zählt zu den Merkmalen unserer rechten Gehirnhälfte. Dort entwickeln sich nach dem Nobelpreisträger Roger SPERRY unsere assoziativen Erkenntnisse. Intuition läßt sich aber plausibel machen. Und das ist das Thema dieses Kapitels. Zwei Faktoren bewirken die intuitive Akzeptanz:

1. *Integrativer Faktor*

   Das Objekt-Paradigma vereint in sich die „fundamentalen Ideen der Informatik" [Claus & Schwill, 1993]: Es kapselt die „Idee der Algorithmisierung". Im Objekt findet der prozedurale Wissensfundus seine gewohnte Anwendung. Es unterstützt die „Idee der Sprache". Die Semantik objektorientierter Programmiersprachen deckt sich mit den Konzepten und Begriffen aus Analyse & Design. Und es verkörpert die „Idee der Zerlegung". Klassen und Objekte fördern die Modularisierung auf der begrifflichen Ebene.

2. *Kognitiver Faktor*

   Die objektorientierten Konzepte greifen bereits in der zeitlich und logisch frühesten Phase des Problemlösens, in der gedanklichen Wahrnehmung und Aufbereitung eines Problems. Sie erschließen die (denk-)psychologische Dimension der Begriffsbildung, in der man auf Strukturen und Operationen stößt, die sich auch in den technischen Phasen des Problemlösens bewährt haben.

Eine „Psychologie des Programmierens" wurde schon in der Vergangenheit verfolgt [Shneiderman, 1980; Weinberg, 1971]: Der *Great Designer* [Brooks, 1987] und der *Super-*

programmer [Molzberger, 1984] stehen exemplarisch für das Phänomen des außergewöhnlichen Problemlösers. Auch wurde die Motivations-Psychologie ins Spiel gebracht: Der *Superprogrammer* strebt nach der Erfüllung seiner *ego needs* in der Spitze der Bedürfnispyramide, der *Hacker* nach sozialer Anerkennung auf unterer Ebene (siehe Bild 4.1 auf Seite 81). Damals, in den frühen 80er Jahren, waren die objektorientierten Methoden noch unausgereift oder der Implementierung zu sehr verhaftet. Mit dem Methodenboom[7] der 90er Jahre rückt die Kreativität des Programmierers wieder in den Blickpunkt: Intuitive Methoden schaffen Akzeptanz. Objektorientierte Methoden sind intuitiv. Mit der breiten Akzeptanz des Objekt-Paradigmas könnte eine Renaissance der *Human-factors*-Bewegung anbrechen. *Intuitives Programmieren* stünde auf ihren Fahnen.

[7] In dieser Studie wurden Entwurfsmethoden untersucht, die auf Monographien verweisen können: [Blaha *et al.*, 1991; Booch, 1991; Coad & Yourdon, 1991a; Coad & Yourdon, 1991b; Mellor & Shlaer, 1992; Wilkerson & Wirfs-Brock, 1989].

8

VLSI/CAD-Weltmodelle

Aufbauend auf den Erkenntnissen zur kognitiven Entwurfsmodellierung skizzieren wir in diesem Kapitel eine objektorientierte Entwurfsmethode. Wir lassen uns dabei leiten von den Prinzipien der Komplexitätsbewältigung aus dem ersten Teil der Studie und von der Synergie der diskutierten Modellierungsvorschläge aus dem zweiten Teil. Unser Ansatz wurzelt im Motivationsumfeld des Forschungsprojekts DASSY [Quibeldey-Cirkel, 1993c] und der EDIF-TSC „Information Modelling Group“ [Wilkes (Hrsg.), 1992].

Das Problem „VLSI-Entwurf“ ist nach unserer Erfahrung gekennzeichnet durch fortgesetzte Fehlanpassungen in der Sequenz aus Problembewußtsein, -formulierung und -lösung. Der Entwurfspfad von der Idee zum Produkt führt über kognitive, strukturelle und methodische Einbrüche: vom Problembewußtsein als kognitivem Prozeß über die Problemformulierung in den formalen Strukturen eines Datenmodells zur Problemlösung durch Abbildung auf die Objekte und Prozesse der Implementierung (der Programmierung und Datenhaltung). Je größer die Sprünge in der Entwurfslinie, desto schwieriger die Aufgabe, *Datenintegrität* global zu gewährleisten. Dieser Einsicht entspringt das Bedürfnis nach einer *übergreifenden* und *durchgängigen* Entwurfsmethode, die den dualen Aspekt der Datenintegrität — methodisch und operativ — berücksichtigt. Hier soll gezeigt werden, daß durch die objektorientierte Modellierung des Entwurfsprozesses und seiner Schnittstellen zu einem CAD-Framework eine derartige Methode möglich und produktiv ist.

Übersicht Abschnitt 8.1 skizziert das Objektmodell für VLSI-spezifische Entwurfsdaten. Es ist kongruent mit den Konzepten des Objekt-Paradigmas. Abschnitt 8.2 geht auf das objektorientierte Beschreibungsmittel ein, das derzeit die Informationsmodellierung im VLSI-Entwurf dominiert: die Sprache Express. Am Beispiel des Datenaustauschformats EDIF zeigen wir den konkreten Einsatz im Abschnitt 8.3. Der Abschnitt 8.4 schließlich umreißt die objektorientierten Ansätze, die den prozeduralen Schnittstellen in einem CAD-Framework zugrundeliegen. Als ein Beispiel für VLSI/CAD-Weltmodelle dient das DASSY-Datenmodell.

8.1 Der objektorientierte VLSI/CAD-Entwurf

Die Parallelen zwischen Soft- und Hardware-Entwurf sind offensichtlich: Ob *System-on-Chip* oder *Programming-in-the-Large*, in beiden Fällen handelt es sich um das Unterfangen, hochkomplexe Systeme überprüfbar zu entwerfen. Hardware-Aspekte, wie Flächenbedarf, Laufzeit, Gatteräquivalente und Teststrukturen, werden in der Regel erst auf den implementierungsnahen Ebenen mit einbezogen; sie stellen physikalische Parameter oder Entwurfsregeln für den Software-Entwurf dar. Eingedenk dieser Analogie führen wir die Objektorientierung als probates Mittel zur Komplexitätsbewältigung in den VLSI-Entwurfsprozeß ein.

Objektorientierung[1] als Leitmotiv einer VLSI-Entwurfsmethode führt auf die von Grady BOOCH formulierte „kanonische Form" wohlstrukturierter komplexer (Software-)Systeme: Klassenstrukturen als hypotaktische *kind-of*- und Objektstrukturen als parataktische *part-of*-Hierarchien [Booch, 1991, S. 12f]. Die grundsätzliche Vorgehensweise bezeichnet BOOCH als *round-trip gestalt design*:

> „A style of design that emphasizes the incremental and iterative development of a system, through the refinement of different yet consistent logical and physical views of the system as a whole; the process of object-oriented design is guided by the concepts of round-trip gestalt design; round-trip gestalt design is a recognition of that fact that the big picture of a design affects its details, and that the details often affect the big picture." [Booch, 1991, S. 517]

Der Kern des Problems liegt in der Dekomposition in geeignete Objekte und deren Klassifizierung. BOOCH stellt die objektorientierte Dekomposition der algorithmischen gegenüber:

> „**algorithmic decomposition**: The process of breaking a system into parts, each of which represents some small step in a larger process. The application of structured design methods leads to an algorithmic decomposition, whose focus is upon the flow of control within a system." [Booch, 1991, S. 512]
>
> „**object-oriented decomposition**: The process of breaking a system into parts, each of which represents some class or object from the problem domain. The application of object-oriented design methods leads to an object-oriented decomposition, in which we view the world as a collection of objects that cooperate with one another to achieve some desired functionality." [Booch, 1991, S. 516]

[1] Auch wenn sich derzeit eine Flut von Publikationen über das Thema ergießt, so sind hauptsächlich nur Einzelanwendungen betroffen: in erster Linie objektorientierte Programmiersprachen [Cox, 1987; Dahl & Nygaard, 1966; Goldberg, 1984; Meyer, 1988; Stroustrup, 1986] und erst zögernd Analyse- und Designmethoden. In [Quibeldey-Cirkel, 1994c] findet man eine erste systematische Gesamtdarstellung des Objektansatzes, die dessen Bedeutung für die kognitiven, methodischen und technischen Prozesse im Software-Lebenszyklus hervorhebt.

Die eingefahrene algorithmische Zerlegung des Steuerflusses hat die klassischen Entwurfswerkzeuge der Analyse und Synthese hervorgebracht. Prinzipiell lassen sie sich dennoch, wie James DANIELL in seiner Abhandlung *An Object Oriented Approach to CAD Tool Control* zeigt, als abstrakte Objekte im Sinne der Objektorientierung auffassen [Daniell, 1989].

Der objektorientierte Entwurf umfaßt neben dem Prozeß der Dekomposition auch eine entsprechende Notation (zum Beispiel *boochgrams* [Booch, 1991]), die es erlaubt, die logisch-physikalischen sowie die statisch-dynamischen Modelle des zu entwerfenden Systems formal zu beschreiben. Als Ergebnis dieser Entwurfsmethode ergeben sich Klassen, Beziehungstypen und Vererbungsstrategien, die bei einer objektorientierten Implementierung *1-zu-1* auf die Ausdrucksmittel der Programmier- oder Datenbanksprache abbildbar sind.

8.1.1 VLSI/CAD- versus Objektmodell

Die junge Vorgehensweise, Entwurfswerkzeuge herstellerübergreifend in standardisierten Infrastrukturen zu integrieren (Schlagwort *CAD-Framework*: „offene" Dialog-, Werkzeug- und Datenhaltungs-Schnittstellen, siehe Abschnitt 3.1), stellt eine Reihe von Anforderungen an die Daten- und Prozeßmodellierung. VLSI/CAD-spezifische Anforderungskataloge wurden zum Beispiel als Meilenstein im DASSY-Projekt aufgestellt [DASSY, 1990] oder sind in der neueren Literatur der Datenbanktechnik zu finden [Batory & Kim, 1985; Brauer, 1990; Encarnação & Lockemann, 1990]. Sie stützen sich sowohl auf Untersuchungen über die Datenstrukturen und Operationen typischer VLSI-Werkzeuge (Datenmodellierung) als auch auf die spezifischen Belange einer integrierten Entwurfsverwaltung und -datenhaltung (Entwurfsprozeß). Wir wollen die wichtigsten Anforderungen skizzieren und den Nachweis führen, daß die inhärenten Eigenschaften eines Objektmodells diese erfüllen.

Der folgende Anforderungskatalog[2] ist nicht unbedingt auf VLSI-Anwendungen beschränkt, sondern gilt auch für andere *Non-Standard*-Datenbankanwendungen, wie Maschinenbau und Geo-Datenbanken.

Generalisierung/Spezialisierung: Die Welt ist vernetzt, besonders die technische. Technische Objekte hierarchisch ordnen zu können, setzt ihre Klassifikation voraus. Als leistungsfähiges Mittel der Modellierung dient die Abstraktion. Sie führt auf „crisply-defined conceptual boundaries relative to the perspective of the viewer" [Booch, 1991, S. 512]. Global spiegelt sich die Generalisierung/Spezialisierung in den VLSI-Abstraktionen wider: *system*, *algorithmic*, *micro-architectural*, *logic* und *circuit levels* nach GAJSKI; lokal in den

[2] Der Abschnitt ist Teil eines Anwenderberichts, veröffentlicht im CAD-CAM Report [Quibeldey-Cirkel, 1993c].

Begriffspyramiden: beispielsweise in *figure* als geometrischen Oberbegriff von *circle* und *polygon* [EDIF-Manual, 1990].

Das Wesentliche einer Klassenhierarchie ist ihre semantische Fügung, das heißt die *kind-of*-Beziehungen der Klassen untereinander. Sie erlauben die *Vererbung* von Eigenschaften, was methodisch die Komplexität verringert und technisch das *code sharing* in Programmen und Datenbanken fördert. Oberklassen stehen für Abstraktionen ihrer Unterklassen. Unterklassen übernehmen die Verhaltens- und Strukturmuster ihrer Oberklassen, verändern sie, fügen bei Bedarf eigene hinzu. Die Generalisierung/Spezialisierung gewährt somit ein Höchstmaß an Ausdrucksökonomie.

Aggregation: Innerhalb der Objektwelt bedarf es der Konstruktion komplexer Objektverbände: *Part-of*-Beziehungen ermöglichen das Zusammensetzen und „Schachteln" von Objekten. So aggregiert eine *Netzliste* die funktionalen Schaltungskomponenten zu einem Objektverband, der als Komplex verwaltet wird. Mit der Art der Aggregation eng verbunden ist die auf ihr definierte *operative* Semantik, zum Beispiel die Frage: Sollen aggregierte Objekte (Unterobjekte) beim Löschen des Aggregats (Oberobjekts) mit gelöscht werden? Im DASSY-Datenmodell wurden hierfür VLSI-spezifische Beziehungstypen vordefiniert, die vom Anwender mit eigener operativer Semantik angereichert werden können [DASSY, 1992a].

Abstrakte Datentypen: Diese Forderung geht zurück auf die unzureichenden Datentypen in herkömmlichen Datenhaltungen: Ganz-, Festkommazahlen und Zeichenketten reichen nicht für CAD-Anwendungen. Abstrakte Datentypen erhöhen überdies die Integrität des Datenbestandes, da Operationen und Daten aneinander gebunden sind. Datenstrukturen und Operationscode bleiben dem Anwender verborgen: PARNAS-Prinzip des *Information hiding* (Abschnitt 2.3.3). Beispiel für einen abstrakten CAD-Datentyp: *Punktlisten* zur Polygon-Beschreibung, wie sie Layouteditoren im allgemeinen verwenden, mit Operationen, die eine Selbstüberschneidung des Polygons prüfen [Brauer, 1990, S. 151f].

Objektausprägung: Der Übergang von der analytischen Phase der Klassenbildung zu den entwurfstechnischen Phasen heißt Exemplarbildung oder Objektausprägung. Dieser Schablonenmechanismus (*instance-of*-Beziehungen zwischen Objekt- und Klassenwelt) unterstützt die *Software-Wiederverwendung* entscheidend und ist für zellbasierte Entwurfstechniken (Standardzellen- und Gate-Array-Entwurf) charakteristisch. Exemplare besitzen eine Identität und können beliebig oft durch Referenzieren der zugehörigen Klassen für die Anwendung verfügbar gehalten werden — bei *generischen* Klassen auch unter Angabe von Parametern. Beispiel: Logik-Gatter und Register-Transfer-Bausteine mit *wahlfreier* Anzahl der Eingänge oder *Fan-out*-Treiberfähigkeit.

Sichten: Entwurfswerkzeuge greifen nur auf bestimmte Repräsentationen eines Entwurfsobjekts zu: Schaltkreisextraktoren benötigen eine andere Objektbeschreibung (Maskengeometrien und Entwurfsregeln) als Logiksimulatoren (Netzlisten und Verhaltensbeschreibungen). Im großen wird der Entwurfsraum nach GAJSKI unter den Blickwinkeln „Verhalten", „Struktur" und „Physik" betrachtet. Im kleinen kennt EDIF zehn *viewTypes*, die hauptsächlich die Konnektivität, Geometrie und Dokumentation der Objekte beschreiben [Quibeldey-Cirkel & Wahl, 1990a].

Objektversionierung: Die Archivierung von CAD-Objekten erfordert einen Mechanismus, der die *zeitlichen* und *alternativen* Richtungen in der Objektentwicklung aufzeichnet. Die Versionsverfolgung ist von zentraler Bedeutung bei der Koordinierung eines Team-Projekts [Katz, 1990]. Damit eng verknüpft sind Entwurfs-Transaktionen, ein Konzept der Datenhaltung, um die Datenkonsistenz bei konkurrierenden Zugriffen und im Fehlerfall zu bewahren [Gray, 1981; Katz *et al.*, 1986; Katz, 1990; Kim *et al.*, 1983]. Besonders wichtig ist hier das VLSI-spezifische Konzept der „langen Transaktion": Für die Dauer einer Transaktion (Entwurfssitzung) werden die aktuell bearbeiteten Entwurfsdaten aus der Datenhaltung ausgelagert. Sie unterliegen in dieser Zeit nicht der Integritätssicherung. Erst wieder beim Einlagern der Daten wird deren Integrität geprüft.

Interaktion: Konventionelle Anstoßtechniken, wie der *Event/Trigger*-Mechanismus in Datenbanken [Köhler, 1992; Kotz, 1989], setzen Kommunikationsmittel und -pfade zwischen den Objekten voraus. Die Datenintegrität zu sichern, ist hier das vorrangige Ziel. Die Korrektheit der Daten muß unter den Aspekten von Aggregation, Typisierung, Sichten, Versionen und Transaktionen gewährleistet sein.

Modellierbarkeit passiver und aktiver Eigenschaften: Struktur und Verhalten der Objekte müssen *gleichermaßen* und *gemeinsam* beschreibbar sein, um die statisch-dynamischen Aspekte innerhalb der Objektmengen — die Aktionen und Reaktionen auf den Objektstrukturen — zu erfassen. Diese Forderung geht über die Leistungen *semantischer Datenmodelle* hinaus, beispielsweise des Entity-Relationship-Modells von Peter CHEN und seiner zahlreichen Erweiterungen, die sich aber im allgemeinen auf die statische Strukturbeschreibung beschränken [Chen, 1976; Hull & King, 1987].

Es ist kein Zufall, daß die Anforderungen an die Modellierbarkeit der CAD-Entwurfsobjekte inhaltlich von den zentralen Konzepten der Objektorientierung erfüllt werden [Booch, 1991; Henderson-Sellers, 1992; Korson & McGregor, 1990; Meyer, 1988; Winblad *et al.*, 1990]. Zum einen resultieren diese Anforderungen — analog zur Evolution des Objekt-Paradigmas [Quibeldey-Cirkel, 1994a] — aus den systemtheoretischen Erkenntnissen der Komplexitätsbewältigung. Zum anderen unterstützt der interdisziplinäre Charakter des

Objekt-Paradigmas unseren intuitiven Zugang zu komplexen Sachverhalten und vereint zugleich traditionelle und innovative Techniken des ingenieurmäßigen Systementwurfs. Eine Gegenüberstellung zwischen VLSI/CAD-Spezifika und Objektmodell ist folglich nahezu deckungsgleich: Tabelle 8.1.

VLSI/CAD-Modell	Objektmodell
Hypotaktische Ordnung: Generalisierung/Spezialisierung	Klassenstrukturen mit Vererbung: *kind-of*-Hierarchien
Parataktische Ordnung: Aggregation	Klassenstrukturen mit definierter operativer Semantik: *part-of*-Hierarchien
abstrakte Datentypen	zu Klassen typisierte Objekte mit Kapselung von Struktur und Verhalten
Objektausprägung	dito
Sichten	spezialisierte abstrakte Klassen
Objektversionierung	Alternativen als spezialisierte Klassen, Versionen als Objekte einer Alternative
Interaktion/Kommunikation	Nachrichtenaustausch über Protokolle (*Message passing* und *Polymorphie*)
passive Eigenschaften aktive Eigenschaften	Klassen- und Objektvariable Klassen- und Objektoperationen

Tabelle 8.1: Zur Kongruenz zwischen VLSI/CAD-Spezifika und Objektmodell

Strukturmuster

Als gemeinsames Strukturmuster der Klassen betrachten wir die „Zelle“. Sie stellt den Grundbaustein der Entwurfsmodellierung und der Implementierung dar — parallel zur Zell-Doktrin der Biologie, nach SIMON „a law of qualitative structure“ [Newell & Simon, 1975, S. 290]. Wir assoziieren also mit diesem Begriff nicht ausschließlich funktionale Module, wie Logik-Gatter und Prozessor-Bausteine in kommerziellen Zellbibliotheken, sondern betrachten die „Zelle“ als ein Generalschema, das die komplementären Prinzipien von Abstraktion und Kapselung reflektiert:

„abstraction focuses upon the outside view of an object and *encapsulation* — also known as *information hiding* — prevents clients from seeing its inside view, where the behavior of the abstraction is implemented. In this manner, encapsulation provides explicit barriers among different abstractions." [Booch, 1991, S. 45]

Unsere Auffassung steht in Übereinstimmung mit den Konzepten abstrakter Datentypen [Liskov & Zilles, 1974] und der Zellstruktur in EDIF [Quibeldey-Cirkel & Wahl, 1990a]. Charakteristisch für eine Zelle ist folglich ihre Dichotomie: *öffentliches* Nachrichtenprotokoll (Schnittstelle) und *private* Implementierung (Inhalt).

„In practice, this means that each class must have two parts: an interface and an implementation. The *interface* of a class captures only its outside view, encompassing our abstraction of the behavior common to all instances of the class. The *implementation* of a class comprises the representation of the abstraction as well as the mechanisms that achieve the desired behavior. This explicit division of interface/implementation represents a clear separation of concerns: the interface of a class is the one place where we assert all of the assumptions that a client may make about any instances of the class; the implementation encapsulates details about which no client may make assumptions." [Booch, 1991, S. 45f]

Das Begriffspaar Kapselung-Vererbung birgt ein Dilemma des Objekt-Paradigmas [Snyder, 1986]: Beides sind wünschenswerte Eigenschaften. Je mehr man aber eine Klasse durch Kapseln von äußeren Einflüssen abschottet, um die lokale Wartbarkeit und Erweiterbarkeit zu erhöhen, desto stärker schränkt man die Vererbungsmöglichkeiten und damit die Ressourcen-Nutzung ein. Umgekehrt gefährdet ein freier Zugriff auf die Klassen-Interna deren Integrität. Eine hermetische Kapselung würde schließlich jegliche Vererbung unterbinden. Deshalb und mit Blick auf die Implementierung differenzierter Vererbungsstrategien (siehe unseren Vorschlag zur VLSI-Versionierung im Objektmodell weiter unten) unterscheiden wir mehrere Grade der Kapselung in der Schnittstellen-Definition einer Zelle (Bild 8.1) — analog zum Stufenkonzept der Klassenschnittstelle in C++ [Stroustrup, 1986].

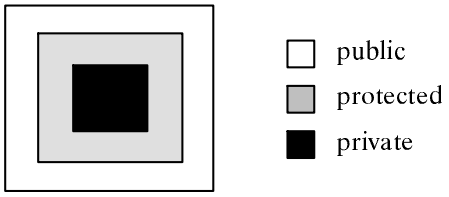


Bild 8.1: Zur selektiven Sichtbarkeit einer Zelle

- *public*
  uneingeschränkt sichtbar für alle Klienten dieser Klasse,

- *protected*
  nur sichtbar für Unterklassen und

- *private*
  nur sichtbar für die Klasse selbst, zum Beispiel die nach außen geschützte Repräsentation des Objektzustands als Konstanten und Variablen-Deklaration (Prinzip der *lokalen Wartbarkeit*).

VLSI-Klassenstruktur

Die Analogie zwischen Generalisierung/Spezialisierung einerseits und Klassenstrukturen andererseits (siehe Tabelle 8.1) führt zwanglos auf ein einfaches Diagramm für ein allgemeines objektorientiertes VLSI-Entwurfsmodell: Bild 8.2 auf der nächsten Seite. Die grafische Notation ist an BOOCH angelehnt. Das Piktogramm für Klassen — „amorphous blob, some call it a *cloud*“ [Booch, 1991, S. 158] — symbolisiert die bereits zitierten *crisply-defined boundaries* einer Abstraktion. Die gestrichelte Umrandung soll daran erinnern, daß Kunden (*clients*) im allgemeinen nur auf den Objekten einer Klasse operieren und nicht auf der Klasse selbst. Die einzige Klassenbeziehung, die wir verwenden, ist die *Vererbung*, dargestellt als Pfeile auf den „Erblasser“. Die VLSI-Stratifikation und die wesentlichen VLSI-Sichten sind der Arbeit von GAJSKI entlehnt, da sie nach unserer Erfahrung den Stand der Technik wiedergeben: „key abstractions (that) reflect the vocabulary of the problem domain“ [Booch, 1991, S. 150].

Die skizzierten Klassen sind als *abstract classes* zu verstehen, das heißt sie besitzen keine rechnerausführbaren Objekte. Sie werden in der Erwartung eingeführt, daß ihre Unterklassen, die sich im Diagramm hinter den Abstraktionsebenen und Sichten verbergen, zu Struktur und Verhalten ihrer Oberklassen beitragen, in der Regel durch Implementieren der typischerweise unvollständigen Methoden (*virtual functions* in C++). Die in der Hierarchie oberste *system-level*-Klasse nimmt als sogenannte *base class* die globale Entwurfsspezifikation auf. Je nach Zielvorgabe wird der Entwurfsauftrag im Ableitungsgraphen bis zur geforderten Spezialisierungstiefe verfeinert (beispielsweise bis zum Layout in der *physical-circuit-domain*-Klasse). Der Übergang von den VLSI-Klassen zur Objektwelt realisiert das Versionskonzept nach Bild 8.3.

VLSI-Versions-Struktur

Während die Klassenstruktur mehr die *statischen* Aspekte der Datenmodellierung widerspiegelt, berücksichtigt die Objektstruktur zur Versionierung die *dynamischen* Aspekte des

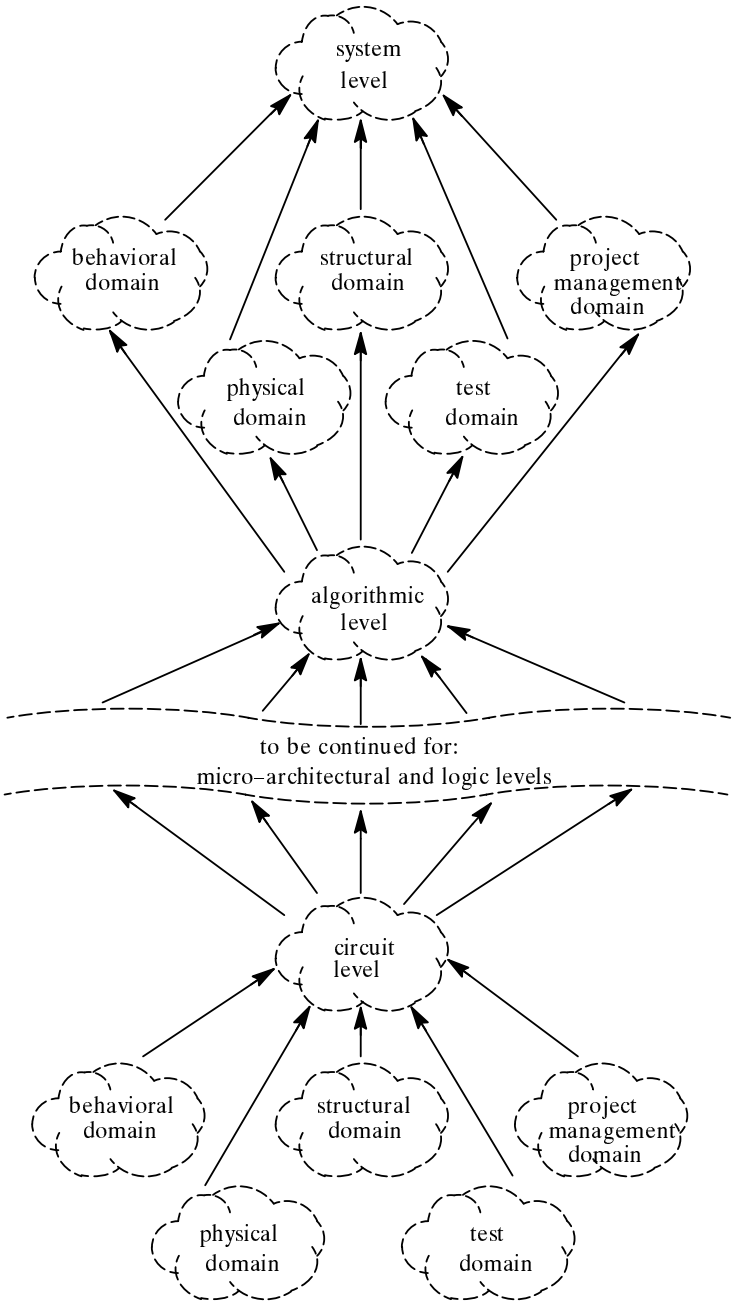


Bild 8.2: VLSI-Klassenstruktur

Entwurfsprozesses und der Datenhaltung. Objekte werden grafisch durch eine durchgezogene Umrandung von Klassen unterschieden. Exemplare werden gebildet durch Implementieren einer *instance-of*-Beziehung zur entsprechenden Klasse: *create object of class* ... Um möglichst einfache Verhältnisse in der Verwaltung zeitlicher und alternativer Abfolgen von

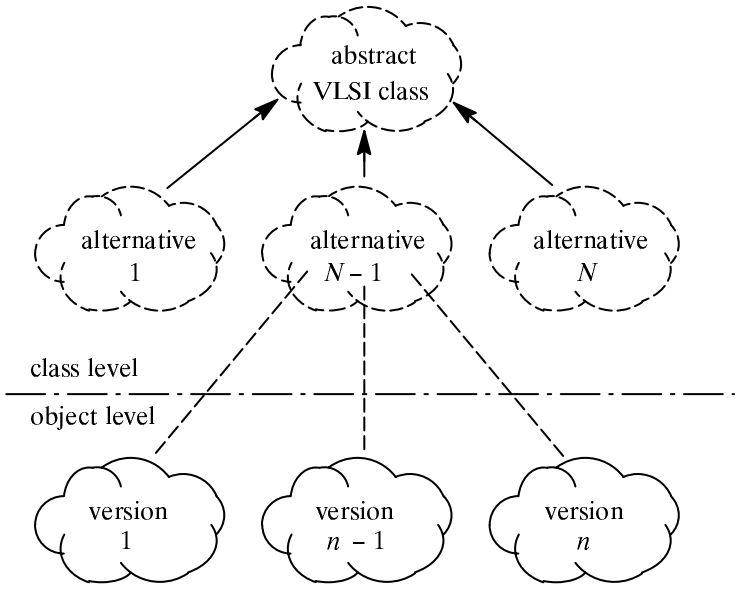


Bild 8.3: VLSI-Versionsstruktur

Entwicklungszuständen zu erreichen, unterscheiden wir in der Objektevolution kategorisch zwischen Versionen und Alternativen. Alternativen sind abstrakte Klassen im obigen Sinne, die alternative Entwurfsstrategien, -algorithmen und -kriterien beinhalten. Der Übergang von der Klassen- in die Objektwelt, in der letztlich der Entwerfer mit seinen Werkzeugen arbeitet, führt auf Objektversionen, die differenziert — zum Beispiel fehlerbereinigt oder leistungsgesteigert — die Operationen und Datenstrukturen der übergeordneten abstrakten Klassen implementieren. Versionen sind somit Exemplare einer Alternative. Falls von einer gegebenen Version wiederum alternative Wege der Implementierung beschritten werden sollen, so wird die Version durch Typisierung zur Klasse erhoben. Im Bild 8.3 wäre dies die Alternative $N + 1$ (*prototypische* Klassenbildung). Von dort können dann wieder weitere Versionen entworfen werden.

8.2 Informationsmodellierung

Ursprünglich als formale Sprache für die Definition eines allgemeinen Produktmodells entwickelt, avancierte Express innerhalb weniger Jahre zum Standard-Medium der *Informationsmodellierung* in CFI und anderen Framework-Initiativen. Express knüpft unmittelbar an die Methoden und Techniken der semantischen Datenmodellierung an, wie sie beispielsweise in den erweiterten Entity-Relationship-Modellen oder in der Modellierungssprache

IDEF1x entwickelt wurden [IDEF, 1985]. Im Vordergrund steht dabei die formale Darstellung von CAD-Daten zusammen mit deren genauen Bedeutung und Bezug untereinander.

Was ist nun Express? Zunächst die Negativ-Definition: Express ist keine Programmiersprache, auch wenn sie ausführbare Anweisungen enthält. Sie ist keine Datenbank-Abfragesprache, auch wenn sie SQL-ähnliche Elemente besitzt. Sie ist auch nicht eine komplexe Datenstruktur im Sinne von Datensätzen, Feldern oder Tabellen. Nun die Positiv-Definition: Express ist eine normative Modellierungssprache, textuell und grafisch. Sie formalisiert (a) die Beschreibung der Objekte eines Diskursbereichs (*universe of discourse*) und deren Beziehungen untereinander und (b) die Beschränkungen (*constraints*), denen die strukturellen Eigenschaften unterliegen.

Die Gründe für eine vorausgehende Informationsmodellierung[3] in der Framework-Technik sind zwingend:

- Konsensfindung unter Experten verschiedener Entwurfsbereiche über den Bedeutungsinhalt der Entwurfsdaten: normative Entwurfssemantik
- Kommunikationsmittel, um ein Problem zu analysieren und eine Lösung zu spezifizieren
- Grundlage für die Implementierung als Datenaustauschformat und Datenbankschema: Referenzmodell
- konzeptioneller Unterbau von CAD-Frameworks und deren Schnittstellen

Das Inventar eines VLSI-*Weltmodells* verlangt eine diskrete Natur der Darstellungsmittel. Darüber hinaus erhöht eine grafische Form die Anschaulichkeit und unterstützt das Abstraktionsvermögen des Betrachters. Bild 8.4 gibt ein illustratives Beispiel für die Informationsmodellierung in Express-G, der grafischen Untermenge von Express [Schenk, 1990].

Das Modell beschreibt den Begriff `Person`. Eine `Person` besitzt gewisse Eigenschaften, wie Vor- und Zunamen, optional einen Rufnamen, Haarfarbe, Geburtsdatum und ein daraus abgeleitetes (DERived) Alter. Eine `Person` ist eine abstrakte (ABStract) Oberklasse in dem Sinne, daß es kein direktes Exemplar von ihr gibt. Erst die weitere Spezialisierung von `Person` führt auf ausprägbare Unterklassen, nämlich männliche und weibliche Personen. Zwischen diesen können monogame Beziehungen bestehen, was dann zusätzliche

[3] Informationsmodellierung und konzeptionelle Modellierung sind nicht synonym, obwohl sie oft einander gleichgesetzt werden. Informationsmodellierung weist eine rein *strukturelle* Zielrichtung auf (die statische Beschreibung eines Diskursbereichs), konzeptionelle Modellierung eine *strukturell-operative* Zielrichtung (statische und dynamische Beschreibung).

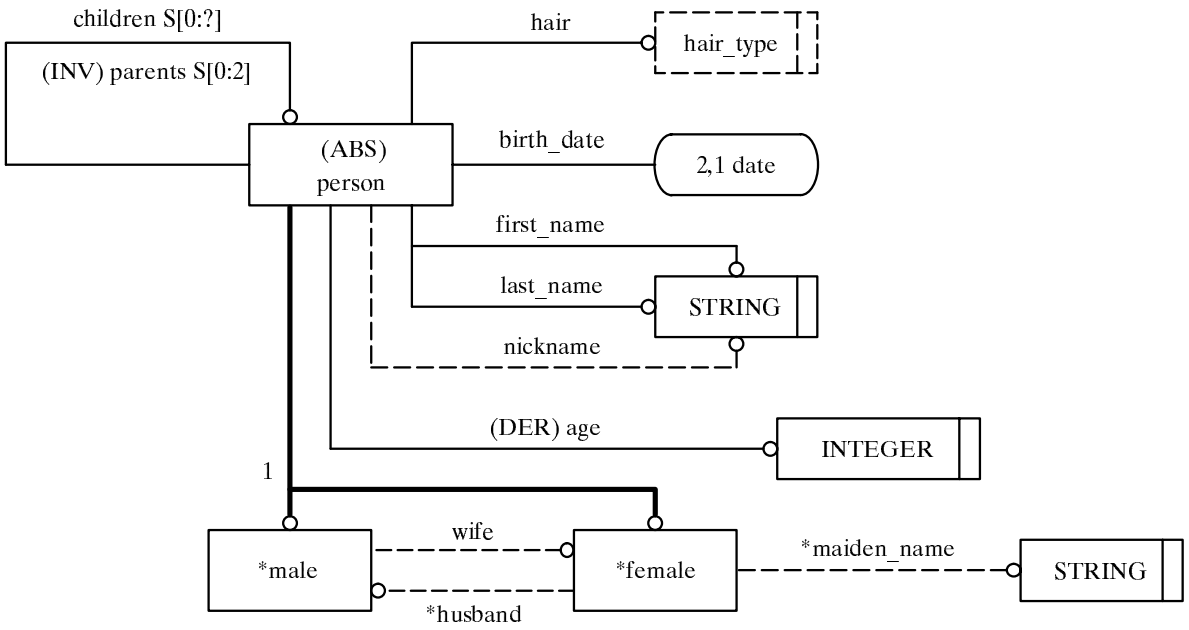


Bild 8.4: Express-G Beispiel

Eigenschaften in der Namensführung der Eheleute mit sich bringt (die Modellierung des deutschen Namenrechts dürfte sich aufwendiger gestalten). Eine `Person` kann natürlich Kinder haben, die wiederum Personen mit obigen Eigenschaften und Beziehungen sind. Der Rollenname der Eltern/Kind-Beziehung gibt Auskunft über die jeweilige Sichtweise.

Daß einige Eigenschaften bestimmten (nur in der textuellen Form von Express sichtbaren) Beschränkungen unterliegen, wird durch das vorgestellte Zeichen * vermerkt. Klassen (*entities*), die eine Menge von konzeptionellen oder realen Objekten mit gleichen Eigenschaften repräsentieren, werden in Express-G durch Rechtecke gekennzeichnet. Beziehungen sind durch Linien zwischen den Rechtecken erkennbar. Strichstärke und Strichart stehen dabei für Art und Stärke der Beziehung.

Will man VLSI-Schnittstellen modellieren, so sind grundsätzlich zwei funktionale Komponenten zu unterscheiden:

1. Werkzeug-Schnittstellen

2. Datenaustausch-Schnittstellen

Werkzeug-Schnittstellen sind prozedural und im Zugriff *on-line*. Von der Konzeption her stellen sie abstrakte Datentypen dar: Das Werkzeug kennt nur die Operatoren zum Abfragen und Manipulieren des Datenbestands. Wie der Datenzugriff organisiert ist, wie beispielsweise die Datenstrukturen oder der Mechanismus, um die Objektidentität zu gewährleisten, realisiert sind, bleibt dem Werkzeug verborgen. Datenaustausch-Schnittstellen dagegen sind deskriptiv und arbeiten in der Regel *off-line*: Der Datenaustausch erfolgt geschlossen in einer formatierten ASCII-Datei. EDIF und VHDL sind hier die Standardformate. Für beide Aspekte einer CAD-Schnittstelle müssen geeignete Sprachkonzepte angeboten werden. Express mangelt es derzeit noch an Ausdrucksmitteln für Modell*verhalten*. Prozedurale Schnittstellen lassen sich (noch) nicht beschreiben. Von Seiten der DASSY-Projektpartner wurden entsprechende Erweiterungsvorschläge in die Express-Normungsgremien getragen.

8.3 Datenaustauschformate

Die Notwendigkeit, die Syntaxinflation einzudämmen — den Informationsgehalt der Entwurfsdaten zu modellieren bevor die Form der Daten festgelegt wird —, wurde in den internationalen Standardisierungsgremien erst in letzter Zeit erkannt. Im Bestreben, Form und Inhalt, also Syntax und Semantik, nicht zu vermischen, nimmt das Austauschformat für elektronische Entwurfsdaten EDIF die Vorreiterrolle ein. Um die von vielen Anwendern prognostizierte Syntax-Krise zu überwinden, hat das EDIF Technical Committee bereits 1989 entschieden, daß zukünftige Erweiterungen des Standards nur noch auf der Grundlage eines wohlfundierten Informationsmodells zugelassen werden. Als formales Beschreibungsmittel wurde hierfür Express ausgewählt.

Aber nicht nur der Informationsgehalt zukünftiger Erweiterungen, sondern auch die Semantik der aktuellen EDIF-Version soll im nachhinein modelliert werden. Die spärlichen und zum Teil widersprüchlichen Erläuterungen im EDIF-Referenzhandbuch bieten für das Informationsmodell nur Anhaltspunkte, aber keine konsistente Gesamtbeschreibung. Wie sehr der Interpretationsbedarf angewachsen ist, zeigt der Umfang des fünfbändigen Sammelwerks „EDIF Questions and Answers“ von 1988 bis 1991. In der Sache „EDIF-Version 2.0.0“ war folglich *Reverse Engineering* vonnöten. Hierzu wurde Ende 1989 eine „EDIF European TSC Information Modelling Working Group“ (EDIF IMG) ins Leben gerufen, die sich ausschließlich mit der Informationsmodellierung der EDIF-Domäne befaßt. Die Domäne elektronischer Entwurfsdaten wurde ansatzweise auch in anderen Gremien und Projekten modelliert: *ECAD Domain Model* (EIS-Projekt [Gottlieb *et al.*, 1989]), *Conceptual Model* (ECIP [Chalmers & Meys, 1989]) und *Test Model* (EVEREST [Wahl (Hrsg.), 1992], siehe Abschnitt 3.1.2).

Im wesentlichen dreht es sich bei elektronischen Austauschformaten um die *Repräsentation* von (a) Grafik, wie Symbolen, Stromlaufplänen und Maskenlayouts, (b) logischer und struktureller Konnektivität[4] und (c) Signalmustern, wie Simulations- und Testdaten.

Die Express-G-Modelle in den Bildern 8.5 und 8.6 zeigen einige Arbeitsproben der EDIF-IMG: Bild 8.5 gibt die klassische Sichtenbetrachtung einer Zelle wieder und steht für die Über-Alles-Modellierung der EDIF-Welt. Bild 8.6 gibt ein Beispiel für das *Reverse Engineering* syntaktischer Konstruktionen. Hier wurde versucht, das Dickicht der grafischen EDIF-Primitive zu lichten, um eine übersichtliche Darstellung zu erreichen. Die textuelle und grafische Gesamtdokumentation der EDIF-Informationsmodelle kann [Wilkes (Hrsg.), 1992] entnommen werden.

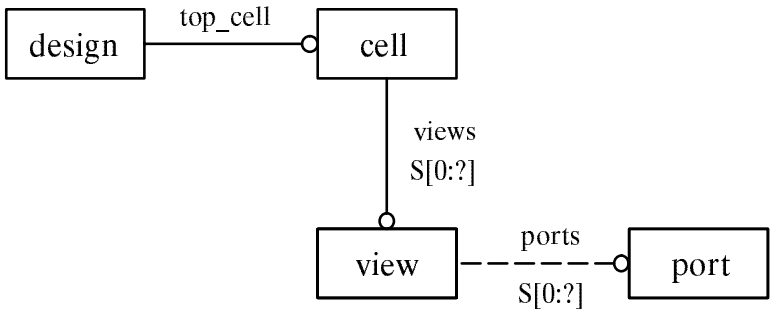


Bild 8.5: Ausschnitt aus dem EDIF-Informationsmodell: Zellen und Sichten

Zusammengefaßt können wir die Motivation für die Informationsmodellierung von EDA-Austauschformaten wie folgt skizzieren:

- Sie führt auf ein syntax-unabhängiges Modell für CAD-Objekte,
- formalisiert die Zusammenhänge und Restriktionen in der Objektwelt,
- spielt beispielsweise eine Referenzrolle, um den Informationsgehalt formatierter Daten zu vergleichen, und
- sie sollte als konzeptionelle Grundlage einer Syntax-Definition dienen.

In diesem Sinne nützt die Informationsmodellierung im Kontext des Datenaustausches sowohl den Entwicklern als auch den Anwendern der Framework-Technik.

[4]Logische Konnektivität meint *Signale*, definiert als die Menge aller elektrisch verbundener Anschlüsse; strukturelle Konnektivität meint *Netze*, verstanden als die räumlich-strukturelle Implementierung eines Signals.

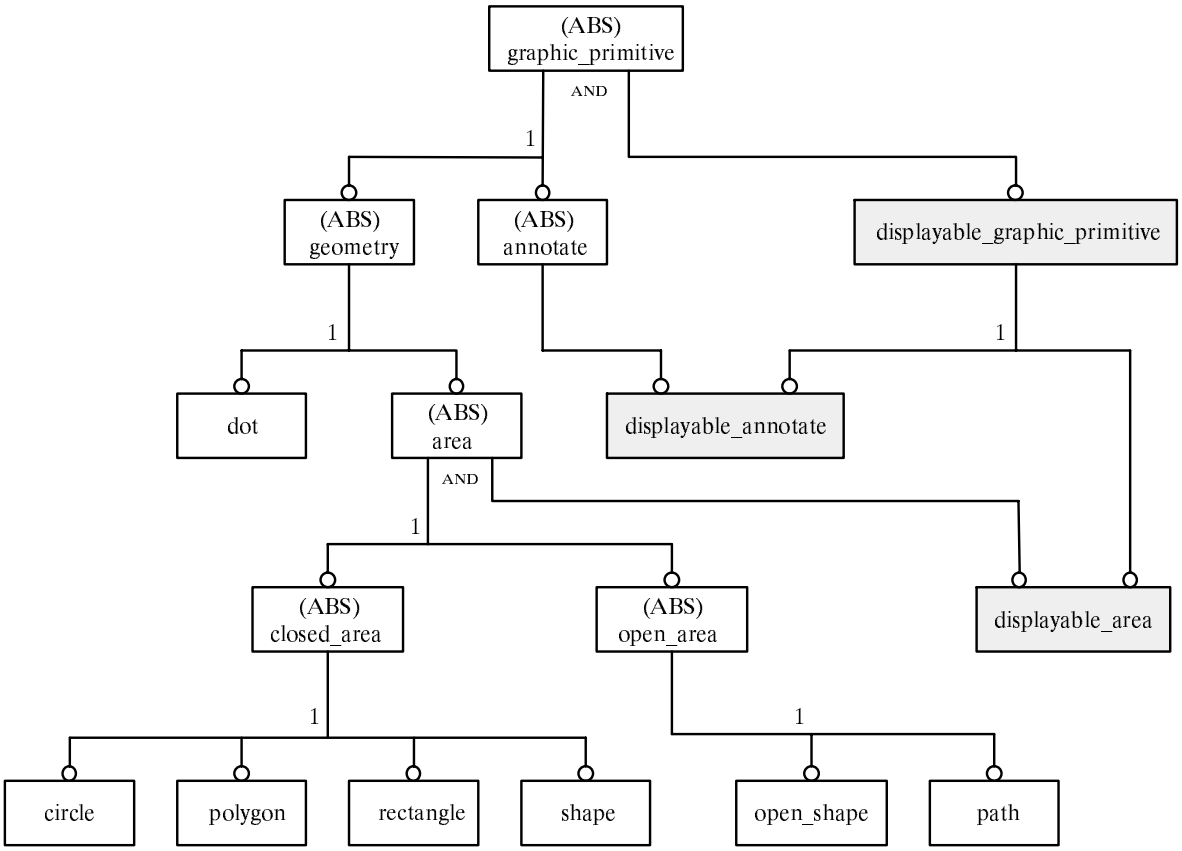


Bild 8.6: Typhierarchie der grafischen Grundelemente in EDIF

8.4 Prozedurale Schnittstellen

Wir haben bereits darauf hingewiesen: In einem Framework erfolgt die Schnittstellenanpassung nicht allein dadurch, daß Entwurfsdaten als *blobs* (*binary large objects*) transferiert werden. Vielmehr muß auch die Möglichkeit bestehen, diese Daten einzeln und im interessierenden Zusammenhang zu interpretieren und zu manipulieren. Mit anderen Worten: Entwurfswerkzeuge sollen über die Framework-Schnittstellen miteinander operieren und gegebenenfalls austauschbar sein, um so die Integrationskosten gering zu halten. Prozedurale Werkzeugschnittstellen leisten das Gewünschte. Bild 8.7 zeigt den Aufbau einer solchen, wie sie im DASSY-Projekt realisiert wurde.

Angelehnt an die Terminologie der Datenbanktechnik sprechen wir hier von einem *Datenmodell*, das es für prozedurale Werkzeugschnittstellen zu entwerfen gilt. Die Aufgabenstel-

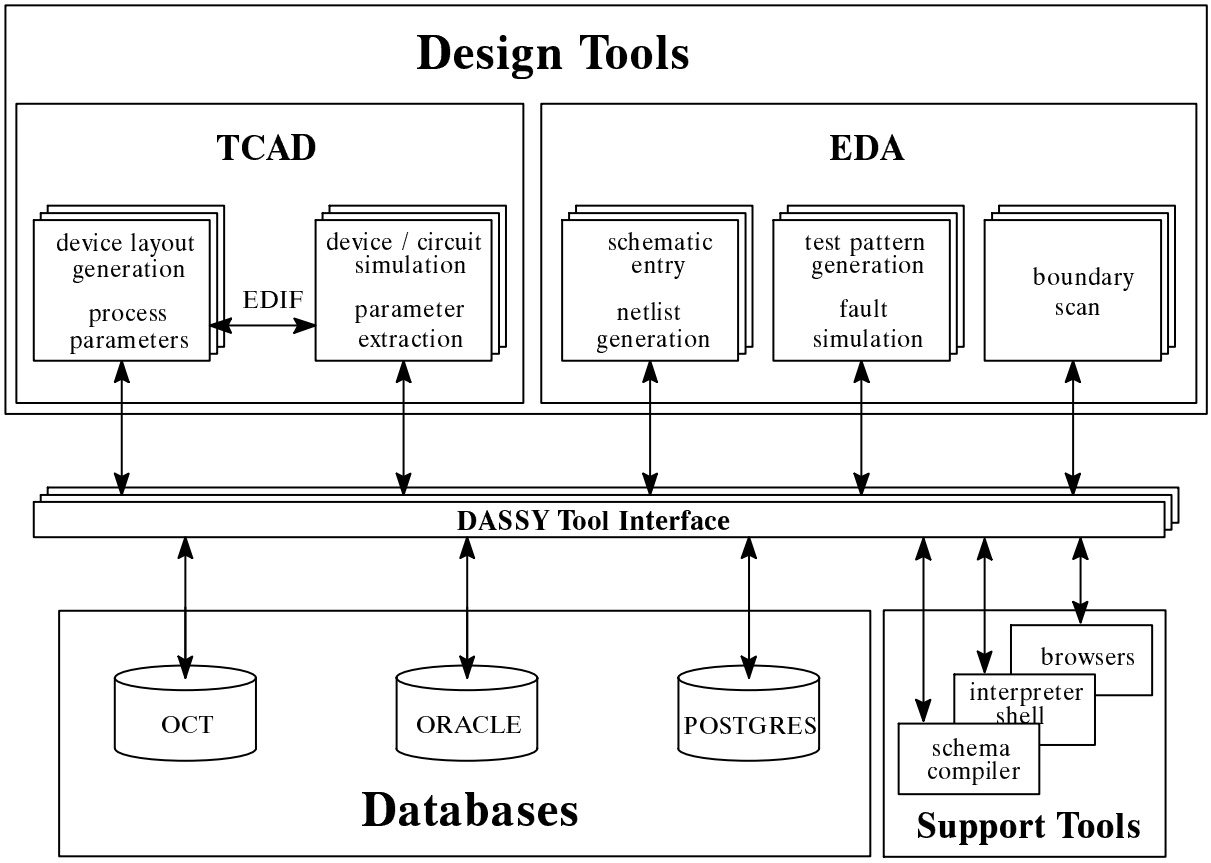


Bild 8.7: Architektur der DASSY-Werkzeugschnittstelle

lung ist ähnlich wie bei der geschilderten Informationsmodellierung für Datenaustauschformate, nur daß hier der *dynamische* Aspekt des Datenzugriffs hinzukommt. Wir sprechen deshalb auch von *konzeptionellen Modellen* in Abgrenzung zu den Informationsmodellen, die ausschließlich die statischen Aspekte eines Weltausschnitts repräsentieren. Prozedurale Werkzeugschnittstellen und ihre konzeptionelle Modellierung sind Brennpunkte der Entwicklungsaktivität in CFI und anderen Framework-Initiativen [Goldfein *et al.*, 1990]. Im folgenden wollen wir stellvertretend für die Anforderungen aus diesem Bereich das DASSY-Datenmodell (*DaDaMo*) vorstellen.

DaDaMo ist ein effizienter Konzeptmix aus der semantischen und objektorientierten Datenmodellierung. Seine Implementierbarkeit wurde prototypisch nachgewiesen [DASSY, 1992b] und seine Bedeutung für die Framework-Standardisierung international anerkannt. DaDaMo bietet erweiterte Konzepte für die Definition und Manipulation komplexer Entwurfsdaten:

Entwurfsdaten, wie Schaltungskomponenten und Verbindungen, werden grundsätzlich als *Objekte* im Sinne des Objekt-Paradigmas modelliert (Klassen und Kapselung) und gehandhabt (abstrakte Datentypen und Nachrichtenaustausch). Die strukturellen Eigenschaften eines Objekts werden durch seine attributierbaren Beziehungen zu anderen Objekten beschrieben, die dynamischen durch benutzer-definierte Operationen. Constraints erlauben zusätzlich, die Objekteigenschaften individuell einzuschränken, um den besonderen Anforderungen der Entwurfswerkzeuge zu genügen. Der Konsistenzmechanismus in DaDaMo fußt auf Constraints. Sie werden ereignisabhängig ausgelöst (*Event*→*Trigger*) und durch benutzerdefinierte Operationen befriedigt.

DaDaMo unterstützt die Modellierung von VLSI-spezifischen Weltausschnitten, indem es die Mehrfachvererbung und die freie Wahl der Beziehungstypen zuläßt. Welche Semantik eine Beziehung trägt und welche Objektstrukturen darstellbar sind, bestimmt der Anwender. Im Gegensatz zu anderen Vorschlägen drückt DaDaMo Beziehungen nicht allein durch Objektverweise aus, verbirgt das, was in den Beziehungen geschehen soll, nicht in unzugänglichem Operationscode oder beläßt es gar auf bloße Hinweise durch entsprechende Bezeichner. Beziehungstypen, die oft im Mikroelektronik-Entwurf auftreten, sind in DaDaMo bereits eingefügt:

- *relate*
  symmetrischer Beziehungstyp (nur ein Rollenname für beide Richtungen), ohne besondere operative Semantik, dient der Navigation auf assoziierten Objektmengen durch Funktionen wie *set_relation, get_relation, modify_relation* und *remove_relation*;

- *contain*
  abgeleitet vom Typ *relate*, beschreibt die Beziehung zwischen einem übergeordneten Container-Objekt, dem Aggregat, und den aggregierten inneren Objekten, wobei ein Zugriff auf das Container-Objekt auch auf die inneren Objekte und deren Beziehungen wirkt (durch Überschreiben von *get_relation*);

- *shared contain*
  abgeleitet vom Typ *contain*, die Existenz der inneren Objekte hängt aber vom Container-Objekt ab: wird es gelöscht, so auch seine Unterobjekte (durch Überschreiben von *remove_relation*), es sei denn, sie sind gleichzeitig innere Objekte anderer Container;

- *strong contain*
  abgeleitet vom Typ *shared contain*, wobei ein inneres Objekt durch spezielle Klassen-Constraints nur einem Container-Objekt angehört und somit grundsätzlich mit diesem gemeinsam gelöscht wird.

Aus den vorgegebenen Beziehungstypen können weitere abgeleitet und durch dedizierte Operationen in ihrer Semantik differenziert werden.

„Berechnete Attribute“ verringern die Redundanz und erhöhen die Konsistenz. Aktuelle Entwurfsentscheidungen können durch Attribute, berechnet auf der Objektebene, das heißt wertmäßig zur Laufzeit der Anwendung, unmittelbar weitergegeben werden. Dies vereinfacht das Management für eine Objektversionierung erheblich [Wilkes, 1988]. Die Berechnungsformeln, erforderlich für die Ableitung eines Attributwertes von dem eines anderen Objekts, werden im Schema angegeben. VLSI-Beispiel: kapazitive Belastung eines Schaltungsknotens durch die Anzahl der aktuell verdrahteten Zellanschlüsse.

Nach außen bietet DaDaMo zwei Sprachschalen für die Schemabeschreibung an: Zum einen steht eine Express-orientierte zur Verfügung, um Express-Informationsmodelle nahtlos in die Werkzeugschnittstelle einzufügen, und zum anderen eine an C++ angelehnte Sprachschale, da viele Framework-Anwendungen in C oder C++ geschrieben sind.

Fazit

Die objektorientierten Weltmodelle der Framework-Anwendungen umfassen also den statischen Informationsgehalt einer Objektmenge, vermehrt um den dynamischen Zusatz aus der konzeptionellen Modellierung der Datenhaltungs- und Werkzeugschnittstellen. Daß diese Weltmodelle nicht nur aus der Feder des Systemanalytikers stammen, sondern auch ihre Gültigkeit und praktische Bedeutung für die Entwurfs- und Implementierungsebenen besitzen, ist bislang noch kein Allgemeingut des Entwicklers. Hier greift die vielerorts geübte Kritik am mangelnden „Systemdenken“ des Entwerfers und Programmierers [Müller, 1993].

Zusammenfassend nennen wir die wichtigsten Vorteile der aus einer objektorientierten Entwurfsmethode resultierenden Klassen- und Objektdiagramme. In der Summe motivieren sie die Forschungsarbeiten zur Objektorientierung als methodisch-operativer Ansatz zur Integritätssicherung im VLSI-Entwurf.

- methodische Vorteile:
  - umfassende logische Sicht auf das zu entwerfende System und seine Komponenten
  - Strukturierung der Entwurfsidee, der Detail-Spezifikationen und der ersten Implementierungsansätze im gesamten Entwurfsraum
  - Mischsichten (keine schematisch starren Entwurfsdomänen, alle Sichten konvergieren in eine abstrakte Klasse)

- implementierungsnahe Vorteile:
  - Abbildbarkeit der Klassen- und Objektdiagramme auf die Ausdrucksmittel objektorientierter Programmiersprachen; Kongruenz zwischen Klassendiagrammen und konzeptionellen, nach Klaus DITTRICH [Dittrich, 1989] „voll objektorientierten“ Datenbankschemata (kein Strukturbruch zwischen Problemformulierung und Problemlösung)
  - Traversieren in Komplexstrukturen (gedanklich und mit Hilfe der Visualisierung in beispielsweise *Hypermedia*-Systemen: grafische *Browser* [Cordes *et al.*, 1989])
  - strukturell-operative Konsistenzsicherung durch die Vererbung von *Constraints* in der Klassen- und durch *Message passing* in der Objektwelt: Weitergabe und Einhaltung von regelhaften Zusammenhängen und Einschränkungen

Auf die implementierungsnahen Vorteile gehen wir im folgenden dritten Teil der Studie ein. Dort eruieren wir den *operativen* Aspekt der objektorientierten Integritätssicherung.

Teil III

Objektorientierung als operativer Ansatz

> *„As AI increasingly confronts real world problems, in expert systems and robotics, for example, we are increasingly likely to encounter situations where, rather than searching for a solution to a problem, we must, in a sense, search for a problem we can solve.“*
>
> Eugene FREUDER [Freuder, 1989]

Im VLSI-Entwurf bedeutet Entwerfen das hierarchische Modellieren von Objekten, die strukturell und verhaltensmäßig vorgegebene *Bedingungen* erfüllen. Es beginnt mit dem Modellieren der *algorithmischen* Entwurfsobjekte (Steuer- und Operationswerk, Register) und endet im *Full-custom*-Fall mit dem Modellieren der *geometrischen* Layoutobjekte (Logikzellen und Leiterbahnen). Der hier benutzte Objektbegriff zeichnet sich also durch *drei* Aspekte aus: Struktur, Verhalten und Beschränkung. Aus ontologischer Sicht sind sie zugleich die drei Charakteristika aller realen Objekte unserer Welt. Objekte nehmen wir erst durch ihre typischen Eigenschaften wahr, durch ihre Struktur- und Verhaltensmuster. Objekte ohne Eigenschaften sind weder wahrnehmbar noch vorstellbar, im ontologischen Sinne auch nicht existent. Objekte treten nicht isoliert auf; sie stehen in *Wechselwirkung* mit anderen Objekten. Sind diese Objekte konkreter Natur, so beschreiben physikalische Gesetze ihre Wechselwirkungen; konzeptionelle Objekte dagegen genügen unseren vernunftmäßigen Gesetzen.

Die *Gesetze*, denen der Entwerfer unterliegt, sind sowohl selbstbestimmt durch seine Anlagen und Fähigkeiten — Ratio, Intuition und Kreativität — als auch fremdbestimmt durch die technischen und wirtschaftlichen Vorgaben. Im folgenden betrachten wir die Bedeutung und Funktion dieser Gesetze näher. Wir werden sie als *Constraints* bezeichnen und dadurch einen großen Literaturfundus der Künstlichen Intelligenz erschließen. Die These lautet: Entwerfen ist ein *Constraint-Satisfaction*-Problem (CSP). Im *objektorientierten CSP-Ansatz* liegt ein wenig erforschtes Synergiepotential: Ein um das Constraint-Konzept erweitertes Objektmodell für VLSI-Entwurfsdatenbanken bietet den operativen Rahmen, um große Datenbestände *integer* zu halten.

9

Constraint-basiertes Entwerfen

Entwerfen ist kein Optimierungsprozeß. Gäbe es eine globale Zielfunktion, läge kein Entwurfsproblem, sondern eine *algorithmisch* lösbare Aufgabe vor (siehe Abschnitt 7.1). Einzelne Entwurfsschritte im globalen Suchraum können dagegen sehr wohl eine Aufgabe der Optimierung sein. Insgesamt aber ist Entwerfen das *exploratorische* Suchen nach einer *zufriedenstellenden* Lösung, die der Menge von selbst- und fremdbestimmten Entwurfskriterien genügt. Die gefundene Lösung erhebt keinen Anspruch, „perfekt“ oder „optimal“ zu sein. Herbert SIMON nennt ihre Eigenschaft *satisficing* — ein für die Suchprobleme der KI bezeichnendes Kunstwort: *satisfice* = *satisfy* + *suffice* [Gross, 1986, S. 126].

Die Suche im Entwurfsraum, der Menge aller theoretisch denkbaren Lösungsschritte, kann allein schon wegen der begrenzten Ressourcen nicht erschöpfend sein. Zum einen sind unsere kognitiven Möglichkeiten begrenzt (*bounded rationality* in der Lesart von SIMON): Einen exponentiellen Zustandsraum gedanklich aufzuspannen und systematisch sämtliche Lösungspfade abzuschreiten, übersteigt unsere kognitive Verarbeitungskapazität (siehe Abschnitt 2.1). Zum anderen sind das Projektbudget und die verfügbare Rechenleistung harte Randbedingungen: Sie diktieren den Kompromiß nach dem Kosten-Nutzen-Verhältnis. Anstelle einer kombinatorisch erschöpfenden Suchstrategie leiten Intuition und Heuristik den Entwerfer. Er gibt sich mit einer Lösung *zufrieden*, die den vorgegebenen Beschränkungen — Suchraum, Suchzeit und Suchmittel — *genügt*.

Entwerfen ist ein zielgerichteter Prozeß. Des Entwerfers Ziel sind *perspektivische* Beschreibungen für die technische Konstruktion eines Artefakts. Für VLSI-Artefakte sind dies die verhaltensmäßigen, strukturellen und physikalischen Sichten (siehe das Y-Diagramm des VLSI-Entwurfs auf Seite 127). Die Detaillierung muß dabei eine Tiefe erreichen, in der die technische Konstruktion des Artefakts beginnen kann. Derlei Detaillierungstiefen sind zum Beispiel für den ASIC-Designer die simulierte Netzliste, für den *Full-custom*-Designer das simulierte IC-Layout oder für den Software-Entwerfer der Quellcode des Programms. Insgesamt bestimmen drei Faktoren den Zielkorridor des Entwerfens: (a) das zu lösende Problem: die Spezifikation, (b) das Medium der Lösung: die Implementierung und (c) der Entwerfer selbst: seine Entwurfsentscheidungen. Der KI-Wissenschaftler Jack MOSTOW, auf der Suche nach *Better Models of the Design Process*, führt weiter aus:

„Design can be viewed as a dialectic between the designer and what is possible [...] The purpose of design is to construct a structure (artifact) description that:

1. Satisfies a given (perhaps informally) functional specification
2. Conforms to limitations of the target medium, *e.g.*, is an executable program in a given language, or is a chip layout for some fabrication technology
3. Meets implicit or explicit requirements on performance and recource usage, *e.g.*, time, space, power, cost
4. Satisfies implicit or explicit design criteria on the form of the artifact, *e.g.*, style, simplicity, testability, maintainability, reliabilty, reusability, manufacturability, modularity, etc.
5. Satisfies restrictions on the design process itself, such as its length or cost, or the tools available for doing the design (*e.g.*, drafting table versus graphic editor).“

[Mostow, 1985, S. 44]

Übersicht: Abschnitt 9.1 erläutert das constraint-basierte Entwerfen als kreativ-evolutionäres Anpassen zwischen Problem und Lösung — in der Interpretation von Christopher ALEXANDER: als *Synthesis of Form*. Um die Vielfalt der Lösungen auf eine zufriedenstellende einzuschränken, wird die *Constraint-Propagierung* eingeführt und ihre Wirkung an einem Beispiel aus dem VLSI-Entwurf gezeigt. Abschnitt 9.2 relativiert den scheinbaren Konflikt zwischen Kreativität und Constraint und erörtert die begriffliche Verwandtschaft zwischen Constraint und Integrität. Abschnitt 9.3 schließlich skizziert das thematische Umfeld des Constraint-Begriffs in den Entwurfsdisziplinen.

9.1 „Synthesis of Form“

Entwerfen ist im allgemeinen ein Syntheseproblem: Anfangs- und Zielzustand im Zustandsraum sind bekannt, wenn auch vage formuliert. Unbekannt sind die Sequenzen der Operatoren oder die Operatoren selbst, die den Anfangs- in den Zielzustand überführen (siehe Abschnitt 7.1.2). Christopher ALEXANDER umschreibt das Gemeinsame aller Syntheseprobleme mit den Worten: „to achieve fitness between two entities: the form in question and its context“. Der *Kontext* definiert das Problem; die Lösung wird durch die *Form* repräsentiert. „The real object of discussion is not the form alone, but the ensemble comprising the form and its context“ [Alexander, 1964, S. 15]. Bild 9.1 veranschaulicht ALEXANDERs Vorstellung vom allgemeinen Entwerfen als *adaptivem* Vorgang: „We want to put the context and the form into effortless contact or frictionless coexistence“ [Alexander, 1964, S. 19].

Der Prozeß der Anpassung verläuft *evolutionär*: Die Anforderungen sind zu Beginn des Entwurfsprozesses ungenau und unvollständig, gangbare Lösungswege *a priori* unbekannt.

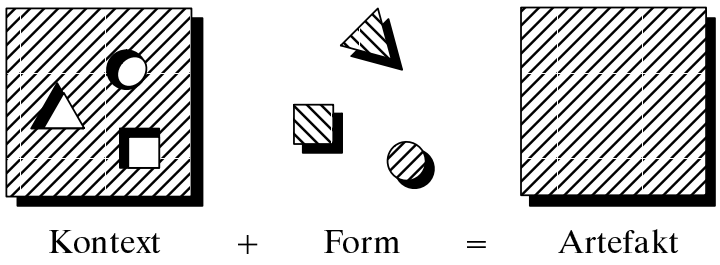


Bild 9.1: „Fitting Form & Context“

Die Beschränkungen sowohl in der Spezifikation des Kontextes (des Problems) als auch in der Realisierung der Form (der Lösung) müssen zyklisch geprüft, modifiziert und verfeinert werden. Der Anpassungsprozeß grenzt den Suchraum soweit ein, bis der Entwerfer einen Lösungsweg findet, der allen Beschränkungen „zufriedenstellend“ genügt.

Das Anpassen ist dabei nicht als *iterativer* Vorgang zu sehen. Die Unterscheidung zwischen Iteration und Evolution ist hier bedeutsam: Iteration steht für „Wiederholung“. Als Schleifen-Ablaufstruktur kennt die Iteration ihre Wiederholungsbedingung *im voraus*. *Bedingung* und *Prozeß* sind in jedem Schleifendurchlauf gleich: `repeat` *Prozeß* `until` *Bedingung* oder `while` *Bedingung* `do` *Prozeß*. Im Entwurfsprozeß dagegen ändern sich Prozeß und Bedingung mit jedem neuen Entwurfszyklus, zumal auch die Prozeßsicht und der Abstraktionsgrad wechseln (vergleiche mit der Vorgehensweise im Y-Diagramm oder dem Spiralmodell nach BOEHM [Boehm, 1988]). Anforderungen und Entwurf sind im Fluß.[1] Entwerfen ist somit evolutionär, das heißt spiralförmig „entwickelnd“. Wäre Entwerfen iterativ, bliebe es auf einer Ebene der Entwurfsspirale stehen.

9.1.1 Reduktion der Vielfalt: Constraint-Propagierung

An jeder Pfadkreuzung im Lösungslabyrinth muß sich der Entwerfer erneut entscheiden, welchem Pfad er folgen will. Bei n Kreuzungen mit m Verzweigungen sind das m^n verschiedene Pfade! Mit jeder Entscheidung jedoch reduziert sich die Zahl der noch verbleibenden Lösungspfade drastisch (in symmetrisch binären Entscheidungsbäumen zum Beispiel um die Hälfte). Anfänglich steht dem Entwerfer eine prinzipiell unbegrenzte Vielfalt an alternativen Lösungspfaden zur Wahl; letztlich reduziert sich diese Vielfalt in Folge der getroffenen Entscheidungen auf eine Restmenge. Aus ihr wird das letzte Pfadstück bestimmt. Bild 9.2 verdeutlicht die reduktionistische Eigenschaft dieser Strategie. Die Entscheidung

[1] Siehe auch [Balzer & Swartout, 1985]: *On the Inevitable Intertwining of Specification and Implementation.*

für eine Alternative auf der Entwurfsebene i reduziert die Entscheidungsvielfalt auf der Ebene $i-1$. Die Indizierung der Ebenen ist als Zielentfernung zu interpretieren (etwa im Sinne des Kapitels 7, Seite 133): $i=0$ ist die Ebene des Zielmediums (der Quellcode eines Programms oder die simulierte Netzliste einer Schaltung), $i \to \infty$ weist in Richtung der kognitiven Ebene unseres Problembewußtseins.

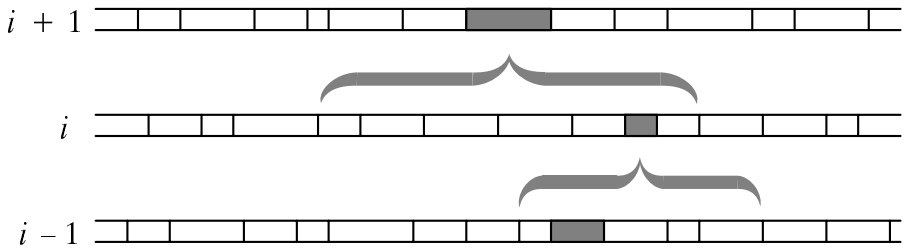


Bild 9.2: Entwerfen als Reduktion der Vielfalt

Die Reduktion kann sich durchaus *bidirektional* auswirken, worauf schon Karl KLINGSHEIM hinweist:

> „... the choices made during data transformations and the subsequent refinement of constraints *invalidate* a number of options previously available. Information is accrued in the context of antecedent decisions and it is typical *specific* to this context. Deciding on a particular clocking scheme on a VLSI chip, for example, reduces the variety of suitable architectures at higher levels.“ [Klingsheim, 1990, S. 13 (Hervorhebungen im Original)]

Wegen seiner reduktionistischen Ausrichtung stellt der Entwurfsprozeß ein klassisches Problem der Künstlichen Intelligenz dar — ein *Constraint-Satisfaction*-Problem, kurz CSP.[2] Constraints, aus dem Englischen ins Deutsche übertragen „Zwang, Beschränkung, Einschränkung“, sind gesteckte Entwurfsziele auf der operativen Ebene: technische und wirtschaftliche Anforderungen, Regeln, Konventionen oder Randbedingungen. Problemspezifische Constraints werden mit implementierungsspezifischen in Einklang gebracht, indem sie von oben (Spezifikation) nach unten (Implementierung) und umgekehrt von unten (technische Vorgaben) nach oben weitergereicht werden: *Constraint-Propagierung*. Scheitern alle kombinatorisch möglichen Versuche, die Constraints zu befriedigen, liegt eine „Überspezifikation“ vor. Das Problem ist dann unlösbar.

Die KI definiert ein CSP aus mengentheoretischer Sicht:

[2] Bezeichnenderweise ist es die KI, die sich der Theoriebildung zum allgemeinen Entwerfen verschrieben hat. Die heutigen KI-Impulse für eine *Science of Design* [Simon, 1982] kommen maßgeblich aus dem VLSI-Entwurf [AI magazine, 1990]

„A Boolean constraint satisfaction problem (CSP) is characterized as follows: given is a set V of n variables $\{v_1, v_2, \ldots, v_n\}$, associated with each variable v_i is a Domain D_i of possible values. On some specified subsets of those variables there are constraint relations given that are subsets of the Cartesian product of the domains of the variables involved. The set of solutions is the largest subset of the Cartesian product of all the given variable domains such that each n-tuple in that set satisfies all the given constraint relations. One may be required to find the entire set of solutions or one member of the set or simply to report if the set of solutions has any members — the decision problem. If the set is empty, the CSP is unsatisfiable.“ [Mackworth, 1987]

Nehmen wir zunächst ein anwendungsneutrales Beispiel, bevor wir den VLSI-Entwurf als CSP erläutern. Bild 9.3 zeigt das Constraint-Netz für zwei algebraische Constraints c_1 und c_2, die wie folgt definiert sind:[3] c_1: $v_1^2 + v_2^2 + v_3^2 = 1$ und c_2: $v_4 = f(v_1, v_2, v_3)$.

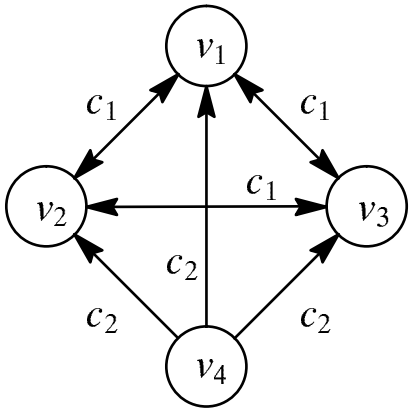


Bild 9.3: CSP

Ein Netzknoten steht für eine Variable v_i mit ihrem Wertebereich, der *Domäne* D_i. Eine Netzkante bezeichnet die Constraint-Relation $c_j(v_k, v_l)$ über den Wertebereichen D_k und D_l der Variablen v_k und v_l. Gerichtete Kanten implizieren einen kausalen Zusammenhang: Die Kante $a \longrightarrow b$ symbolisiert die Lesart „a hängt von b ab“, die Kante $a \longleftrightarrow b$ „a und b sind zwei gegenseitig abhängige Variable“. Der besondere Vorteil der Graphen-Repräsentation liegt in der Auswertung topologischer Eigenschaften: Widersprüchliche (überspezifizierte), fehlende (unterspezifizierte) und redundante Constraints können durch *Traversiermethoden* [Gossard & Serrano, 1987] entdeckt werden. Um eine Aussage über die Lösbarkeit und den Umfang der Lösung zu erhalten, müssen — beginnend mit einer bekannten oder

[3] Der Gleichheits-Operator „=“ in Constraint-Netzen und constraint-basierten Programmiersprachen meint stets eine Relation (Abbildungsvorschrift). Er ist kein Zuweisungs-Operator wie in imperativen Sprachen: „An assignment operator is unnecessary in a constraint language; the constraint-satisfaction mechanism ‚assigns‘ values to variables by finding values for the variables that make the equality relationships true“ [Leler, 1988, S. 3].

willkürlichen Anfangsbelegung einiger Variablen — allen Variablen nacheinander Werte zugewiesen werden, die den jeweiligen Constraints genügen. Im Falle zyklischer Constraints müssen die Zuweisungen gleichzeitig erfolgen, was den Rechenaufwand potenziert. Gemäß den topologischen Eigenschaften propagieren die Wertzuweisungen solange im Netz, bis eine Lösungsmenge gefunden wird, deren Elemente das CSP global lösen. Ist die Lösungsmenge leer, handelt es sich um ein überspezifiziertes, das heißt unlösbares Problem.

9.1.2 Der VLSI-Entwurf als CSP: ein Beispiel

Der Constraint-Ansatz betrachtet den Vorgang des Entwerfens als Erforschen und Ausloten von Entwurfsalternativen in einem eingeschränkten Entwurfsraum [Gross, 1986]. Die Dimensionen des Entwurfsraums sind die Entwurfsvariablen. Sie stehen für die Menge aller möglichen Entwurfsentscheidungen, zum Beispiel für die Entscheidung, eine bestimmte Logik in CMOS- oder bipolarer Technik zu realisieren. Jeder Punkt im Entwurfsraum stellt eine Entwurfsalternative dar. Ob diese das Problem löst, ist zunächst unbestimmt. Erst die Constraints des Problems grenzen den Entwurfsraum auf den Lösungsraum ein. Ein Punkt im Lösungsraum beschreibt dann eine Lösung des Entwurfsproblems, das heißt eine ausgezeichnete Menge von Entwurfsentscheidungen, die alle Constraints befriedigen. Constraint-basiertes Entwerfen vollzieht sich also in drei Schritten:

1. Definieren der Entwurfvariablen, die relevante Entwurfseigenschaften beschreiben,
2. Spezifizieren der Constraints als Relationen über den Variablen und
3. Festlegen der Anfangsbelegung einiger Variablen, um eine spezielle Lösung zu entwickeln.

Diese Schritte werden zyklisch durchlaufen. Wertzuweisungen und Constraints werden dabei revidiert und gegebenenfalls modifiziert.[4] Dimensionen und Begrenzung des Lösungsraums ändern sich sukzessive, bis alle Constraints befriedigt sind. Die Komplexität des Entwurfsproblems — die Menge der Variablen und Constraints kann im allgemeinen nicht simultan revidiert werden — setzt eine Strukturierung in Teilentwürfen voraus. Entwurfsvariable und Constraints werden dabei in Teilmengen zusammengefaßt, die untereinander nur lose gekoppelt sind. Wertzuweisungen haben dann nur *lokale* Auswirkungen. Das Problem der Constraint-Propagierung wird überschaubarer und berechenbarer. Bild 9.4 zeigt einen Ausschnitt aus einem constraint-basierten VLSI-Entwurfsprozeß [Girczyc & Ly, 1988].

[4] Gewisse Beschränkungen sind natürlich problemspezifisch, das heißt *invariant*. Sie *definieren* das Problem. Das gilt primär für die vom Anwender geforderte *Systemfunktion*.

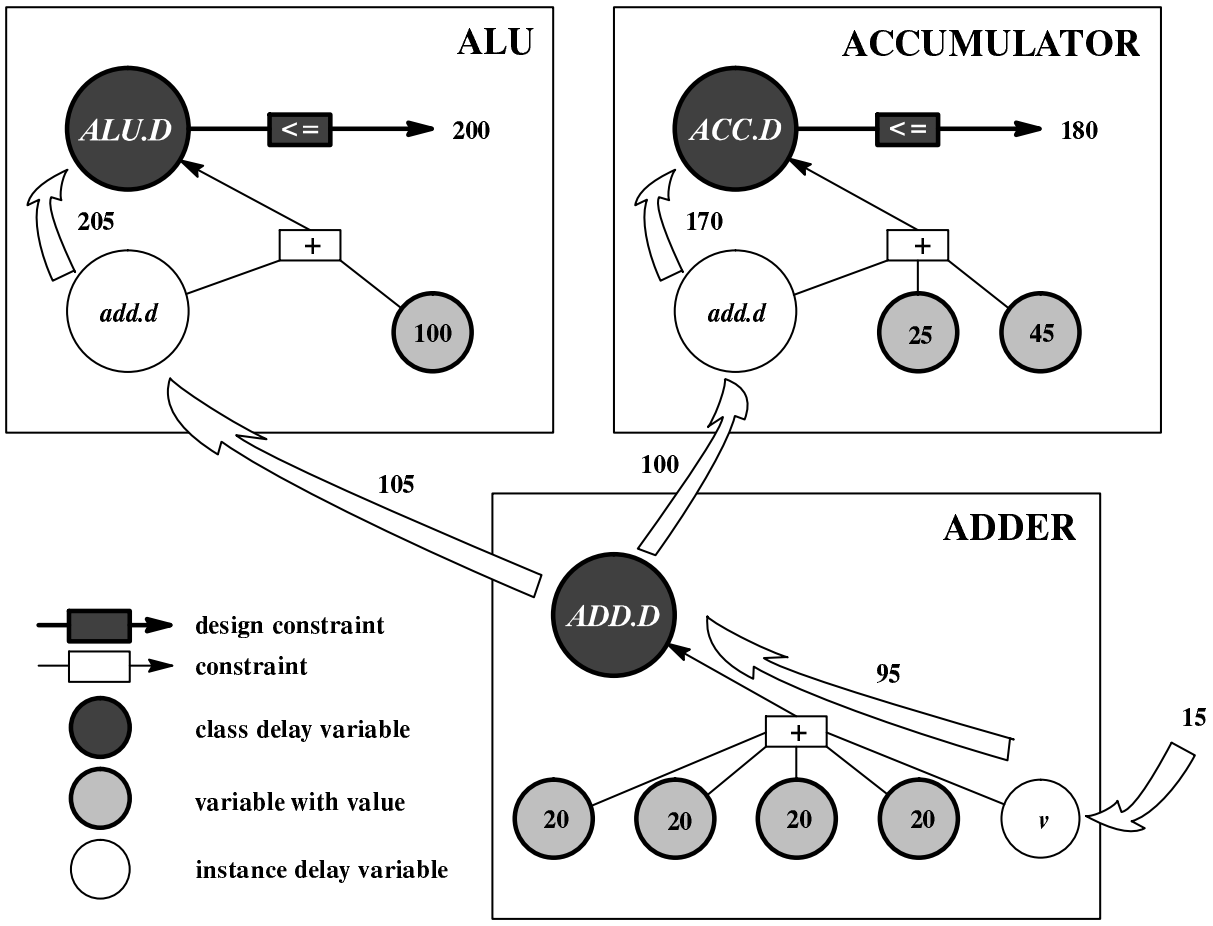


Bild 9.4: CSP: VLSI-Beispiel

Gezeigt wird die Phase, in der drei objekt- und constraint-basierte Teilentwürfe bearbeitet werden, die untereinander gekoppelt sind. Die Kopplung beschränkt sich auf eine Entwurfsvariable[5] für die Laufzeitverzögerung der Addierer-Komponente (*add.d*). Sowohl der Teilentwurf für die ALU-Klasse als auch der Teilentwurf für die Akkumulator-Klasse benutzen als Komponente eine Ausprägung (Objekt) der Klasse Addierer. Zunächst einige Erläuterungen zum dargestellten Ausschnitt: Die Laufzeit-Klassenvariablen *ALU.D* und *ACC.D* sind unterschiedlichen Entwurfsanforderungen unterworfen. Die ALU soll eine Laufzeitverzögerung von höchsten 200 ns aufweisen, der Akkumulator von höchstens 180 ns. Mit Ausnahme der jeweiligen Addierer-Komponente sind die additiven Laufzeitbeiträge der relevanten Entwurfsvariablen (sie stammen aus dem internen Entwurf von

[5] Großbuchstaben bezeichnen eine Klassenvariable, Kleinbuchstaben eine Objektvariable à la Smalltalk [Goldberg, 1984].

ALU und Akkumulator) bereits fixiert. Auf den Addierer-Komponenten lastet also eine Laufzeitanforderung von höchstens 100 ns im ALU- und 110 ns im Akkumulatorentwurf.

Der Wert der Laufzeitvariablen in der Addierer-Komponente wird durch den Mechanismus der Constraint-Propagierung bestimmt.[6] Im Addierer-Entwurf entscheidet letztlich die Objektvariable v über den Laufzeitbeitrag. Im Bild wurden der Variablen v der Wert 15 ns zugewiesen (dieser könnte beispielsweise aus dem Entschluß für eine *Carry-look-ahead*-Generierung resultieren). Die Laufzeitbeiträge summieren sich nun zur Gesamtlaufzeit der Addierer-Klasse. Die Summe wird an die Objekte dieser Klasse in den ALU- und Akkumulatorentwürfen weitergeleitet. Daß hierbei unterschiedlich hohe Werte an die jeweiligen Variablen *add.d* propagiert werden, geht auf die nicht näher dargestellten Impedanz-Einflüsse zurück (Ein- und Ausgangswiderstände und -kapazitäten der Zellen). Insgesamt führt die Constraint-Propagierung auf die Aussage, daß der ALU-Entwurf die Laufzeitbedingung verletzt, während der Akkumulatorentwurf sie erfüllt. Der Entwerfer hat nun die Wahl zwischen einem Redesign der Addierer- und einem Redesign der ALU-Klasse.

Der Mechanismus der Constraint-Propagierung akzeptiert also die *deklarative* Formulierung der Entwurfsbeschränkungen. Ihre prozedurale Bewertung obliegt dem sogenannten Constraint-Manager. Die Trennung zwischen der deklarativen Beschreibung des Entwurfsziels und der prozeduralen Prüfung seiner Realisierbarkeit erhöht die Flexibilität des Entwerfers. Er kann sich nunmehr auf das „Was“ des Entwerfens (auf das *knowing that* der *declarativists*) konzentrieren, ohne zwanghaft das „Wie“ des Machbaren im voraus zu bedenken (das *knowing how* der *proceduralists* [Winograd, 1975b]). Alan MACKWORTH umreißt den Wissensaspekt der Constraint-Modellierung mit den Worten: „Much of one's knowledge of the world is best expressed in terms of what is allowed or, conversely, what is not allowed“ [Mackworth, 1987, S. 210].

Die Constraint-Propagierung entfaltet ihre Vorteile im *bidirektionalen* Prozeß: Absteigend (*top-down*) werden die höheren Entwurfsspezifikationen auf die Implementierungsebene der Entwurfszellen und deren Komponenten propagiert. Aufsteigend (*bottom-up*) gelangen die Charakteristika der Zellkomponenten und deren Interaktionen auf die Spezifikationsebene. Einerseits sichert die *Top-down*-Spezifikation die Flexibilität in der Erforschung der Entwurfsalternativen. Andererseits reduziert die *Bottom-up*-Propagierung der Zellparameter (Laufzeit, Chipfläche, Leistung und mehr) die Gänge in der Entwurfsspirale, das heißt die Zahl der Redesigns. Die Entwurfsspezifikation muß nicht erst soweit verfeinert werden, bis erkennbar ist, ob sie realisierbar ist oder in eine technische Sackgasse führt. Die kritischen Teile einer Zelle werden nur soweit wie erforderlich entworfen. Sie begleiten und überwachen die Verfeinerung der Spezifikation für den Rest der Zelle. Diese Vorgehensweise wird als *Least-Commitment*-Strategie in der VLSI-orientierten KI-Literatur bezeichnet [Girczyc

[6] Zum Prinzip der lokalen Constraint-Propagierung: Sobald ein Knoten im Constraint-Netz genügend Information über seine Kanten erhält, berechnet er die Werte seiner noch nicht belegten Knotenvariablen und sendet diese Werte an seine Nachbarknoten. Die berechneten Werte propagieren dann entlang der Kanten und lösen die Berechnung in anderen Knoten aus, und so weiter.

& Ly, 1988; Stefik, 1981]: Entwurfsentscheidungen werden so lange wie möglich ausgesetzt. Die Propagierung kritischer Constraints von unteren auf höhere Ebenen wirkt als *Korrektiv* der Spezifikation — und das zu einem sehr frühen Zeitpunkt, wo die Kosten eines Redesigns noch gering sind.

Die bidirektionale Propagierung der Constraints kommt besonders im *bausteinbasierten* Entwurf zum Tragen. Als wiederverwendbare Bausteine fungieren im VLSI-Entwurf bekanntlich die Logik- und Layoutzellen, im Software-Entwurf die Programmodule (Klassen oder Funktionsmakros). Zwei Constraint-Ströme beschneiden den Entwurfsraum: zum einen die Entwurfsanforderungen, formuliert als Constraint-Netz, und zum anderen die technischen Parameter der verfügbaren Bausteine. Hier wird wieder der eingangs skizzierte zweiseitige Anpassungsprozeß zwischen „Form & Kontext“ deutlich: den Kontext bilden die Anforderungen, die Form soll sich aus den Bausteinen ergeben. Jean Patrick TSANG spricht von einem „Boomerang-Effekt“ der Propagierung [Tsang, 1990, S. 146]: Die *to-propagation* (in Richtung der Bausteinbibliothek) selektiert die Bausteine gemäß dem aktuellen Wertebereich der Entwurfsvariablen. Die *from-propagation* (von den Bausteinen kommend) wiederum beschränkt die Wertebereiche der Entwurfsvariablen gemäß der restriktiven Bausteineigenschaften. Die *to-propagation* der Entwurfsconstraints wirkt über die *from-propagation* der Bausteinconstraints auf sich selbst zurück: das ist der Boomerang-Effekt. Die Entwurfsdomänen werden zusätzlich beschnitten. Auf diese rückwirkende „Reduktion der Vielfalt“ hatte schon KLINGSHEIM hingewiesen (Seite 181).

Auf den höheren, vom Zellentwurf weiter entfernten Entwurfsebenen, wie zum Beispiel auf der Ebene der Systemarchitektur, gelten in der Regel andere „reasoning paradigms“ — traditionell die funktionale Dekomposition (siehe Fußnote 9 auf Seite 82), zukünftig die objektorientierte. Sie lassen sich aber sehr wirkungsvoll in das Constraint-Konzept integrieren, wie TSANG für die funktionale Dekompositionsmethode zeigt [Tsang, 1990]. Wir werden im nächsten Kapitel die Integration des Constraint-Konzepts in das Objekt-Paradigma zeigen. Zunächst aber sollen zwei wesentliche Aspekte des constraint-basierten Entwerfens skizziert werden.

9.2 Constraint-Aspekte

Die Funktion eines Constraints ist dual: Einerseits ist ein Constraint *deskriptiv*: Was ist der Zweck oder das Ziel eines Objekts? — andererseits *imperativ*: Um den Zweck oder das Ziel zu erreichen, halte folgende Rand- und Nebenbedingungen ein! Zum einen scheint die Constraint-Funktion in Konflikt zu stehen mit der kreativen Freiheit des Entwerfers; zum anderen verträgt sie sich offenbar mit der geforderten Integrität der Entwurfsdaten. Wir wollen den Konflikt mit der Kreativität relativieren und die Nachbarschaft zur Datenbankintegrität hervorheben.

9.2.1 Constraints versus Kreativität

Das englische Wort „Constraint“ und das entsprechende deutsche Wortfeld zwischen „Beschränkung“ und „Zwang“ sind für viele negativ besetzt: constraint-basiertes Entwerfen assoziiert eine fremdbestimmte Einengung der schöpferischen Freiheit. Wir könnten gleichwohl und der besseren Akzeptanz wegen Entwerfen mit positiv besetzten Adjektiven definieren: als *ziel-* oder *entscheidungs*-orientiert. Die synonyme Gleichsetzung scheint durchaus legitim: Wegen der extensionalen (eingrenzenden) und der intensionalen (inhaltlichen) Eigenschaft beschreiben Constraints indirekt auch die Entwurfsziele und Entwurfsentscheidungen. Da für unsere Begriffsbestimmung aber eine technische Komponente wichtig ist und wir in diesem Teil der Arbeit die *operative* Integritätssicherung diskutieren (im Kontext technischer Entwurfsdatenbanken), bleiben wir fortan beim Terminus technicus *Constraint*. Durch diese Wahl können wir auf einen großen Bestand der Informatikliteratur zurückgreifen — speziell auf die KI-Literatur zur *Constraint-Programmierung* [Borning, 1981; Leler, 1988; Saraswat, 1993; Stallman & Sussman, 1977; Steele & Sussman, 1980].

Formal betrachtet besitzt ein Constraint-Netz einen mathematisch exakt definierten Grad an schöpferischer Freiheit: das kartesische Produkt der Wertebereiche aller Entwurfsvariablen. Mit jeder Entwurfsentscheidung, das heißt mit jeder Festlegung einer Entwurfsvariablen auf einen Wert ihres Wertebereichs, verringert sich dieser Freiheitsgrad. Besinnen wir uns des Entropiebegriffs (Cees KOOMEN interpretiert die Entropie als Maß für Entwurfsfreiheit, siehe Abschnitt 6.2.2), so können wir auch formulieren: Mit jeder Wertzuweisung oder Einschränkung des Wertebereichs einer Variablen reduzieren Constraints die Entropie im Entwurfsprozeß. Die Entropie geht gegen Null, wenn die nach der Constraint-Propagierung gefundene Lösungsmenge nur ein Element besitzt. Anderenfalls verbleibt eine Restentropie, das heißt eine letzte Wahlfreiheit unter mehreren Entwurfsalternativen.

Stehen sich somit Constraints und Kreativität als Pole gegenüber? Zunächst die extremen Auswirkungen der Constraints: Eine *Überspezifikation* des Entwurfsproblems besitzt keine Lösung: widersprüchliche Constraints sind weder rational durch Expertenwissen noch kreativ durch Intuition aufzulösen. Das andere Extrem, die *Unterspezifikation* am Anfang des Entwurfsprozesses, wird gerade vom Laien als belastend empfunden: Im Gegensatz zum Experten verfügt er noch nicht über ein breites Erfahrungswissen, das heißt über effektive Entscheidungskriterien, ökonomische Präferenzen und prototypische Vergleichsentwürfe. Ohne systematische Anleitung wird ihn die exponentielle Vielfalt ungeprüfter Lösungswege überfordern. Seine Kreativität kann sich erst gar nicht entfalten. Über- und Unterspezifiaktion durch zu viele oder zu wenige Constraints sind also gleichsam kontraproduktiv und der Kreativität abträglich.

Im constraint-basierten Entwurf bleibt die Kreativität nicht zwangsweise auf der Strecke: Die unscharfe und unvollständige Spezifikation der Constraints läßt in der Regel genügend viele Freiheitsgrade. Überdies sind die Entwurfsbeschränkungen nicht absolut festgeschrie-

ben: Entwerfen versteht sich als *exploring constraints* [Gross, 1986]. Vorgeschriebene, bekannte, aber auch neue, erst während des Entwurfsprozesses erkennbare Constraints obliegen der Auswahl und Gestaltung durch den Entwerfer. Welche und wieviele Constraints, auf welche Weise der Entwerfer erforscht, ist ein Akt seines kreativen Willens. Bild 9.5 soll die Freiheitsgrade verdeutlichen.

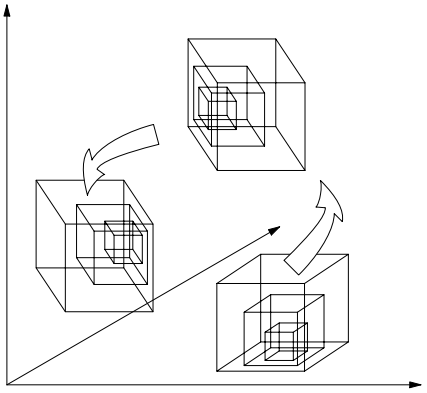


Bild 9.5: Entwerfen als Auswählen und Ausloten von Constraints

Constraints sind zweischneidig: einerseits beschränken sie die Freiheitsgrade im Entscheidungsraum, andererseits beschreiben sie gerade diese Freiheitsgrade. Sie zügeln unsere Kreativität, weisen ihr aber zugleich neue Dimensionen. Nehmen wir zum Beispiel die Maxime des modernen VLSI-Entwurfs: *Design for Testability* (DFT) [Wunderlich, 1991]. Die Maxime beschränkt die Menge aller möglichen Entwürfe auf die Menge der prüfbaren: ein wirtschaftliches Erfordernis höchstintegrierter Schaltungen. Zugleich aber wird die Kreativität des Entwerfers kanalisiert auf die im Sinne dieses Erfordernisses erfolgreichen Entwürfe. Im DFT-Entscheidungsraum kann sich die Kreativität dann wieder frei entfalten. Die ungezählten Konferenzen und Publikationen über DFT-Entwurfsmethoden, -Techniken und -Richtlinien sind Beleg hierfür. Die Zweischneidigkeit eines Constraints spiegelt sich auch in der „makroskopischen“ Ebene des Entwerfens: *technology push* und *market pull* „beschränken und beflügeln“ die Kreativität. Innovative Soft- und Hardware-Unternehmen definieren sich gerade über diese duale Eigenschaft.

Ein weiterer Aspekt zwischen Constraints und Kreativität ist zu nennen: Komplexe Artefakte genügen dem SIMONschen Grundprinzip der *Beinahen-Zerlegbarkeit*, vorausgesetzt ihre Komponenten unterliegen starken und schwachen Constraints: einer starken Kohäsion *in* den und einer schwachen Kopplung *zwischen* den Komponenten. Die Beinahe-Zerlegbarkeit kommt der rationalen Beschränkung[7] des Entwerfers entscheidend entgegen (siehe Abschnitt 2.2.1). Die vollständige constraint-freie Zerlegbarkeit widerspräche dem

[7] Auch ohne die neurophysiologische Kanalkapazität ist die Ratio beschränkt: Unsere Kognition wird auch durch Emotion und Motivation bestimmt [KI, 1994].

Systemgedanken: Das Ganze ist mehr als die Summe seiner Teile. Die schwachen Interaktionen zwischen den Systemkomponenten sind ursächlich für das makroskopische Systemverhalten. Um dieses „Mehr“ zu entwerfen, bedarf es nicht nur eines Erfahrungs- und Methodenwissens: Um die „kognitive Lücke“ zwischen dem Problem und seiner Formulierung zu überbrücken, ist der kreative Akt unabdingbar.

Es bleibt das Dilemma des Entwerfers: Er sucht den Ausgleich zwischen Anpassung und Selbstbestimmung. Ein Dilemma ist dadurch charakterisiert, daß es zwei erstrebenswerte Ziele gibt, das eine aber nur auf Kosten des anderen erreichbar ist und umgekehrt. Anpassung ist einerseits ein technisch und wirtschaftlich lohnendes Ziel: Produktstandards und Entwurfsregeln sichern eine kurze Markteinführungszeit. Selbstbestimmung andererseits schafft erst die Voraussetzung für neue Entwürfe: Neues entwickelt sich stets aus der Kombination des Alten. Neue Entwürfe aus alten zu kombinieren, ist ein assoziativer Prozeß und somit Ausdruck unserer Kreativität. Entwerfen ist also das Suchen nach dem zufriedenstellenden *Kompromiß* im Dilemma zwischen normativem und kreativem Verhalten.

9.2.2 Constraints versus Integrität

Es gibt aber auch, und das ist in unserer Betrachtung besonders wichtig, eine *bewahrende* und *sichernde* Funktion der Anpassung. In dieser Funktion spielt der Constraint-Begriff die zentrale Rolle in der Datenbanktechnik: *consistency constraints* dienen der Wahrung und Sicherstellung der Integrität großer Datenbestände [Köhler, 1992; Kotz, 1989]. Die Datenintegrität definiert sich über die Einhaltung von Constraints, das Gegenteil, die Dateninkonsistenz, über die Verletzung von Constraints.

Wie verhält sich das Constraint-Konzept zum Integritätsbegriff des Abschnitts 3.2? Dort wurde ein Datenbankschema als Quadrupel definiert: $DB =< \mathcal{S}, \mathcal{O}, \mathcal{C}, \mathcal{T} >$. Das Management der Transaktionen $\mathcal{T}$ gehört zu den Aufgaben des Datenbank-Managementsystems (DBMS). Für unsere Diskussion verkürzt sich also die formale Beschreibung der Datenbank auf das Tripel $\widetilde{DB} =< \mathcal{S}, \mathcal{O}, \mathcal{C} >$. Die Objekte $\mathcal{O}$ und die Strukturen $\mathcal{S}$ können als Netz dargestellt werden. Diesem Netz gegenübergestellt wird nun das Constraint-Netz aus $\mathcal{C}$: Constraint-Knoten korrespondieren mit Objektknoten. Im allgemeinen müssen die beiden Graphen nicht isomorph sein: Zum einen ist die Abbildung nicht unbedingt injektiv; ein Constraint-Teilnetz kann verschiedenen Teilschemata zugeordnet sein (Beispiel: die Zuordnung von Prioritätsschlüsseln). Zum anderen ist auch Surjektivität nicht gefordert; es gibt durchaus Constraints, die keine Bildelemente von Datenbankobjekten sind: So können anwendungsneutrale Constraints vom Projekt- oder Datenbank-Management vorgegeben sein, zum Beispiel Kostenfaktoren oder Constraints zur Vermeidung von Versionskonflikten. Es unterliegt dem Gutdünken des Entwerfers und orientiert sich am Grad der Rechnerunterstützung (CAx), wie homomorph (= strukturähnlich) die Abbildung sein soll oder muß.

In der KI unterscheidet man mehrere Konsistenzebenen eines Constraint-Netzes [Mackworth, 1987]. Sie klassifizieren den Begriff entlang der Linie *lokale* → *globale*-Konsistenz:

1. Unäre[8] Constraints typisieren eine Variable unabhängig von anderen. Eine Integer-Variable v_i zum Beispiel ist typisiert durch ein einstelliges Prädikat $P_i(v_i)$, das nur ganzzahlige Werte erlaubt. Die Einhaltung unärer Constraints in einem Constraint-Netz wird als *knotenbezogene* Konsistenz bezeichnet (*node consistency*). Der algorithmische Aufwand, um die Knotenkonsistenz zu berechnen ist linear: $O(n)$ [Tsang, 1990, S. 138].[9]

   Definition für Knotenkonsistenz:

   $$\forall x.\ x \in D_i \Rightarrow P_i(x). \tag{9.1}$$

   In Worten: Für jedes x im Wertebereich D_i ist das Prädikat $P_i(x)$ wahr.

   Operation, um Knotenkonsistenz zu erreichen:

   $$D_i \leftarrow D_i \cap \{x | P_i(x)\}. \tag{9.2}$$

2. Binäre Constraints setzen zwei voneinander abhängige Variable in Relation. Die Einhaltung binärer Constraints gewährleistet die *kantenbezogene* Konsistenz (*arc consistency*). Die Komplexität entsprechender Alogrithmen liegt in $O(ea^2)$ [Tsang, 1990, S. 138].[10]

   Definition für Kantenkonsistenz:

   $$\forall x.\ x \in D_i \Rightarrow \exists y.\ y \in D_j \wedge P_{ij}(x, y). \tag{9.3}$$

   In Worten: Für jedes x in D_i gibt es mindestens ein y in D_j, so daß das zweistellige Prädikat $P_{ij}(x, y)$ wahr ist.

   Operation, um Kantenkonsistenz zu erreichen:

   $$D_i \leftarrow D_i \cap \{x | \exists y.\ y \in D_j \wedge P_{ij}(x, y)\}. \tag{9.4}$$

[8] Constraints können unäre, binäre oder allgemein n-äre (n-stellige) *Relationen* sein. Sie werden in der Regel prädikativ beschrieben (Prädikatenlogik erster Ordnung).

[9] In den Aufwand für einen Konsistenzalgorithmus geht *nicht* der Aufwand für die Berechnung der Prädikate P ein. Die Größenabschätzung bezieht sich stets nur auf die formale Struktur eines Constraint-Netzes mit n als der Zahl der Netzknoten (*nodes*), e der Zahl der Kanten (*edges*) und a dem Umfang des größten Wertebereichs einer Variablen. Ein einzelnes Prädikat kann dabei *beliebig* komplex sein: Repräsentiert zum Beispiel ein Objekt eine komplexe Schaltung, stehen seine Prädikate womöglich für sehr komplexe Entwurfsbedingungen.

[10] Die hier von TSANG zitierten Konsistenzalgorithmen gehen vom Begriff *support* aus: Der Wert c_i einer Variablen v_i wird unterstützt (*supported*), wenn es einen Wert c_j der Variablen v_j gibt, so daß $P_{ij}(c_i, c_j)$.

3. Bei mehr als zwei gleichzeitig eingeschränkten Knoten spricht man von *pfadbezogener* Konsistenz (*path consistency*). Die Bestimmung entsprechender Konsistenzalgorithmen und deren Komplexität ist ein zentrales KI-Thema [Freuder, 1989; Mackworth, 1977; Shapiro (Hrsg.), 1987]. TSANG zitiert Quellen, nach denen die Komplexität entsprechender Algorithmen in $O(n^3a^3)$ liegt.

   Ein Pfad der Länge $l = 2$ vom Knoten i durch Knoten m zum Knoten j ist konsistent, wenn:

$$\forall x \forall z.\ P_{ij}(x,z) \Rightarrow \exists y.\ y \in D_m \wedge P_{im}(x,y) \wedge P_{mj}(y,z). \tag{9.5}$$

   In Worten: Für jedes x und für jedes z, für die $P_{ij}(x,z)$ wahr ist, gibt es mindestens ein y in D_m, und $P_{im}(x,y)$ und $P_{mj}(y,z)$ sind wahr.

4. k-Konsistenz verallgemeinert die Begriffe: Knotenkonsistenz führt auf 1-Konsistenz, Kantenkonsistenz auf 2-Konsistenz und Pfadkonsistenz der Länge $l = 2$ auf 3-Konsistenz. k-Konsistenz mit $(k \leq n)$ besagt, daß alle Constraints der Stelligkeit kleiner als k erfüllt sind. TSANG zitiert Quellen, nach denen die Komplexität entsprechender Algorithmen in $O(n^2(a+1)^n)$ liegt. *Streng* k-konsistent ist ein Constraint-Netz, wenn es j-konsistent für alle $j \leq k$ ist. FREUDERS Algorithmus für k-konsistente Netze liegt in $O(n^k)$.

Übertragen wir die Constraint-Kategorien auf die Integritätsaspekte in der Objekt-Beziehungs-Modellierung der Datenbankwelt (*Entity-Relationship-Modellierung* nach Peter CHEN [Chen, 1976]), so gilt die folgende Analogie:

1. Die Knotenkonsistenz bezieht sich auf die *individuelle* Integrität der Datenbankobjekte eines gegebenen Datenbankschemas. Dabei werden alle Objekte isoliert betrachtet, das heißt ohne den Einfluß ihrer Beziehungen zu anderen Objekten.

2. Die Kantenkonsistenz bezieht sich auf die *affine* Integrität der Objektpaare, die direkt zueinander in einer Nachbarschaftsbeziehung stehen. Die Einhaltung der Kantenkonsistenz wird in der Theorie relationaler Datenbanksysteme als *semijoin* bezeichnet (siehe zum Beispiel [Schlageter & Stucky, 1983]).

3. Die Pfadkonsistenz bezieht sich auf die Integrität eines Objektverbands aus mehr als zwei Objekten. Der Objektverband weist dabei eine maximale Pfadlänge $l \leq n$ auf. Die Pfadkonsistenz schließt die Knoten- und Kantenkonsistenz in diesem Verband mit ein. Vollständigkeit des Graphen eines Datenbankschemas vorausgesetzt, ist die Sicherung der Pfadkonsistenz über der Gesamtlänge $l = n$ (strenge n-Konsistenz) gleichbedeutend mit der Sicherung der *globalen* Datenintegrität der Datenhaltung.

Die *Gewährleistung* globaler Pfadkonsistenz läuft auf den Lösungsversuch des gesamten Constraint-Satisfaction-Problems hinaus. Die hierfür in der Literatur vorgeschlagenen Konsistenzalgorithmen sind NP-vollständig und damit impraktikabel (zum Beispiel nach FREUDER: $O(n^n)$). Wären sie praktikabel, wäre der Entwurfsvorgang, sobald das Constraint-Netz vollständig spezifiziert vorläge, automatisierbar. Die Komplexität des VLSI-Entwurfs ($n \gg 100$) läßt dies nicht zu.

Ein Constraint-Netz ist knoten-, kanten- und pfadkonsistent, wenn jeder Knoten, jede Kante und jeder Pfad des Netzes konsistent sind. Auf eine constraint-basierte Datenbankanwendung bezogen, läge in diesem Fall ein *integrer* Datenbankinhalt vor: Der datenbankorganisierte Entwurf wäre konsistent im Sinne aller definierten Constraints.[11] Das allgemeine Konsistenzproblem wird in der Praxis des Entwerfens drastisch gemildert: Zum einen reduziert die *Reihenfolge* der Maßnahmen zur Konsistenzsicherung den Aufwand beim *Backtracking* (dem Rückziehverfahren in der KI): Die einfache Knotenkonsistenz schließt einen Großteil aller unzulässiger Variablenwerte im vorhinein aus, so daß die mit diesen Werten gekoppelten Teilsuchräume gar nicht erst für das Backtracking durchsucht werden. Das gleiche gilt für die Kantenkonsistenz: Ihre Sicherung ist zwar aufwendiger als die vorangegangene Sicherung der Knotenkonsistenz, aber auch hier werden wieder große Teilsuchräume im voraus ausgeschlossen, die wegen ihrer pfadinkonsistenten Ausgangswerte nur Irrwege bieten.

Zum anderen reduziert die *Rangfolge* der Constraints die Constraint-Propagierung. Zum Beispiel sind technische *Worst-case*-Parameter oder physikalische Gesetze *harte* Constraints, verglichen mit den *weichen* Constraints ästhetischer Gestaltungsaspekte, beispielsweise einer grafischen Benutzungsschnittstelle. Hierarchische Constraints (zur formalen Begriffsdefinition siehe [Borning *et al.*, 1987]) schließen all jene Suchräume aus, die zwar auf der betrachteten Abstraktionebene möglich wären, aber wegen der Restriktionen tieferliegender Ebenen nicht möglich sind. Der Boomerang-Effekt, wie TSANG ihn formuliert hat (siehe Seite 186), ist ein solches Phänomen hierarchischer Constraints.

Hierarchische Constraints sind für den VLSI-Entwurf typisch: Er ist organisiert nach Sichten (Verhalten, Struktur, Physik) und Abstraktionsebenen (Architektur-, Register-Transfer- und Gatterebenen). Die den Sichten und Abstraktionen zugeordneten Variablen und Constraints sind also *partiell geordnet*. Ein integrer Weltausschnitt besteht folglich im allgemeinen aus einem Quintupel: $< V, D, C, \leq_V, \leq_C >$, wobei $\leq_V$ und $\leq_C$ für die partiellen Ordnungen über den Variablen und den Constraints stehen. Das Ordnen der Variablen V entlang zunehmender Größe ihrer Wertebereiche D zahlt sich aus für das Backtracking: Das Ordnen maximiert den durchschnittlichen Suchraum, der nach dem

[11] Nach dem Theorem von MONTANARI kann die Betrachtung des *globalen* Konsistenzproblems weiter eingeschränkt werden: Wenn jeder Pfad der Länge $l = 2$ eines vollständigen Constraint-Netzes pfadkonsistent ist, so auch das Netz [Montanari, 1974].

Scheitern eines Prädikats von der weiteren Suche ausgeschlossen wird. Auch wenn ein CSP NP-vollständig ist, profitiert eine Datenhaltung vom Constraint-Ansatz:

> „The general lesson is that by doing a limited amount of local computation at each level using, say, linear, quadratic, or cubic time, one can optimize substantial improvement in performance on some difficult problems [...]." [Mackworth, 1987, S. 209]

9.3 Variationen des Themas

Das Modell des constraint-basierten Entwerfens geht aus einem interdisziplinären Ansatz hervor. Das Thema variiert in den Entwurfsdisziplinen der Künstlichen Intelligenz, der Architektur, der rechnergestützten Konstruktion (CAx) und der Softwaretechnik. In der grafischen Datenverarbeitung spielen Constraints die zentrale Rolle [Köhler, 1992; Sapossnek, 1989]. Wir wollen die wichtigsten Parallelen und Unterschiede zusammenfassen.

9.3.1 Constraints und Problemlösen

Constraint-basiertes Entwerfen, verstanden als *Human Problem Solving* [Newell & Simon, 1972], ist ein Hauptgegenstand der KI. Herbert SIMON beschreibt den allgemeinen Entwurfsprozeß als *globales* Suchproblem[12] in einem geschlossenen Suchraum, abgegrenzt durch Constraints (siehe Abschnitt 5.1). Im Gegensatz zum hier diskutierten Constraint-Modell versteht SIMON die Vorgaben für das Entwerfen — Entwurfsziel und Constraints — als *starre* Anfangsbedingungen. Typische Entwurfssituationen sind aber nicht durch wohldefinierte und vollständig im voraus bekannte Constraints charakterisiert. Das gilt im allgemeinen auch für das Entwurfsziel. Das Problem ist nicht so sehr der Weg, sondern die exakte Beschreibung des berechenbaren Ziels. Mark GROSS formuliert dies so:

> „Deciding on an objective function is part of the design process. Also, constraints and objectives are not fixed at the outset of design, but they are introduced and changed by the designer continuously throughout the exploration for satisfactory variants." [Gross, 1986, S. 126]

Was GROSS für das Problemlösen in der Architektur anmerkt, gilt unserer Erfahrung nach auch für die Ziele und Bedingungen des Soft- und Hardware-Entwurfs. Sie sind keine *globalen* Größen, die *a priori* vollständig bekannt und für den globalen Entwurfsprozeß

[12] In der KI und der Kognitionswissenschaft dominiert die Gleichsetzung: „Problemlösen = Suchen" (siehe Diskussion in [KI, 1994]).

definierbar und dauerhaft gültig wären. Vielmehr sind dynamische und hierarchische Eigenschaften für jedes constraint-basierte Entwerfen typisch: Nicht eine globale Zielfunktion, sondern viele zum Teil konkurrierende lokale Teilziele werden verfeinert, optimiert und das unter dynamisch sich ändernden Constraints (siehe wieder Bild 9.5 auf Seite 188).

Problemlösen mit symbolischen Constraints

Sind die Wertebereiche der Variablen eines Constraint-Netzes *endlich* (zum Beispiel die Zeichen eines Alphabets ohne numerische Interpretation), so spricht man von *symbolischen* Constraints. Im Fall *unendlicher* Wertebereiche sind die Constraints *algebraischer* Natur: Gleichungen und Ungleichungen. Beide Constraint-Arten spielen eine wichtige Rolle in konkreten Anwendungen.

Bereits in den 70er Jahren verwendete David WALTZ symbolische Constraints für die Bildanalyse der *Blockwelt* [Waltz, 1972]. Diese stark eingeschränkte Lern- und Spielwelt der KI ist charakterisiert durch einfache Linienzeichnungen aus Polyedern. WALTZ identifizierte Objekte der Blockwelt anhand ihrer Konturen. Er entwickelte einen Constraint-Algorithmus, der die Konturinformation der Objektkanten auswertet. Im ersten Ansatz beschränkte sich WALTZ auf lücken- und schattenlose Zeichnungen aus Polyedern, auf Kantenlinien also, die in Knoten zusammentreffen. Um die Zeichnungen physikalisch interpretieren zu können, wurden die Polyederkanten *markiert*: Innenlinien sind entweder konkav (-) oder konvex (+); Begrenzungslinien werden als umlaufende Pfeile dargestellt (siehe Bild 9.6, aus [Winston, 1987] entnommen).

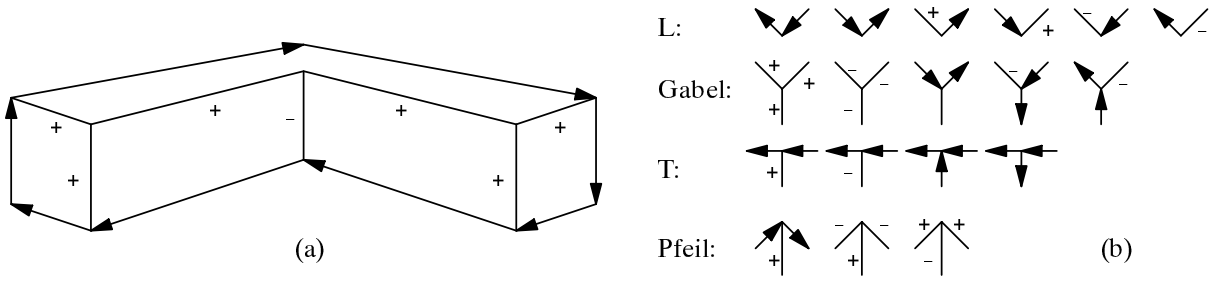


Bild 9.6: Symbolische Constraints in der Bildanalyse nach WALTZ

Die Knoten werden nach der Anzahl der zusammenlaufenden Kanten sowie den Winkeln zwischen den Kanten kategorisiert: *Gabel*, *Pfeil* und weitere. Insgesamt identifizierte WALTZ maximal 18 sinnvoll markierbare Knotentypen für Polyeder-Zeichnungen (Teilbild (b) in Bild 9.6), vorausgesetzt, jeder Knoten verbindet genau drei Flächen. Diese sinnvollen Knotentypen bilden die symbolischen Constraints in der hier betrachteten Block-

welt. Die Aufgabe des WALTZ-Algorithmus besteht nun darin, die Lösung für eine konsistente Markierung aller Knoten des Constraint-Netzes zu berechnen — im Prinzip eine kombinatorische Aufgabe, die für größere Beispiele schnell kombinatorisch „explodiert". Durch Ausnutzung lokaler Information grenzt aber der WALTZ-Algorithmus den kombinatorischen Suchraum ein und arbeitet dadurch effizient (linear bezogen auf die Zahl der Constraints [Mackworth, 1987, S. 208]): Es werden ausschließlich die unmittelbaren Nachbarknoten ins Kalkül gezogen. Nur der Wertebereich der gerade betrachteten Kantenvariablen eines Knotens unterliegt der Constraint-Propagierung: Prinzip der *lokalen* Propagierung.[13] Im Sinne der Spezifikation 9.3 und 9.4 auf Seite 190 sichert der WALTZ-Algorithmus die *kantenbezogene Konsistenz* — vom Prinzip her anwendungsunabhängig.

Wegen seiner Eleganz und grundlegenden Bedeutung für höhere Konsistenzalgorithmen zitieren wir WALTZ' *Filterverfahren* der lokalen Constraint-Propagierung (man lese anstelle von „junction" „Knoten", anstelle von „label" „Wert" und anstelle von „branch" „Kante"):

> „[The result] is obtained by going through the junctions in numerical order and:
>
> (1) Attaching to a junction all labels which do not conflict with junctions previously assigned, i.e., if it is known that a branch must be labeled from the set S, do not attach any junction labels which would require that the branch be labeled with an element not in S.
>
> (2) Looking at the neighbors of this junction which have already been labeled; if any label does not have a corresponding assignment for the same branch, then eliminate it.
>
> (3) Whenever any label is deleted from a junction, look at all its neighbors in turn, and see if any of their labels can be eliminated. If they can, continue this process iteratively until no more changes can be made. Then go on to the next junction (numerically)." (zitiert in [Mackworth, 1977, S. 105])

Als ein zweites Beispiel für das Problemlösen mit symbolischen Constraints sei auf die Arbeit von Mark STEFIK verwiesen [Stefik, 1981]. MOLGEN ist ein constraint-basiertes System für die *hierarchische Planung* von Klon-Experimenten in der Molekulargenetik. Eine hierarchische Planung unterscheidet zwischen Konzept und Detail: Beschreibungen für abstrakte Zustände (Konzepte) werden erzeugt, und die Planungsaufgabe in Teilprobleme gegliedert, in denen später die abstrakten Zustände verfeinert werden (Details). In einer komplexen Planung interagieren die Teilprobleme in der Regel (siehe SIMONs These zur „Beinahen Zerlegbarkeit" komplexer Systeme). Diese Interaktionen modelliert STEFIK mit Constraints. Sie werden dynamisch formuliert und über die hierarchischen Planungsebenen hinweg weitergereicht (in der Lesart von STEFIK: *constraint posting*). Mit anderen Worten: In MOLGEN koordinieren Constraints die Lösungen beinahe unabhängiger Teilprobleme.

[13] Die lokale Berechnung erlaubt den Einsatz parallel arbeitender Prozessoren.

Problemlösen mit algebraischen Constraints

Während der WALTZ-Algorithmus von aufzählbaren symbolischen Variablenwerten ausgeht ($\{+, -, \leftarrow, \rightarrow\}$), verlangen numerische Probleme algebraische Constraints. Im allgemeinen können diese nicht durch die explizite Aufzählung ihrer Tupel beschrieben werden, das heißt durch ihre *Extension*. Die *intensionale* Definition und Handhabung algebraischer Constraints ist stets indirekt und in der Regel aufwendiger. Am Beispiel des regelbasierten Systems EL von Richard STALLMAN und Gerald SUSSMAN wollen wir das constraint-basierte Problemlösen in einer elektrotechnischen Anwendung charakterisieren [Stallman & Sussman, 1977].

EL zeichnet sich durch zwei zukunftsweisende Eigenschaften aus:

1. *forward reasoning*

   EL folgert Schlüsse auf der Grundlage von Fakten, Regeln und Behauptungen. Der Anwendungsbereich Schaltungsanalyse legt diese Struktur der Wissensbasis nahe: *Allgemeines* Wissen wird durch die implementierten Regeln der Anwendung codiert. Darunter fallen Baustein-Charakteristiken (beispielsweise linearisierte Arbeitskennlinien von Transistoren) und Grundgesetze nach KIRCHOFF und OHM. *Spezifisches* Wissen über die konkrete Schaltung wird in Form von Fakten und Behauptungen repräsentiert: die Netzliste, Baustein-Parameter, Ströme und Spannungen zum Beispiel.

2. *dependency-directed backtracking*

   Ein wesentlicher Fortschritt, inspiriert durch EL, ist die drastische Einengung des kombinatorischen Suchraums.[14] Wissensbasierte Problemlöser wie EL sind lernfähig: Führt der Suchweg in eine Sackgasse, so wird diese markiert und von der weiteren Suche ausgeschlossen. Mehr noch: Die Ableitungskette aus Regeln und Fakten, die zum Scheitern führte, wird in einer Datenbank protokolliert; jede gescheiterte Schlußfolgerung wird somit revidierbar („recovering gracefully from failure" [Saraswat, 1993, S. 19]). Den entsprechenden Datensatz kann dann der EL-Benutzer analysieren, falsche Behauptungen revidieren. Die *Erklärungs*komponente wurde später in der KI zu einem eigenständigen problemunabhängigen Bestandteil wissensbasierter Systeme: *truth maintenance systems* [Doyle, 1979].[15]

[14] Die ersten constraint-basierten Problemlöser, zum Beispiel Micro-Planner von 1971, praktizierten ein Backtracking mit *chronologischer* Ablaufstruktur („a brute force approach" [Güsgen & Hertzberg, 1992]). Die Suchstrategie folgt dabei der Organisation eines Kellerspeichers: Die zuletzt gescheiterte Wertzuweisung wird zurückgenommen (vergleiche mit dem D-Algorithmus, um Testmuster für integrierte Schaltungen zu erzeugen [Wojtkowiak, 1989]).

[15] Das *dependency-directed backtracking* ist rückwärtsgerichtet in der Zeit. Vorwärtsgerichtete Techniken und Heuristiken „for selecting the right constraints and the right values for these constraints" steigern zusätzlich die Effizienz [Güsgen & Hertzberg, 1992, S. 65].

Die Constraint-Propagierung in EL geht zunächst von einer *1-Schritt*-Ableitung aus: Stromstärken oder Spannungen werden nach den OHMschen und KIRCHOFFschen Gesetzen bei Kenntnis aller notwendigen Argumente direkt berechnet. Fehlt aber ein Potentialwert an einem oder die Stromstärke durch ein Schaltungsbauteil, scheitert im allgemeinen die einschrittige Ableitung der gesuchten Größe. In diesem Fall wird dem Knoten eine (Hilfs)-Variable zugeordnet. Nach mehreren Ableitungsschritten im Schaltnetz weist die zuvor noch unbestimmte Knotenvariable möglicherweise verschiedene Variablenausdrücke auf mit Werten, aus denen dann die ursprüngliche Hilfsvariable eliminiert werden kann. Das Zusammentreffen verschiedener Bestimmungsausdrücke bezeichnen STALLMAN und SUSSMAN als *Koinzidenz*. Mit diesen beiden Begriffen — Koinzidenz und 1-Schritt-Ableitung — kann der Constraint-Algorithmus in EL beschrieben werden: Bild 9.7 aus [Fidelak & Güsgen, 1991].

1. Ordne einem Knoten das Potential 0 zu.
2. Schleifenanfang:
   - 2.1 Falls eine 1-Schritt-Deduktion existiert:
     - 2.1.1 Wähle eine solche aus und führe sie durch.
     - 2.1.2 Falls eine Koinzidenz auftritt, löse die entsprechende Gleichung — wenn möglich — nach einer Variablen auf und entferne diese aus dem Schaltkreis.
     - 2.1.3 Gehe zum Schleifenanfang.
   - 2.2 Falls es einen Knoten ohne Potential gibt:
     - 2.2.1 Wähle einen solchen Knoten aus und weise ihm eine neue Variable zu.
     - 2.2.2 Gehe zum Schleifenanfang.
   - 2.3 Ansonsten beende das Verfahren.

Bild 9.7: Constraint-Algorithmus in EL

9.3.2 Constraints und architektonisches Entwerfen

Christopher ALEXANDER hat ein Leitthema aus Herbert SIMONs *The Architecture of Complexity* [Simon, 1962] aufgegriffen und am Beispiel der Architektur wissenschaftlich vertieft: die Dekomposition eines komplexen Problems in kleine und einfache Teilprobleme. Auch ALEXANDER beschreibt das Entwurfsproblem als eine Menge von Anforderungen (= Constraints) an das zu entwerfende Objekt. Die Kernaussage seiner *Notes on the Synthesis of Form* [Alexander, 1964] lautet: Die Zerlegung des Entwurfsproblems in überschaubare und handhabbare Teile ist eine zentrale und zunehmend schwierige Aufgabe des Entwerfens.

Folglich entwirft ALEXANDER einen entsprechenden Algorithmus für die Zerlegung der Entwurfsanforderungen. Sein Algorithmus findet Teilmengen von Anforderungen, die in sich logisch gekoppelt sind und von anderen Anforderungen isoliert betrachtet werden können. Voraussetzung hierfür ist eine Tabelle der *links* (= Abhängigkeiten) zwischen den Anforderungen: „... two requirements are linked if what you do about one of them in a design necessarily makes it more difficult or easier to do something about the other“ [Alexander, 1964, S. 107].

Auch wenn das *Divide-et-Impera*-Prinzip mindestens so alt ist wie Julius CAESAR, ist es das Verdienst ALEXANDERs, dieses Prinzip in die Architektur eingeführt zu haben: Er lehrt als erster die systematische Anwendung der algorithmischen Dekomposition. In der Folge seiner Lehre sind zahlreiche Dekompositions- oder *Cluster*-Algorithmen entwickelt worden, zitiert zum Beispiel in [Ervin & Gross, 1987]. Verschiedene Algorithmen gruppieren Constraints auf verschiedene Weisen. Dadurch öffnen sie alternative Sichten auf das zu entwerfende Objekt. Mark GROSS beschreibt in seiner Abhandlung eine Theorie des architektonischen Entwerfens, die sich ausschließlich an der constraint-basierten Dekomposition orientiert: *Design as Exploring Constraints* [Gross, 1986].

Steven ERVIN und Mark GROSS untersuchen die Effizienz des Constraint-Ansatzes für den Entwurf von Verkehrswegen [Ervin & Gross, 1987]. Hier ist das bereichsspezifische Wissen fast vollständig durch geometrische Constraints darstellbar, so daß Constraints nicht nur unterstützend wirken, sondern die Vorgehensweise insgesamt bestimmen. Die meisten Variablen nehmen nur einfache Werte an: Koordinaten, Längen, Weiten, Kurven, Steigungen der Wegsegmente. Andere Dimensionen des Entwerfens gelten der Entwurfs*politik*, wie Verkehrsfluß und Umweltverträglichkeit. Die damit verbundenen Constraints sind topographischer Art. Entwerfen bedeutet hier primär das Erforschen der Auswirkungen alter und neuer Constraints: Variablen werden mit vorläufigen Werten belegt, Constraints hinzugefügt, Werte propagiert, neue Variablen identifiziert und so weiter. Constraint-basierte Expertensysteme mit annähernd abgeschlossenem bereichspezifischen Wissen erweisen sich in der Praxis als besonders effizient.

So wenden Boi FALTINGS et al. das Constraint-Modell geschlossen für den Brückenentwurf an [Faltings *et al.*, 1990]. Sie erweitern hierfür den allgemeinen CSP-Ansatz um den Aspekt der Dynamik: Nicht alle Constraints sind im voraus bestimmt; während des Entwerfens werden erweiterte, eingeschränkte oder neue Constraint-Sätze erforscht. Die jeweiligen Entwurfsentscheidungen bestimmen die Gültigkeit bisheriger Constraints und diktieren gegebenenfalls andere Constraint-Sätze (siehe wieder Bild 9.5). Im Constraint-Modell nach FALTINGS et al. werden numerische Intervalle propagiert. Das Constraint-Netz aus algebraischen Constraints verändert sich dynamisch mit den Entwurfsentscheidungen. Dynamische CSP treffen die Natur des ingenieurmäßigen Entwerfens.

9.3.3 Constraints und Softwaretechnik

Ivan SUTHERLAND war seiner Zeit weit voraus; sein Programm Sketchpad nahm die heutige Rechnerentwicklung vorweg: interaktive Grafik, beliebig reproduzierbare Objekte (*Master* → *Instance*-Kopien), generische Hierarchien, Bausteinbibliotheken, Listenverarbeitung und — hier besonders wichtig — constraint-basiertes Programmieren. „... in the past, we have been writing letters to rather than conferring with our computers", umreißt SUTHERLAND die Zeit der Mensch-Maschine-Kommunikation vor Sketchpad [Sutherland, 1963, S. 329]. Der Sketchpad-Benutzer beginnt mit einfachen Skizzen aus Punkten, Linien und Kreisbögen. Durch grafisches Zuordnen von Constraint-Piktogrammen werden daraus mit relativ wenig Aufwand komplexe und geometrisch exakte Zeichnungen. Bild 9.8 zeigt das Prinzip:

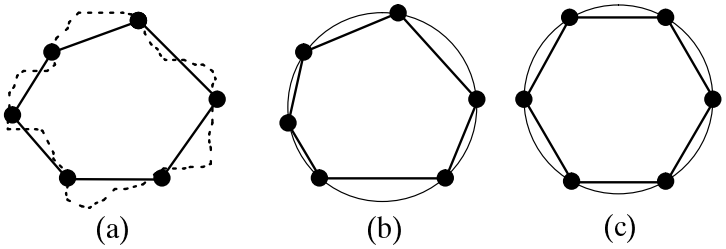


Bild 9.8: Grafische Constraints in Sketchpad

Aus der willkürlichen Freihandzeichnung eines Sechsecks, mit dem Lichtgriffel skizziert (a), wird mit Hilfe der Constraints „alle Ecken auf einem Kreis" (b) und „alle Kanten gleich lang" (c) automatisch ein reguläres Hexagon. Erst zwanzig Jahre später werden die bahnbrechenden Grafikeigenschaften von Sketchpad aktualisiert: Als Erweiterung von Smalltalk vereint Alan BORNINGs ThingLab die objektorientierte mit der constraint-basierten Programmiermethode [Borning, 1981]. Wir werden im nächsten Kapitel den vereinenden Ansatz vertiefen. Constraints in Sketchpad werden intern als Fehlerausdruck dargestellt. Ein Constraint ist befriedigt, wenn der Fehler gegen Null geht (durch lokale Constraint-Propagierung oder, falls diese scheitert, durch eine numerische *Relaxionsmethode*). Die *primitiven* Constraints sind hartcodiert; weitere lassen sich durch die *Makro*-Option, angewandt auf benutzerdefinierte Constraints, simulieren.

Sketchpad ist die erste constraint-basierte Programmiersprache. Ihre Anwendung beschränkt sich aber auf Grafikeditoren. Die Sprache „Constraints", von Guy STEELE und Gerald SUSSMAN entwickelt, verallgemeinert den Constraint-Ansatz für beliebige Anwendungen mit algebraischen Constraints [Steele & Sussman, 1980]. Bild 9.9 zeigt ein Beispiel. Die algebraischen Constraints „Addierer" (+) und „Multiplizierer" (×) verfügen jeweils über drei verschiedene Regeln, um den fehlenden Wert an einem der drei Anschlüsse zu

berechnen, wenn die Werte der beiden anderen Anschlüsse bekannt sind. Konkret bedeutet das: je nach der Verteilung der bekannten Werte und ihrer arithmetischen Funktion arbeitet der Addierer auch als Subtrahierer, der Multiplizierer auch als Dividierer. Sind zum Beispiel die Summe und einer der beiden Summanden bekannt (ihre Werte wurden an die entsprechenden Anschlüsse propagiert), wird der fehlende Summand durch eine Subtraktion bestimmt.

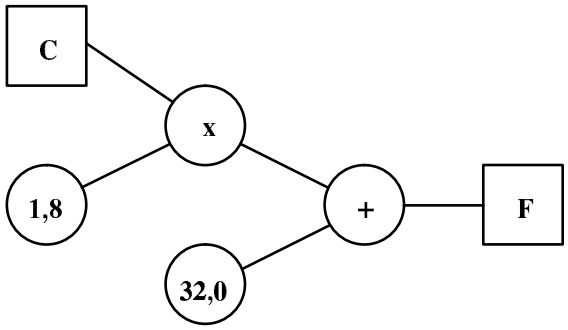


Bild 9.9: Constraint-basierter Konverter: Celsius ↔ Fahrenheit

Das Beispiel verdeutlicht zugleich den Unterschied zwischen einer *imperativen* Sprache, wie C oder Pascal, und einer *deklarativen* Constraint-Sprache, wie ThingLab, Constraints oder Bertrand [Leler, 1988]: Der in Bild 9.9 als Constraint-Netz beschriebene Konverter mit Konstanten (Kreisen), Variablen (Quadraten) und Operatoren ($+$, $\times$) erfüllt die Gleichung: $F = C \times 1,8 + 32$. In einer imperativen Sprache formuliert, wären hierfür neben der Fallunterscheidung zwei Anweisungen erforderlich: $C := \frac{F-32}{1,8}$ und $F := C \times 1,8 + 32$. Der prozedurale Ablauf müßte für jede Richtung des Datenflusses explizit festgelegt werden. Der deklarative Constraint-Ansatz hingegen trennt die Formulierung des Problems von der konkreten Lösung. Der sogenannte Constraint-Manager übernimmt die erforderlichen Zuweisungen und koordiniert den Ablauf, entlastet somit den Anwender: Dieser kann sich mehr auf das *Was* des Problems als auf das *Wie* einer Lösung konzentrieren.

Fazit

Eine jede Theorie des Entwerfens wird Constraints berücksichtigen. Constraints stehen für Entwurfsprinzipien und Entwurfsregeln, für technische und wirtschaftliche Randbedingungen, für Naturgesetze und Kulturnormen. Entwerfen ist das *Erforschen* dieser Constraints mit dem Ziel, eine zufriedenstellende Lösung des Problems zu finden, die allen oder möglichst vielen Constraints genügt. Wenn auch Constraints den Vorgang des Entwerfens prägen, eine *kreative* Vielfalt der Lösungen bleibt — zumindest für *komplexe* Probleme. Ob gleiche oder verschiedene Constraints, ob gleiche oder verschiedene Ziele: Entwerfer werden stets zu *individuellen* Lösungen gelangen. Das Anpassen zwischen „Form & Kontext", zwischen Problem und Lösung, ist ein kreativer Prozeß!

Constraints zählen zur ontologischen Begriffswelt eines jeden Menschen — nicht nur des Entwerfers. Wir sahen dies im Abschnitt 5.3. Das gleiche gilt für den Objektbegriff: Er zählt zur kognitionspsychologischen Begriffswelt. Wir sahen dies im Abschnitt 7.4.1. Was liegt nun näher, als beide Grundbegriffe zu einem gemeinsamen Konzept zu vereinen? Constraint-basiertes und objektorientiertes Entwerfen folgt unserer Intuition unmittelbar. Wir werden diesen Aspekt im nächsten Kapitel beleuchten.

10

Das Tripel des Objekts

In diesem Kapitel interpretieren wir das Objekt als *konzeptionelle Grundeinheit* des Entwerfens. Die Kapselung von Struktur, Verhalten und Beschränkung ist vollständig und sinnvoll im Sinne der Ontologie: Das ontologische Objektmodell verzahnt die statischen und dynamischen Eigenschaften durch *Gesetze*. (Der Anhang B erläutert die ontologischen Grundbegriffe und skizziert das ontologische Objektmodell.) Gesetze stehen für das *Wissen*, was die Dinge eines Weltausschnitts sein oder was sie nicht sein können. Sie schränken die kombinatorische Zustandsmenge eines Dings auf die in der Perspektive des Betrachters zulässigen Zustände ein. Gleiches gilt für das Wissen über die zulässigen Ereignisse, also was die Dinge eines Weltausschnitts werden und was sie nicht werden können. Gesetze üben somit die Funktion von Constraints aus: Struktur- und Dynamikbegriffe sind durch den Gesetzesbegriff gekoppelt.

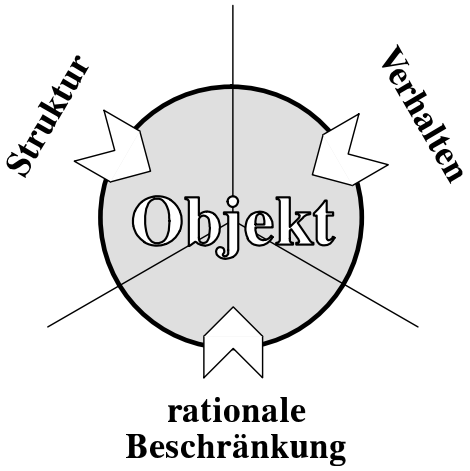


Bild 10.1: *Mind-map* zum Tripel des Objekts

Bild 10.1 macht den Grundgedanken anschaulich: Wir erweitern den klassischen Dualismus des Objekt-Paradigmas — die Kapselung von Struktur und Verhalten — um die *rationale* Komponente, ausgedrückt durch Constraints, zum *Tripel* des Objekts. Vom „Tripel des Objekts“ zu sprechen und dieses Tripel für die Modellierung eines Weltausschnitts anzu-

setzen, ist also philosophisch gerechtfertigt. Daß es auch *pragmatisch* für den Soft- und Hardware-Entwurf gerechtfertigt ist, zeigen wird im folgenden.

Übersicht: Inspiriert durch die Parallelen zwischen dem KI-Konzept der Wissensrepräsentation und den Konzepten des Objekt-Paradigmas, zeigen wir in Abschnitt 10.1 die Synergie objektorientierter und constraint-basierter Konzepte an einem frühen Beispiel. Die weiteren Abschnitte beleuchten dann die drei Aspekte eines Objekts getrennt: Der Abschnitt 10.2 charakterisiert den *Strukturaspekt*. Die Prinzipien, um die Objekte eines Weltausschnitts zu strukturieren, werden sprach- und anwendungsneutral erläutert: Abstraktion, Klassifikation, Generalisierung, Komposition und Teilungsmechanismen wie Vererbung und Delegation. Der Abschnitt 10.3 erweitert die Objektsicht um den *Verhaltensaspekt*: Funktionsbibliotheken werden von Klassenbibliotheken unterschieden und die Bedeutung klassenbasierter Verhaltensmuster für die Wiederverwendung herausgestellt. Als Kontrast zum konventionellen Operator-Operanden-Modell stellen wir das objektorientierte Kommunikationsmodell, den Austausch von Nachrichten. Wir abstrahieren von den Protokollen der Objektkommunikation, indem wir die Metapher von der „Dienstleistung" verwenden. Der Abschnitt 10.4 schließlich ergänzt um den in der Objekt-Literatur häufig ignorierten *Constraint-Aspekt*: Erst der Einschluß der rationalen Beschränkung macht die Objektsicht ganzheitlich. Im Kontext einer objektorientierten Entwurfsdatenbank skizzieren und bewerten wir die prinzipiellen Constraint-Modelle. Sie dienen uns zur Klassifikation objektorientierter Constraint-Systeme.

10.1 Synergie aus KI und Objektorientierung

Es gibt deutliche Parallelen zwischen der Wissensrepräsentation im KI-Ansatz und der konzeptionellen Modellierung im Objektansatz: Das Kapitel 4 über „Konzeptionelles Modellieren" und das Kapitel 7 über „Entwerfen als Problemlösen" haben bereits einige Parallelen impliziert; weitere expliziert die Tabelle 10.1.

Die Analogie findet ihren programmiertechnischen Niederschlag in den zahlreichen objektorientierten KI-Dialekten von Lisp: Flavors, Loops, ObjTalk und weiteren [Reimer, 1991, S. 240]. Die CLOS-Initiative (*Common Lisp Object System*) integriert beide Begriffswelten und standardisiert sie [Moon, 1989]. Die Analogie setzt sich fort im aktuellen Forschungsgebiet der *Non-Standard*-Datenbanken [Encarnação & Lockemann, 1990]. Hier verwischen zunehmend die Grenzen zwischen den KI-Modellen der Wissensrepräsentation [Reimer, 1991] und den semantischen Datenmodellen [Hull & King, 1987]. In [Patel-Schneider, 1990] findet man eine detaillierte Gegenüberstellung *objektzentrierter* Repräsentationsmodelle und *objektorientierter* Programmiersprachen. Dieses Kapitel erörtert nun

Wissensrepräsentation	*KI-Ansatz*	*Objekt-Ansatz*
kognitionspsychologisches Gedächtnismodell	*Schema* (stereotypes Erinnerungsmuster) [Bartlett, 1932]	*Semantisches Netz* [Quillian, 1967]
Repräsentationsformat	*Frame* [Minsky, 1981]	*Molecular Object* [Batory & Buchmann, 1984]
Abstraktionen (semantische Beziehungen)	Klassifikation, Assoziation, Aggregation, Generalisierung [Hull & King, 1987; Smith & Smith, 1977]	
Strukturaspekte	*Slots/Rollen*	*Attribute*
Verhaltensaspekte	*Attached Procedures* [Winograd, 1975b]	*Methods* [Goldberg, 1984]

Tabelle 10.1: Konzept-Analogie

die Integration zwischen dem *constraint-basierten* Entwurfsmodell der KI und dem *objekt-orientierten* Entwurfsmodell der Softwaretechnik.

10.1.1 Ein frühes Beispiel: ThingLab

Einen ersten und zugleich richtungsweisenden Versuch, Constraints und Objekte ganzheitlich zu betrachten, stellt Alan BORNINGs Programmiersprache ThingLab dar [Borning, 1981; Borning *et al.*, 1987]. Sie erweitert die objektorientierten Konzepte aus Smalltalk, in der sie implementiert wurde, um die Möglichkeit, Constraints zu spezifizieren und automatisch zu befriedigen. ThingLab wurde als constraint-basierte und objekt-orientierte Simulationsumgebung konzipiert mit einer interaktiven grafischen Benutzungsoberfläche. Diese steht in der Tradition von Ivan SUTHERLANDs Sketchpad (Seite 199). Die Objekte (*things*) in physikalischen und geometrischen Experimenten lassen sich in ThingLab grafisch simulieren: Beispiele sind die Simulation einfacher elektrischer Schaltungen wie in EL von STALLMAN und SUSSMAN (Seite 196) oder die Kräfteanalyse mechanischer Brückenkonstruktionen. Ein Constraint, dem die geometrischen und physikalischen Objekte unterliegen, wird durch eine *Regel* und durch eine Menge von *Operationen* repräsentiert. Die Regel dient dem Constraint-System als „prozeduraler Test“, ob der Constraint befriedigt ist. Ist der Test negativ, können die Constraint-Operationen individuell aufgerufen werden, um den Constraint zu befriedigen. Den Grad der Befriedigung drückt eine *Fehler*-Meldung aus: Der Fehler ist Null, falls der Constraint vollständig befriedigt wurde.

Es liegt in der Verantwortung des Entwerfers, die Constraints der Entwurfsobjekte korrekt zu spezifizieren. Das System dagegen ist verantwortlich, die Constraints zu befriedigen oder das Scheitern zu analysieren (widersprüchliche, unvollständige oder zirkulare Constraints).

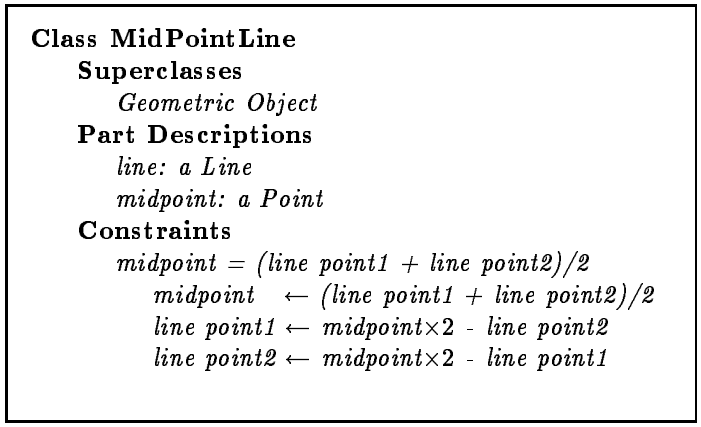


Bild 10.2: Constraints und Objekte in ThingLab

Ein Beispiel für die objektorientierte Constraint-Deklaration zeigt Bild 10.2 aus [Borning, 1981]: Statische Constraints[1] beschreiben hier die ständig gültige Bedingung zwischen den Komponenten (*parts*) eines geometrischen Objekts. Die beiden Teilobjekte der Klasse **MidPointLine** unterliegen der Einschränkung, daß das Punktobjekt auf halbem Weg zwischen den Endpunkten des Linienobjekts liegt. Die erste Zeile der Constraint-Deklaration beschreibt den „prozeduralen Test" für die Verknüpfungsbedingung; die restlichen Zeilen beschreiben die Alternativen, um den Test beim nächsten Versuch zu bestehen. Die Constraint-Operationen weisen entweder dem Mittelpunkt oder einem der Endpunkte einen Wert zu, der sich aus der Berechnung der bekannten Werte der beiden anderen Punkte ergibt.

Einerseits bietet die objektorientierte Einbettung des Constraint-Ansatzes einen wichtigen Vorteil, den BORNING so formuliert:

> „The notion of an *object* provides a basic organizational tool; in particular, the modularity gained by the use of object-oriented programming techniques is important for constraint satisfaction, where it is essential to know what is affected by a given change." [Borning, 1981, S. 353]

Andererseits stehen Constraints und Objekte auch in einem prinzipiellen Spannungsverhältnis:

> „Object-oriented languages generally emphasizes a very localized approach to interaction within a program: an object interacts with other parts of the system only by sending and

[1] Dynamische Constraints wurden erst im ThingLab-Abkömmling Animus realisiert [Borning & Duisberg, 1986]. Hier modellieren *triggerbare* Constraints die Entwicklung eines Objekts: Ein *Ereignis-Antwort*-Paar beschreibt den Triggermechanismus.

> receiving messages to other objects that it knows about. On the other hand, it is very difficult to do constraint satisfaction in a purely local way: there are problems of circularity and the like that are better spotted by a more global analysis. There is a tension between the object and constraint metaphors; the integration of these approaches in ThingLab is one of its points of interest." [Borning, 1981, S. 356]

Das Spannungsverhältnis untersuchen wir später anhand von Fallstudien. Im folgenden geht es uns zunächst um die „konzeptionelle Grundeinheit" des Entwerfens, um das Tripel des Objekts. Diese ganzheitliche Sichtweise hat, wenn überhaupt, dann nur partiell den Entwurf heutiger objektorientierte Analyse- und Designmethoden bestimmt (siehe die Methodenstudie in [Quibeldey-Cirkel, 1994c]) — obwohl Constraints und Objekte unserer Intuition am nächsten stehen. Zwar werden Constraints vereinzelt eingesetzt, um die Vor- und Nachbedingungen der Objektoperationen zu beschreiben (zum Beispiel in der Sprache Eiffel von Bertrand MEYER [Meyer, 1988]), als entwurfstheoretische Einheit aber sind Constraints und Objekte bisher nicht betrachtet worden. Dieses Manko sollen die folgenden Abschnitte offenlegen; die Sichtenintegration zum Tripel des Objekts soll das Manko beheben.

10.2 Struktur

Die Realität einer Anwendung wird durch Abstraktion und Ausschnittsbildung modelliert. Die statischen Strukturen in diesem modellierten Realitätsauschnitt, auch *Miniwelt* genannt, werden durch „repräsentative" Objekte, ihre Attribute und wechselseitige Beziehungen erfaßt. Die Warte des Systementwerfers ist dabei die Warte des Anwenders, damit die Anwendung selbst: Um sich zu lösen von den physikalischen und logischen Datenmodellen der Informationssysteme (siehe Bild 3.3 auf Seite 62), steht die *semantische* Datenmodellierung im Vordergrund. Sie vermeidet die Beschränkungen der konventionellen Datenmodelle, die sich an *Datensätzen* (*records*)[2] orientieren und nicht an der Semantik der Miniwelt.

Wir haben bereits einige für die VLSI-Repräsentation charakteristische Datenmodelle kennengelernt: Informationsmodelle in Abschnitt 8.3 in der objektorientierten Beschreibungssprache Express und objektorientierte VLSI-Weltmodelle in Abschnitt 8.1.1. In diesem Abschnitt wollen wir die allgemeinen Prinzipien hervorheben, um einen Weltausschnitt objektorientiert zu strukturieren. Wir stellen also den Objektbegriff in den Mittelpunkt.

[2] Richard HULL und Roger KING charakterisieren konventionelle Datenmodelle (relationale, netzartige und hierarchische [Schlageter & Stucky, 1983]) als *record and value oriented* und semantische Datenmodelle als *object oriented* [Hull & King, 1987]. Die konventionellen Datenmodelle identifizieren die Datensätze anhand bestimmter Schlüsselwerte (*key values*), während in semantischen Datenmodellen die Objekte wertunabhängig identifiziert werden (durch systemweit festgelegte eindeutige Objektbezeichner, auch *Surrogate* genannt).

10.2.1 Objektorientiertes Strukturieren

Bis vor wenigen Jahren wurde „Objektorientierung“ ausschließlich mit „Programmiersprachen“ assoziiert. Erst in letzter Zeit wird das Objekt-Paradigma auch konzeptionell erschlossen. Wegen seines pragmatischen Bezugs zur Klassifikationstheorie konnte es zu einer angewandten Theorie des Systementwurfs aufsteigen: „O-O is a way of organizing our thoughts about our world“ [Odell, 1992]. Diese Einsicht hätte man bereits in den Anfängen objektorientierter Sprachen gewinnen können, denn Simula-67 entsprang dem Bedürfnis, Systeme gleich welcher Art zu *simulieren* statt zu programmieren. Die Schöpfer der Sprache, Kristen NYGAARD und Ole-Johan DAHL, gaben den richtungsweisenden Ansatz für den Systementwurf: Nicht in der Beherrschung eines großen Prozesses, im Wechselspiel vieler Komponenten liegt die Lösung. Diese Komponenten gehen nicht auf eine Ad-hoc-Modularisierung der Systemfunktion zurück (funktionaler Ansatz, siehe Fußnote 9 auf Seite 82), sondern basieren auf *simulierbaren* Einheiten, *Klassen*[3] genannt. Erst das Klassenkonzept erlaubt die einfache, lokal beschränkte Simulation von Struktur und Verhalten.

Simula hat nicht nur eine neue Ära in der Programmiermethodik eingeleitet (alle objektorientierten Sprachen haben hier ihre Wurzeln), das Erbe ist vielmehr techno-philosophischer Art: es kann auf die Entwicklung beliebiger Systeme, künstlicher wie natürlicher, angewandt werden. Objektorientierte Abstraktionsprinzipien, Teilungsmechanismen und Kommunikationsmodelle bilden die gemeinsame Grundlage für Analyse, Entwurf und Implementierung. Wir gehen also bewußt einen anderen Weg, abseits von Ansätzen, die sich am Zielmedium der Implementierung orientieren. Wir vermeiden so den Einfluß der VON-NEUMANN-Rechnerorganisation auf die Entwurfsphasen, das heißt den Einfluß der strikten Trennung zwischen Struktur und Verhalten (siehe das Zitat von Linda WEISER FRIEDMAN auf Seite 15). Wir modellieren die Wirklichkeit der Miniwelt, so wie sie vom Menschen gesehen wird. Die Sicht des Menschen ist die Grundlage des objektorientierten Entwerfens. Dieser *humanistische* Ansatz erlaubt die Modellierung beliebiger Weltausschnitte, nicht nur informationstechnischer.

> „In other words, human programmers aren't TURING machines — and the less their programming systems require TURING machine techniques the better.“ [Kay, 1993, S. 81]

10.2.2 Abstrahieren

Die wohl bedeutendste geistige Leistung des Menschen ist die Abstraktion. Sie hat uns einerseits alles-und-nichts-sagende Abstrakta wie *System, Struktur* und *Prozeß* beschert und

[3] David KOEPKE datiert die Einführung des Klassenkonzepts auf eine frühe Veröffentlichung von C. A. R. HOARE: „Record Handling“ [Hoare, 1965].

nährt ohne Unterlaß die geisteswissenschaftlichen Disziplinen. Zugleich aber ist sie grundlegend für die Konstruktion technischer Artefakte aus Soft- und Hardware. Das extreme Streben der Künstlichen Intelligenz, unser Denken auf die Manipulation von Symbolen, definiert über dem binären Alphabet $\{0, 1\}$, zu verkürzen [Wojtkowiak, 1990], steht exemplarisch für die Spannweite der Abstraktion. Erstaunlich auch, daß sich abstrahierende Vorgänge auf wenige Prinzipien zurückführen lassen. Auf die wichtigsten mit technischer Bedeutung gehen wir im folgenden ein.

Zunächst eine Lexikon-Definition zur Einstimmung:

> „Abstraktion (von lat. abziehen), das Heraussondern bestimmter Merkmale in der Absicht, das Gleichbleibende und Wesentliche verschiedener Gegenstände zu erkennen, um so zu allgemeinen Begriffen und Gesetzen zu kommen, vor allem im wissenschaftlichen Denken. Welche Merkmale für wesentlich gehalten werden, hängt einerseits von der sachlichen Fragestellung, andererseits von Aufmerksamkeit, Interesse, Einsicht und Bildung ab. Man kann die Abstraktion immer weiterführen und bis zu allgemeinsten Begriffen gelangen, z. B. Hund, Tier, Lebewesen, Seiendes, Etwas." [Brockhaus-Enzyklopädie, 1990]

Mit dem Ziel, Analyse und Entwurf auf eine formale Grundlage zu stellen, hat James ODELL, auf ihn werden wir uns berufen, die Erkenntnisse der Kognitionsforschung interpretiert und die wichtigsten Zusammenhänge der Begriffsbildung herausgestellt [Odell, 1991]. Danach ist Abstraktion eine *Relation* zwischen Begriff und Objekt.

Was ist ein „Begriff", was ein „Objekt"? Ein Begriff (englisch: *concept*) ist das Ergebnis eines kognitiven Vorgangs, einer *Apperzeption* der realen Welt. Die Apperzeption folgt der *Perzeption*, der rein sensorisch-sinnlichen Wahrnehmung eines Weltausschnitts. Wir wissen um einen Begriff, wenn wir ihn anwenden können. Ein Beispiel: Um sagen zu können, wir besitzen die begriffliche Vorstellung „Person", bedarf es lediglich unserer sensorischen Fähigkeit, ein Exemplar von „Person" zu identifizieren: unseren Nachbarn beispielsweise anhand gewisser Merkmale. Die Begriffsbildung ist ein kognitiver Prozeß der *Mustererkennung*, der Erkennung stereotyper Strukturen und Verhaltensweisen. Begriffe helfen uns, unsere Welt zu ordnen: Mit dem Begriff „Person" können wir über mehrere Milliarden Individuen auf dieser Erde *in einem Gedankengang* sinnieren. Die Evolution der Begriffsbildung erläutert ODELL an einem vertrauten Bild:

> „ ... when a baby starts out in life its world is a buzzing confusion. At a very young age, it develops the notion of being fed. Soon after, it learns to differentiate the sounds of its mother and father. Humans seem to possess an innate capacity to perceive regularities and to recognize the many objects our world offers (...) Eventually, we develop concepts like ‚blue' and ‚sky'. As we get older still, we construct elaborate conceptual constructs that lead to increased meaning, precision and subtlety. For example, ‚The sky is blue only on cloudless days' and even ‚The sky is not really blue: it only looks blue from our planet Earth because of atmospheric effects.' " [Odell, 1991, S. 76][4]

[4] Die Fußnote 4 auf Seite 42 führt dieses Beispiel noch weiter aus.

Begriffe formen unsere Wahrnehmung der Wirklichkeit. Zur bewußten Wirklichkeit zählen allein die Objekte, auf die wir Begriffe anwenden können. Ein Begriff wiederum beruht auf *Tests*[5]. Sie bestimmen, wann der Begriff auf ein Objekt zutrifft. Die Wirklichkeit wird somit durch subjektive Tests auf Zugehörigkeit gefiltert: Objekte, die einen bestimmten Test bestehen, werden von uns als Ausprägung des zugehörigen Begriffs wahrgenommen (sprachlich: „sie fallen unter den Begriff"). Objekte ohne Begriff entziehen sich unserer Wirklichkeit! Die aktive Verbform *begreifen* ist hier in ihrer wörtlichen Bedeutung zu verstehen. Dagegen ist ein Begriff ohne Objekte durchaus denkbar: In diesem Fall beziehen wir den Begriff auf ein *virtuelles Objekt*, auf etwas, das es nur in unserer Vorstellung gibt (ein „Perpetuum mobile" zum Beispiel) oder erst in Zukunft geben wird (ein „Silicon-Compiler" vielleicht).[6]

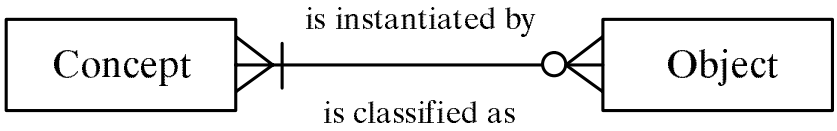


Bild 10.3: Abstraktion als Relation zwischen Begriff und Objekt

Abstraktion ist unsere geistige Fähigkeit, die Komplexität der Objekte in der Welt zu beherrschen. Sie erlaubt uns, unendlich viele individuelle Phänomene mit einem Begriff zu erfassen, zum Beispiel die (abzählbar) unendliche Menge der natürlichen Zahlen $\{1, 2, 3, \ldots\}$ mit dem Symbol $\mathcal{N}$. Bild 10.3 verdeutlicht dies in grafischer Notation: Die *Krähenfuß*-Symbolik besagt, daß ein Begriff entweder ohne Objektbezug ist oder sich auf ein oder mehrere Objekte bezieht (kenntlich durch einen leeren Kreis am Objektsymbol). Umgekehrt muß ein Objekt mindestens einem Begriff angehören (symbolisiert durch einen vertikalen Strich; zur grafischen Notation siehe [Knolmayer & Myrach, 1990]).

Objekte bilden keine Mengen von sich aus. Erst der Prozeß der Abstraktion führt auf Mengen von Objekten. Die Relation zwischen Begriff und Objekt besagt mit anderen Worten: Alle Objekte, auf die derselbe Begriff zutrifft, sind *analog*. Ohne die Bildung von Analogien wüßten wir nur, daß *alles* verschieden ist. Wir würden die „infantile Phase" unserer Wahrnehmung nie überwinden: unsere Welt bliebe eine „buzzing confusion"!

[5] Hartmut WEDEKIND bezeichnet sie als „Verwendungsnormen" oder „Regelwerke" [Wedekind, 1990, S 82].

[6] In objektorientierten Sprachen lassen sich Objekte nur erzeugen und manipulieren, wenn zuvor ihre *Klassen* vereinbart wurden (Ausnahmen lernen wir in Abschnitt 10.2.3 kennen). Eine Klasse ist, wie wir im nächsten Abschnitt sehen werden, die „Extension" eines Begriffs.

Klassifikation: Intension und Extension eines Begriffs

> „Klassifikation (aus lat. classis, Klasse)
> 1) die Einteilung von Dingen oder Begriffen, die durch gemeinsame Merkmale miteinander verbunden sind, in Klassen. Sie werden in Teilklassen unterschiedlichen Grades gegliedert. Wissenschaft und Praxis bedienen sich der Klassen, um einen Arbeitsbereich überschaubar zu machen." [Brockhaus-Enzyklopädie, 1990]

James ODELL setzt eine „Klasse" in den Kontext von „Begriff" und „Objekt". Dazu führt er zwei komplementäre Eigenschaften an: Ein Begriff hat eine *Intension* und eine *Extension*.[7] Die Intension (synonym: Definition, Zweck, Verständnis) beschreibt, ob der Begriff auf ein Objekt zutrifft, das heißt, ob das Objekt der begrifflichen Vorstellung entspricht. Die Intension versteht sich also als ein Test, angewandt auf Objekt-Kandidaten, um deren Zugehörigkeit zu prüfen. Mit anderen Worten: Die Intension eines Begriffs bestimmt das *Muster* für seine Objekte. Die Extension (Umfang) eines Begriffs steht für die *Menge* aller identifizierten Objekte, die der Intension genügen. Die Extension eines Begriffs heißt *Klasse*. Eine Klasse ist also eine Menge von Objekten, die sich durch ein gemeinsames Muster (Struktur- oder Verhaltensmuster) auszeichnen.

In diesem Sinne wird ein Begriff als *Wissenseinheit* aus zwei Komponenten eingesetzt: 1. Definition der *Zugehörigkeit* und 2. *Identifikation* zugehöriger Objekte. Zusammen mit einer symbolischen Darstellung — Symbole (Bezeichner) erleichtern die Kommunikation, ohne auf längliche Definitionen oder umfangreiche Objektmengen verweisen zu müssen — umfaßt ein Begriff also drei Aspekte, eine *Triade des Wissens*. Begriffstriaden sind grundlegend für die objektorientierte Analyse eines Weltausschnitts. Sie unterstützen den Systemanalytiker in seinem Bemühen, die Begriffe in einer Problembeschreibung zu verstehen und sich mit anderen über sie zu verständigen.

Um Mehrdeutigkeit auszuschließen, sei ab jetzt die folgende Notation vereinbart:

- « Begriff »: die französischen Anführungszeichen sollen die Abstraktion hervorheben;
- „ Intension ": die deutschen Anführungszeichen stehen für den verbalen Charakter einer Intension;
- { Extension/Klasse }: die geschweiften Klammern betonen die Eigenschaft der Extension/Klasse, eine Menge im mathematischen Sinne zu sein; und
- < Objekt >: die französischen halben Anführungszeichen deuten auf die Konkretisierung eines Begriffs.

[7] Klassische Termini der Logik: Intension meint den Inhalt, Extension den Wahrheitswert einer Aussage.

Das Beispiel in Bild 10.4 zeigt eine Begriffstriade aus dem logischen Schaltungsentwurf. Dargestellt wird der Begriff « Inverter » unter drei Aspekten: Die boolesche Gleichung „ $y = \overline{x}$ “ definiert die Eigenschaft eines Objekts, « Inverter » zu sein, das DIN-Symbol minimiert den Kommunikationsaufwand und als Beispiele für identifizierte Inverter-Objekte stehen Beschaltungen von Logikgattern, wie den Objekten < NAND >, < NOR > und < EXOR >.

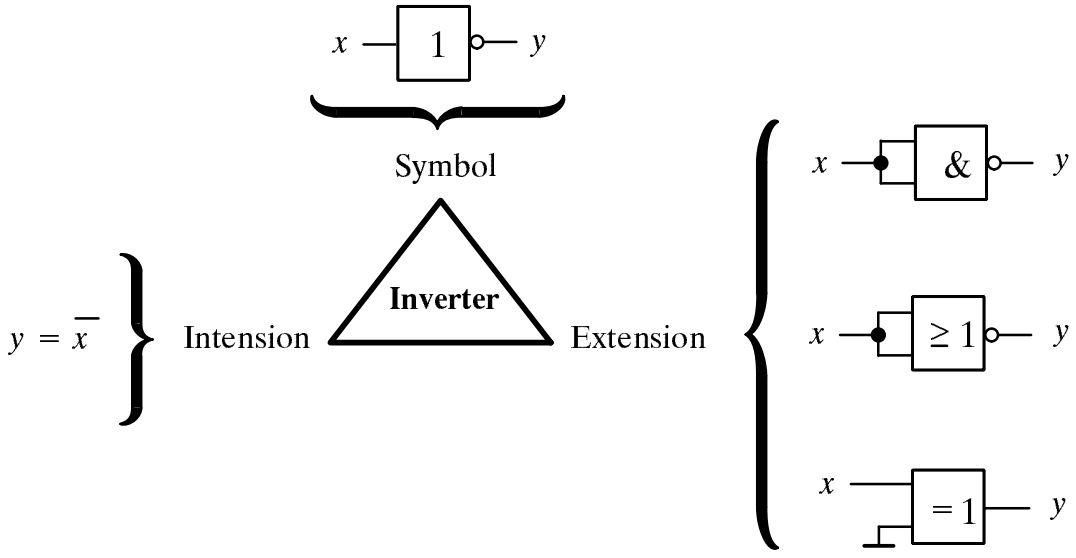


Bild 10.4: Begriffstriade: Intension-Symbol-Extension

Eine Begriffstriade muß nicht unbedingt vollständig sein: Zum ersten bedürfen Begriffe und Objekte keiner besonderen Namen. Kognitive Prozesse laufen auch ohne Symbole ab. Als Kommunikationsmittel sind sie aber der Effizienz wegen erforderlich. Zum zweiten ist ein Begriff ohne Extension, wie wir bereits gesehen haben, durchaus denkbar und wird in der Objektorientierung als *abstrakte Klasse* bezeichnet, als eine Klasse *ohne* Objekte. Sie dient der Klassifikation in der Analyse, hat aber keine eigenen ausführbaren Rechnerobjekte (siehe unser Modell zur Versionierung in Abschnitt 8.3). Der dritte Fall einer unvollständigen Begriffstriade ist unbedingt zu vermeiden: Es werden zwar Objekte einer mutmaßlichen Klasse identifiziert, deren dokumentierte Definition ist aber unauffindbar. Dieser Fall ist typisch für unvollständig dokumentierte Entwürfe. Die Klassendefinition, das heißt die Vorstellung eines Begriffs, existierte zwar zu einem früheren Zeitpunkt, verschwand aber mit dem Entwerfer der Klasse. Ein Begriff ohne Intension ist kein Begriff, sondern mit ODELLs Worten: *meaningless noise*.

Es sind also Begriffe, die unsere Wirklichkeit definieren. In der Analyse eines Problems, in der Verständnisphase, sind sie die Schlüsselabstraktionen (*key abstractions* im Sinne von

Grady BOOCH [Booch, 1991]), auf denen die weiteren Entwurfsschritte aufbauen. Im Systementwurf orientieren wir uns primär an den Extensionen der Begriffe, den Klassen. Um *reaktive* Systeme zu entwerfen [Harel, 1992], steht das Zustandsverhalten der Klassenobjekte im Mittelpunkt: Objekte werden erzeugt, ausgewählt, verändert, schließlich gelöscht. Klassen bilden dabei eine semantische Beschreibungsebene für den Objekt-Lebenszyklus, eine *Meta*-Ebene: Struktur und Verhalten der Objekte werden *mittelbar* durch ihre Klassen beschrieben. Die Klassenbeschreibung umfaßt neben den Datenstrukturen, den Zugriffs- und Änderungsoperationen im allgemeinen auch die Konstruktoren und Destruktoren zum Erzeugen und Löschen der Klassenobjekte (so zum Beispiel in der Sprache C++ [Stroustrup, 1986]). Aus dieser Sicht erscheinen Klassen als eine Organisationsform für die Objekte des betrachteten Weltausschnitts. Aus einer semantisch höheren Sicht können Klassen auch eine Meta-Ebene für andere Klassen beschreiben. Da sie auf *Mengen* von Objekten definiert sind, lassen sich folglich *Ordnungsrelationen* über Klassen konstruieren: Für die Problemanalyse sind *Generalisierung* und *Komposition* die wichtigsten.

Grundrelation I: Generalisierung

Ein Begriff ist das Ergebnis einer Abstraktion. Wir können unsere Vorstellung von der Miniwelt, gewonnen durch Abstraktionen, weiter ordnen, indem wir die Abstraktionen darin unterscheiden, ob sie allgemeiner sind als andere: *Generalisierung*. Generalisierte Abstraktionen lassen sich hierarchisch ordnen. Abstraktion und Generalisierung, trotz ihrer lateinischen Wurzeln, haben nichts Mystisches an sich. Sie sind kognitive Mechanismen und daher für den Menschen intuitiv verständlich (siehe Kapitel 7). Im Grunde handelt es sich um Relationen, die sich auf Begriffe und Objekte beziehen. Wie alle Relationen können wir auch die Generalisierung modellieren: Bild 10.5 zeigt hierfür das Meta-Modell[8].

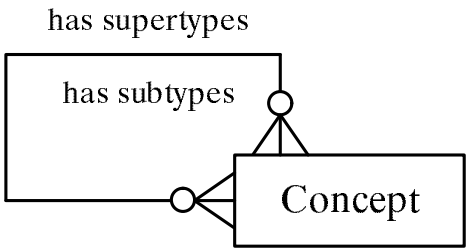


Bild 10.5: Generalisierung: Relation zwischen Unter- und Oberbegriff

[8]Meta-Modelle beschreiben das Modell eines Sachverhalts auf einer im Abstraktionsgrad höheren Darstellungsebene.

Generalisierte Abstraktionen heißen *Oberbegriffe*, spezialisierte Abstraktionen folglich *Unterbegriffe*. Die grafische Notation des Bildes 10.5 besagt, daß ein Begriff keinen oder beliebig viele Ober-und Unterbegriffe besitzen kann (bei ODELL heißen sie *types*). Bringen wir wieder die zwei Aspekte eines Begriffs ins Spiel, so können wir sagen, daß die Generalisierung den Begriffsumfang, die *Extension*, erweitert, indem sie die Begriffsdefinition, die *Intension*, aufweicht. Umgekehrt führt die Spezialisierung zu einer schärferen Begriffsintension und damit auch zu einer eingeengten Begriffsextension. Generalisierung und Spezialisierung sind konträre Blickrichtungen derselben Ordnungsrelation. Bild 10.6 zeigt hierfür ein Beispiel aus der Biologie: Die Generalisierung verallgemeinert das Besondere, und die Spezialisierung abstrahiert vom Allgemeinen auf das Besondere.[9]

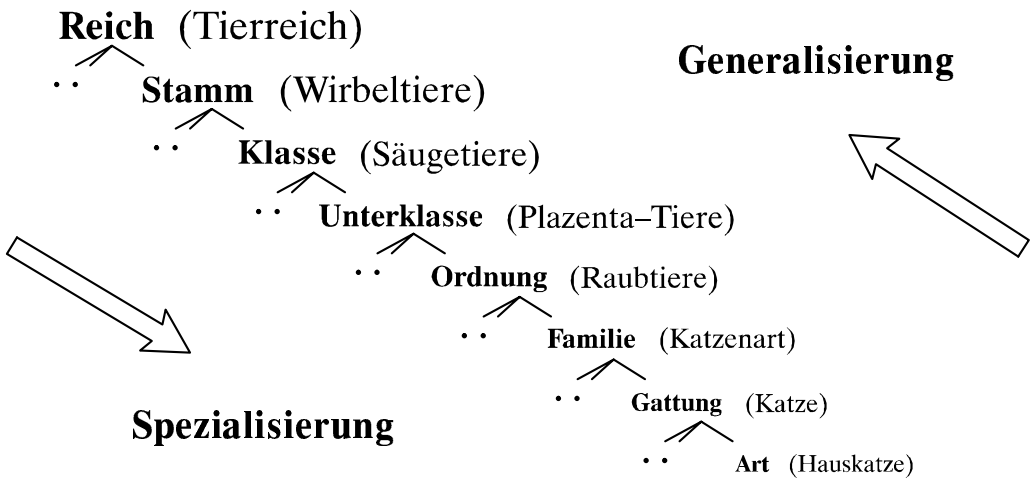


Bild 10.6: Generalisierung versus Spezialisierung

Berücksichtigen wir wieder, daß im Systementwurf primär die Extensionen der Begriffe betrachtet werden, so führt der Vorgang der Generalisierung auf eine Hierarchie aus Ober- und Unter*klassen*. Nehmen wir als simples Beispiel die Klassifikation der Zahlen in reelle ($\mathcal{R}$), natürliche ($\mathcal{N}$) und prime Zahlen ($\mathcal{P}$). Es verdeutlicht die VENN-Vorstellung generalisierter Klassen als „Mengen von Objekten in Mengen von Objekten“ und unterstreicht die *Transitivität der Inklusion*: da $\mathcal{P} \subseteq \mathcal{N}$ und $\mathcal{N} \subseteq \mathcal{R}$, so auch $\mathcal{P} \subseteq \mathcal{R}$ [Whitesitt, 1973].

[9] Wie die komplementären Prinzipien Generalisierung und Spezialisierung für die Spezifikation von Software angewandt werden können, zeigen BORGIDA et al. in [Borgida *et al.*, 1984].

Grundrelation II: Komposition

Das Ganze aus seinen Teilen zu entwickeln, hat mehrere Bezeichnungen. Sie leiten sich aus der gewählten Sicht für Analyse und Entwurf ab: Wird eine aufsteigende, das heißt *Bottom-up*-Strategie verfolgt, so sprechen wir von *Komposition*. Eine absteigende, das heißt *Top-down*-Vorgehensweise legt die Bezeichnung *Partition* oder *Dekomposition* nahe. Wird ein *zusammengesetztes* Objekt in einer Datenbank verwaltet, so heißt der Objektverband *Aggregation* [Smith & Smith, 1977]. Der Allgemeingültigkeit wegen wollen wir die Bezeichnung Komposition verwenden, zumal die *Bottom-up*-Strategie grundlegend für die objektorientierte Systementwicklung ist: Denn unter Ausnutzung wiederverwendbarer Ressourcen, das heißt in Bibliotheken archivierter Klassen, sollen neue Anwendungen aus alten konfiguriert werden. Bild 10.7 zeigt das Meta-Modell der Komposition: eine Relation zwischen Begriffen, die als Teile oder als Ganzes auftreten.

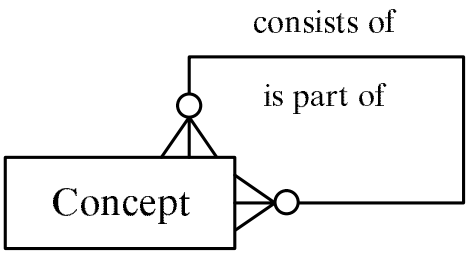


Bild 10.7: Komposition: Relation zwischen dem Ganzen und seinen Teilen

Die Komposition reduziert die Komplexität eines Weltausschnitts, indem die Konfiguration vieler Dinge als ein Begriff betrachtet wird. Wir benutzen diese Relation, wenn wir das Ganze meinen, ohne unser Wissen oder Unwissen über die Teile stets explizit anzugeben. Wann und wie wir die Komposition anwenden, beruht auf sehr subjektiven Erwägungen, an die sich eine Reihe von Fragen knüpfen: Sollen die Eigenschaften des Ganzen auch auf seine Teile übergehen? Sollen die Teile existentiell vom Ganzen abhängen, sollen also beim Löschen des Ganzen auch seine Teile gelöscht werden? Woran erkennen wir das Ganze in einem komplexen Objekt? Welche Teilobjekte dürfen sich zeitlich ändern, ohne die Komposition aufzulösen? Welche Teilobjekte sind substantiell, ohne die das Ganze seine Daseinsberechtigung verwirkt? All diese Fragen zur *referentiellen* und *existentiellen* Konsistenz müssen für jede Komposition getrennt beantwortet werden. Im folgenden wollen wir wegen der Vielfalt der möglichen Entscheidungen nur die Komposition betrachten, die sich auf *unveränderliche* Objekte oder Begriffe bezieht: *invariante Komposition* genannt.

Erläutern wir zunächst den einfachsten Fall einer *binären Relation*: Hier definieren zwei Komponenten die Komposition. Zum Beispiel hat die Relation « Ehe » nur bestand, wenn

beide Ehepartner leben, nicht getrennt und nicht geschieden sind. In diesem Fall geht die auflösende Wirkung vom Teil auf das Ganze über, wo sonst die umgekehrte Richtung vorherrscht. Die Relation « Ehe » ist also eine Assoziation zwischen zwei Begriffen, « Mann » und « Frau », die durch die Relation als Einheit betrachtet werden. Eine Relation, als Begriff verstanden, hat folglich eine *Intension* („ verheiratet sein, was Einfluß hat auf das Namen-, Erb- und Steuerrecht, auf Unterhaltsansprüche und mehr “) und eine *Extension*, die alle Ausprägungen von paarweise bezogenen « Eheleuten » umfaßt. Die Ausprägungen einer Relation heißen *Tupel*. Die Extension einer Relation ist somit eine Klasse von Tupeln.

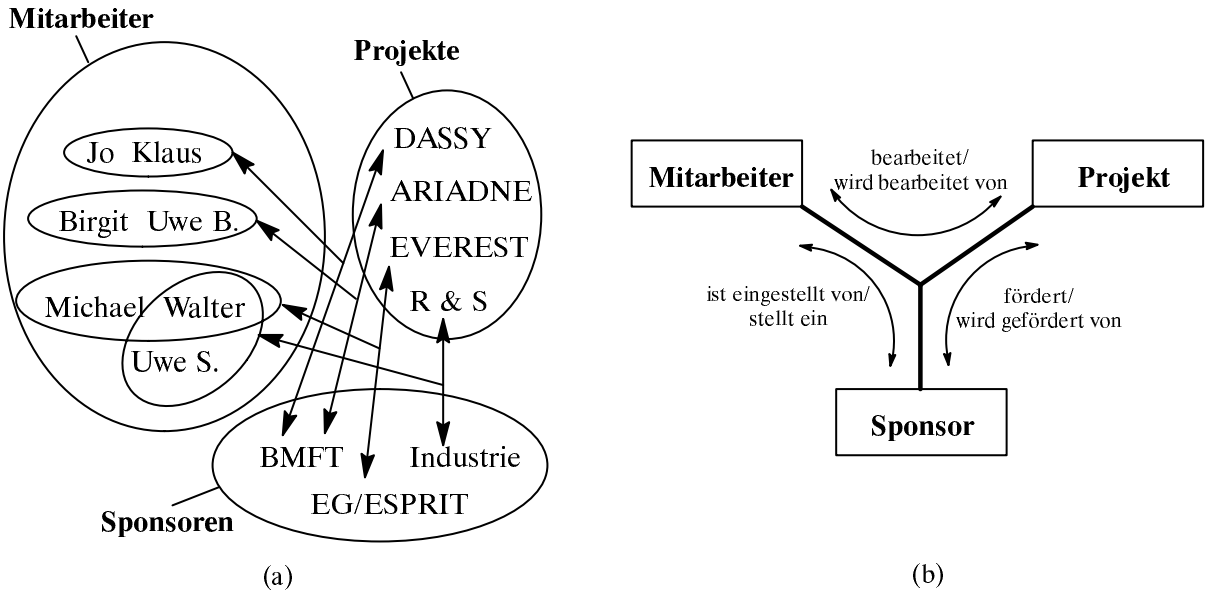


Bild 10.8: Beispiel einer ternären Relation

Relationen können *unär*, *binär* oder im allgemeinen *n-stellig* sein: Ein komplexes Objekt kann sich aus beliebig vielen Objekten zusammensetzen. Derart definierte Objekte heißen auch *n-Tupel*. Bild 10.8 gibt ein Beispiel für eine *ternäre* Relation zwischen den Begriffen « Mitarbeiter », « Projekt » und « Sponsor ». Teilbild (a) zeigt Tupel der ternären Relation « Projektzuordnung ». Das Y-Kantennetz in Teilbild (b) faßt die Tupel symbolisch zusammen. Die an die Kanten geschriebenen Verbformen stehen für die *Funktionen*, in der KI und Datenmodellierung auch *Rollen* genannt, die die Objekte aufeinander abbilden. Mit anderen Worten: Eine Funktion ist ein Prozeß, der für ein gegebenes Objekt eine Menge von Objekten liefert. Diese Menge kann auch leer sein. Mathematisch gesehen, korrespondiert eine Funktion mit einer binären Relation, das heißt, eine Funktion kann als Relation zwischen zwei Klassen behandelt werden. Funktionen sind gerichtet ({ BMFT } *fördert* { DASSY }) und haben eine Umkehrung, die *inverse* Funktion ({ DASSY } *wird gefördert vom* { BMFT }). Dadurch zeichnet sich eine Komposition insgesamt durch zwei

Eigenschaften aus: Transitivität (wie auf Seite 213 definiert) und Antisymmetrie. Antisymmetrie bedeutet: wenn $X \subset Y$, dann gilt $Y \not\subset X$ [Whitesitt, 1973].

Zusammenfassend können wir die wichtigsten Begriffsrelationen mit charakteristischen Verben beschreiben:

- Abstraktion ist eine *is-a*-Assoziation: { DASSY } *ist ein* { Projekt }.
- Generalisierung ist eine *is-a-kind-of*-Assoziation: { DASSY } *ist eine Art* { öffentlich gefördertes Projekt }.
- Komposition ist eine *consists-of*-Assoziation: { DASSY } *besteht aus* { zwei Projekt-Teilen: (a) Datentransfer und (b) Schnittstellen in offenen VLSI-Entwurfssystemen }.

Es gibt weitere wichtige Assoziationen, die in der klassenbasierten Analyse und in der Informationsmodellierung selten erwähnt werden, aber im Zusammenhang mit der Integritätssicherung großer Objektbestände bedeutsam sind. Hierzu zählen vor allem Assoziationen, auf denen Anstoßtechniken, wie der Event-Trigger-Mechanismus und die Constraint-Propagierung, beruhen. Diesen Aspekt werden wir im nächsten Kapitel untersuchen.

Typkonzept

Abschließend zu den Abstraktionsprinzipien der Objektorientierung skizzieren wir die Verwandtschaft zu einer Software-Abstraktion, die heute den höchsten Rang einnimmt in der Programmiermethodik: dem „Abstrakten Datentyp“ (ADT).[10] Wie wir bereits in Kapitel 4 gesehen haben, verläuft die Entwicklung der Softwaremethoden entlang einer Linie wachsender Abstraktion. Jeder Aspekt in einer „maschinenfernen“ Programmiersprache, wie Pascal oder Eiffel, ist abstrakt: Variable und ihre Werte, Funktionen, Prozeduren und Programme sind allesamt Abstraktionen ihrer tatsächlichen Darstellungen im Medium der Implementierung. Letztlich abstrahieren diese Softwarebegriffe von den Bits und Bytes der Hardware und den in ihr implementierten Maschinenbefehlen. Nach DIJKSTRA können wir Software als eine Schichtung abstrakter Maschinen auffassen:

> „[software] ... can be regarded as structured in layers. We conceive an ordered sequence of machines: $A[0], A[1] \ldots A[n]$, where $A[0]$ is the given hardware machine ... the software of layer i will use some of the resources of machine $A[i]$ to provide resources for $A[i+1]$: in machine $A[i+1]$ and higher these used resources of machine $A[i]$ must be regarded as no longer there!“ (zitiert in [Koepke, 1985, S. 21])

[10] Scott DANFORTH und Chris TOMLINSON haben die Parallelen und Unterschiede zwischen verschiedenen Typtheorien und der objektorientierten Programmierung in einem Übersichtsaufsatz herausgearbeitet [Danforth & Tomlinson, 1988].

Das Konzept des abstrakten Datentyps geht noch einen Abstraktionsschritt weiter: es vereint abstrakte Daten (elementare Datentypen und daraus zusammengesetzte Datenstrukturen) mit den auf sie wirkenden abstrakten Operationen (Funktionen und Prozeduren). Barbara LISKOV und Stephen ZILLES gaben 1974 die erste formale Darstellung in ihrem Aufsatz *Programming with Abstract Data Types* [Liskov & Zilles, 1974]. Danach geht die Entwicklung abstrakter Datentypen auf drei Strömungen zurück:

1. Datentypen, die erst der Benutzer definiert: *records* in Cobol (1961)
2. Klassenkonzept in Simula: NYGAARD und DAHL (1967)
3. Information hiding und Datenkapselung: PARNAS (1971)

```
structure LBK = (mode Δ, nat max: max > 0, based on boolean, based on nat):
        mode        lbs Δ;
        functions   () lbs Δ empty,
                    (lbs Δ) boolean isempty, isfull,
                    (lbs Δ) Δ top,
                    (lbs Δ) lbs Δ pop,
                    (lbs Δ, Δ) lbs Δ push,
                    (lbs Δ) nat length;
        laws        LEER:   (x = empty) ⇔ isempty(x),
                    VOLL:   (length(x) = max) ⇔ isfull(x),
                    LIFO:   not isfull(x) ⇒ pop(push(x, d)) = x,
                    KON:    not isempty(x) ⇒ push(pop(x), top(x)) = x,
                    OBEN:   not isfull(x) ⇒ top(push(x, d)) = d,
                    ANZ:    not isfull(x) ⇒ length(push(x, d)) = length(x) + 1,
                    ANZ0:   length(empty) = 0

end structure
```

Bild 10.9: Klassisches Beispiel eines abstrakten Datentyps

Besonders das Prinzip des Information hiding (siehe Abschnitt 2.3.3) hat das ADT-Konzept beeinflußt: Datenstrukturen sollen die Details ihrer Implementierung und ihrer Manipulation in einem gemeinsamen Modul verbergen. Der Anwender *abstrahiert* von der Realisierung eines Datentyps und den zugehörigen Operationen. Bild 10.9 zeigt den Aufbau eines abstrakten Datentyps, entnommen dem Informatik-Duden: „längenbeschränkter Keller", LBK [Claus & Schwill, 1993]. Durch mode und functions wird die *Signatur* vorgestellt,

das heißt die Liste der Bezeichner für Wertebereiche und Operationen. Auf die Signatur folgen die Axiome oder Gesetze: laws.[11] Das Beispiel des Kellerspeichers (*stack*) als ADT wurde erstmals von LISKOV und ZILLES als Erklärungsvehikel eingeführt und fehlt seitdem in keinem Nachschlagewerk der Informatik.

Zurück zur Objektorientierung: Objekte als Exemplare (englisch: *instances*) ihrer Klassen gehen auf Simula zurück. Objekte kapseln einen Zustand zusammen mit den Operationen, die eine Zustandsänderung bewirken können. Objekte bieten also Datenabstraktion durch ihre Schnittstelle. Da die Klasse die Objektschnittstelle festlegt, kann sie folglich als abstrakter Datentyp aufgefaßt werden. In der Tat ist die Ähnlichkeit so groß, daß vielerorts objektorientiertes Programmieren im Sinne von LISKOV und ZILLES verstanden wird: „Programming with Abstract Data Types". Klassen implementieren aber nicht nur das Konzept des abstrakten Datentyps, sie überwinden auch zugleich einen negativen Aspekt der Kapselung: Sie vergeben Nutzungsprivilegien an ihre Anwender und erfüllen so eine zentrale Forderung des ökonomischen Software-Entwurfs: die Wiederverwendung oder das Teilen der Ressourcen. *Vererbung* (fehlt im ADT-Konzept!), *Exemplarbildung*, *Delegation* und *Prototyp* stehen hier für das Konzept-Repertoire an objektorientierten Teilungsmechanismen, auf die wir nun näher eingehen.

10.2.3 Teilen

Mechanismen, die es den Objekten erlauben, strukturelle Eigenschaften untereinander zu teilen, gehören zu den nützlichsten und am meisten debattierten Merkmalen objektorientierter Sprachen. Die Debatte setzte ein, als die Schöpfer und Verfechter neuer objektbasierter Sprachen scheinbar konträre Ansätze für eine Ressourcenteilung vertraten. Zwei Lager waren bald mit Schlagwörtern ausgemacht: *Klassen* und *Vererbung* auf der einen Seite, *Prototypen* und *Delegation* auf der anderen. Stellvertretend für diese Begriffe stehen Simula [Dahl & Nygaard, 1966] und Smalltalk [Goldberg, 1984] bzw. Actors [Agha, 1987] und Self [Ungar & Smith, 1987]. Das Trennende der beiden Lager liegt im Zeitpunkt der Wiederverwendung: Da ist zum einen die *erwartete Teilhabe* an Struktur und Verhalten. Hier dienen vor allem Klassen als Mechanismus, um die Teilhabe der Objekte an den gemeinsamen Klasseneigenschaften zu codieren. Zum anderen gibt es den Fall der *unerwarteten Teilhabe*, der folglich auch nicht im voraus programmiert werden kann und der von den traditionellen Vererbungsmechanismen à la Smalltalk unzureichend unterstützt wird.

Die traditionelle Vererbungslinie Oberklasse-Unterklasse-Objekt verlangt die *textuelle*, das heißt *statische* Unterscheidung zwischen „Erblasser" und „Erben": also zwischen den Klassen, die das Erbgut bilden, und den Objekten, die erben sollen. In prototypischen Ent-

[11] Also werden auch im ADT-Konzept Struktur, Verhalten und Constraints gekapselt.

wurfssituationen, wo der Dialog zwischen Anwender und Entwerfer noch andauert, die Anforderungen an das zu entwerfende System für eine Spezifikation noch vage und im Fluß sind, wird eine größere Flexibilität des Teilens gefordert: Die *dynamische* Vererbung (David LIEBERMAN hat sie unter der Bezeichnung „Delegation“ in die Diskussion eingebracht [Lieberman, 1986]) erlaubt die unerwartete Teilhabe an den dynamisch vorhandenen Ressourcen. Neue Objekte können zur *Laufzeit* des Programms das Verhalten existierender Objekte nutzen, ohne daß zur Übersetzungszeit die exakten Pfade der Vererbung oder der Delegation bekannt sein müssen.

Auf den ersten internationalen ACM-Konferenzen über *Object-Oriented Programming, Systems, Languages and Applications* (OOPSLA), 1986 und 1987, wurde die Debatte weitergeführt und endete 1988 mit einem Konsens: *The Treaty of Orlando* [Stein *et al.*, 1989]. Lynn STEIN, Henry LIEBERMAN und David UNGAR kommen dort zu der Überzeugung, daß der Entwurfsraum objektorientierter Sprachen nicht zweigeteilt ist. Die in der Debatte über statische und dynamische Vererbung aufgekommene Zweiteilung wurde überwunden: Die „Unterzeichner des Vertrags von Orlando“ erkennen zwei grundlegende Mechanismen des Teilens, die den Rahmen für einen Sprachenvergleich bilden: *Schablonen* und *Empathie*. Sie sind in ihren Definitionen unabhängig und in der Kombination ein Kriterium für Objektorientierung. Objektorientierte Sprachen unterstützen *beide* Mechanismen, wenn auch im unterschiedlichen Maße. Jeder Teilungsmechanismus für sich hat mehrere Freiheitsgrade, die wir skizzieren wollen.

Schablonen

Schablonen (im Englischen heißen sie *templates*) erzeugen Objekte gleicher Form und bieten somit Garantien für die Objekt*ähnlichkeit*. Eine Schablone enthält alle Festlegungen, die für die Definition neuer Objekte notwendig sind, also Deklarationen der Datenstrukturen (Objektvariablen), Zugriffsoperationen (Objektoperationen) und Verweise auf Elternklassen. Können keine weiteren Eigenschaften hinzugefügt oder alte entfernt werden, heißt die Schablone starr, andernfalls flexibel. Es gibt Sprachen, die keine Schablonen unterstützen, wie Actors und DELEGATION. In anderen Sprachen wie Self ist die Schablone ein Laufzeitobjekt. Gewöhnlich aber sind Schablonen in *Generator*objekten eingebettet, sie heißen *Klassen*. Dinge, die mit Klassenschablonen ausgestanzt werden, heißen Objekte oder Exemplare. Eine Klasse garantiert die Einheitlichkeit und Unabhängigkeit ihrer „schablonierten“ Objekte.

Empathie

Mit Empathie („Sich-Hineinversetzen-Können", „Einfühlungsvermögen") kann ein Objekt so agieren, als wäre es ein anderes. Empathie schafft somit die Möglichkeit, Struktur- und Verhaltensmuster zu *teilen*. Die Empathie unterliegt beiden Konzepten: sowohl der Vererbung als auch der Delegation. Sie kann wie folgt definiert werden: „Man sagt, Objekt A versetzt sich in Objekt B bezüglich der Nachricht M, wenn A über kein eigenes Protokoll verfügt, um auf M zu antworten. A antwortet auf M, als ob es sich das Antwortprotokoll von B ausliehe." Vererbung und Delegation, beides Formen der Empathie, unterscheiden sich im „Wann und Wie" ihrer Festlegung. Empathie kann explizit sein: „Execute `thisObject:thisRoutine` in my environment" oder implizit: „Anything I can't handle locally, look up in `myParent` (and execute in my environment) with `SELF`[12] `= me`" [Stein *et al.*, 1989, S. 37]. Empathie kann dynamisch zur Laufzeit oder statisch zum Zeitpunkt der Übersetzung festgelegt werden, für ein einzelnes Objekt oder für eine Gruppe von Objekten. Die Vielfalt der Optionen — Variationen der Paare explizit/implizit, dynamisch/statisch und Einzelobjekt/Objektgruppe — haben die heutige Fülle objektorientierter Sprachen hervorgebracht.

Bei der Vererbung im Stil von Simula und Smalltalk liegt der Verteilungspfad im voraus fest. Die Verteilung bezieht sich stets auf Gruppen von Objekten. Der Smalltalk-Stil der Vererbung trennt teibare Komponenten von unteilbaren. Was teilbar ist, bestimmt global für eine Gruppe von Objekten die Klassendefinition (Definition der Klassenvariablen und Klassenoperationen). Was unteilbar ist, bestimmt individuell die Objektdefinition (Definition der Objektvariablen). Dagegen ist Vererbung im Stil der Sprachen Actors und DELEGATION nicht an statisch vorgegebenen Verteilungspfaden gebunden. Was und wie teilbar ist, entscheidet sich dynamisch auf der Objektebene: Objekte haben keine Eltern (Klassen). Objekte werden erzeugt, indem ein vorhandenes Objekt — der Prototyp — „geklont" und um individuelle Eigenschaften erweitert wird.

Abschließend zur Diskussion der objektorientierten Teilungsmechanismen zeigt die Tabelle 10.2 eine entsprechende Spracheneinteilung mit den jeweiligen Freiheitsgraden. Danach sind Actors und DELEGATION dynamische Empathie-Sprachen und offerieren die größte Entscheidungsfreiheit, was das Teilen betrifft. Die Sprache Self fügt das Schablonenkonzept in Form von *templates* hinzu, bleibt aber flexibel, da sie hierfür keine starren Klassen voraussetzt. Die traditionellen objektorientierten Sprachen, Simula und Smalltalk, sind in ihren Empathie- und Schablonenmechanismen statisch und starr. HYBRID schließlich vereint die Synergie beider Sprachlager: uneingeschränkte Flexibilität der Schablonen — Klassen oder Prototypen — und dynamische Vererbung durch Delegation.

[12] Pronomen wie „self", „me" und „this" werden in objektorientierten Sprachen als formale Parameter einer Operation benutzt. Sie beziehen sich auf das Objekt, auf dem die Operation ausgeführt wird.

Sprache	*Bestimmung der Empathie*			*Schablonenmechanismus*	
	Wann	*Wie*	*Wer*	*Was*	*Wie*
Actors	Laufzeit	explizit	Objekt	fehlt	
DELEGATION	Laufzeit	beides	Objekt	fehlt	
Self	Laufzeit	implizit	Objekt	templates	flexibel
Simula	Übersetzung	implizit	Gruppe	Klassen	starr
Smalltalk	Objekterzeugung	implizit	Gruppe	Klassen	starr
HYBRID	Laufzeit	beides	beides	beliebig	flexibel

Tabelle 10.2: Schablonen und Empathie als Sprachkriterien

10.2.4 VLSI-Datenmodell nach Batory & Kim

Den Strukturaspekt im Objektmodell wollen wir an einem VLSI-Beispiel zusammenfassen: D. S. BATORY und W. KIM haben ein VLSI-Datenmodell vorgeschlagen, das den meisten VLSI/CAD-Forderungen[13] nach Abstraktion, Generalisierung, Komposition und Vererbung genügt [Batory & Kim, 1985]. Johannes BRAUER stellt dieses Modell in den Mittelpunkt seiner Untersuchung über die „Datenhaltung in VLSI-Entwurfssystemen“ [Brauer, 1990]. Wir übernehmen sein Beispiel zur Modellierung der EDIF-Netzliste (siehe [EDIF-Manual, 1990]) und erläutern daran die VLSI-spezifische Strukturierung. Bild 10.10 zeigt die Entity-Relationship-Notation nach Peter CHEN [Chen, 1976]: Rechtecke stehen für *Objekttypen*, Rauten für *Beziehungstypen* und Ovale für Attribute, wobei das Zeichen # auf einen eindeutigen Bezeichner verweist [Brauer, 1990, S. 126].[14]

- Abstraktion

  BATORY und KIM haben ihr Objektmodell an die Datenstrukturen im Austauschformat EDIF angelehnt: Analog zur EDIF-Zweiteilung einer *Zellsicht* (*view*) wird auch hier die Objektbeschreibung in *Schnittstelle* und *Implementierung* unterschieden. Auf der semantisch höheren Beschreibungsebene abstrahiert die Schnittstelle eines Objekts von der tatsächlichen Implementierung: sie ist „das Erscheinungsbild des Objekts nach außen“ [Brauer, 1990, S. 120]. Das Teilschema (a) in Bild 10.10 beschreibt die Schnittstelle einer EDIF-Zelle: Zellen sind charakterisiert durch ihren Typ und identifizierbar durch ihren Bezeichner c_t#. Sie stehen mit n Zellanschlüssen

[13] Der Abschnitt8.1 beschreibt die Forderungen ausführlich und stellt sie einem allgemeinen Objektmodell gegenüber.

[14] Die ER-Notation für das BATORY-KIM-Modell hat einen pragmatischen Grund: Ziel ist die Realisierung des Objektmodells mit Hilfe eines relationalen Datenbanksystems — ein Ansatz, der in den 80er Jahren intensiv verfolgt wurde und im DASSY-Projekt seinen bisherigen Abschluß gefunden hat [Quibeldey-Cirkel, 1993a].

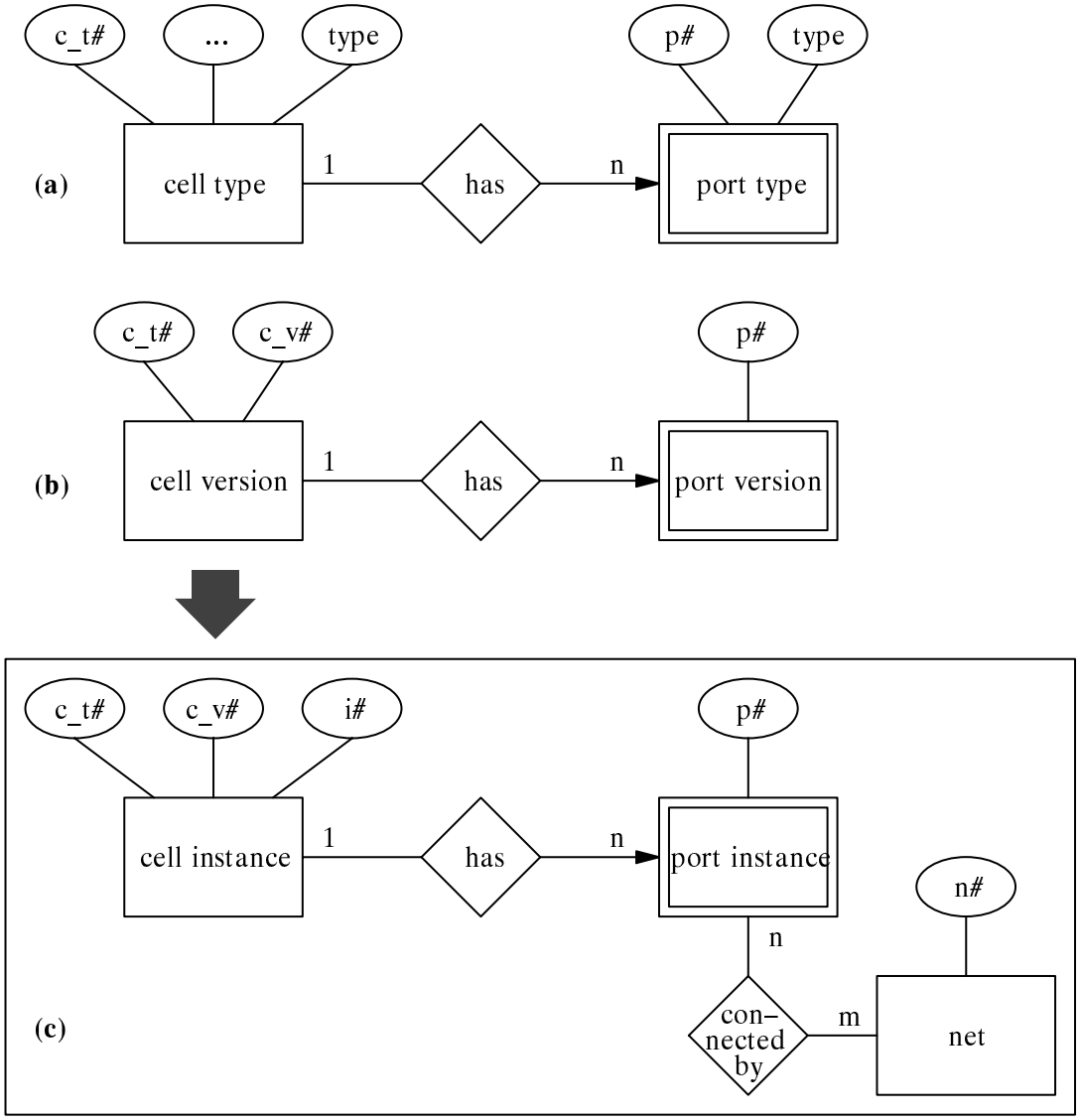


Bild 10.10: BATORY-KIM-Objektmodell: Schema einer EDIF-Netzliste

(*ports*) in Beziehung. Der Doppelrahmen kennzeichnet sogenannte „schwache“ Objekttypen, die existentiell von anderen Objekttypen abhängen. Da für die Benutzung einer Zelle innerhalb der Netzliste einer anderen Zelle nur die Information über die Zellanschlüsse benötigt wird, genügt die Typisierung der Zellanschlüsse für die vollständige Schnittstellenbeschreibung.

- Generalisierung

  So wie eine EDIF-Zelle mehrere Sichten haben kann, so kann im BATORY-KIM-Modell dieselbe Schnittstelle mehrere Implementierungen haben, *Versionen* genannt. Die Unterscheidung wird durch die Zusätze *type* und *version* bei gleichem Objektnamen notiert. Das Teilschema (a) ist somit die für alle Versionen gemeinsame Schnittstellenbeschreibung. Die Verfasser nennen ihr Versionskonzept *version generalization*. Das Teilschema (b) beschreibt die versionsspezifischen Daten der Schnittstelle.

- Komposition

  In beiden Fällen wird das Konzept der *molecular aggregation* angewandt: Objekte verschiedenen Typs werden zu einem Objekttyp auf höherer Ebene zusammengefaßt. Die Aggregation, das ist der datenbanktechnische Terminus für die Komposition [Smith & Smith, 1977], wird durch die Umrandung des Teilschemas (c) angedeutet. Der Inhalt des Rahmens beschreibt die Implementierung der im Teilschema (b) beschriebenen Schnittstelle. Die Aussage „Die Schnittstelle abstrahiert von der Implementierung“ wird durch den Pfeil symbolisiert. Die Objekt- und Beziehungstypen des Teilschemas (c) bilden zusammen das Aggregat „Netzliste“. Hierbei wird zugleich das VLSI-typische Prinzip der Objektausprägung (*instantiation* = Exemplarbildung) genutzt: Bereits definierte Zellen werden für die Definition neuer Zellen wiederverwandt, wobei die wiederverwandten Zellen nur referenziert, nicht aber kopiert werden. Das vermeidet redundante Objektbeschreibungen und erhöht die Zugriffseffizienz der Datenhaltung. Exemplare sind am Namenszusatz *instance* kenntlich; sie haben natürlich auch einen eigenen Bezeichner: i#.

- Vererbung

  Die Angabe des Typs c_t# und der Version c_v# des gewünschten Objekts impliziert das *Teilen* aller Attribute des Objekttyps und der Objektversion: das Exemplar *erbt* deren Struktureigenschaften. In diesem Zusammenhang greift ein weiteres Modellierungskonzept von BATORY und KIM, das sie *parameterized version* nennen. Die Versionierung erfährt durch dieses Konzept eine methodische Aufweitung, die in der Objektorientierung als *generische* oder *abstrakte* Klassifikation bezeichnet wird (siehe Seite 211). Einzelne Parameter einer Version werden erst bei einer Ausprägung der Version gebunden. Da die Parametrisierung hierarchisch fortgesetzt werden kann (aus parametrisierten Versionen werden neue parametrisierte Versionen erzeugt) unterstützt dieses Konzept zugleich die *Top-down*-Vorgehensweise in der Entwurfsspezifikation: Die Dekomposition eines komplexen Entwurfsobjekts in Teilobjekte und deren Beziehungen kann zunächst *unvollständig* beschrieben werden: Die Teilobjekte müssen nur typisiert sein, das heißt die Beschreibung beschränkt sich auf ihre Schnittstellen. Die Implementierung dieser Schnittstellen kann zunächst verschoben oder delegiert werden.

Bei aller strukturellen Übereinstimmung mit dem Ideal unseres VLSI-Weltmodells (siehe Tabelle 8.1 auf der Seite 163) bleibt ein Manko: Zum einen lehnt sich das Versionskonzept von BATORY und KIM starr an das Sichtenkonzept von EDIF. Dessen Semantik ist aber unzureichend definiert. BRAUER weist bereits daraufhin: In der Praxis der VLSI-Datenmodellierung genügt es nicht, nur die Implementierung einer Zelle zu versionieren und ihre Schnittstelle einheitlich zu halten. Verschiedene Repräsentationen desselben Entwurfsobjekts unterscheiden sich unter Umständen auch in der Schnittstellenbeschreibung, beispielsweise wenn sichtenabhängige Attribute hinzukommen.[15] Zum anderen fehlt dem BATORY-KIM-Modell jede Option, die *operative Semantik* der Objekt- und Beziehungstypen zu spezifizieren (wie in Abschnitt 8.4 für das VLSI-Weltmodell gefordert): Vorgesehen sind nur elementare Operationen der Datenhaltung, wie das Erzeugen, Verändern und Löschen der Entwurfsobjekte. Dieses Manko charakterisiert das Datenmodell als lediglich *strukturell* objektorientiert; *vollständig* objektorientierte Datenmodelle dagegen schließen den Verhaltensaspekt mit ein (die Einteilung in strukturell, verhaltensmäßig und vollständig objektorientierte Datenmodelle geht auf Klaus DITTRICH zurück [Dittrich, 1989]).

10.3 Verhalten

Im Gegensatz zur bloßen strukturellen Zweiteilung nach BATORY und KIM oder nach dem Sichtenkonzept von EDIF folgt die *verhaltensmäßige* Objektmodellierung dem ADT-Konzept: Es wird von den implementierten Datenstrukturen abstrahiert und die Semantik des Entwurfsobjekts allein durch die Semantik der Schnittstellenoperationen beschrieben. In diesem Punkt unterscheiden sich die objektorientierten Datenmodellen deutlich von den *semantischen* (siehe zum Beispiel [King, 1989]). Auch wird das PARNAS-Postulat des *Information hiding* erfüllt und die Daten entsprechend gekapselt (siehe Abschnitt 2.3.3). Entwurfsobjekte in *vollständig objektorientierten* Datenhaltungen zeigen individuelle *Actio-Reactio*-Verhaltensmuster, also Muster der Art und Weise, wie ein Objekt mit seiner Umgebung interagiert, das heißt, wie es Eingaben auf Ausgaben abbildet.[16] Die Empathie der Objekte — ihr Vermögen, Struktur und Verhalten zu teilen — erlaubt die flexible *Wiederverwendung* als Entwurfsbausteine.

[15] Die Logiksimulation mit *Timing*-Analyse ist ein Beispiel für Sichtenabhängigkeit: Durch die Rückführung (*back annotation*) der Signallaufzeiten, berechnet aus dem geometrischen Layout, werden die Sichten *Layout* und *Logik* vermischt, was folglich die Schnittstellenbeschreibung der Logikzellen verändert.

[16] In objektorientierten Systemen bestimmt der Nachrichtenaustausch (*Message passing*) die *Actio-Reactio*-Verhaltensmuster: Die Nachrichten sind die aktivierenden, die Objektoperationen die reagierenden Muster.

10.3.1 Funktions- versus Klassenbibliothek

Seit es programmgesteuerte Rechenanlagen gibt, sind Bausteinbibliotheken in der Diskussion. Schon 1949 plädierten WILKES, WHEELER und GILL für das Konzept des Unterprogramms und der mathematischen Bibiliothek [Endres, 1988]. Aber erst zwanzig Jahre später wurde das Bausteinprinzip durch MCILROY zur wirtschaftlichen Prämisse einer Software-Industrie erhoben. Form und Inhalt der Bausteine waren dabei lange Zeit auf die funktionale Abstraktion ausgerichtet. Der Ausgang y einer funktionalen Komponente f hängt ausschließlich vom Eingang x ab: $y = f(x)$. Prozeduren und Unterprogramme galten als *das* pragmatische Mittel, um Programmcode wiederzuverwenden, wobei in den Anfängen meistens eine *cut-paste-edit*-Strategie praktiziert wurde. Die systematische Einbindung von Bibliotheksfunktionen als *Black boxes* war eher die Ausnahme.

Die fehlende „Umgebungsinformation" einer Funktionskomponente bedeutet eine vermeidbare Einschränkung im Bestreben um Software-Wiederverwendung: Nur Funktionen, deren Werte von der Einsatzumgebung unabhängig sind, können ohne Anpassung wiederverwandt werden (das erklärt auch die weite Verbreitung mathematischer Bibliotheken). Um Module mit eigener parametrisierbarer Umgebung zu schaffen (aus Variablen und Konstanten), bedarf es höherer Abstraktionen. Die Lösung wird heute im Konzept des abstrakten Datentyps gesehen, genauer: in dessen implementierter Form als *Klasse*.

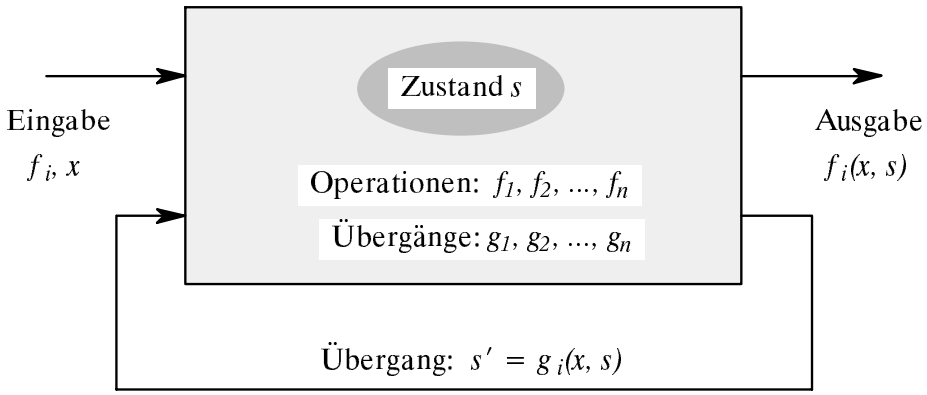


Bild 10.11: Objektbasierte Datenabstraktion: MEALY-Automat

Angelehnt an den zentralen Gegenstand der Automatisierungstechnik, stellt der heutige Evolutionsstand wiederverwendbarer Softwarekomponenten, die *objektbasierte Datenabstraktion*, einen MEALY-Automaten dar [Seifart, 1988]. Beispiele hierfür: *packages* in Ada [Booch, 1983], *modules* in Modula [Wirth, 1985] und *classes* in Smalltalk [Goldberg, 1984]. Bild 10.11 zeigt die MEALY-Eigenschaft der Datenabstraktion.

Der interne Zustand s eines MEALY-Automaten fungiert als „Gedächtnis“ für die Wirkungen vergangener Operationen. Der Ausgang der Komponente hängt nicht mehr allein vom augenblicklichen Eingangswert ab, sondern auch von der Zustands-Chronik: $y = f_i(x, s)$. Funktionale Abstraktion und Datenabstraktion bestimmen den gegenwärtigen Paradigmenkonflikt in der Softwaretechnik: Sollen Softwaremodule funktions- oder objektorientiert entwickelt und für eine spätere Wiederverwendung konfektioniert werden? Während die funktionale Abstraktion die Wiederverwendung der Funktionen anstrebt, zielt die Datenabstraktion auf die Wiederverwendung der Datenobjekte. Oder mit dem Vokabular des Marktes [Wegner, 1989]: Im Falle der funktionalen Abstraktion stellen die Programme die primären Produktionsmittel dar und die Daten die Verbrauchsgüter. Die Verbrauchsgüter werden vom Verbraucher geliefert und an diesen wieder ausgegeben. Im Falle der Datenabstraktion stellen die Datenobjekte die primär wiederverwendbare Ressource dar. Hier sind die Funktionen die Verbraucher. Sie weisen eine kürzere „Lebensdauer“ auf als die Objekte, auf denen sie operieren.

Für datenintensive Anwendungen birgt die Datenabstraktion das größere Potential der Wiederverwendung. Es stellt sich die Frage, welchen Kriterien die Datenabstraktion genügen sollte, um den größtmöglichen Nutzen aus ihr zu ziehen. Die Frage zielt auf die *Modularität*, auf das Maß zwischen dem inneren Zusammenhalt eines Moduls und seinen Bindungen nach außen, also zwischen Kohäsion und Kopplung. Herbert SIMON gab die allgemeingültige Antwort: Minimale Kopplung und maximale Kohäsion sind die Kriterien der „beinahen Zerlegbarkeit“ aller komplexen Gebilde in Natur und Technik (siehe Abschnitt 2.2.1). Auch softwaretechnische Bausteine müssen natürlich diesen Forderungen genügen. Mit der objektbasierten Datenkapselung erfüllt die Softwaretechnik diese Forderungen mit aller Konsequenz: Datenstrukturen und Algorithmen können ohne Änderung der Modulschnittstelle variiert werden. Zusammengehörige Prozeduren sind gebündelt. Prozeduren werden ohne Kenntnis der konkreten Implementierung aufgerufen. Überdies erlaubt es der konzeptionelle „Bypass“ der Kapselung — die Vererbung von Datenstrukturen und Operationen —, die Gemeinsamkeiten auch *zwischen* den Modulen rationell auszuschöpfen.

Bausteinbibliotheken in der Softwaretechnik müssen folglich *Klassen*bibliotheken sein, will man die wiederverwendbaren Ressourcen möglichst vieler *datenintensiver* Anwendungen erschließen. Auf die wichtigsten organisatorischen und technischen Aspekte zum Aufbau und zur Wartung von Klassenbibliotheken geht MEYER in [Meyer, 1990] ein. *Large-scale reuse* durch ein Klassenmanagement für *software communities* ist Gegenstand eines Übersichtsartikels von GIBBS et al. [Gibbs *et al.*, 1990]. Objektorientiertes *Megaprogramming* zielt in die gleiche Richtung [Johnson *et al.*, 1992]. Übereinstimmend wird eine Standardisierung der Klassenbibliotheken als wichtigste Voraussetzung für den Erfolg dieses Konzepts gewertet.

10.3.2 Operator-Operanden-Modell versus Smalltalk-Modell

Vor rund 50 Jahren setzte die *prozedurale Abstraktion* den Anfang der wissenschaftlichen Auseinandersetzung mit der Rechnerprogrammierung. Bis in die Gegenwart hinein hat sie die Informatikmethoden geprägt (siehe Abschnitt 4.3.1). Brad COX spannt den zeitlichen Bogen des Operator-Operanden-Modells wie folgt:

> „We are taught very early that computers do *operations* on *operands*, and this computational model is preserved through everything we learn subsequently. Once we know that larger operators can be built out of stored sequences of operators, our progress as programmers becomes a matter of increasing the number of different operators we know — text editors, command languages, machine languages, linkers, debuggers, high-level languages, subroutine libraries, application packages, and so forth. We think of the computer as two disjoint compartments, one containing operators and the other operands, and we express our intentions by selecting what operators are to be applied and to what operands in what order." [Cox, 1984, S. 152]

Das prozedurale Modell findet seine Anwendung auf allen Ebenen: Auf der Maschinenebene haben wir die Trennung zwischen Prozessorbefehlen und binären Daten, auf der Sprachebene zwischen Ausdrücken und Variablen, auf der Bibliotheksebene zwischen Funktionen und Argumenten und auf der Ebene des Betriebssystems zwischen Programmen und Dateien. Die Implementierung ist jeweils eine andere, aber die Konzepte sind gleich: Operatoren sind aktiv und manipulieren in vorbestimmter Weise bestimmte Operanden. Operanden sind passiv und ändern ihren Zustand nur unter dem Einfluß bestimmter Operatoren. Es ist die *Umgebung*, die bestimmt, welche Operatoren auf welche Operanden in welcher Reihenfolge angewandt werden. Drei Aufgaben leiten sich daraus für die Umgebung ab: (a) Koordinierung der Ereignisfolge, (b) Verschiebungen im Adreßraum der beteiligten Komponenten und (c) Typ-Prüfung der Operanden. Letztes ist bezeichnend für das prozedurale Modell: Operatoren sind *monomorph*, das heißt, ihre korrekte Anwendung ist streng mit dem Datentyp des Operanden gekoppelt und läßt keine Typ-Variationen zu (andernfalls wären sie *polymorph*).[17] Texteditoren zum Beispiel können nur Textdateien lesen und scheitern in der Regel an binären Eingaben. Bild 10.12 symbolisiert die enge Zuordnung zwischen Operatoren (O) und Operanden (D wie Daten) und ihre lokale Zufälligkeit.

Die Konzeption eines *Remote Procedure Call* (RPC) lehnt sich gleichfalls an das Operator-Operanden-Modell, erweitert dieses aber über die Prozeß- und Hardwaregrenzen hinweg: Wird eine Prozedur auf dem Zielrechner aufgerufen, geht die Kontrolle der aufrufenden auf die aufgerufene Umgebung über. Die Parameter werden über das Netz in die Zielumgebung weitergereicht. Dort wird die Prozedur ausgeführt und die Kontrolle mit den Ergebnisparameter in die Quellumgebung zurückgegeben. Auch hier muß die Umgebung

[17] Das Thema „Typen, Datenabstraktion und Polymorphie" wird von Luca CARDELLI und Peter WEGNER in [Cardelli & Wegner, 1985] erschöpfend behandelt.

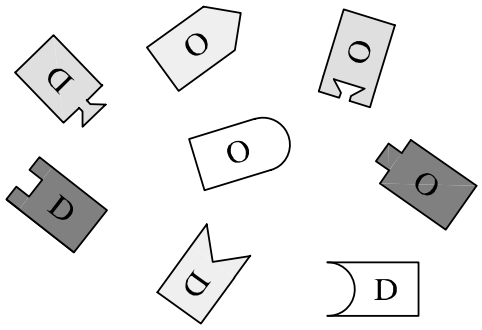


Bild 10.12: Das prozedurale Modell: seine Komponenten und deren Verteilung

sicherstellen, daß der Prozeduraufruf (*call*) typmäßig an die richtige Operation gebunden wird. Dazu muß sie die Zielmaschine identifizieren und gewährleisten, daß die zu übertragenden Parameter an der aufrufenden Stelle die gleiche Semantik haben wie an der aufgerufenen, da wegen der räumlich getrennten Arbeitsspeicher eine Typ-Prüfung durch den Übersetzer ausscheidet. Weiterhin muß die aktive Umgebung die Datenrepräsentation der Quell- oder der Zielmaschine in die Repräsentation der Protokollebene konvertieren. Die Maschinenschnittstellen konsistent zu halten, ist entsprechend aufwendig [Frank, 1992].

Dem Operator-Operanden-Modell stellt COX das objektorientierte Smalltalk-Modell gegenüber. Die Umgebung dieses Modells ist lediglich für die Koordinierung der Ereignisse verantwortlich. Das Objekt, verstanden als „some private memory and the set of operations provided by its class“ [Cox, 1984, S. 153], trägt die Verantwortung für die Ausführung der geeigneten Operationen. Seine Klassendefinition lokalisiert die Typ-Vereinbarungen. Polymorphe Nachrichten werden durch die Vererbungshierarchie an die entprechende Implementierung der zugehörigen Operation dynamisch gebunden.

Ein Beispiel soll den Unterschied klären zwischen dem prozeduralen und dem objektorientierten Modell: Auszuführen sei die Addition zweier Summanden: $A + B$. Im Operator-Operanden-Modell werden der +-Operator und die Operanden A und B so behandelt, als seien sie unabhängig voneinander. Die Umgebung, in der die Operation $A + B$ ausgeführt werden soll, Modul genannt, wird durch die Implementierung bestimmt. In der Umgebung sind die Datentypen von A und B deklariert. Die Typ-Deklaration wiederum impliziert den Typ der Addition: Integer-Addition für Integer-Operanden, Matrizen-Addition für Matrizen-Operanden oder eine Verkettung bei Zeichen-Operanden. Im Gegensatz dazu steht das Smalltalk-Modell: Die Nachricht „ addiere A“, an das Objekt B gerichtet, kann zum Aufruf verschiedener Operationen führen, in Smalltalk *methods* genannt.[18] Nach-

[18]Leider wurde der Smalltalk-Begriff *method* unreflektiert als Anglizismus ins Deutsche übernommen, obwohl „Methode“ etwas gänzlich anderes meint als „Operation“: siehe Fußnote 5 auf Seite 6.

richten in Smalltalk sind Prozeduraufrufe, die erst zur Laufzeit des Programms gebunden werden. Die dynamische (späte) Bindung wird durch die Klassenzugehörigkeit des Objekts A bestimmt. In der Umgebung, in der das Paar „Nachricht-Empfänger“ auftritt, muß keine Typ-Deklaration von A und B vorliegen.

Der Unterschied liegt im Grad der Durchdringung eines Programms mit zahlreichen und redundanten Typ-Deklarationen. Jedes Mal, wenn der +-Operator angewandt wird, müssen die Typen der Operanden zuvor vereinbart werden. Im Smalltalk-Modell dagegen sind die Abhängigkeiten zwischen Operatoren und Operanden in den Objekten verborgen, und breiten sich nicht im System aus. Mit anderen Worten: die Integration neuer Objekte in ein bestehendes System ist prinzipiell ohne Code-Änderung möglich. Weiterhin öffnet der Austausch von Objekten anstelle von getrennten Verweisen auf Daten und Funktionen die Perspektive auf eine komfortable und mit Blick auf Sicherheit und Wartbarkeit effektive Kommunikation. Die Kapselung der Abhängigkeiten zwischen Operatoren und Operanden gewährt eine größere Unabhängigkeit des Systems von den Objekten, die es enthält. Die Vielgestaltigkeit der Operatoren, die Polymorphie, gewährleistet somit „schlanke“ und redundanzarme Programme. Auch wenn der Nachrichtenaustausch zur Laufzeit auf einen mittelbaren Prozeduraufruf reduziert wird, hebt diese Metapher die Objekt-Interaktion auf eine kommunikative Ebene, auf der komplexe Protokolle gefahren werden können. Die Abstraktionsprinzipien und Teilungsmechanismen der vorigen Abschnitte können hier konsequent angewandt werden.

10.3.3 Client-Server-Vertrag

Ausgehend von der BROOKSschen Erkenntnis, daß die Produktivität einer Entwicklungsabteilung im wesentlichen ein Problem der Koordination ist [Brooks, 1975], fordert COX ein objektorientiertes Kommunikationsmodell, das die Bedürfnisse koordinierender Werkzeuge befriedigt: Zum einen gilt es, Definition und Austausch beliebiger Objekte zu ermöglichen, zum anderen sollen diese benutzerdefinierten Objekte auch über den eigenen Adreßraum hinweg zwischen verschiedenen organisatorischen Einheiten bewegt werden können. Die Technik, die das vermag, geht auf den Interaktionsmechanismus in objektorientierten Sprachen zurück: Nachrichtenaustausch in Objektwelten.[19]

Die anthropomorphe Metapher „Nachrichtenaustausch“ ist in zwei Punkten irreführend und soll im folgenden durch die Client-Server-Metapher ersetzt werden. Erstens assoziiert die Kommunikation durch Nachrichten einen nebenläufigen Prozeß, was zwar wünschenswert ist, auf VON-NEUMANN-Rechnern aber nur mit zusätzlichem Aufwand an Soft- und

[19] Dennis TSICHRITZIS, zitiert in [Frank, 1992, S. 34] versteht unter einer Objektwelt (*object world*) die Vision, daß anstelle von Daten, die der Sender in vorbestimmter Weise an den Empfänger verschickt, Objekte durch das Informationssystem *migrieren*. Sie suchen sich ihre Zielobjekte selbst und kommunizieren mit diesen eigenständig.

Hardware realisierbar wäre. Zweitens suggeriert das Senden einer Nachricht einen einzelnen Empfänger an einem festen Ort, was die arbeitsteilige und räumlich verteilte Beantwortung der Nachricht ohne Not einschränkt. Die Client-Server-Metapher dagegen vermeidet weitgehend sprachliche Einengungen und etabliert die Kommunikation zwischen Objekten auf einer semantisch hohen Ebene. Bild 10.13 zeigt die Client-Server-Konstellation [Wilkerson & Wirfs-Brock, 1989]. Die metaphorische Funktion dieses Bildes haben wir bereits in Kapitel 7 diskutiert.

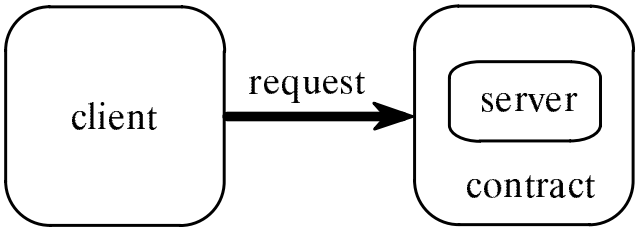


Bild 10.13: Die Metapher des Client-Server-Vertrags

Objekte können eine von zwei Rollen in der Kommunikation einnehmen, wobei synonyme Rollennamen in der Literatur auftreten:

1. Kunden-Objekt (*client*, *customer*, Auftraggeber, Sender)
2. Anbieter-Objekt (*server*, *supplier*, Auftragnehmer, Empfänger)

Die Client-Server-Beziehung zwischen Objekten ist grundlegend, weil sich das Systemverhalten objektorientierter Anwendungen in den Objekt-Interaktionen äußert. Die interaktive Kommunikation hat zwei Elemente: 1. den ein- oder beidseitig gerichteten Datenfluß und 2. die Synchronisation. Das Grundmodell objektorientierter Kommunikation bezieht sich auf ein Objektpaar, das in einem Dienstleistungsverhältnis steht: ein Objekt fordert einen Dienst an („Dienst“ ist Platzhalter für ein Programmodul mit Datenzugriff) und das aufgeforderte Objekt führt den Dienst aus oder delegiert ihn auf andere Objekte. Das anfordernde Objekt nennt stets den Adressaten; umgekehrt weiß der Adressat nicht unbedingt, von wem die Anforderung stammt.

Um eine Dienstleistung anfordern zu können, muß der Anbieter für den Kunden „sichtbar“ sein. In der Regel ist die Sichtbarkeit in einer Aggregation gegegeben: aggregierte Objekte haben die gleiche Lebensdauer wie das Aggregat und sind für dieses sichtbar. Objekte können aber auch Objekte außerhalb ihres Aggregats benutzen, man spricht dann

von einer globalen Sichtbarkeit entlang von assoziativen Objektbeziehungen. Weiterhin kann zwischen statischer und dynamischer Sichtbarkeit unterschieden werden: dynamische Sichtbarkeit liegt zum Beispiel vor, wenn ein Anbieter-Objekt erst zur Laufzeit dem Kunden-Objekt durch einen Parameteraustausch bekannt wird. Die Sprache C++ bietet hier eine Palette von Sichtbarkeitsoptionen an: *public*, *protected*, *private* und *friend* (siehe [Stroustrup, 1986]).

Das zweite Element einer Kommunikation ist deren Synchronisation. Eine *synchrone* Kommunikation verlangt vom Client-Objekt, daß es seinen Prozeß solange anhält, bis seine Anfrage an das Server-Objekt beantwortet wurde. Dagegen erlaubt eine *asynchrone* Kommunikation die Fortsetzung des Prozesses unabhängig vom Bearbeitungszustand der Anfrage. Der synchrone Fall kann weiter differenziert werden: (a) *balking*: der Client zieht seine Anfrage zurück, wenn der Server nicht zum Zeitpunkt der Anfrage dienstbereit ist; (b) *time-out*: dito, aber erst nach Ablauf einer festgelegten Zeitspanne; und (c) *reliable*: die Anfrage bleibt bestehen, bis der Server dienstbereit ist.

Analog zur RPC-Erweiterung des prozeduralen Kommunikationsmodells lassen sich auch im objektorientierten Client-Server-Modell die Maschinengrenzen überwinden: *Object Request Broker* (ORB) koordinieren die Kommunikation in verteilten heterogenen Netzen. Im Gegensatz zum RPC-Ansatz sind die Maschinengrenzen aber für den Entwickler transparent: Mit einem ORB muß er den genauen Speicherort der Datenbestände im Netz, ob eigene oder entfernte Maschine, nicht mehr kennen. Daraus folgt: ein Wechsel des Speicherorts im Netz ist ohne Programmänderung möglich. Der Wartungsaufwand bei organisatorischen Maßnahmen, wie zum Beispiel einem *Downsizing* der Rechnerwelt, bleibt somit minimal. Für ORB-Architekturen wurde von der OMG ein Standard formuliert und veröffentlicht: CORBA, *Common Object Request Broker Architecture* [Soley (Hrsg.), 1992]. Alle im Netz verfügbaren Server-Objekte sind beim „Makler“ registriert. Anfragen werden über den Makler abgewickelt, das heißt, sie werden an die geeigneten Server-Objekte weitergeleitet. Deren Antwort leitet der Makler wieder an das Client-Objekt zurück. Die Größe der durchgängig verfügbaren Objektwelt bestimmt den Nutzen der Client-Server-Programmierung: C++ beschränkt die Objektwelt auf die Laufzeit eines einzelnen Lademoduls, Smalltalk auf einen einzelnen Arbeitsplatz; im OMG-Modell aber ist die Größe der Objektwelt beliebig: Für jeden Maschinentyp muß lediglich ein einfacher Object-Request-Broker geschrieben werden [Rösch, 1993].

Der Client-Server-Vertrag steht nicht nur als Metapher für die objektorientierte Kommunikation, er suggeriert zugleich bindende und zusichernde Elemente, die Pflichten und die Rechte der Vertragspartner. Dies führt uns auf den noch fehlenden Objektaspekt — die Beschränkung.

10.4 Rationale Beschränkung

Die Repräsentation von Struktur- und Verhaltensmuster allein ist nur bedingt ausreichend. Ulrich REIMER über die Eigenschaften einer Wissensrepräsentation:

> 1. „Eine Repräsentation erfaßt einige Merkmale der repräsentierten Welt.
> 2. Eine Repräsentation erfaßt nicht zwangsläufig (in der Regel nie) alle Merkmale der repräsentierten Welt.
> 3. Nicht alle Merkmale einer Repräsentation stehen zwangsläufig für Merkmale der repräsentierten Welt."
>
> [Reimer, 1991, S. 9]

Die Repräsentation eines Weltausschnitts erfordert also sowohl Repräsentationsstrukturen als auch eine *Vorschrift*, wie diese Strukturen zu *interpretieren* sind. Diese Vorschriften gestatten erst die Formulierung und Auswertung von Anfragen oder Änderungswünschen [Reimer, 1991, S. 9]. Da Constraint-Netze regelhafte Zusammenhänge und einschränkende Bedingungen ausdrücken, können sie die Funktion der Interpretationsvorschriften übernehmen. *Explizit* treten Constraints als Funktionen, Prozeduren, Regeln, logische Prädikate oder Relationen (Tabellen) auf. *Implizit* sind sie in den semantischen Beziehungstypen enthalten. Zum Beispiel verbergen sie sich in der *transitiven* Eigenschaft der Generalisierung: wenn A Unterklasse von B ist und B Unterklasse von C, dann ist auch A Unterklasse von C.

Wir wollen den Aspekt der *wissentlichen* Einschränkung von Struktur- und Verhaltensmustern verallgemeinern und von der „rationalen Beschränkung" eines Objekts sprechen. Wir interpretieren dabei SIMONs Begriff der *bounded rationality* aus der Objektsicht: Wir meinen genau die strukturellen und verhaltensmäßigen Beschränkungen, die das Objekt erfüllen muß, um unserer Vorstellung von ihm zu genügen. Im Sinne der Definition des Abschnitts 10.2.2 spezifiziert also die rationale Beschränkung die begriffliche *Intension* eines Objekts, das heißt dessen Verwendungsnorm, Regelwerk und Test auf Zugehörigkeit. Auch korrespondiert die rationale Beschränkung mit dem ontologischen Begriff des *Gesetzes*, erläutert im Anhang B: Wie Gesetze beschränken auch Constraints die Zustandsmenge und die Zustandsübergänge (= Ereignisse) der repräsentierten Objekte. Betroffen sind sowohl stoffliche Dinge, sie unterliegen den Natur- und Technikgesetzen, als auch gedankliche Dinge, sie unterliegen unserer Ratio.

Weiterhin können Constraints die *deontische Modalität* eines Sachverhalts definieren: Deontische Constraints beschränken die Vielfalt des Möglichen auf das „Obligatorische", „Gebotene" und „Verbotene". Auf ihren modalen Charakter haben wir bereits im Referenzmodell zum Integritätsbegriff hingewiesen (Abschnitt 3.3). Verfahrenstechnisch beschrei-

ben Constraints die Reduktion der Variationsvielfalt, wie es Bild 9.2 auf Seite 181 veranschaulicht. Die jeweilige Rechtfertigung spiegelt sich im Integritätsaspekt wider, der durch die Prüfung und Befriedigung spezieller Constraints gesichert werden soll: Semantische, methodische und pragmatische Integritätsbedingungen, wie im Referenzmodell definiert, können durch Constraints *objektzentriert* werden. Die dann *lokal* operierende Constraint-Propagierung steigert entscheidend die Effizienz der Integritätssicherung.

Fassen wir die Mehrfachfunktion eines objektorientierten Constraints zusammen:

- Dokumentierung der Intension (Begriffsinhalt): Welche Eigenschaften besitzen Objekte der Klasse? Welche Semantik tragen die Objektbeziehungen?
- Einschränkung der Extension (Begriffsumfang): Welche Objekte gehören zur Klasse? Welche Objekte gehören zur Objektaggregation? Wieviele Objekte müssen mindestens, wieviele können maximal in Beziehung zueinander stehen (Kardinalität der Objektbeziehung)?
- Repräsentation von Wissen: Constraints sind deklarativer oder prozeduraler Ausdruck regelhafter Zusammenhänge und einschränkender Bedingungen.

10.4.1 Beispiele zur rationalen Objektbeschränkung

Der Constraint-Aspekt wurde in der einschlägigen Literatur zur Objektorientierung bisher nur wenig berücksichtigt. Berührt wird er im wesentlichen nur von der objektorientierten Metapher des Client-Server-Vertrags. Hier berücksichtigt Bertrand MEYER den Constraint-Aspekt zumindest programmiertechnisch durch das Konzept der „Zusicherung". Funktional gesehen stellt die Zusicherung einen positiv formulierten Constraint dar: die Einhaltung einer *invarianten* Größe.

Auf die Analogie zur algebraischen Spezifikation eines abstrakten Datentyp haben wir bereits hingewiesen: Dort übernehmen *Axiome* oder *Laws* die Funktion der Constraints (siehe das ADT-Beispiel auf Seite 217). Weitere Parallelen verbergen sich in den Klassenhierarchien: Spezialisierung und Generalisierung sind struktureller Ausdruck vermehrter oder verminderter Constraints. Generaliserte Klassen entstehen durch Entfernen von Constraints, das heißt es wird von einigen der regelhaften Zusammenhänge und Einschränkungen in den Unterklassen abstrahiert. Umgekehrt gilt für spezialisierte Klassen, daß sie mehr Constraints aufweisen als ihre Oberklassen. In dem Maße, wie die Extension mit der Spezialisierung zunimmt, nimmt die Intension mit der Generalisierung ab. Constraints bestimmen das Maß dieser Wechselwirkung.

Zwei weitere Beispiele zur rationalen Objektbeschränkung wollen wir skizzieren: Das eine ist wichtig für die objektorientierte Produktmodellierung, das andere für die softwaretechnische Konsistenzsicherung.

Constraints und Express

In der objektorientierten Modellierungssprache Express, die derzeit einen hohen Rang in der Produktmodellierung einnimmt ([McLaren, 1992] und Abschnitt 8.2), lassen sich rationale Beschränkungen nur spärlich beschreiben: Lokale Constraints werden innerhalb einer Objekttypbeschreibung (*entitiy*) definiert. Sie schränken den Wertebereich einzelner Objektattribute ein: *where*-Klausel in Express. Globale Constraints beschreiben prozedurale *if*-Regeln und Gesetzmäßigkeiten zwischen mehreren Objekttypen. Sie werden im Schema definiert, somit außerhalb der Objekttypbeschreibung.

Um nun das *Data dictionary* aussagekräftig zu gestalten, wurde in [Quibeldey-Cirkel & Wahl, 1990b] ein Standardisierungsvorschlag publiziert: Die in den Express-Modellen verteilt spezifizierte Struktur- und Verhaltensinformation wird automatisch extrahiert und die Zusammenhänge dokumentiert. Auf diese Weise werden die Querbezüge transparent gehalten und unvollständige oder widersprüchliche Spezifikationen identifizierbar. Das Schema des *Data dictionary* eignet sich als Grundlage für die Konzeption semantischer Prüfprogramme.[20] Bild 10.14 zeigt das verkürzte Schema für die extrahierte Objektinformation. Ein Novum für Modellierungssprachen stellt die *Rationale*-Klausel dar: Sie schränkt die Interpretationsbreite der Objekttypbeschreibungen ein und hebt — zumindest verbal — die Bedeutungsabsicht des Entwerfers hervor: Die Daseins- und Zweckbestimmung entsprechender Objekte dieses Typs und der Bezug zu Objekten anderen Typs werden kontextbezogen dokumentiert. Das genügt dem in Abschnitt 10.2.2 motivierten Dokumentationsgebot für den *Intensions*-Aspekt eines Objekts.

Pflichten, Zusammenarbeit und Verträge

„Wunsch und Wirklichkeit“ heißen in der Informatik „Spezifikation und Implementierung“. So gilt ein Programm als fehlerfrei (korrekt), wenn es implementiert, wofür es spezifiziert wurde. Ein pragmatischer Weg, die Diskrepanz zwischen Spezifikation und Implementierung gering zu halten, ist das Konzept der „Zusicherung“. Hier werden Elemente der Spezifikation — Vorgaben und Randbedingungen — *invariant* bis zur untersten Ebene der Implementierung beibehalten: Die Operationen einer Klasse sind *Vor*- und *Nachbedingun-*

[20] Derzeit verfügbare grafische Entwurfsumgebungen bieten nur selten eine semantische Prüfoption auf strukturelle Konsistenz: Bestenfalls handelt es sich um *notationssensitive* Grafikeditoren [Knolmayer & Myrach, 1990].

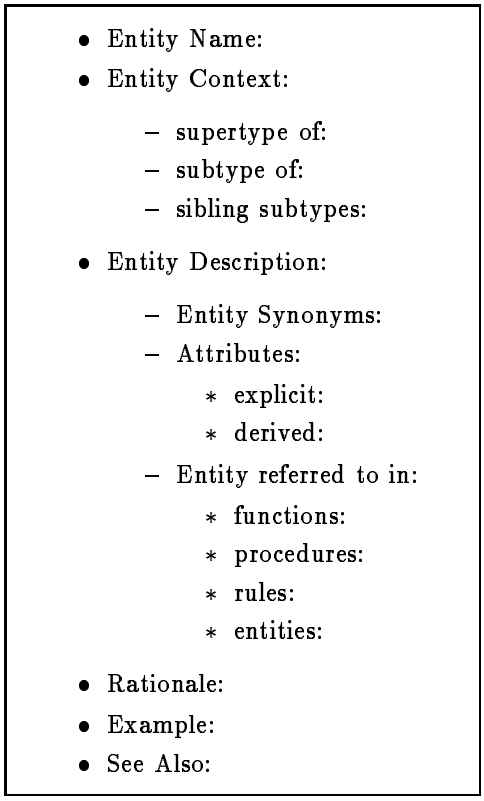


Bild 10.14: „Rationale Beschränkung“ in Express

gen unterworfen; logische Schranken der Klasseneigenschaften werden als *Invarianten* spezifiziert (BOOLEsche Gleichungen, mathematische (Un-)Gleichungen). Die Möglichkeit, die Zweckbestimmung und die Aussagen über die Korrektheit eines Programmelements direkt in die Implementierung einzubringen, ist von zentraler Bedeutung für das objektorientierte Vertragskonzept: *Programming by Contract* [Meyer, 1988].

MEYER hat das Vertragskonzept bereits durch entsprechende Ausdrucksmittel in seiner objektorientierten Sprache Eiffel verankert: Zusicherungen heißen dort *assertions*. Die Klassenoperationen werden in (optionalen) Vor- und Nachbedingungen sowie invarianten Bedingungen eingebettet: Vorbedingungen binden das Client-Objekt. Sie legen fest, ob die Anforderung eines Dienstes im Sinne des Vertrags legal ist (*require*-Klausel). Nachbedingungen binden die Klasse und damit alle Server-Objekte dieser Klasse. Sie legen fest, in welchem legalen Zustand sich das Server-Objekt nach der Dienstleistung befindet (*ensure*-Klausel). Invariante Bedingungen müssen für die Dauer der Anfrage bis zu ihrer Beantwortung ständig erfüllt sein (*invariant*-Klausel). Die Prüfung der Korrektheitsbedingungen einer Operation, oder allgemeiner des Objektzustands, ist somit im Objekt *zentriert* und wird mit jeder Service-Anforderung veranlaßt. Tabelle 10.3 zeigt am Bei-

spiel der *push*-Operation eines Keller-Objekts die Matrix aus Verpflichtung und Nutzen für beide Vertragspartner.

	Pflichten	*Rechte*
Kunde	Vorbedingungen garantieren (Aufruf von *push(x)*, nur wenn der Stack dabei nicht überläuft)	garantierte Nachbedingungen (x ist oberes Stack-Element)
Anbieter	Nachbedingungen garantieren (sicherstellen, daß x oberes Stack-Element wird)	garantierte Vorbedingungen (keine eigene Prüfung, ob Stack überläuft)

Tabelle 10.3: Matrix eines Klassenvertrags

Das „Vertragskonzept im kleinen“, also auf der Klassenebene, fördert den einfachen und effektiven Programmierstil. Die systematische Formulierung von Vorbedingungen reduziert die vorbereitenden Maßnahmen der Programmierung. Es entfällt die ständige Prüfung, ob die zu verarbeitenden Daten die Voraussetzungen für eine korrekte Verarbeitung erfüllen. Fehlen formal vereinbarte Pflichten zwischen dem Benutzer eines Programmoduls und dessen Programmierer, wird entweder gar nicht (scheinbare Sicherheit) oder aber mehrere Male geprüft (beiderseitige Unsicherheit). MEYER beschreibt die Risiken:

> „The distribution of redundant checks all over a software system destroys the conceptual simplicity of the system, increases the risk of errors, and hampers such qualities as extendability, understandability and maintainability.“ [Meyer, 1988, S. 117]

Das Vertragskonzept zwischen Client- und Server-Objekten ist also methodisch und pragmatisch zugleich. Es unterstreicht den Anspruch objektorientierter Konzepte, die Phasen des Systementwurfs durchgängig zu unterstützen.

10.4.2 Constraint-basierte Integritätssicherung

Spezifikationssprachen für komplexe Systeme müssen den dreifachen Aspekt eines jeden Entwurfsobjekts einfangen und prägnant formulieren können: Gleichgültig auf welcher Abstraktionsebene des Entwerfens das Entwurfsobjekt gerade betrachtet wird, alle drei Aspekte müssen einheitlich und in geschlossener Form erfaßbar sein: Struktur, Verhalten

und rationale Beschränkung. Bisher sind aber kaum entsprechende Sprachen und Methoden entwickelt worden. Auf die repräsentativen Ausnahmen gehen wir nun ein. Da der allgemeine VLSI-Entwurf vor allem ein Problem der integren Datenhaltung ist, werden wir entsprechende Constraint-Modelle im Kontext von *Entwurfsdatenbanken* erörtern. Die VLSI-spezifischen Entwurfs*transaktionen* und *-versionen* klammern wir aus: ihre Modellierung ist zum einen hinreichend dokumentiert [Gray, 1981; Katz *et al.*, 1986; Katz, 1990; Kim *et al.*, 1983] und fällt zum anderen in die Verantwortung des Transaktions- und Versionsmanagement eines Datenbanksystems.

Non-Standard-Datenbanken

Die semantische Integrität einer Datenbank definiert sich über die Einhaltung *anwendungs*spezifischer Constraints. Sie müssen folglich durch den Benutzer definierbar sein. Herkömmliche Datenbanken, wie die relationalen, netzartigen und hierarchischen [Schlageter & Stucky, 1983], erlauben nur im bescheidenen Maße die Definition anwendungsspezifischer Constraints. Was die Constraint-Modellierung komplexer Entwurfsobjekte betrifft, so ist diese in konventionellen Datenmodellen stark eingeschränkt: Die *modellinhärente* Konsistenz (siehe Kapitel 3.2) ist selbstredend nur so mächtig wie das Datenmodell, auf das sie sich bezieht. Daß es hier für Non-Standard-Anwendungen wie den VLSI-Entwurf eklatante Modellierungsdefizite gibt, ist hinlänglich bekannt [Batory & Buchmann, 1984; DASSY, 1990; DASSY, 1992a; Encarnação & Lockemann, 1990]. Die Defizite lassen sich zwar durch geeignete *1-zu-n*-Abbildungen oder Modellerweiterungen prinzipiell kompensieren (so zum Beispiel beim Einsatz relationaler Datenbanken (RDBMS) für den VLSI-Entwurf [Brauer, 1990]), das Antwortverhalten der Datenhaltung ist für interaktive Anwendungen aber inakzeptabel [Maier, 1989]. Objektorientierte Datenmodelle wurden konzipiert und realisiert (Postgres, Gemstone und andere), um die Laufzeit- *und* Modellierungsdefizite zu kompensieren. Auch wenn die Entwicklung objektorientierter Datenbanken (OODBMS) derzeit vom Prototypenstadium zur Marktreife schreitet, bleibt aber auch bei diesem Datenbanktypus ein prinzipieller Konflikt: das Spannungsdreieck der konkurrierenden Anforderungen (Bild 10.15)

- Will man Effizienz und Flexibilität in der Constraint-Modellierung, so müssen die Definition und Sicherstellung der Datenintegrität in die Anwendung ausgelagert werden. Der Versuch, komplexe CAD-Funktionen als datenbankinterne Constraints in die Datenbank einzubringen, zum Beispiel die Prüfung der Entwurfsregeln und der elektrischen Regeln (DRC/ERC), potenziert natürlich die Antwortzeiten. Die primäre Aufgabe einer Entwurfsdatenbank bleibt die effiziente Speicherung und der effiziente Datenzugriff. Wollte man iterative CAD-Operationen (zum Beispiel die Simulation) als Daten-Constraints verwalten, wäre die primäre Aufgabe nicht mehr zu erfüllen.

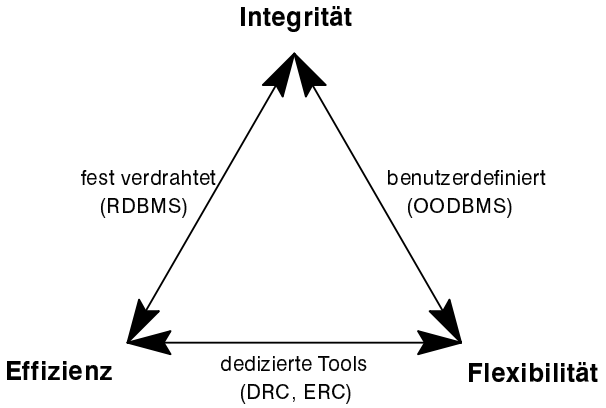


Bild 10.15: Trilemma der Datenbanktechnik

- Will man Effizienz und Datenintegrität, so ist dies nur auf Kosten der Flexibilität erreichbar: Wie wir bereits ausführten, lassen sich zwar auch komplexe Integritätsbedingungen mit laufzeiteffizienten Datenmodellen, wie dem relationalen, realisieren, allerdings nur durch modellimmanente Erweiterungen, durch „fest verdrahtete" Constraints also. Entwerfen ist aber ein Erforschen *a priori* unbekannter Constraints, wie wir es im vorigen Kapitel eruierten. Eine flexible Definition neuer und eine flexible Redefinition alter Constraints ist daher unverzichtbar.

- Will man nun Datenintegrität und Flexibilität, so geht dies nur durch benutzerspezifizierbare, sogenannte *explizite* Constraints. Objektorientierte Datenmodelle erlauben die Definition und Ausführung beliebig komplexer Constraints: Beliebige Prozeduren und Programme können nach dem Kapselungsprinzip lokal an die Datenobjekte gebunden werden. Es ist eine Frage der Zentralisierung, ob komplexe Constraints die Effizienz des Datenbanksystems soweit verschlechtern, bis sie inakzeptabel wird. Für lokal wenig rechenintensive Constraints, scheint dieser Weg gangbar. Hier ist ein Kompromiß zu suchen. Er wird sich offenbar im Mittelpunkt des Spannungsdreiecks finden lassen: Objektorientierte Constraint-Modelle, die lokal beschränkte Constraint-Mechanismen nutzen, wie die lokale Constraint-Propagierung, sind ein datenbanktechnischer Kompromiß im Trilemma von Integrität, Effizienz und Flexibilität.

Im Abschnitt 9.2.2 haben wir bereits die Constraint-Kategorien der KI auf die Integritätsaspekte strukturell objektorientierter Datenbanken übertragen. Der Effizienz wegen scheidet auch hier die Berechnung der *globalen* Konsistenz einer Datenhaltung aus: Algorithmen für *strenge Pfadkonsistenz* sind NP-vollständig. Das Potential des objektorientierten Constraint-Ansatzes liegt in den beiden *lokal* beschränkten Constraint-Kategorien: So korrespondiert die Knotenkonsistenz mit der Typprüfung (Prüfung auf Klassenzugehörigkeit)

eines Datenbankobjekts. Die Kantenkonsistenz korrespondiert mit der *affinen* Integrität einer Objektaggregation, in der also die Objekte unmittelbar benachbart sind. Für die beiden letzten Fälle existieren effiziente Constraint-Algorithmen. Es scheint deshalb lohnend, objektorientierte Constraint-Modelle als Datenbankansatz zu verfolgen.

10.4.3 Objektorientierte Constraint-Modelle

Eisuke MUROGA hat diesen Ansatz in seiner Abhandlung untersucht [Muroga, 1990]. Anwendungsunabhängig unterscheidet er zwischen zwei Grundmodellen: Constraints, die in der Klassendefinition eingebettet sind (*embedded constraints*), und *eigenständige* Constraint-Objekte. Wir übernehmen diese prinzipielle Unterscheidung, um einige Fallbeispiele zur constraint-basierten Objektmodellierung zu klassifizieren. Zunächst erläutern wir die Charakteristika der beiden Grundmodelle.

Embedded Constraints versus Constraint-Objekte

Wie können Constraints in einem objektorientierten Datenmodell integriert werden? Prinzipiell auf zwei Arten: Entweder sind sie *integraler* Bestandteil der Klassendefinition eines Objekts, das heißt sie werden im Datenbankschema definiert, oder sie sind *eigenständige* Objekte. Für den ersten Fall sind all diejenigen Constraints prädestiniert, die im voraus bestimmbar sind: *antizipierte* Constraints. Alle Objekte einer Klasse mit eingebetteten Constraints sind dann der gleichen, statisch vordefinierten Struktur unterworfen. Der zweite Fall erlaubt die Differenzierung auf der Objektebene: Nur ausgewählte Objekte einer Klasse werden den Einschränkungen der Constraint-Objekte unterworfen. Diese Constraints wirken also dynamisch; sie sind *ad hoc* definierbar. Bild 10.16 gibt ein Beispiel für beide Grundmodelle.

Für den einfachen Fall, daß Constraints nur auf elementare Objekte wirken, ist eine Unterscheidung unnötig. Erst wenn Objekte in Beziehung zu anderen Objekten treten, sie also komplexe, das heißt zusammengesetzte Objekte bilden, gibt es prinzipiell zwei Lokalisierungen für die Information über diese Beziehung. Die Constraints stehen dann für *mehrstellige* Prädikate über dem Objektverband. Sollen nun mehrstellige Constraints in den Teilobjekten lokalisiert werden, so muß jedes Objekt seine Zugehörigkeit zum Objektverband durch individuelle Constraints reflektieren. Die lineare Gleichung $y = mx + b$ im Bild 10.16 findet sich also in jedem Objekt wieder, allerdings aus der Warte des Objekts gesehen (Teilbild a): Das Objekt x zum Beispiel wird seinen Part am Constraint als $x = \frac{y-b}{m}$ ausdrücken. Jedes partizipierende Objekt muß darüber hinaus alle Formen des Constraints aufnehmen, so wie sie aus der Warte eines jeden Objekt gelten. Denn im Sinne des Constraint-Satisfaction-Problems ist jede Form in jedem beteiligten Objekt-

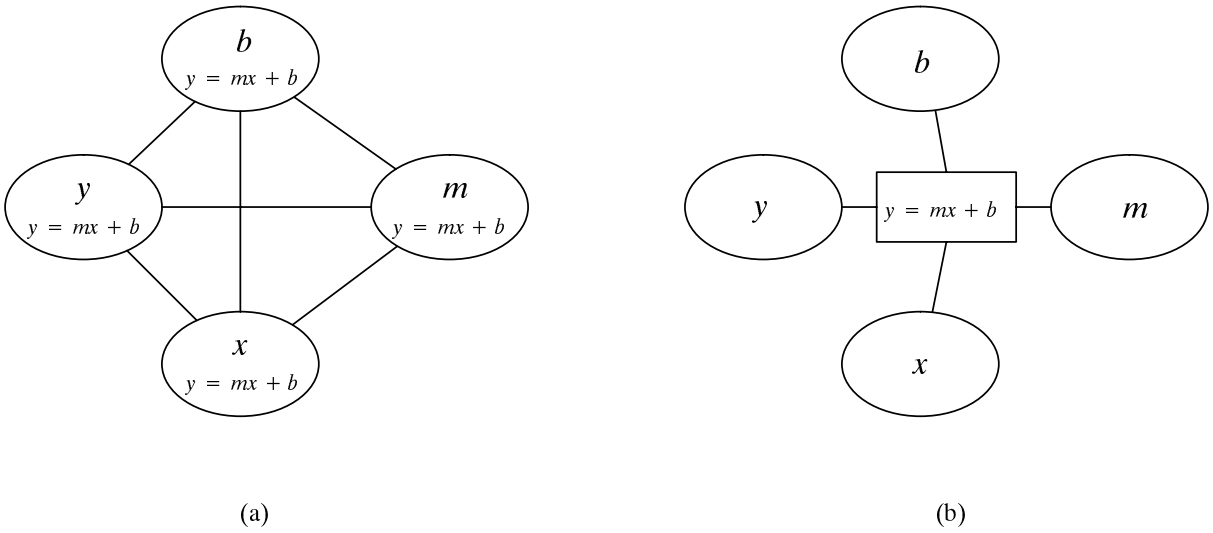


Bild 10.16: *Embedded Constraints* versus *Constraint-Objekte*

knoten notwendig: einmal um einen Wert für den Knoten selbst zu berechnen und zum anderen um den berechneten Wert an die Nachbarknoten zu propagieren (durch Triggern einer entsprechenden Änderungsoperation, siehe Bild 10.17).

Ein Constraint-Objekt dagegen ist die *zentralisierte externe* Repräsentation eines mehrstelligen Constraints (Teilbild b in Bild 10.16).[21] Durch die zentrale Stellung des Constraint-Objekts erhöht sich offensichtlich der Kommunikationsaufwand, um den Constraint in allen beteiligten Objekten zu validieren und zu erfüllen (Bild 10.18). Dem Constraint-Objekt kommt eine umfassende Monitorfunktion zu: Es muß eventuelle Wertänderungen in allen, dem gemeinsamen Constraint unterworfenen Objekten überwachen. Das Constraint-Objekt selbst wird verwaltet wie jedes andere adressierbare Datenbankobjekt; der Constraint-Mechanismus behandelt Daten- und Constraint-Objekte gleich. Als selbständiges Objekt verfügt das Constraint-Objekt über eigene Operationen (Methoden) und Variablen.

Beide Grundmodelle sind wichtig: Entwerfen ist zum Teil ein vorbestimmter, zum Teil aber auch ein dynamisch entschiedener Prozeß: So sind die technischen Parameter vom Hersteller der Entwurfszellen vorgeschrieben (= vorbestimmt). Dasgleiche gilt für wirtschaftliche und zeitliche Randbedingungen, die das Projektmanagement vorgibt. Anders die „selbstbestimmten“ Constraints unserer Intuition und Kreativität: Sie treten während des Entwerfens zutage und sind ihrer Natur nach zufällig. Vorbestimmte Constraints können statisch definiert werden durch eingebettete Constraints. Da sie auf der Schema-Ebene definiert

[21] Einstellige Constraints stellen einen einfachen Sonderfall dar.

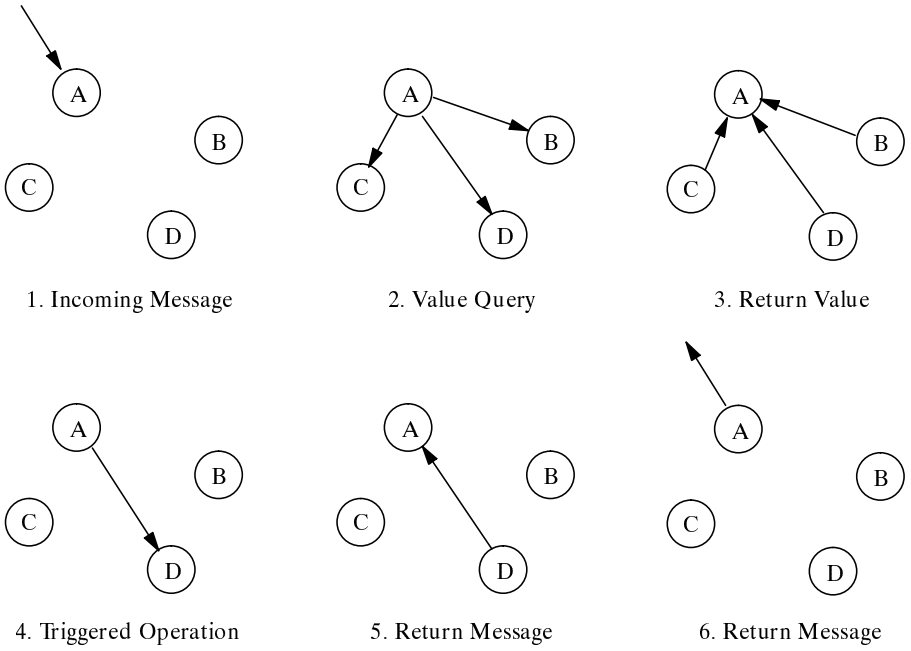


Bild 10.17: Protokoll für *Embedded Constraints*

werden, charakterrisieren sie alle Objekte einer Klasse. Die Abbildung mehrstelliger Constraints auf Constraints, eingebettet in jedem beteiligten Objekt, resultiert in redundante verteilte Constraint-Definitionen. Die perspektivische Constraint-Formulierung (siehe das Beispiel der linearen Gleichung) bringt es mit sich, daß jedes Objekt einer anderen Unterklasse angehört. Eingebettete Constraints können folglich auch nur aufwendig durch eine Schemamodifikation geändert werden. Im Nachteil des einen liegt der Vorteil des anderen Grundmodells: Constraint-Objekte lassen sich dynamisch modifizieren, ohne das Schema, das heißt die Klassendefinition zu ändern. Die Constraint-Spezifikation kann dadurch dynamisch der Entwicklung des Datenbestands angepaßt werden.

Was den Nachrichtenaustausch betrifft, so sind für die beiden Grundmodelle dedizierte Protokolle zu entwickeln — je nach der Strategie, wann und wie die Constraints überwacht und befriedigt werden sollen. MUROGA hat bereits entsprechende Protokolle entwickelt, auf die wir hier aber nur verweisen können. Die beiden objektorientierten Constraint-Modelle dienen uns zur Klassifikation objektorientierter Constraint-Systeme. Die Tabelle 10.4 nennt das Ergebnis. Die dort eingeordneten objektorientierten Constraint-Systeme sind repräsentatitv: Sie erlauben die *vollständige* Spezifikation des *Objekt-Tripels*. „Vollständig objektorientierte“ Datenmodelle im Sinne von DITTRICH erfassen dagegen nur die Struktur- und Verhaltensaspekte eines Entwurfsobjekts; sie ignorieren oder vernachlässigen zumindest den Constraint-Aspekt.

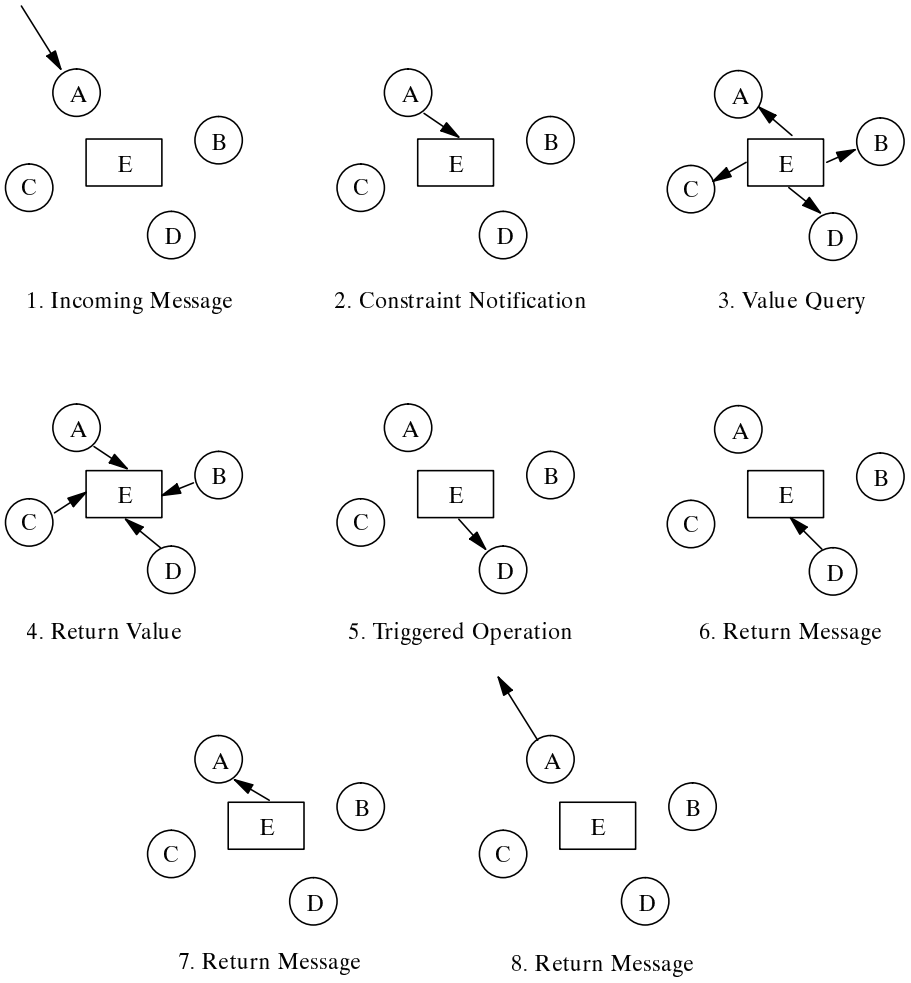


Bild 10.18: Protokoll für *Constraint-Objekte*

Grundmodell	*Beispiele*
Eingebettete Constraints	ThingLab [Borning, 1981], STEM [Girczyc & Ly, 1988] DOC [Sapossnek, 1989], FREEDOM [Yokoyama, 1990]
Constraint-Objekte	PROSE [Berlandier, 1991], JASMINE [Ishikawa *et al.*, 1991] OAR [Arbab & Wang, 1991], CRUX [Köhler, 1992]

Tabelle 10.4: Einordnung objektorientierter Constraint-Systeme

Die Möglichkeit, Aussagen über die rationale Beschränkung eines Objekts zu spezifizieren, ist aber nur ein Teil der Forderung. Die spezifizierte Beschränkung auch *operativ* zu prüfen

und gegebenenfalls zu sichern, ist die ganze Forderung. Daß der VLSI-Entwurf nicht über alle Abstraktionsebenen hinweg als ein Constraint-Satisfaction-Problem behandelt werden kann, haben wir mehrfach betont. Es ist auch nicht das Ziel der in der Tabelle genannten Constraint-Systeme, den Entwurfsprozeß als CSP zu *automatisieren*. Vielmehr zeigen sie aspekthaft die Einsatzmöglichkeiten auf: Constraint-Systeme als Unterstützungssysteme ist das Thema.

Fazit

> „Da dieses Kapitel viele die Menschen erregende Gedanken beschreibt, ist dies eine geeignete Gelegenheit, zur Vorsicht zu mahnen. Die Erfahrung zeigt, daß viele Ideen der Künstlichen Intelligenz uns ähnlich wie sagenhafte Sirenen zu Fehlanwendungen verführen. Folglich müssen wir darauf achten, Probleme zu lösen und nicht nach Möglichkeiten Ausschau halten, um Ideen nur aus dem einzigen Grund anzuwenden, weil sie phantastisch sind. Der beste Weg, dabei achtzugeben, ist, die folgenden methodischen Gebote zu befolgen:
>
> 1. Zunächst *stellen Sie die Art des Problems fest.*
> 2. Dann *wählen oder erfinden Sie eine geignete Repräsentation.*
> 3. Als nächstes *stellen Sie Beschränkungen oder Gesetzmäßigkeiten heraus.*
> 4. Erst jetzt sind *bestimmte Verfahren zu kreieren.*
> 5. Schließlich *muß durch Experimente alles überprüft werden.*“
>
> [Winston, 1987, S. 102]

Unser Ansatz, das Constraint-Konzept der KI mit den Konzepten des Objekt-Paradigmas zu integrieren, folgt den „methodischen Geboten“, wie sie Patrick WINSTON für die KI formuliert hat. 1. Das Tripel des Objekts findet seine Anwendung beim besonders *datenintensiven Problem* des VLSI-Entwurfs. 2. Wie wir im Kapitel 8 zeigten, ist die hierfür *geeignete Repräsentation* die objektorientierte. 3. Die VLSI-Technik zeichnet sich durch eng gekoppelte geometrische und physikalische *Beschränkungen und Gesetzmäßigkeiten* aus (*design* und *electrical rules*). Neben der Struktur und dem Verhalten der VLSI-Objekte ist also auch der Aspekt der Beschränkung ein *integrer* Bestandteil der Objektsicht — rational sowohl im Sinne der Anwendung als auch technisch im Sinne des Machbaren. 4. Setzt man das „Tripel des Objekts“ als konzeptionelle Grundeinheit des VLSI-Entwurfs, so können effektive *Verfahren* der Integritätssicherung *kreiert* werden. 5. Dies ist prototypisch geschehen, was WINSTONs *Experimenten* entspricht; wir klassifizierten entsprechende Prototypen aus der Literatur.

Der Ansatz „Entwerfen ist ein *Constraint-Satisfaction*-Problem“ führt letztlich auf wissensbasierte Entwurfsdatenbanken mit einer Ableitungskomponente: Expertensysteme für CAD. Die dargelegten Erkenntnisse sind als Vorstufe in diese Richtung zu werten. Aus

pragmatischen Gründen ist eine derartige Vorstufe in der Mentalität des Entwerfers und im Techniktransfer der industriellen Praxis unabdingbar: Die Akzeptanz neuer Methoden und Techniken ist um so größer, je intuitiver der evolutionäre Übergang erfolgt. Überdies ist der allgemeine Entwurf ein NP-vollständiges Problem, so daß die mangelnde Effizienz den Akzeptanzverzug zusätzlich vergrößert.

Ein weiterer Gesichtspunkt ist die *Diversifikation* der Entwurfsbereiche. Ein „General Problem Solver“ in der Gestalt eines universellen „Silicon-Compilers“ wird und kann es nicht geben. Allgemeine Entwürfe sind die Ausnahme. Der VLSI-Entwurf lebt von innovativen und sehr anwendungsspezifischen Ideen, die es gilt, kurzfristig als Produkt in den Markt einzuführen. Dabei ist der Entwerfer permanent zwei Kräften ausgesetzt: der Schubwirkung modernster Techniken und der Sogwirkung rasch wechselnder Marktbedürfnisse. Er wird sich schon deshalb mit technisch nicht-optimalen Entwürfen begnügen, um marktwirtschaftlich erfolgreich zu sein. Die objektorientierte Constraint-Modellierung unterstützt die Suche nach zufriedenstellenden Lösungen. Sie sichert die Integrität der Datenbankobjekte, so wie sie der Entwerfer definiert.

Der Entwurf komplexer Artefakte ist auch durch objektorientierte Constraint-Systeme nicht automatisierbar. Entwerfen bleibt ein intellektueller und kreativer Akt. Objektorientierte Constraint-Systeme schaffen aber eine kognitiv adäquate Modellierungsumgebung, in der sich die Kreativität des Entwerfers frei (freier als in nicht objektzentrierten Entwurfssystemen) entfalten kann. Die mühseligen und langweiligen Entwurfs*aufgaben* werden an den Rechner delegiert, das Entwurfs*problem* bleibt.

11

Resümee und Ausblick

Entwerfen ist ein offener, exploratorischer Prozeß mit dem Ziel, einen zufriedenstellenden Kompromiß zu finden unter allen technischen, wirtschaftlichen und rationalen Beschränkungen. Was nun den Fortschritt im rechnergestützten Entwerfen betrifft — ausgeklügelte CAD/CAE-Entwurfssysteme für Analyse und Synthese —, so ist dieser *werkzeugzentrierte* Fortschritt Ausdruck eines Mythos: Denn Werkzeuge allein machen aus einem Laienentwerfer noch keinen Experten.[1] Es gibt eine *Kreativitätslücke*: Bild 11.1. Ihr eines Ende, zugewandt der Entwurfsautomatisierung, ist zwar bestimmt durch den „Stand der Technik", ihr anderes Ende aber bleibt operativ unerreichbar. Das Erkennen und Formulieren eines Problems und der Ansatz für seine Lösung sind dort angesiedelt. Die Trendkurve im Bild 11.1 nähert sich nur asymptotisch dem Ideal des *Silicon-Compilers*: Es gibt ihn nur für *wohlstrukturierte* Anwendungen, deren Constraints *vollständig im voraus* spezifizierbar sind (Beispiele findet man in [Gajski, 1988]).

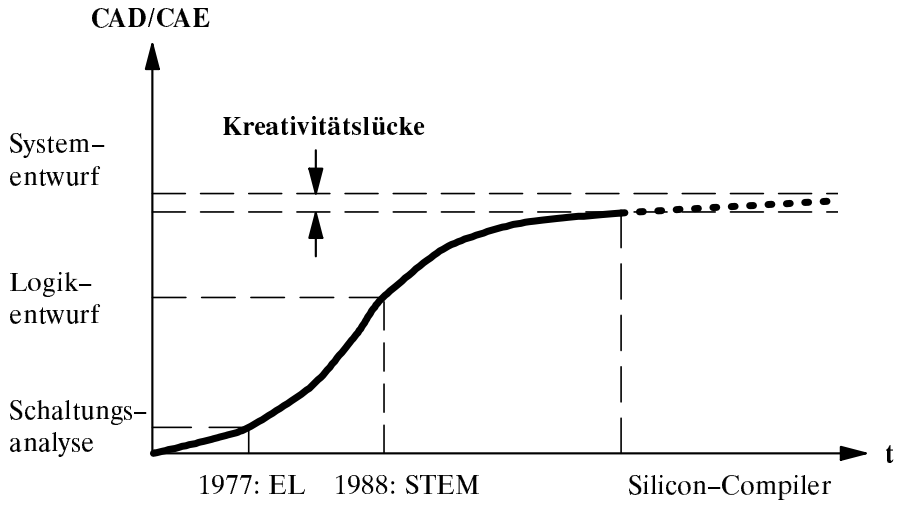


Bild 11.1: Constraint-basierte Entwurfsunterstützung

[1] Ein englisches Sprichwort sagt es treffend: „A fool with a tool is still a fool." Ähnlich prägnante Formulierungen findet man in *The Mythical Man-Month* [Brooks, 1975].

In die Kreativitätslücke fällt das Thema dieser Abhandlung: die Erstellung und Integritätssicherung von VLSI-Weltmodellen. Hier eruiert die Studie das *Objekt-Paradigma* als methodisch-operativen Ansatz. Die methodischen und pragmatischen Vorteile der Objektorientierung sind dem Programmierer hinlänglich bekannt; dem Systementwerfer dagegen selten oder unzulänglich. Unseres Wissens gibt es keine andere Studie, die Beiträge leistet, um das fehlende Kettenglied zu überbrücken (*the missing link* der Entwurfsmethodik). Es fehlt die methodische Brücke zwischen den kognitiven Prozessen während des Entwerfens und dem konzeptionellen Modellieren als Einstieg in die objektorientierte Entwurfslinie. Deren Glieder scheinen vollzählig zu sein: Analyse (OOA) → Design (OOD) → Programmierung (OOP) → Datenverwaltung (OODBMS).

Im Mittelpunkt dieser Arbeit steht die Komplexitätsbewältigung in den *konzeptionellen* Phasen des VLSI-Entwurfs.[2] Im Entwurf komplexer Schaltungen nimmt die Integritätssicherung der Datenflut die wirtschaftliche Schlüsselrolle ein. Deswegen verstehen wir den Begriff des „VLSI-Weltmodells“ als konzeptionelles Datenmodell für VLSI-Entwurfsdatenbanken. Der Hauptbeitrag dieser Studie liegt nun einerseits in der Status-quo-Analyse der VLSI-Entwurfs- und Datenmodellierung. Andererseits werden die erprobten Einzelkonzepte der Softwaretechnik und der KI vereint zu einem ganzheitlich durchgängigen Objektansatz. Wir öffnen eine neue umfassende Perspektive, wie die Entwurfskomplexität der nächsten VLSI-Systemgeneration angegangen werden kann: Anstatt nach Rezepten und Regelwerken für das Problemlösen zu forschen, sollte das *Problemverständnis* des Entwerfers gefördert werden. Die *constraint-basierte Objektmodellierung* ist derzeit ein vielversprechender Ansatz.

11.1 Zusammenfassung der Ergebnisse

Nach der Falsifikationsthese von Karl POPPER haben alle Aussagen (also auch die Theorien und Gesetze der Natur- und Technikwissenschaften) *hypothetischen* Charakter [Popper, 1963]. Der Wahrheitswert kann nicht bewiesen, sondern nur widerlegt (falsifiziert) werden. Die Thesen dieser Studie, wie im Exposé formuliert, wurden weder durch das gesichtete Material noch durch Projekterfahrungen im Soft- und Hardware-Entwurf widerlegt. Vielmehr wurde eine Reihe von Belegen und Argumenten identifiziert, die sich als Kernaussagen wie folgt darstellen (als Numerierung dienen die Kapitel der Studie):

[2] Der Trend zum *Codesign* verwischt zunehmend die Grenzen zwischen Soft- und Hardware-Entwurf. Die Konzepte zur Komplexitätsbewältigung sind sicherlich gleich: Abstraktion, Modularisierung, Klassifikation, Komposition und Generalisierung. Als VLSI-spezifisch gilt vor allem das Konsistenzproblem der Datenhaltung (Non-Standard-Datenbanken), der Datenaustauschformate und -schnittstellen.

I: Das Problem: Komplexität versus Integrität

1. Komplexität ist keine isolierbare Eigenschaft des zu entwerfenden Objekts. Als *deskriptiver Pluralismus* stellt sie vielmehr eine zentrale Entwurfsgröße dar. Die klassische Komplexitätslehre kennt zwar Verfahren zum Abschätzen der technischen Kostenfaktoren (Rechenzeit und Speicherbedarf der Software, Chipfläche und Signallaufzeit der Hardware), für die marktwirtschaftlich entscheidenden Faktoren, *Man-power* und *Time-to-Market*, stehen aber kaum vergleichbare Metriken zur Verfügung.
2. Reduktionistische Konzepte und Methoden zur Bewältigung der Komplexität sind seit langem bekannt. Autoren der Soft- und Hardwaretechnik zitieren aber nur selten ihre Quellen: Ungenannt bleiben zum Beispiel George MILLERs Erkenntnis zur kognitiven Verarbeitungskapazität des Menschen, Herbert SIMONs und Albert ANDOs Theoreme zur Architektur und Zerlegung der Komplexität oder David PARNAS' Konzept des Information hiding. Zwar wendet der Entwerfer reduktionistische Methoden meist intuitiv an, eine systematische Entwurfsmethode aber — das gilt für Soft- und Hardware gleichermaßen — sollte ihre grundlegenden Prämissen stets offenlegen.
3. Ein pragmatisches Konzept, Komplexität zu reduzieren, heißt *CAD-Framework*: Standardisierte Infrastrukturen für Entwurfswerkzeuge, die auf einer gemeinsamen Datenbasis operieren, versprechen „Integrität durch Integration". Das Konzept linearisiert den Aufwand für die Datenkonvertierung in heterogenen Entwurfsumgebungen von $O(n^2)$ auf $O(n)$ (in Dateisystemen mit n Orten, die sowohl Datenquellen als auch Datensenken sind). Die redundanzfreie Speicherung und Verwaltung der Entwurfsdaten sowie der schematisierte und konfliktfreie Datenzugriff der Werkzeuge sind die operativen Hauptforderungen an neue Entwurfssysteme.
4. Der aktuelle Paradigmenwechsel in der Informatik heißt „SA/SD nach OOx". Nach dem Struktur-Paradigma, das mit strukturierten Methoden der Analyse, des Entwurfs und der Programmierung den Ausweg aus der Softwarekrise der 60er Jahre weisen sollte, weist derzeit das Objekt-Paradigma den Ausweg aus der Wiederverwendungskrise. Der objektorientierte Ansatz, um die Entwurfskomplexität zu reduzieren und integre Datenobjekte zu entwerfen, führt auf die fachübergreifende Methode des *konzeptionellen Modellierens*. Objektorientierte „Weltmodelle" sind die Leitmetaphern im komplexen Entwurfsprozeß.

II: Objektorientierung als methodischer Ansatz zur Integritätssicherung

5. Das Curriculum einer „Wissenschaft des Entwerfens" im Sinne von Herbert SIMON und Heinz ZEMANEK ist ein zentrales fachübergreifendes Thema in der Informatik. Das Objekt-Paradigma, verstanden als *disziplinäre Matrix*, als die Gesamtheit dessen, was die (fiktive) Gemeinschaft der Entwerfer an kognitiven

und sozialen Elementen verbindet, vereint die entwurfstheoretischen Leitideen aus der Künstlichen Intelligenz, der Architektur und der ontologischen Philosophie.

6. Konventionelle Entwurfsmodelle haben ein kognitives Manko: sie sind streng systemorientiert. Informationstheoretische und verfahrenstechnische Modellierungsansätze lassen die denkpsychologische Disposition des Entwerfers außer acht. Die Integrität des zu entwerfenden Produkts ist folglich systembestimmt. Integritätsbedingungen, die bereits in der frühesten Phase des Entwerfens erkannt werden, in der kognitiven Wahrnehmung, gehen für die weitere informationstechnische Verarbeitung verloren. Da durch die Wahl der Methode (= des Weges zum Ziel) die meisten semantischen Integritätsverletzungen *a priori* ausgeschlossen werden können, sind ganzheitliche, psychologisch fundierte Methoden vorzuziehen.

7. Kognitionspsychologische Belange des Entwerfens sind ein zentrales Thema des Objekt-Paradigmas. Dessen Intuitivität geht darauf zurück: Die ureigenen Denk- und Handlungsmuster des Menschen orientieren sich am Objekt. Zwei Faktoren bewirken die intuitive Akzeptanz objektorientierter Konzepte und Methoden, der eine bezieht sich auf Erfahrung, der andere auf Erkenntnis: Der *integrative* Faktor vereint die „fundamentalen Ideen“ der Informatik. Der *kognitive* Faktor erschließt die denkpsychologische Begriffsbildung. Hier stößt man auf Strukturen und Operationen, die sich auch in den technischen Phasen des Problemlösens bewährt haben. Diese zweifache Motivation — kognitiv und pragmatisch — fördert entscheidend die Kreativität des Entwerfers.

8. Objektorientierte VLSI-Weltmodelle sind technischer Ausdruck der kognitiv-pragmatischen Bedeutung des Objekt-Paradigmas. Die Informationsmodellierung für datenintensive Anwendungen orientiert sich am Datenobjekt. Die VLSI-Spezifika lassen sich eindeutig und effizient im Sinne einer methodischen Integritätssicherung auf ein Objektmodell abbilden. Deskriptive und prozedurale VLSI-Schnittstellen — also Datenaustauschformate und Werkzeuganbindungen — lassen sich mit Hilfe des Objektmodells vereinfachen und vereinheitlichen.

III: Objektorientierung als operativer Ansatz zur Integritätssicherung

9. Entwerfen ist kein Optimierungsprozeß. Das allgemeine Entwurfsproblem, *ill-defined* und *ill-structured* [Simon, 1982], läßt sich nicht algorithmisch lösen. Entwerfen ist ein zielgerichteter Prozeß, dessen Zielkorridor durch *Constraints* eingeengt wird: selbstbestimmte durch Ratio, Intuition und Kreativität und fremdbestimmte durch technische und wirtschaftliche Vorgaben. Das *constraint-basierte* Entwerfen ist ein kreativ-evolutionäres Anpassen zwischen „Form & Kontext“, zwischen Problem und Lösung. Nicht die optimale, sondern die *zufriedenstellende* Anpassung ist das Ziel.

10. Der softwaretechnische Objektbegriff — die Kapselung von Struktur und Verhalten — läßt sich *ontologisch* stimmig und *pragmatisch* zugleich um den Constraint-Aspekt erweitern: zum *Tripel des Objekts*. Die KI-Konzepte der Wissensrepräsentation und der Constraint-Propagierung entfalten im Verbund mit den objektorientierten Konzepten ihre Synergie: Objektorientierte Constraint-Modelle sind Ausdruck dieser Synergie. Die Constraint-Modelle bieten den operativen Rahmen zur Integritätssicherung großer Datenbestände.

11.2 Einordnung der Ergebnisse

Karl KLINGSHEIM hat in seiner Abhandlung über *VLSI Design Theory* ein Entwurfsmodell vorgeschlagen, das sich für die Einordnung unserer Ergebnisse gut eignet [Klingsheim, 1990, S. 31]. Das Bild 11.2 zeigt eine erweiterte Darstellung.

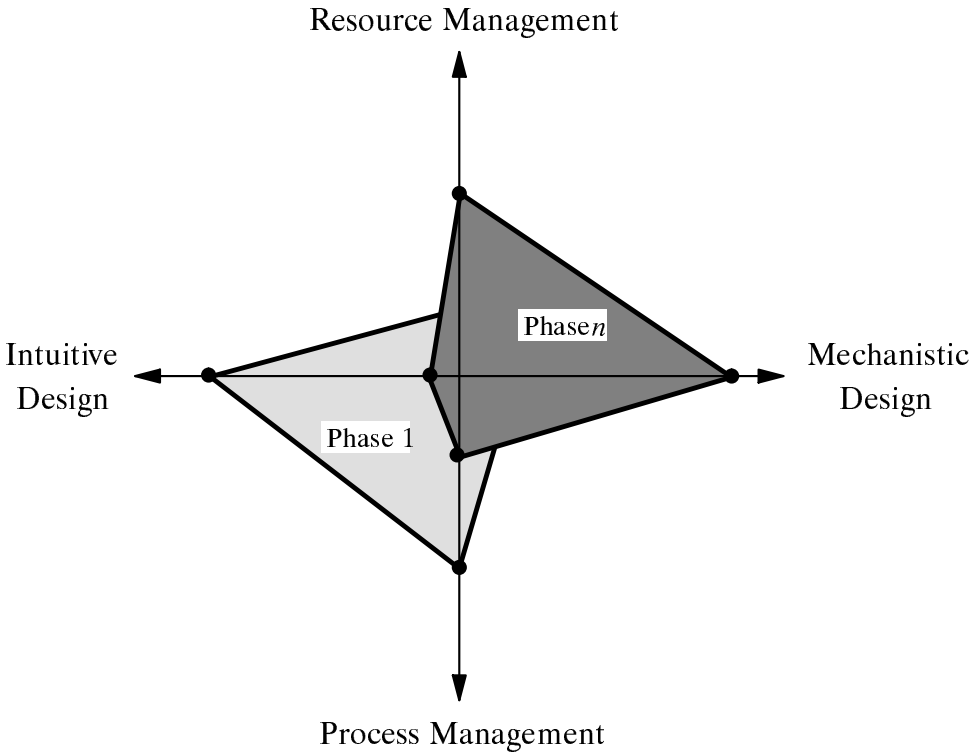


Bild 11.2: „A Universal Model of VLSI Design“ nach KLINGSHEIM

Das Modell beschreibt die Verteilung der Aktivität während der Systementwicklung. Gezeigt werden zwei Augenblicksaufnahmen eines Projekts: Das Viereck der Phase 1 kenn-

zeichnet die Aktivitätsverteilung in einer frühen, das Viereck der Phase n in einer späten Phase des Projekts. Während der gesamten Projektdauer verläuft der Schwerpunkt der Verteilung von unten links im Bild 11.2 nach oben rechts, und zwar *nichtmonoton* im Sinne eines beiderseitigen Anpassens zwischen Problem und Lösung. Form und Schwerpunkt der aufgespannten Flächen charakterisieren die aktuellen Anforderungen an die entwurfs- und verwaltungstechnischen Fähigkeiten. Die Tabelle 11.1 faßt die verschiedenen Aspekte der Projektaktivität zusammen. Ergänzend ordnen wir hier die Kapitel der Arbeit ein. Ihre Einordnung verdeutlicht die thematische Ausrichtung auf die beiden Entwurfsaspekte.

Activity	*Issues & Concerns*	*Focus*	*behandelt in*
Intuitive Design	creativity and innovations; artistic and subjective design; conjectures based on experience	ideas and concepts; high-level design; initial constructs	Kapitel 4, 5, 7, 9, 10
Mechanistic Design	data transformations; procedural and algorithmic design; deterministic and causal	explicit design data; data formats; netlists & layout	Kapitel 3, 6, 8
Process Management	designing the design process; communication within the team; design process quality assurance	designer's prosperity; the design *process*; quality control	
Resource Management	organization and resource allocation; schedules and financial constraints; external interface and legal issues	resources of all kinds; planning & scheduling; budgets & cost	

Tabelle 11.1: Projektaspekte im VLSI-Entwurf nach KLINGSHEIM

Die horizontale Dimension des Universalmodells beschreibt das Kontinuum der Entwurfsaktivität, die Spanne zwischen den funktionalen Systemanforderungen bis zum simulierten Layout: *from artistic creativity to shuffling polygons* [Klingsheim, 1990, S. 32]. Die vertikale Dimension spiegelt die Aktivitätsspanne des Projektmanagement. In ihr finden sich die *externen* Beschränkungen des Entwurfsprozesses wieder: Projektorganisation und Budgetplanung, Logistik, Qualitätskontrolle und mehr. Der *Time-to-Market*-Faktor korreliert beide Dimensionen: Gemeinsam entscheiden sie über den Erfolg eines industriellen Projekts. Karl KLINGSHEIM betont die ganzheitliche Sicht auf den Entwurfsprozeß:

> „Intuitive Design and Process Management depend heavily on the ingenuity of the designers involved, while Resource Management and especially Mechanistic Design are more ‚procedural' and methodical. Traditionally, research and development in VLSI systems design have focused almost exclusively on the *mechanistic* part of the process, including schematic capture, logic synthesis, simulation, routing, polygon manipulation, etc., because this is the part of the process that is most easily defined and comprehended. Design has been seen as largely an *analytical* task, requiring objective observation and data collection. We maintain that this approach is no longer sufficient, and that also the *intuitive* aspects of designing, involving creativity and subjective judgement must be accounted for in models of design processes." [Klingsheim, 1990, S. 32]

Die Managementaspekte werden in dieser Arbeit nicht explizit behandelt, obwohl auch die wirtschaftlichen und organisatorischen Vorzüge des Objekt-Paradigmas immer mehr erkannt und industriell genutzt werden. Gerade der hohe Grad an Modularität prädestiniert klassenbasierte Softwarebausteine für die industrielle Wiederverwendung im großen Stil: *Megaprogramming* [Boehm & Scherlis, 1987; Johnson *et al.*, 1992]. Diese Aspekte waren in einem früheren Entwurf der Arbeit integriert, wurden dann aber ausgegliedert[3], um so konzentriert die Objekt-Parallelen zur horizontalen Dimension in Bild 11.2 (und hier besonders in Richtung des intuitiven Entwerfens) herauszuarbeiten. Hier liegt, wie KLINGSHEIM schon bemerkte, ein Defizit in der konventionellen VLSI-Entwurfsmodellierung. Für die weitere Forschung zum Objekt-Paradigma ist es sicherlich geboten, beide Dimensionen eines VLSI-Projekts — Entwerfen und Verwalten — gemeinsam zu betrachten.

11.3 Weiterführende Forschung: „Informatik-Systemtechnik“

Um die „Defizite im Software-Engineering“ zu überwinden, bedarf es der Einsicht, daß „die Probleme des Software-Engineering nicht gelöst werden können, solange sie nicht als Probleme des Systems-Engineering erkannt sind“ [Wendt, 1993, S. 34]. In den letzten Jahren wurden international Stimmen laut, die Kenntnis- und Ausbildungsdefizite in der Systemanalyse und der Systemmodellierung nachhaltig zu kompensieren. Eine neue Ingenieurdisziplin wird gefordert, die sich der Komplexitätsbeherrschung diesseits und jenseits des gegenwärtigen Soft- und Hardware-Entwurfs annimmt. Die Förderung des „Systemdenkens“ unter Informatikern und Ingenieuren steht im Mittelpunkt [Müller, 1993]. Für die wissenschaftliche Fundierung der geforderten „Informatik-Systemtechnik“ [Schweizer & Thomé, 1993] wird dem Objekt-Paradigma die zentrale Rolle zugeschrieben: Es ist eine problemadäquate *Philosophie* und zugleich das *Instrumentarium*, um „Informatiksysteme“ konsistent und wirtschaftlich zu entwickeln. Als Ausblick unserer Studie skizzieren wir den Forschungsbezug zur neuen Ingenieurdisziplin.

Die IEEE-Initiative ECBS

Moderne rechnergestützte Anlagen und Geräte sind komplexe und heterogene Gebilde: Sie sind komplex, da sie sich aus verteilten und parallelen Systemen zusammensetzen, die wiederum verteilte Systeme sein können. Sie sind heterogen, da sie soft- und hardwaretechnische Komponenten umfassen, wie Rechner und Peripherie, Datenbanken, Benutzungsschnittstellen, Sensoren und Aktoren. Die Systemteile können geographisch verteilt

[3] Das gesamte Bedeutungsspektrum objektorientierter Konzepte, Methoden, Modelle und Techniken wird in [Quibeldey-Cirkel, 1994c] als ETHOS-Präsentation dargestellt (siehe die Kurzfassung auf Seite 85).

oder lokal eng gekoppelt sein. Einige Beispiele mögen diese Systemklasse verdeutlichen: integrierte Fertigungssteuerung, Leittechnik für Kraftwerke, Flugleit- und Telekommunikationssysteme. Für diese Klasse rechnergestützter Systeme wird der Begriff „Informatiksysteme“ vorgeschlagen, im Englischen CBS: *Computer-Based Systems* [Lawson, 1990]. Die Ingenieurdisziplin, die sich mit der Konzeption und Entwicklung von Informatiksystemen befaßt, heißt „Informatik-Systemtechnik“, englisch ECBS: *Engineering of Computer-Based Systems* [Thomé, 1993; White, 1993]. Bild 11.3 skizziert ihre Einordnung. Demnach ist die Informatik-Systemtechnik eine Teildisziplin der allgemeinen Systemtechnik (*Systems-Engineering*). Sie integriert die Methoden, Werkzeuge und Produkte etablierter Ingenieurtechniken, um rechnergestützte Multisysteme zu entwerfen. Diese steuern Anlagen und Geräte oder sind in Geräten und Anlagen eingebettet.

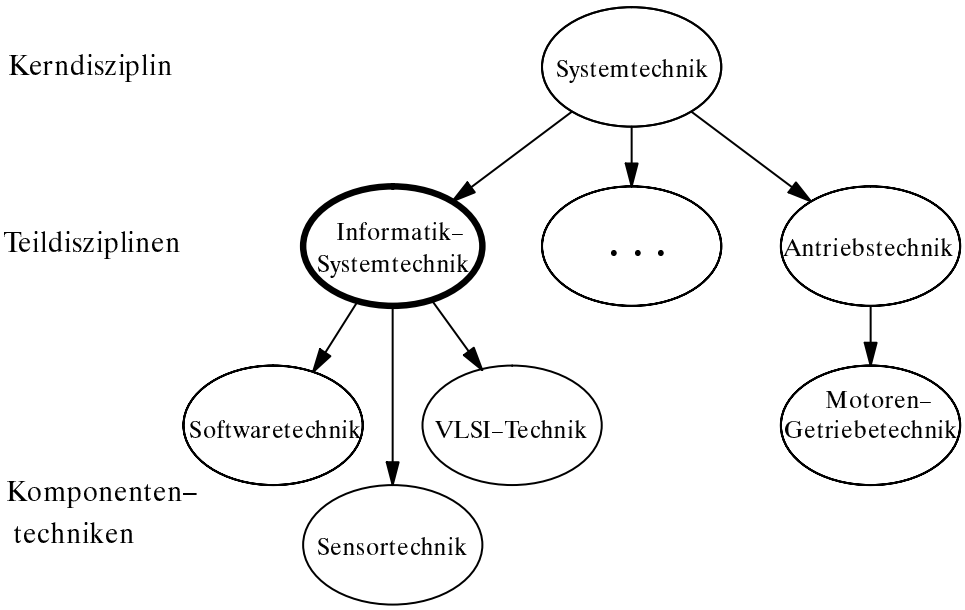


Bild 11.3: Einordnung der Informatik-Systemtechnik

Die Komplexität und Heterogenität der Informatiksysteme erfordert übergeordnete und ausdrucksmächtige Modelle, effiziente Methoden und Werkzeuge. Hier sehen wir die zukünftige Hauptanwendung der objektorientierten Systemmodellierung, und zwar aus zwei Gründen: Erstens, die Struktur- und Verhaltensmerkmale der Informatiksysteme sind konzeptionell und methodisch durch das Objekt-Paradigma erfaßbar und in Modelle für den Rechner umsetzbar.[4] Zweitens, das Problem der heterogenen Fachsprachen und Kom-

[4] Auf dem ersten ECBS-Workshop 1991 wurden die Struktur- und Verhaltensmerkmale von Informatiksystemen diskutiert und ein erster Konsens gefunden [Agrawala *et al.*, 1991]. Die Merkmale decken sich im wesentlichen mit denen der objektorientierten Client-Server-Architektur, wie sie von der OMG vorgeschlagen wird [Soley (Hrsg.), 1992; Rösch, 1993].

munikationsschnittstellen — das *Turm-zu-Babel*-Syndrom — ist durch die objektorientierte Terminologie überwindbar. Wir wollen den Forschungsbedarf in dieser Aussage motivieren.

Objektorientierung als eine Philosophie der Informatik-Systemtechnik

Der Pragmatiker Harold LAWSON hat den Anspruch der Softwaretechnik der späten 80er Jahre *ad absurdum* geführt. Seine fundamentale Kritik belegt er anhand der Ursachenanalyse gescheiterter Großprojekte.

> „In recent years, attention has focused on software engineering — especially its computer-aided aspects — as a ‚cure‘ for the life-cycle problems of complex computer-based systems. Unfortunately, this emphasis is like placing the cart before the horse. Software engineering methods and tools are important, but they should be the natural result of a well-developed philosophy for solving the application domain.“ [Lawson, 1990, S. 52]

Unter einer „Philosophie“ versteht LAWSON die gemeinschaftliche Sicht aller Mitglieder eines Projekts, wie ein Problem oder eine Klasse von Problemen *prinzipiell* angegangen werden sollte. Die Gemeinsamkeit der Sicht basiert auf den gemeinsamen Begriffen und Strategien aller Beteiligten: Auftragnehmer wie Auftraggeber und „all other parties with vested interests“. Die Methoden, Mechanismen und Werkzeuge einer ingenieurmäßigen Systementwicklung sollten einer *problemrelevanten* Philosophie entspringen; die meisten gescheiterten Projekte begannen ohne Philosophie — mit einer rühmlichen Ausnahme, die für unsere Arbeit Symbolcharakter hat: die Entwicklung von Simula. LAWSON, ein Skandinavier, erhellt die Philosophie dieser skandinavischen Entwicklung:

> „This programming language was designed in the early 1960s as a result of a philosophy for simulating industrial work environments and social systems. DAHL, MYHRHAUG, and NYGAARD, the Simula founders, later assisted Norwegian labor unions in gaining insight into various production models. They evolved concepts they felt were important for programming industrial work environment simulators, including the notions of classes, subclasses, objects with attributes, and inheritance. These concepts became the heart of the Simula philosophy, which lives on among avid Simula users and is perpetuated by the Simula User‘s Group.“ [Lawson, 1990, S. 53]

LAWSON hat drei Grundeinstellungen für den erfolgreichen Entwurf von Informatiksystemen identifiziert, die seinem Begriff von Philosophie genügen [Lawson, 1990]:

- *Management of Complexity*

  Das Entwurfsproblem und die zu entwerfenden Artefakte werden als kompliziert eingestuft. Ein Projektmanagement wird eingerichtet, um die Komplexität zu bewälti-

gen. Neue Artefakte sind erforderlich, um die zu entwerfenden Artefakte zu beherrschen (CASE-Werkzeuge zum Beispiel).

- *Management of Formalism*

  Linguistische Notationen und ihre Semantik werden eingesetzt, um das zu entwerfende System zu spezifizieren und das entworfene System zu verifizieren.

- *Management of Creativity*

  Der Projekterfolg hängt von kreativen und visionären Entwerfern ab. Nur sie sind fähig, die Philosophie im Projekt zu etablieren und verständlich zu machen. Das Projektmanagement sollte diesen Schlüsselpersonen vertrauen und ihnen Verantwortung übertragen.

Es ist die Objektorientierung, die LAWSON für eine geeignete Philosophie hält. Sie wird den drei Grundeinstellungen gerecht und kann die Entwicklung von Informatiksystemen leiten. Fördert man die Kommunikation und das Verständnis aller Projektbeteiligten, werden auch alle Phasen im Lebenszyklus eines Informatiksystems durchgängig unterstützt. Die *gemeinsame* Philosophie ist der wichtigste Aspekt im Entwurf. Artefakte, wie Unterstützungswerkzeuge und Programmiersprachen sind wichtig, aber sekundär. Auch die IEEE-Task-Force zur ECBS-Initiative kommt zu dieser Auffassung: Eine ECBS-Philosophie muß primär die Kommunikation und das Verständnis im Projektteam fördern; hierzu bedarf es einer gemeinsamen *intuitiven* Terminologie, um die *holistic properties of CBS* zu erfassen. Eine derartige Philosophie ist auch für die Ingenieurausbildung bedeutsam:

> „The basic scientific knowledge required covers many scientific domains, which a single individual cannot grasp in depth. Hence, a proper balance has to be found in curricula to provide adequate knowledge in each. In some cases, finding unifying concepts could possibly reduce the learning load, and this is certainly an object for ECBS research." [Agrawala *et al.*, 1991, S. 161]

Wie die Studie aspekthaft für den Software- und VLSI-Entwurf gezeigt hat (beides sind Komponententechniken der Informatik-Systemtechnik, siehe Bild 11.3), verfügt das Objekt-Paradigma über die *unifying concepts*, um Informatiksysteme ganzheitlich zu verstehen, zu spezifizieren und zu implementieren.

Die *kommunikative* Bedeutung der Terminologie wollen wir nochmals unterstreichen: Die Vermengung der Begriffe ist umgangssprachlich und auch in der Informatik weitverbreitet. So spricht man fälschlicherweise von „Informationsverarbeitung", was aus der unzulässigen Gleichsetzung resultiert: „Daten = Information". Daten *codieren* aber nur eine verbale oder nonverbale Nachricht, und erst diese ist kommunikativer Ausdruck einer Information. Eine Information konstituiert sich auf der Wissensebene, Nachrichten konstituieren sich

auf der Kommunikationsebene und Daten auf der Maschinenebene. Den Zusammenhang zwischen Information (I), Nachricht (N) und Daten (D) zeigt Bild 11.4, angelehnt an [Barkow *et al.*, 1994a, S. 42].

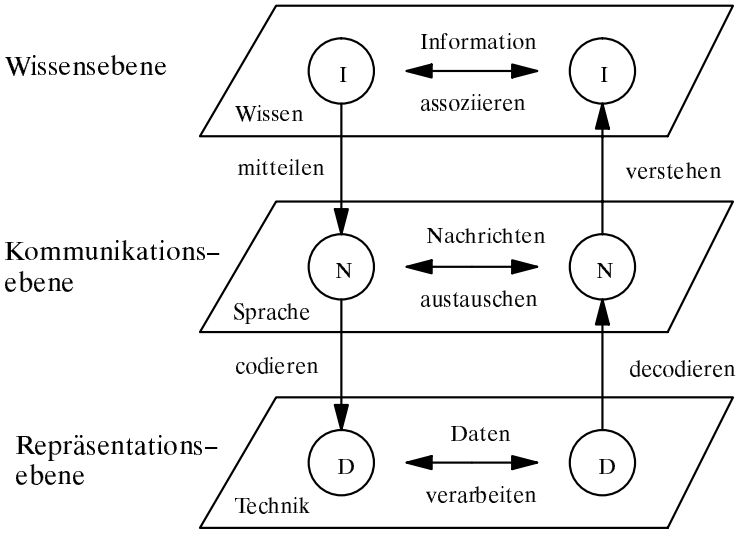


Bild 11.4: Zum Umfeld einer durchgängigen Terminologie

Die Abbildungen zwischen den Ebenen wirken im allgemeinen verkürzend, sind redundant und mehrdeutig. Dies um so mehr, wenn Personen verschiedener Fachgruppen Wissensaspekte eines Diskursbereichs austauschen und dabei auf verschiedenen Sprachebenen kommunizieren und verschiedene technische Repräsentationen verwenden. Will man also komplexe und heterogene Anwendungssysteme gemeinsam analysieren und modellieren, bedarf es eines einheitlichen Begriffssystems (in unserem Sprachgebrauch eines „Weltmodells“). Die Studie hat gezeigt, daß gerade die objektorientierten Begriffe, wie Klassen, Objekte, Nachrichten, Generalisierung und Komposition, die gewünschte Kongruenz zwischen den Ebenen sichern.

Analog zur konventionellen Terminologie der Softwaretechnik [Barkow *et al.*, 1994a; Barkow *et al.*, 1994b], wäre eine *objektzentrierte* Terminologie für Informatiksysteme und deren Technik geboten. Die konsistente und systematische Begriffsbildung ist ein wissenschaftlicher Anspruch einer jeden Disziplin. Im Fall der Informatik-Systemtechnik kommt die *Konsolidierung* hinzu. Hier stoßen die Begriffswelten verschiedener Fachgruppen aufeinander: zum einen bei der *perspektivischen* Analyse und Modellierung der Anwendung und zum anderen bei der Übernahme der Fachkonzepte in die Fachsprachen der Entwicklungsabteilungen. Eine objektzentrierte Terminologie könnte die Abteilungsgrenzen durchgängig halten: ohne Sprach-, Methoden- und Strukturbrüche. Eine konsolidierte Begriffssystematik des Objekt-Paradigmas steht aber noch aus.

Anhang A

Metaphorik der Objektorientierung

Der bildhafte, sinnfällige Ausdruck ist nicht nur ein Stilmittel der Rhetorik, ohne ihn kämen wir im sprachlichen Alltag nur mühsam zurecht. Mehr noch: wir sind uns kaum mehr der Metaphern in unserer Alltagsrede bewußt. Längst ist der sprachliche Alltag auch in die Informatik eingekehrt: Wir tragen uns ein in das *Logbuch* unseres Rechners (*log in*) und geben ihm unser *Kennwort*. Läßt er uns *passieren*, beginnen wir eine *Sitzung*, in der wir *Fenster öffnen* und *schließen*, per *Mausklick* ein *Menü* wählen, wie in einer Speisekarte (englisch: *menu*), und schließlich *loggen* wir uns wieder aus. Warum nur dieser allseits verbreitete Metaphern-*Salat* aus Allegorie und Periphrase, von den Katachresen in der unbedarften Sprache des Informatikers ganz zu schweigen?[1]

A.1 Metapher: Sprechblume oder Bedeutungssprung?

Wolf SCHNEIDER bringt die Bedeutung der Metapher auf den Punkt:

> „Springt hier eine Bedeutung über auf ein Ding, das bisher keinen Namen hatte — oder ist die Metapher lediglich eine schmückende Dreingabe, ein Sprachzierat, eine Sprechblume (Jean PAUL)? Sprechblumen sind manchmal schön, aber Bedeutungssprünge sind großartig. Sie waren, sie sind die typische, oft die einzig mögliche Art, Entdeckungen, Erfindungen, neue Einsichten, unvermutete Entwicklungen sprachlich zu bewältigen oder Stimmungen und Ahnungen zu Begriffen zu verdichten. Das eben heißt ‚Metapher': Übertragung — die vom Alten zum Neuen, vom Konkreten zum Abstrakten, vom Grobschlächtigen zum Filigran." [Schneider, 1988, S. 239]

[1] Allegorie meint die Metapher, wenn sie einen abstrakten Begriff durch eine Sache oder ein Lebewesen ausdrückt: „Fenster" oder „Maus" in unserem Beispiel. Periphrase meint die Metapher, wenn sie Einfaches kompliziert umschreibt: „eine Sitzung beginnen" statt einfach „anfangen". Katachrese meint die *entgleiste* Metapher, zum Beispiel das weitverbreitete „Debuggen": Grace HOPPER, auf der Fehlersuche in einem Mark-II-Rechner 1947, fand eine tote Motte und entfernte sie mit der Pinzette. Das „Entwanzen" stand fortan „Pate" (was wiederum ein schiefes Bild ist) für das Ausmerzen von Programmierfehlern [Weiser Friedman, 1992, S. 4].

Greifen wir gleich zu Beginn die wichtigste Metapher des Objekt-Paradigmas auf, die *Vererbung*: „*Vererben* war jahrtausendelang ein juristischer Begriff, ehe der Lehrer Gregor MENDEL aus Brünn ihn als Metapher auf die von ihm entdeckten Gesetze — nun, der ‚Vererbung' übertrug, es gibt kein Synonym" [Schneider, 1988, S. 239]. Charles DARWIN erhob den Vererbungsbegriff zum Ordnungsprinzip der Arten, und die Informatik läßt die Bedeutung noch weiter springen: auf unbeseelte Dinge: Datenstrukturen und Algorithmen. Das ist, um mit SCHNEIDER zu sprechen, das „Großartige" dieser Metapher.

Eine Metapher, so ihre Grenzen fließend sind, kann natürlich auch überbeansprucht werden: Sie rastet ein und verliert ihre Kraft (*Topos/Stereotyp*), oder ihre Grenzen sind so weitläufig, daß ihre Assoziationen in die Irre führen oder schlichtweg mehr verwirren als klären. Die objektorientierten Metaphern machen da keine Ausnahme: So weist James RUMBAUGH auf den Mißbrauch des Vererbungskonzepts hin, dem selbst in der einschlägigen Literatur gefrönt wird: Viele Verfasser unterscheiden nicht deutlich zwischen *Vererbung* und *Aggregation*, da die eingerastete Metapher der Vererbung den objektorientierten Entwurf über Gebühr beherrscht [Rumbaugh, 1993].[2] Alan SNYDER meidet die Metapher *Kapselung* wegen ihrer Weitschweifigkeit und schließt sie deshalb aus dem Metaphern-Katalog objektorientierter Begriffe aus [Snyder, 1993].

Unbotmäßige Übertragungen aus dem Englischen tun ein übriges: „Geheimnisprinzip" für „Information hiding" verleiht der pragmatischen Trennung der Belange die Aura der Verschwiegenheit, obwohl in aller Regel in die Implementierung eines Objekts hineingeschaut werden kann, und dies zu verhindern, auch gar nicht beabsichtigt ist. Oder der Anglizismus „Entität" (*entity*), was schlicht Ding, Einheit oder halt Objekt heißt. Entität als „Dasein im Unterschied zum Wesen eines Dinges" (Duden) zeigt zwar den akademischen Anspruch des Benutzers, vernebelt aber vermeidbar das objektorientierte Vokabular um eine weitere philosophische Andeutung.

Kommen wir zur eigentlichen Funktion der Metapher zurück: Was macht eine Metapher so nützlich? Die Antwort gibt uns die Psychologie des Lernens: Wir lernen, indem wir altes Wissen um neues bereichern. Wir entwickeln neue Begriffe über Assoziationen mit bereits bekannten Begriffen. Ein Beispiel, entnommen aus [Booth, 1989, S. 75], verdeutlicht die Rolle der Metapher im Lernprozeß: Wenn wir den Begriff des „Elektrons" lehren oder lernen wollen, können wir drei verschiedene Metaphern bemühen: (a) Ein Elektron verhält sich wie eine „Welle", da es gewisse Eigenschaften mit einer elektromagnetischen Welle, zum Beispiel dem Licht, teilt. (b) Ein Elektron verhält sich wie ein „Partikel" oder „Korpus", da es eine Masse besitzt. (c) Ein Elektron verhält sich wie eine „Wolke", da sich die Wahrscheinlichkeit seines Aufenthalts nicht exakt berechnen läßt. Indem wir also Eigenschaften wiedererkennen, die ein Elektron mit bereits verstandenen Begriffen gemein-

[2] Eine einfache Regel, um den Mißbrauch zu vermeiden, lautet: „Only use inheritance when you want to inherit *all* the properties of a superclass, not just some of them. Use aggregation when one object is made of other objects and *some* properties of the aggregate are shared by the parts" [Rumbaugh, 1993, S. 22].

sam hat, bilden wir einen neuen Begriff auf der Grundlage unseres vorhandenen Wissens. Metaphern haben also eine *Brückenfunktion* im Lernprozeß: Sie machen komplexe Begriffe über die Assoziation mit einfachen Begriffen verständlich. Paul BOOTH:

> „In essence, metaphors enable learning, they provide short-cuts to understanding complex concepts; they can be used to shape user's behaviour in circumstances that are unfamiliar and that they might otherwise find confusing." [Booth, 1989, S. 75]

A.2 Metaphern der Objektorientierung

Eine Metapher hat wie keine andere die objektorientierten grafischen Benutzungsoberfächen geprägt: die Allegorie vom „Schreibtisch". Sie war ein verkanntes Nebenprodukt der Xerox-Forschung im Palo-Alto-Research-Center (PARC), bevor sie Steven JOBS zum Leitmotiv der Apple-Macintosh-Rechner stilisierte [Young, 1988]. Heute ist sie überall verbreitet im namensgleichen Desktop-Publishing. Der Benutzer wird ermutigt, sein Allgemeinwissen intuitiv einzusetzen: *Büro-Utensilien*, wie Dateikarten, Ordner, Ablagen, und *Büroarbeit*, wie Dokumente erstellen, formatieren, drucken und kopieren. Sein kognitives Modell der Bürowelt ist kongruent mit den Objekten und Diensten, die ihm sein elektronischer Schreibtisch suggeriert.

Nehmen wir zum Beispiel die allegorische Mächtigkeit des „Papierkorbs" innerhalb der Schreibtisch-Metapher. Der Papierkorb drückt eine abstrakte Handlung durch eine konkrete Sache aus: „Entsorge den selektierten Datenmüll vorläufig (bäuchiger Papierkorb auf einem Macintosh) und endgültig, wenn ich es will (Menü-Option ‚Papierkorb entleeren ...')." Hierfür wäre, wie die leidliche Erfahrung mit anweisungsgesteuerten Betriebssystemen lehrt (*prompting*), einiges syntaktisches Wissen erforderlich.

Die Tabelle A.1 nennt die Metaphern, die das Objekt-Paradigma verständlich und populär gemacht haben. Die wichtigsten erklären und versinnbildlichen Konzepte, die auf mehreren Beschreibungsebenen der Produktentwicklung gültig sind (Analyse, Design, Programmierung). Das trägt maßgeblich zur Durchgängigkeit der objektorientierten Methode bei. Die Intuitivität der objektorientierten Konzepte geht auf wenige Leitmetaphern zurück: man kann sie verstehen, ohne sie zu reflektieren.

Metapher	*Erklärungsbezug*	*Beschreibungsebene*
Vererbung, Erblinien, Erben, Erbgut	Struktur-, Teilungsmechanismus	alle Phasen
Objektwelt, object society	Anordnung von Modulen	alle Phasen
Klassen, Objekte, Bausteine	Module	alle Phasen
Client-Server-Vertrag	Modulspezifikation	Design
Pflichten, Rechte, Kooperationen	Vor- und Nachbedingungen	Design
Matrize, Schablone	Teilungsmechanismus	Design, Programmierung
Dienst, Service	Prozedur, Unterprogramm	Design, Programmierung
Bibliothek	konfektionierte Modulmenge	Design, Programmierung
Polymorphie	dynamische Modulbindung	Design, Programmierung
Object-Request-Broker	verteilte modulare Systeme	Design, Programmierung
Datenkapsel, Geheimnisprinzip	Datenschutz	Programmierung
Nachrichtenaustausch: Small-Talk	Modulinteraktion, Prozeduraufruf	Programmierung
Pronomen: self, me, this	formaler Modulparameter	Programmierung
Protokoll, Signatur:	Modulschnittstelle	Programmierung
public, protected, private, friend		
data member, member function, slot	Modulprogrammierung	Programmierung

Tabelle A.1: Objektorientierte Metaphern: eine Auswahl

Anhang B

Ontologische Konzepte

Der Objektbegriff ließe sich bis in die griechische Philosophie der Antike zurückverfolgen, zum Beispiel auf DEMOKRITs atomistische Weltsicht, in der unteilbare *Dinge* im Mittelpunkt des Interesses stehen. Einen neuzeitlichen philosophischen Bezug gibt Yair WAND:

> „Is the emergence of the object paradigm an *empirical indication* that humans find it easier to describe perceptions of the world through the notion of objects?“ [Wand, 1989a, S. 538]

WAND leitet ein Objektmodell aus den formalen Aussagen der Ontologie ab, der Wissenschaftsphilosophie von der Begriffs- und Ordnungsbestimmung der Dinge in der realen Welt.[1] Er stützt sein Objektmodell auf den philosophischen Formalismus von Mario BUNGE [Bunge, 1979]. Wir werden sein Objektmodell wegen der mengentheoretischen Strenge im Detail vorstellen. Eine ontologisch formale Auslegung des Objekt-Paradigmas hat zweifachen Nutzen:

- Zum einen kann die Implementierung von der Konzeption klar getrennt werden, was zur Klärung der Begriffe beiträgt. Technische Randbedingungen bleiben außen vor [Wand, 1989b]: Das Objekt wird nicht mehr aus der Perspektive der Programmierung betrachtet; dort ist es den technischen Belangen der Implementierung unterworfen. Der ontologische Objektbegriff verschiebt vielmehr die Perspektive auf die Modellierung eines Weltausschnitts, auf unsere Wahrnehmung der Wirklichkeit.
- Zum anderen unterstützt das so gewonnene Verständnis die Bewertung objektorientierter Analyse- und Designmethoden [Wand & Weber, 1989].[2] Quasi als Nebenprodukt begründet die philosophische Analogie den Anspruch des Objekt-Paradigmas: Begriffe und Konzepte der realen Welt sind auf die softwaretechnische Konstruktion direkt abbildbar, und zwar natürlich, schlüssig und der menschlichen Denkweise konform.

[1] *Konzeptionelle Dinge*, also solche, die sich nach der Perzeption der realen Dinge bilden, das heißt im kognitiven Prozeß der Apperzeption, sind in der weiteren Betrachtung ausgeschlossen. Es geht um die Modellierung der konkreten Dinge unserer Anschauung (*entities* oder *substantial individuals* in der Lesart von BUNGE).

[2] Dagmar KÖHLER verwendet den ontologischen Objektbegriff nach WAND, um den *Constraint*-Begriff in der grafischen Datenverarbeitung zu formalisieren [Köhler, 1992].

B.1 Die Begriffswelt der Ontologie

Vorangestellt seien die wichtigsten ontologischen Prinzipien nach BUNGE, gefiltert durch die Absicht, aus ihnen ein formales Objektmodell abzuleiten:[3]

- Die Welt besteht aus Dingen.
- Form und Gestalt sind Eigenschaften der Dinge.
- Dinge stehen in Wechselwirkung mit anderen Dingen.
- Jedes Ding verändert sich.
- Nichts entsteht aus dem Nichts und kein Ding wird zu Nichts.
- Jedes Ding richtet sich nach Gesetzen.

Strukturbegriffe

Die Grundstruktur in der ontologischen Weltsicht ist das *Ding* (*thing* oder *entity*). Ein Ding ist entweder elementar oder es setzt sich aus anderen Dingen zusammen. Man unterscheidet zwischen den *Eigenschaften* und den *Attributen* eines Dings: Jedes Ding besitzt Eigenschaften, aber nicht alle Eigenschaften sind für uns erkennbar. Wir nehmen nur solche wahr, denen wir zuvor mindestens ein Attribut zugeordnet haben. Ein Attribut ist ein „willkürliches Etikett", mit dem wir eine Eigenschaft bezeichnen. Eigenschaften existieren nicht für sich allein. Sie haften den Dingen an. Umgekehrt ist ein Ding keine Anhäufung von Eigenschaften.

Dinge sind elementar oder zusammengesetzt. Die Eigenschaften zusammengesetzter Dinge werden entweder auf die Komponenten *vererbt*, das heißt, es gibt mindestens eine Komponente mit diesen Eigenschaften, oder sie beziehen sich ausschließlich auf die Zusammensetzung, das heißt, es gibt keine isolierbaren Komponenten mit diesen Eigenschaften. Dinge sind *eindeutig*, im Sinne von *einzigartig*. Wenn wir dennoch zwei Dinge als identisch wahrnehmen, dann nur deshalb, weil wir nicht *allen* Eigenschaften unterscheidbare Attribute zugeordnet haben. Dinge mit „ähnlichen" Eigenschaften können zusammengefaßt werden zu *Klassen*.

[3] Zwei Anmerkungen: 1. Die Ontologie nach BUNGE versteht sich als formale Wissenschaft. Um die Darstellung überschaubar zu halten, verzichten wir auf die mengentheoretische Strenge, die BUNGE und WAND in ihren Postulaten, Definitionen, Theoremen und Lemmata betreiben. 2. Natürlich findet man in erster Linie nur das, was man auch sucht. So auch WAND, der selbstredend nur jene ontologischen Konzepte aufgreift, die auch eine tragfähige Analogie zur Objektorientierung versprechen.

Dinge werden als *Begriff* oder als *Modell* wahrgenommen und in einem *konzeptionellen Schema* formalisiert. Die Perspektive des Betrachters — sein subjektiver *Bezugsrahmen* (zum Beispiel als Hersteller, Vertreiber oder Käufer eines Produkts) — und ein *Funktionensatz* prägen dieses Schema. Der Funktionensatz selbst heißt *funktionales Schema*: $\underline{F} =< F_1, \ldots, F_n >$. Es umfaßt solche Funktionen, die sich auf die relevanten Eigenschaften eines Dings beziehen, indem jede Funktion F_i den Wert einer Eigenschaft zu einem gegebenen Zeitpunkt bestimmt. Jedes Ding kann als funktionales Schema modelliert werden. Weiterhin gilt, daß sich jedes Ding zu einem gegebenen Zeitpunkt in einem eindeutigen *Zustand* befindet. Der Zustand ist bestimmt durch die Werte der *Zustandsvariablen* und durch die *Zustandsfunktionen*. Welche Zustände ein Ding einnehmen kann, wird durch *Gesetze* festgelegt. Gesetze sind Eigenschaften der Dinge. Sie bilden die kombinatorisch möglichen Zustände, definiert durch das funktionale Schema, auf die Menge {„gesetzmäßig", „ungesetzmäßig"} ab.

Dynamikbegriffe

Das Konzept der Gesetze ist grundlegend: Gesetze stehen für das Wissen, was die Dinge sein oder was sie nicht sein können. Sie implizieren die „Dynamik" der Dinge. Vollständiges Wissen über ein Ding setzt die Kenntnis über seine möglichen (diskreten) Zustandsübergänge voraus. Ein Ding ändert sich, wenn sich der Wert einer Eigenschaft ändert. Da jedes Ding sich verändert und sich eine Änderung in einem Zustandswechsel äußert, muß folglich jedes Ding mehrere Zustände einnehmen können. Eine Zustandsänderung wird als *Ereignis* bezeichnet. Ein Ereignis wird als ein geordnetes Paar von Zuständen $< s_1, s_2 >$ beschrieben, wobei s_1 den Zustand vor und s_2 den Zustand nach der Änderung beschreibt. Betrachten wir den Zustandsraum $S(X)$, die Menge aller Zustände eines Dings X, dann ist die Menge aller vorstellbaren Ereignisse $E(X)$ das kartesische Produkt $S(X) \times S(X)$. Da nicht alle Zustände in der Realität erlaubt sind, ist die reale Ereignismenge eingeschränkt. Dies führt auf den Begriff des *gesetzmäßigen* Ereignisses. Gesetzmäßig ist ein Ereignis $< s_1, s_2 >$, wenn beim Übergang von X aus dem Zustand s_1 in den Zustand s_2 kein Gesetz von X verletzt wird, sich X somit immer in einem gesetzmäßigen Zustand befindet.

Das ontologische Prinzip „jedes Ding verändert sich" impliziert die Änderung über der Zeit. Hierfür werden „Linien" (*Trajektorien*) im Zustandsraum definiert (chronologisch geordnete Zustände), entlang derer sich der Werdegang eines Dings (die *Historie*) niederschlägt. Der Begriff der Historie erlaubt die Beschreibung wechselwirkender Dinge. Die Wechselwirkung (*Interaktion*) äußert sich in einer *Abweichung* in den Zustandslinien. Jedes Ding steht in irgend einer Wechselwirkung mit anderen Dingen: Es gibt keine unabhängigen Dinge. Das führt uns zurück auf das ontologische Grundprinzip: „Jedes Ding gehört einem System an".

Der Dualismus von „Ding und Ereignis" wird durch den ontologischen Formalismus von BUNGE aufgelöst:

> „A basic polarity in traditional metaphysics is that of being and becoming: event is opposed to thing, process to stuff, change to structure. This opposition makes no sense in our system, where every change is a transformation of some thing or other, and every thing is in flux. This view [...] is consistent with science: the latter provides no ground for hypothesizing the existence of thingless events anymore than it suggests that there might be changeless things." [Bunge, 1979, S. 273]

B.2 Ein ontologisches Objektmodell

Der Formalismus von BUNGE dient WAND für die Entwicklung eines Objektmodells. Dieses Modell soll möglichst viele Konzepte des Objekt-Paradigmas einfangen. Ausgeschlossen sind nur solche Aspekte, die sich auf die technische Implementierung beziehen. Hierin liegt ein wichtiger Erkenntnisgewinn der ontologischen Analogie.

Grundzüge des Objektmodells

- Statische Eigenschaften
  - Die Welt besteht aus Objekten.

    Objekte beschreiben konkrete Dinge im Gegensatz zu Typen oder Klassen, die generisch (= unspezifisch) Objekte beschreiben.[4]
  - Objekte sind bekannt (oder beobachtbar) durch ihre Eigenschaften.

    Eigenschaften sind entweder *Attribute* oder *Gesetze*. Mit Hilfe von Attributen werden Objekteigenschaften beobachtbar: Ein Attribut bildet eine Objektmenge mit dieser Eigenschaft auf einen gegebenen Wertebereich ab. Gesetze schränken die Kombinationen der Attributwerte ein. Die Darstellung einer Eigenschaft ist somit entweder eine *Funktion* (Wertzuweisung) oder ein *Prädikat*, das anzeigt, ob eine Zustandsbeschränkung eingehalten wird. Die Menge der Eigenschaften, die ein Objekt beschreiben, ist abhängig von der Sichtweise und der Modellierungsabsicht. Diese Menge heißt *funktionales Schema*. Die Werte eines Attributs zu einem bestimmten Zeitpunkt werden durch eine *Zustandsvariable* erfaßt. Die aktuellen Werte aller Zustandsvariablen definieren den *Zustand* eines Objekts. Gesetze bestimmen die *erlaubten* Objektzustände.

[4] Hinsichtlich der (subtilen) Unterscheidung zwischen einem Objekt*typ* und einer Objekt*klasse* sei auf [Danforth & Tomlinson, 1988] verwiesen.

  - Objekte sind elementar oder zusammengesetzt.

    Die Eigenschaften eines zusammengesetzten Objekts gehen entweder auf seine Komponenten über oder sie haften allein der Zusammensetzung an.

- Dynamische Eigenschaften

  - Objekte verweilen nur in gesetzmäßigen Zuständen.
  - Ein Objekt, das sich in einem ungesetzmäßigen Zustand befindet, wird in einen gesetzmäßigen wechseln. Der Wechsel wird eindeutig durch den ungesetzmäßigen Zustand bestimmt.
  - Objekte erreichen ungesetzmäßige Zustände nicht spontan.
  - Ein Objekt kann den Zustand eines anderen Objekts beeinflussen. Dies bewirken Gesetze, die Zustandsvariablen beider Objekte wertmäßig binden.

Formale Definitionen

Objekt-Interaktion: Die Menge aller möglichen Zustände eines Objekts i, seine Zustandsmenge, sei mit $S(i)$ (*states*) bezeichnet.

- Ein *Gesetz* L (*law*) ist eine Funktion auf der Menge von Zuständen.

  $L : S(i) \rightarrow S(i)$

  Wenn $L(s) = s$ gilt, heißt s *gesetzmäßiger* Zustand.[5]
  Wenn $L(s) \neq s$ gilt, heißt s *ungesetzmäßiger* Zustand.
  Die Menge aller gesetzmäßigen Zustände des Objekts i wird mit $S_L(i)$ bezeichnet.

- Seien i, j zwei Objekte. Ein *Schnittstellen-* oder *Interobjekt-Gesetz* (*assertion*)[6] ist eine Funktion:

  $A : S(i) \times S(j) \rightarrow S(i) \times S(j)$

  Im folgenden bezeichne s^k den Zustand des Objekts k. $A((s^i, s^j))$ sei verkürzt auf $(\mathbf{s}^i, \mathbf{s}^j)$.

  Wenn $(\mathbf{s}^i, \mathbf{s}^j) = (s^i, s^j)$ gilt, heißt der Interobjekt-Zustand *stabil*.
  Wenn $(\mathbf{s}^i, \mathbf{s}^j) \neq (s^i, s^j)$ gilt, heißt der Interobjekt-Zustand *instabil*.
  Wenn $\mathbf{s}^i \neq s^i$ gilt, beeinflußt Objekt j das Objekt i.
  Wenn $\mathbf{s}^j \neq s^j$ gilt, beeinflußt Objekt i das Objekt j.

[5] Ein gesetzmäßiger Zustand s ist somit invariant: ein *Fixpunkt* von L.

[6] Gesetze sind zusichernd (*assertional*), nichtprozedural, das heißt, sie sagen nichts über das *Wie* eines Zustandswechsels aus.

Objekt- und Interobjekt-Gesetze erfassen die Dynamik der Objekte, da sie die Information beinhalten, welche Zustandswechsel auftreten können, das heißt welche stabil und gesetzmäßig sind. Die Dynamik der Objekte läßt sich durch deren Zustandswechsel beschreiben. Derartige Wechsel heißen *Ereignisse*.

- Ein *Ereignis* wird als geordnetes Paar $e =< s_1, s_2 >$ notiert; s_1 ist der Zustand vor und s_2 der Zustand nach dem Wechsel.

  Da ein Gesetz die möglichen Zustandswechsel beschreibt, kann ein Gesetz auch explizit als Aufzählung der Ereignisse beschrieben werden, die auf ungesetzmäßige Zustände folgen (die gesetzmäßigen Zustände werden durch das Gesetz nicht verändert, sie sind Fixpunkte).

Mit diesen Begriffen können wir nun die Dynamik einer Objekt-Interaktion (diskret) betrachten: Ein Objektpaar i, j befinde sich in einem instabilen Zustand (s_1^i, s_1^j). Betrachten wir zuerst das Objekt i und unterstellen wir, es befinde sich in einem gesetzmäßigen Zustand $s_1^i \in S_L(i)$. Aufgrund einer Interaktion mit dem Objekt j, das heißt aufgrund eines Schnittstellengesetzes zwischen i und j, wechsle nun i seinen Zustand nach s_2^i. Falls s_2^i gesetzmäßig ist, wird nichts weiteres passieren. Ist aber s_2^i ungesetzmäßig: $s_2^i \notin S_L(i)$, dann wird i seinen Zustand in einen neuen Zustand s_3^i ändern, der konform ist mit seinen Gesetzen $\{L^i\}$. Die Interaktion des Objekts j mit i wird folglich als Folge zweier Ereignisse e_1, e_2 modelliert: $e_1 =< s_1^i, s_2^i >$ gemäß dem Schnittstellengesetz und $e_2 =< s_2^i, s_3^i >$ gemäß den Objektgesetzen. Gleiches gilt für den Zustand des Objekts j.

Klassenhierarchie und Vererbung: Angenommen X sei eine Menge von Objekten und jedem $x \in X$ sei eine Menge $p(x)$ zugeordnet, dessen Elemente die beobachtbaren Eigenschaften von x repräsentieren (Attribute und Gesetze). Wir bezeichnen mit $\mathbf{P}(X)$ die Menge aller Eigenschaften der Objekte aus X: $\mathbf{P}(X) = \bigcup\{p(x)|x \in X\}$. Mit 2^X bezeichnen wir die Potenzmenge von X, also die Menge aller Teilmengen von X.

- Der *Umfang* U einer Eigenschaft $P \in \mathbf{P}(X)$ ist die Teilmenge der Objekte mit der Eigenschaft P:

  $U(P; X) = \{x|x \in X \wedge P \in p(x)\} \in 2^X$

- Sei $\mathbf{p}$ eine Teilmenge von $\mathbf{P}(X)$. Eine *Klasse* C (*class*), bezogen auf $\mathbf{p}$ in X, ist die Menge aller Objekte, die alle Eigenschaften aus $\mathbf{p}$ besitzen:

  $C(\mathbf{p}; X) = \bigcap\{U(P; X)|P \in \mathbf{p}\}$.

  Lemma a: Seien $\mathbf{p}_1$, $\mathbf{p}_2$ zwei Teilmengen von $\mathbf{P}(X)$. Dann ist $\mathbf{p}_1$ eine (echte) Teilmenge von $\mathbf{p}_2$, wenn und nur wenn $C(\mathbf{p}_2; X)$ eine (echte) Teilmenge von $C(\mathbf{p}_1; X)$ ist.

Theorem a: Sei $2^{\mathbf{P}(X)}$ die Menge aller Teilmengen der Eigenschaften der Menge X. Dann ist die Menge $G = \{C(\mathbf{p}; X) | \mathbf{p} \in 2^{\mathbf{P}(X)}\}$ ein gerichteter azyklischer Graph, dessen Knoten in einer Teilmengenbeziehung (Inklusion) zueinander stehen.

Lemma a und Theorem a bilden die formale Grundlage der Begriffe „Klassenhierarchie" und „Vererbung von Eigenschaften". Um dies zu verdeutlichen, betrachten wir zwei Eigenschaftsmengen $\mathbf{p}_1$ und $\mathbf{p}_2$, wobei gilt $\mathbf{p}_1 \subset \mathbf{p}_2$. Wir bezeichnen alle Objekte der Eigenschaften $\mathbf{p}_i$ mit C_i. Jedes Objekt, das alle Eigenschaften $\mathbf{p}_2$ besitzt, besitzt auch alle Eigenschaften von $\mathbf{p}_1$, somit gilt $C_2 \subseteq C_1$. Betrachten wir nun eine Teilmenge von C_1: Alle Objekte dieser Teilmenge besitzen die Eigenschaften $\mathbf{p}_1$. Deshalb können wir sagen, daß C_2 eine Unterklasse von C_1 beziehungsweise C_1 eine Oberklasse von C_2 ist. Nehmen wir weiterhin eine Eigenschaftmenge $\mathbf{p}_3 \subset \mathbf{p}_2$ an. Dann ist C_3 auch eine Oberklasse von C_2. Somit impliziert die Definition von Klassen eine Hierarchie mit „Mehrfachvererbung". „Erben" bedeutet hier also das Teilen gemeinsamer Eigenschaften (Attribute und Gesetze). Die Metapher der Vererbung geht letztlich auf eine Teilmengenbeziehung zurück; darin liegt ihr Potential der Wiederverwendung (siehe Abschnitt 10.2.3). Wegen der Asymmetrie des Teilmengenoperators $\subset$ können Klassen also als gerichteter azyklischer Graph beschrieben werden, in dem die Existenz einer Kante eine Teilmengenbeziehung bedeutet.

Zusammengesetzte Objekte: Folgende Annahme gälte für zusammengesetzte Objekte: Objekte können paarweise durch einen Assoziationsoperator (bezeichnet mit „$\odot$") zu einem anderen Objekt kombiniert werden. Die Operation $\odot$ ist kommutativ und assoziativ. Auf diese Weise kann ein Objekt aus mehreren Objekten zusammengesetzt sein.

- Seien x und y Objekte, für die gälte: $x \odot y = x, x \neq y$. Dann ist y „Teil von" x (*part of*), geschrieben y p-o x.
- Die Zusammensetzung Z eines Objekts ist die Menge seiner Teile:
  $Z(x) = \{y | y \text{ p-o } x\}$

  Theorem b: Die Menge aller Objekte O ist unter der Relation „p-o" partiell geordnet.[7]
- Sei $P \in p(x)$ eine Eigenschaft des Objekts x. Dann ist P eine *vererbbare* Eigenschaft, wenn es ein Objekt $y \in Z(x)$ gibt mit $P \in p(y)$, $y \neq x$. Andernfalls ist P eine Eigenschaft der Zusammensetzung: für alle $y \in Z(x)$ und $y \neq x$ gilt $P \notin p(y)$.

[7] Genauer muß es heißen: Die Relation „p-o" ist eine *irreflexive* (strikte) partielle Ordnung. Für alle $a, b, c \in O$ gelten die Ordnungsaxiome: 1. $\neg(a \text{ p-o } a)$ (Irreflexivität), 2. a p-o $b \wedge b$ p-o $c \Rightarrow a$ p-o c (Transitivität) und 3. a p-o $b \Rightarrow \neg(b$ p-o $a)$ (Asymmetrie).

Fazit

Das Hauptanliegen des ontologischen Objektmodells nach WAND ist die Modellierung der Objekte *frei* von den Belangen der Implementierung. Es formalisiert die Grundprinzipien *Datenkapselung* und *Vererbung*, unterscheidet sich sonst aber signifikant von den in der Literatur vorgeschlagenen Definitionen zur Objektorientierung, zum Beispiel [Blair *et al.*, 1989; Meyer, 1988; Nierstrasz, 1989; Wegner, 1987]. Um eine Objekt-Interaktion zu beschreiben, verzichtet das Objektmodell auf den Begriff der *Objektoperation* (Smalltalk-Terminus: *Methode*) und auf das Kommunikationsmodell des *Nachrichtenaustausches*. Diese von vielen Verfassern als definitorisch eingestuften objektorientierten Konzepte sind *kein* Bestandteil eines ontologischen Objektmodells. WAND stuft sie als Mechanismen ein, die der Implementierung angehören. Um die Objektdynamik zu beschreiben, genügen *Gesetze*. Sie stellen neben der Datenkapselung und der Vererbung ein ontologisches Grundprinzip der Objektorientierung dar.

Weitere Differenzen zu konventionellen Objektdefinitionen fallen auf: Zum ersten wird über einen Mechanismus zum *Erzeugen* oder *Löschen* der Objekte nichts ausgesagt. Zum zweiten ist das Modell nicht vollständig homogen: Nicht alles ist ein Objekt, so die Eigenschaften. Es vermeidet dadurch *zirkulare* Definitionen, die einen willkürlichen Abbruch erfordern: Sind Eigenschaften Objekte, sollten auch diese Objekte Eigenschaften haben, die wiederum Objekte sind und so fort. Zum dritten führt das Objektmodell das Konzept des *zusammengesetzten* Objekts ein und greift damit auch ein zentrales Anliegen der modernen Non-Standard-Datenbanktechnik auf [Alagić, 1989; Dittrich, 1989; Encarnação & Lockemann, 1990].

Abkürzungen

ACM	Association for Computing Machinery
ASIC	Application Specific Integrated Circuit
ATE	Automatic Test Equipment
BMFT	Bundesministerium für Forschung und Technologie
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAx	Computer Aided x
CBS	Computer-Based Systems
CFI	CAD Framework Initiative©
CIF	Caltech Intermediate Format
CSP	Constraint Satisfaction Problem
DADAMO	DASSY-Datenmodell
DASS	Design Automation Systems Subcommittee
DASSY	Datentransfer und Schnittstellen in offenen integrierten VLSI-Entwurfssystemen
DBMS	Database Management System
DDL	Date Definition Language
DFT	Design for Testability
DML	Date Manipulation Language
ECBS	Engineering of Computer-Based Systems
ECIP	European CAD Integration Project
EDA	Electronic Design Automation
EDIF	Electronic Design Interchange Format
EIS	Engineering Information Systems
E.I.S.	Entwurf Integrierter Schaltungen
ESPRIT	European Strategic Programme for Information Technology
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standardization Organization
JOOP	Journal of Object-Oriented Programming
KI	Künstliche Intelligenz
LSI	Large Scale Integration
MSI	Medium Scale Integration
OMG	Object Management Group
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
OODBMS	Object-Oriented DBMS
OOP	Object-Oriented Programming
OOx	Object-Oriented x
ORB	Object Request Broker
RT	Register-Transfer
SA/SD	Structured Analysis/Structured Design
SIGPLAN	Special Interest Group on Programming Languages
SIGSOFT	Special Interest Group on Software Engineering
SQL	Structured Query Language
SSI	Small Scale Integration
STEP	Standard for the Exchange of Product Model Data
TCAD	Technology CAD
VHSIC	Very High Speed IC
VLSI	Very Large Scale Integration

Literaturverzeichnis

Die Arbeit versteht sich im Sinne von Umberto ECO als eine *kompilatorische* Literaturstudie [Eco, 1990]. Folglich legitimiert sie sich durch den Nachweis der zitierten Quellen. Das Literaturverzeichnis belegt nur die Literaturstellen, auf die sich die Zitate und Verweise im Text beziehen: benutzte Literatur.

[ACM, 1987] ACM TURING AWARD LECTURES:
The First Twenty Years: 1966–1985.
Addison-Wesley Publishing Company, Reading, Mass., 1987

[Adey & Sriram (Hrsg.), 1987] ADEY; R. A.; SRIRAM, D. (Hrsg.):
Knowledge Based Expert Systems in Engineering: Planning and Design.
Computational Mechanics Publication, Southampton, 1987

[Agha, 1987] AGHA, Gul A.:
ACTORS: A Model of Concurrent Computation in Distributed Systems.
MIT Press, Cambridge, Mass., 1987

[Agrawala *et al.*, 1991] AGRAWALA, Ashok; BUHR, Raymond; JACKSON, Ken; JACKSON, Michael; LANG, Bernard; LAVI, Jonah Z.:
Computer Based Systems Engineering Workshop.
In: Proc. of the SEI 1991 Conf. on Software Engineering Education, Lecture Notes on Computer Science, Nr. 536, Springer-Verlag, 1991

[AI magazine, 1990] Themenheft:
AI and Design.
In: AI magazine, Jg. 11, H. 4, 1990

[Alagić, 1989] ALAGIĆ, Suad:
Object-Oriented Database Programming.
Springer-Verlag, New York, 1989

[Alexander, 1964] ALEXANDER, Christopher:
Notes on the Synthesis of Form.
Harvard University Press, Cambridge, MA, 1964

[Amkreutz, 1976] AMKREUTZ, J. H.:
Cybernetic Model of the Design Process.
In: Computer Aided Design, Jg. 8, H. 3, 1976, S. 187–192

[Ando & Simon, 1961] ANDO, Albert; SIMON, Herbert A.:
Aggregation of Variables in Dynamic Systems.
In: Econometrica, Jg. 29, 1961, S. 111–138

[Arbab & Wang, 1991] ARBAB, Farhad; WANG, Bin:
A Geometric Constraint-Management System in Oar.
In: Intelligent CAD Systems III: Practical Experience and Evaluation, Hrsg.: ten Hagen, P. J. W.; Veerkamp, P. J., Springer-Verlag, Berlin, 1991, S. 205–231

[Arnold *et al.*, 1991] ARNOLD, Patrick; BODOFF, Stephanie; COLEMAN, Derek; GILCHRIST, Helena; HAYES, Fiona:
An Evaluation of Five Object-Oriented Development Methods.
In: Journal of Object-Oriented Programming: Focus on Analysis & Design, Jg. 4, 1991, S. 107–121

[Athey, 1982] ATHEY, T. H.:
Systematical Systems Approach: An Integrated Method for Solving Systems Problems.
Prentice Hall, Eaglewood Cliffs, 1982

[Atwood, 1991] ATWOOD, T.:
Why the OMG Object Request Broker Should Mean Good News for Object Databases.
In: Journal of Object-Oriented Programming, Jg. 4, H. Juli/August, 1991, S. 8–11

[Balzer, 1985] BALZER, R.:
A 15-Year Perspective on Automatic Programming.
In: IEEE Transactions on Software Engineering, Jg. 11, H. 11, 1985, S. 1257–1267

[Balzer & Swartout, 1985] BALZER, R.; SWARTOUT, W.:
On the Inevitable Intertwining of Specification and Implementation.
In: CACM, Jg. 25, H. 7, 1988, S. 438–440

[Barkow *et al.*, 1994a] BARKOW, Georg; VON BRAUN, Hubert; HESSE, Wolfgang; KITTLAUS, Hans-Bernd; SCHESCHONK, Gert:
Terminologie der Softwaretechnik: Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen, Teil 1: Begriffssystematik und Grundbegriffe.
In: Informatik-Spektrum, Jg. 17, 1994, S. 39–47

[Barkow *et al.*, 1994b] BARKOW, Georg; VON BRAUN, Hubert; HESSE, Wolfgang; KITTLAUS, Hans-Bernd; SCHESCHONK, Gert:
Terminologie der Softwaretechnik: Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen, Teil 2: Tätigkeits- und ergebnisbezogene Elemente.
In: Informatik-Spektrum, Jg. 17, 1994, S. 96–105

[Barnes *et al.*, 1992] BARNES, Timothy; HARRISON, David; NEWTON, Richard; SPICKELMIER, Rick:
Electronic CAD Frameworks.
Kluwer Academic Publishers, Norwell, Massachusetts, 1992

[Bartlett, 1932] BARTLETT, C. F.:
Remembering: A Study in Experimental and Social Psychology.
Cambridge University Press, Cambridge, 1932

[Batory & Buchmann, 1984] BATORY, D. S.; BUCHMANN, Alejandro P.:
Molecular Objects, Abstract Data Types, and Data Models: A Framework.
In: Proc. of the Tenth Int. Conf. on Very Large Data Bases, 1984, S. 172–184

[Batory & Kim, 1985] BATORY, D. S.; KIM, W.:
Modelling Concepts for VLSI CAD Objects.
In: ACM Transactions on Database Systems, Bd. 10, 1985, S. 322–346

[Berlandier, 1991] BERLANDIER, P.:
Constraint Management in Evolving Object Databases.
In: Computational Intelligence, III, Hrsg.: Cercone, N.; Gardin, F., Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1991, S. 151–157

[Bertalanffy, 1973] VON BERTALANFFY, Ludwig:
General System Theory: Foundations Development Applications.
Penguin University Books, London, 1973

[Bertelsmann-Lexikon, 1975] *Lexikothek: Das Bertelsmann Lexikon.*
Bertelsmann-Lexikon-Verlag, Gütersloh, 1975

[Bhat & Taku, 1990] BHAT, Jayaram; TAKU, Fumio:
Frameworks: EDA-Technologie für die 90er Jahre.
In: Elektronik, Jg. 39, H. 21, 1990, S. 88–94

[Biggerstaff & Richter, 1989] BIGGERSTAFF, Ted J.; RICHTER, Charles:
Reusability Framework, Assessment, and Directions.
In: [Biggerstaff & Perlis, 1989], S. 1–17

[Biggerstaff & Perlis, 1989] BIGGERSTAFF, Ted J.; PERLIS, Alan J.:
Software Reusability: Concepts and Models.
Addison-Wesley Publishing Company, Reading, Mass., 1989

[Blaha *et al.*, 1991] BLAHA, M.; EDDY, F.; LORENSEN, W.; PREMERLANI, W.; RUMBAUGH, James:
Object-Oriented Modeling and Design.
Prentice Hall, New Jersey, 1991

[Blair *et al.*, 1989] BLAIR, Gordon S.; GALLAGHER, John J.; MALIK, Javad:
Genericity vs. Inheritance vs. Delegation vs. Conformance vs. ....
In: Journal of Object-Oriented Programming, Jg. 2, H. September/Oktober, 1989, S. 11–17

[Blaauw, 1972] BLAAUW, Gerrit A.:
Computer Architecture.
In: Elektronische Rechenanlagen, Jg. 14, H. 4, 1972, S. 154–159

[Blödel (Hrsg.), 1991] BLÖDEL, Joachim (Hrsg.):
EDIF Device Descriptions, V 2 0 5.
Universität Kaiserslautern, 1991

[Boehm, 1982] BOEHM, Barry W.:
Software Engineering Economics.
Prentice Hall, New Jersey, 1982

[Boehm, 1988] BOEHM, Barry W.:
A Spiral Model of Software Development and Enhancement.
In: Computer, Jg. 21, H. 5, 1988, S. 61–72

[Boehm & Scherlis, 1987] BOEHM, Barry W.; SCHERLIS, William L.:
Megaprogramming.
In: Proc. DARPA Software Technology Conference, 1987

[Booch, 1983] BOOCH, Grady:
Software Engineering with Ada.
The Benjamin/Cummings Publishing Company, Menlo Park, CA, 1983

[Booch, 1991] BOOCH, Grady:
Object Oriented Design with Applications.
The Benjamin/Cummings Publishing Company, Redwood City, 1991

[Booth, 1989] BOOTH, Paul A.:
An Introduction to Human-Computer Interaction.
Lawrence Erlbaum Associates, Hove, 1989

[Borgida *et al.*, 1984] BORGIDA, Alexander; MYLOPOULOS, John; WONG, Harry K. T.:
Generalization/Specialization as a Basis for Software Specification.
In: [Brodie *et al.* (Hrsg.), 1984], S. 87–117

[Borning, 1981] BORNING, Alan:
The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory.
In: ACM Transactions on Programming Languages and Systems, Jg. 3, H. 4, 1981, S. 353–387

[Borning & Duisberg, 1986] BORNING, Alan; DUISBERG, R.:
Constraint-Based Tools for Building User Interfaces.
In: ACM Transactions on Graphics, Bd. 5, H. 4, 1986, S. 345–374

[Borning *et al.*, 1987] BORNING, Alan; DUISBERG, R.; FREEMAN-BENSON, B.; KRAMER, A.; WOOLF, M.:
Constraint Hierarchies.
Proc. of OOPSLA, 1987

[Brachman & Levesque (Hrsg.), 1985] BRACHMAN, Ronald J.; LEVESQUE, Hector J. (Hrsg.):
Readings in Knowledge Representation.
Morgan Kaufmann Publishers, San Mateo, Kalifornien, 1985

[Brauer, 1990] BRAUER, Johannes:
Datenhaltung in VLSI-Entwurfssystemen.
Teubner-Verlag, Stuttgart, 1990

[Brockhaus-Enzyklopädie, 1990] F. A. Brockhaus, Mannheim, 19. Auflage, 1990

[Brodie *et al.* (Hrsg.), 1984] BRODIE, Michael L.; MYLOPOULOS, John; SCHMIDT, Joachim W. (Hrsg.):
On Conceptual Modelling.
Springer-Verlag, New York, 1984

[Bronstein, 1981] BRONSTEIN, Il'ja N.:
Taschenbuch der Mathematik.
Verlag Harri Deutsch, Thun und Frankfurt/Main, 21. Aufl., 1981

[Brooks, 1975] BROOKS, Frederick P.:
The Mythical Man-Month.
Addison-Wesley Publishing Company, Reading, Mass., 1975

[Brooks, 1987] BROOKS, Frederick P.:
No Silver Bullet: Essence and Accidents of Software Engineering.
In: Computer, Jg. 20, H. 4, 1987, S. 10–19

[Brügge, 1993] BRÜGGE, Peter:
Mythos aus dem Computer: Über Ausbreitung und Mißbrauch der „Chaostheorie".
In: Spiegel, Jg. 47, H. 39, 1993, S. 156–164

[Bryant, 1991] BRYANT, Randal E.:
On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication.
In: IEEE Transactions on Computers, Jg. 40, H. 2, 1991, S. 205–213

[Bunge, 1979] BUNGE, Mario:
Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World.
und *Vol. 4: Ontology II: A World of Systems.*
Reidel, Boston, 1979

[CACM, 1990] Themenheft:
Object-Oriented Design.
In: CACM, Jg. 33, H. 9, 1990

[Capurro, 1990] CAPURRO, Rafael:
Ethik und Informatik: Die Herausforderung der Informatik für die praktische Philosophie.
In: Informatik-Spektrum, Jg. 13, 1990, S. 311–320

[Cardelli & Wegner, 1985] CARDELLI, Luca; WEGNER, Peter:
On Understanding Types, Data Abstraction, and Polymorphism.
In: ACM Computing Surveys, Jg. 17, H. 4, 1985, S. 471–522

[Chalmers & Meys, 1989] CHALMERS, David L.; MEYS, Frans A. C.:
Successful Cad Integration Needs a Standard Conceptual Model.
In: [EDIF, 1989], S. II-6–II-10

[Champeaux & Faure, 1992] CHAMPEAUX, D.; FAURE, P.:
A Comparative Study of Object-Oriented Analysis methods.
In: Journal of Object-Oriented Programming, Jg. 5, H. März/April, 1992

[Chen, 1976] CHEN, Peter P.:
The Entity-Relationship Model: Toward a Unified View of Data.
In: ACM Transactions on Database Systems, Bd. 1, 1976, S. 9–36

[Claus, 1989] CLAUS, Volker:
Entwicklung von Informatik-Methoden.
In: Handbuch der Modernen Datenverarbeitung, Jg. 26, H. 150, 1989, S. 26–44

[Claus & Schwill, 1993] *Duden Informatik.*
Hrsg.: Lektorat des B.I.-Wissenschaftverlags unter Leitung von Hermann Engesser, bearbeitet von Volker Claus und Andreas Schwill, 2. Aufl., Mannheim, 1993

[Coad & Yourdon, 1991a] COAD, Peter; YOURDON, Edward:
Object-Oriented Analysis.
Prentice Hall, New Jersey, 1991

[Coad & Yourdon, 1991b] COAD, Peter; YOURDON, Edward:
Object-Oriented Design.
Prentice Hall, New Jersey, 1991

[Codd, 1979] CODD, Edgar F.:
Extending the Database Relational Model to Capture More Meaning.
In: ACM Transactions on Database Systems, Bd. 4, 1979, S. 397–434

[Computer, 1992] Themenheft:
Inheritance and Classification in Object-Oriented Computing.
In: Computer, Jg. 25, H. 10, 1992

[Conant, 1976] CONANT, Roger C.:
Laws of Information Which Govern Systems.
In: IEEE Transactions on Systems, Man, and Cybernetics, Jg. 6, H. 4, 1976, S. 240–255

[Conway & Mead, 1980] CONWAY, Lynn; MEAD, Carver:
Introduction to VLSI Systems.
Addison-Wesley Publishing Company, Reading, Mass., 1980

[Cordes *et al.*, 1989] CORDES, R.; HOFMANN, M.; LANGENDÖRFER, H.:
Hypertext/Hypermedia.
In: Informatik-Spektrum, Jg. 12, H. 4, 1989, S. 218–220

[Courtois, 1977] COURTOIS, P.-J.:
Decomposability: Queueing and Computer System Applications.
Academic Press Inc., London, 1977

[Courtois, 1985] COURTOIS, P.-J.:
On Time and Space Decomposition of Complex Structures.
In: Communications of the ACM, Jg. 28, H. 6, 1985, S. 590–603

[Cox, 1984] COX, Brad J.:
Message/Object Programming: An Evolutionary Change in Programming Technology.
In: IEEE Software, Jg. 1, H. 1, 1984, S. 150–161

[Cox, 1987] COX, Brad J.:
Object-Oriented Programming: An Evolutionary Approach.
Addison-Wesley Publishing Company, Reading, Mass., 1987

[Dahl *et al.*, 1972] DAHL, Ole-Johan; DIJKSTRA, Edsger W.; HOARE, C. A. R.:
Structured Programming.
Academic Press Inc., New York, 1972

[Dahl & Nygaard, 1966] DAHL, Ole-Johan; NYGAARD, Kristen:
SIMULA: An Algol-based Simulation Language.
In: Communications of the ACM, Jg. 9, H. 9, 1966, S. 671–678

[Danforth & Tomlinson, 1988] DANFORTH, Scott; TOMLINSON, Chris:
Type Theories and Object-Oriented Programming.
In: ACM Computing Surveys, Jg. 20, H. 1, 1988, S. 29–72

[Daniell, 1989] DANIELL, James Donald:
An Object Oriented Approach to CAD Tool Control.
Dissertation, Carnegie Mellon University, Pittsburgh, 1989

[DASSY, 1990] DASSY-AUTOREN:
Anforderungen an eine Werzeug-Schnittstelle aus der Sicht der Tools.
Bericht zum 1. DASSY-Meilenstein-Workshop, St. Augustin, April 1990

[DASSY, 1992a] DASSY-AUTOREN:
DaDaMo: The DASSY Data Model.
GMD, St. Augustin, 1992

[DASSY, 1992b] DASSY-AUTOREN:
The DASSY Prototype.
GMD, St. Augustin, 1992

[DeMarco, 1978] DEMARCO, Tom:
Structured Analysis and System Specification.
Yourdon Press, Prentice Hall, New Jersey, 1978

[DeRemer & Kron, 1976] DEREMER, Frank: KRON, Hans H.:
Programming-in-the-Large versus Programming-in-the Small.
In: IEEE Transactions on Software Engineering, Jg. 2, H. 2, 1976, S. 80–86

[Dijkstra, 1965] DIJKSTRA, Edsger W.:
Programming Considered as a Human Activity.
Proc. of the IFIP Congress, Amsterdam, 1965, S. 213–217

[Dijkstra, 1968] DIJKSTRA, Edsger W.:
Go To Statement Considered Harmful.
In: Communications of the ACM, Jg. 11, H. 3, 1968, S. 147–148

[Dillon & Tan, 1993] DILLON, Tharam; TAN, Poh Lee:
Object-Oriented Conceptual Modeling.
Prentice Hall, Sidney, 1993

[Dittrich, 1989] DITTRICH, Klaus R.:
Objektorientierte Datenbanksysteme.
In: Informatik-Spektrum, Jg. 12, H. 4, 1989, S. 215–218

[Dörner, 1976] DÖRNER, Dietrich:
Problemlösen als Informationsverarbeitung.
Kohlhammer, Stuttgart, 1976

[Doyle, 1979] DOYLE, Jon:
A Truth Maintenance System.
In: Artificial Intelligence, Jg. 12, 1979, S. 231–272

[Dreyfus & Dreyfus, 1987] DREYFUS, H. L.; DREYFUS, S. E.:
Künstliche Intelligenz: Von den Grenzen der Denkmaschne und dem Wert der Intuition.
Rowohlt, 1987

[Duden, 1970] *Duden: Stilwörterbuch der deutschen Sprache.*
Hrsg.: Wissenschaftlicher Rat der Dudenredaktion, 6. Aufl., Mannheim, 1970

[Eco, 1990] Eco, Umberto:
Wie man eine wissenschaftliche Abschlußarbeit schreibt: Doktor-, Diplom- und Magisterarbeit in den Geistes- und Sozialwissenschaften.
UTB für Wissenschaft, Uni-Taschenbücher 1512, Heidelberg, 3. Aufl., 1990

[EDIF, 1989] EDIF:
Third European EDIF Forum.
In: Proc. of the 3rd European EDIF Forum, Hrsg.: German EDIF Interest Group in Cooperation with European EDIF Groups, Bonn/Königswinter, 1989

[EDIF, 1990] EDIF:
Fourth European EDIF Forum: Session I: Information Modelling.
In: Proc. of the 4th European EDIF Forum, Hrsg: U.K. EDIF Users' Group; European EDIF Users' Group; EuroEDIF, University of Manchester, GB, 1990, S. 3–43

[EDIF-Manual, 1990] EDIF:
Electronic Design Interchange Format EDIF, Version 2.0.0, EIA/ANSI Standard 548.
Electronic Industries Association, EDIF Steering Committee, Washington D. C., 1990

[Ege, 1991] Ege, R. K.:
Improving Object-oriented User Interfaces with Constraints.
In: Information and Software Technology, Bd. 33, H. 2, 1991, S. 143–150

[Encarnação & Lockemann, 1990] Encarnação, José L.; Lockemann, Peter C.:
Engineering Databases: Connecting Islands of Automation Through Databases.
Springer-Verlag, Berlin Heidelberg, 1990

[Encyclopædia Britannica, 1986] 15. Auflage, 1986

[Endres, 1988] Endres, Albert:
Software-Wiederverwendung: Ziele, Wege und Erfahrungen.
In: Informatik-Spektrum, Jg. 11, 1988, S. 85–95

[Endres & Uhl, 1992] Endres, Albert; Uhl, Jürgen:
Objektorientierte Software-Entwicklung: Eine Herausforderung für die Projektführung.
In: Informatik-Spektrum, Jg. 15, 1992, S. 255–263

[Ervin & Gross, 1987] Ervin, Steven M.; Gross, Mark Donald:
ROADLAB: A Constraint Based Laboratory for Road Design.
In: [Adey & Sriram (Hrsg.), 1987], S. 167–184

[Faltings *et al.*, 1990] Faltings, Boi; Haroud, Djamila; Hua, Kefeng:
Dynamic Constraint Satisfaction in a Bridge Design System.
In: [Gottlob & Nejdl (Hrsg.), 1990], S. 217–232

[Festinger, 1966] Festinger, Leon:
A Theory of Cognitive Dissonance.
Stanford University Press, Stanford CA, 1966

[Fiadeiro & Sernadas, 1991] Fiadeiro, José; Sernadas, Cristina:
Towards Object-Oriented Conceptual Modeling.
In: Data & Knowledge Engineering, Jg. 6, 1991, S. 479–508

[Fidelak & Güsgen, 1991] Fidelak, Manfred; Güsgen, Hans-Werner:
Programmieren mit Constraints.
In: [Struß, 1991], S. 131–144

[Fischer, 1983] FISCHER, G.:
Wie intelligent können und sollen Computersysteme sein?
In: Intelligenztechnologie. Hrsg.: Giloi; Schulze-Vorberg, Teubner-Verlag, Stuttgart, 1983

[Flood, 1987] FLOOD, Robert L.:
Complexity: a Definition by Construction of a Conceptual Framework.
In: Systems Research, Jg. 4, H. 3, 1987, S. 177–185

[Flores & Winograd, 1986] FLORES, F.; WINOGRAD, Terry:
Understanding Computers and Cognition: A New Foundation for Design.
Ablex Publishing Corporation, Norwood, 1986

[Floyd, 1988] FLOYD, C.:
Outline of a Paradigm Change in Software Engineering.
In: SIGSOFT Software Engineering Notes, Jg. 13, H. 2, 1988, S. 25–38

[Floyd, 1994] FLOYD, Christiane:
Software-Engineering — und dann?.
In: Informatik-Spektrum, Jg. 17, 1994, S. 29–37

[Frank, 1992] FRANK, Ulrich:
Integrierte Informationssysteme: Konventionelle Modelle und Perspektiven objektorientierter Kommunikation.
In: Praxis der Informationsverarbeitung und Kommunikation, Jg. 15, H. 1, 1992, S. 29–35

[Freeman & Newell, 1971] FREEMAN, Paul; NEWELL, Allen:
A Model for Functional Reasoning in Design.
In: Proc. Second International Joint Conference on Artificial Intelligence, British Computer Society (Hrsg.), London, 1971, S. 621–640

[Freuder, 1989] FREUDER, Eugene C.:
Partial Constraint Satisfaction.
In: Proc. of the 11th Int. Joint Conf. of Artificial Intelligence, Detroit, Michigan, 1989, S. 278–283

[Gajski, 1988] GAJSKI, Daniel D.:
Silicon Compilation.
Addison-Wesley Publishing Company, Reading, Mass., 1988

[Gajski & Kuhn, 1983] GAJSKI, Daniel D.; KUHN, Robert H.:
Guest Editors' Introduction: New VLSI Tools.
In: Computer, Jg. 16, H. 12, 1983, S. 11–14

[Gardner, 1985] GARDNER, Howard:
The Mind's New Science: A History of the Cognitive Revolution.
Basic Books Inc., New York, 1985

[Gibbs *et al.*, 1990] GIBBS, Simon; TSICHRITZIS, Dennis; CASAIS, Eduardo; NIERSTRASZ, Oscar; PINTADO, Xavier:
Class Management für Software Communities.
In: Communications of the ACM, Jg. 33, H. 9, 1990, S. 90–103

[Girczyc & Ly, 1988] GIRCZYC, Emil F.; LY, Tai A.:
Constraint Propagation in an Object-Oriented IC Design Environment.
In: Proc. of the 25th ACM/IEEE Design Automation Conf., 1988, S. 628–633

[Goldberg, 1984] GOLDBERG, Adele:
Smalltalk-80: The Interactive Programming Environment.
Addison-Wesley Publishing Company, Reading, Mass., 1984

[Goldfein *et al.*, 1990] GOLDFEIN, Arnold; MAZDRA, Ron; BREVARD, Laurence:
Information Model and (Example) Programming Interface.
CAD Framework Initiative Inc., Boulder, CO, 1990

[Gombrich, 1978] GOMBRICH, Ernst H.:
Kunst und Illusion: Zur Psychologie der bildlichen Darstellung.
Belser-Verlag, Stuttgart, 1978

[Gordon, 1961] GORDON, W. J. J.:
Synectics.
Harper & Row, New York, 1961

[Gossard & Serrano, 1987] GOSSARD, D.; SERRANO, D.:
Constraint Management in Conceptual Design.
In: [Adey & Sriram (Hrsg.), 1987], S. 211–224

[Gottlieb *et al.*, 1989] GOTTLIEB, P.; KUMAR, A.; MAFFIT, R. B.:
EIS Engineering Information Model: EDIF Domain Model.
Electronic Technology Division, Microelectronics Center, 1989

[Gottlob & Nejdl (Hrsg.), 1990] GOTTLOB, Georg; NEJDL, Wolfgang (Hrsg.):
Expert Systems in Engineering.
Springer-Verlag, Reihe: Lecture Notes in Artificial Intelligence, Berlin Heidelberg, 1990

[Gray, 1981] GRAY, J. N.:
The Transaction Concept: Virtues and Limitations.
In: Proc. of the 7th Conf. on VLDB, Cannes, 1981, S. 144

[Griethuysen, 1982] VAN GRIETHUYSEN, J. J. [Hrsg.]:
Concepts and Terminology for the Conceptual Schema and the Information Base.
ISO/TC97/SC5/WG3, 1982

[Gross, 1986] GROSS, Mark Donald:
Design as Exploring Constraints.
Dissertation, Massachusetts Institute of Technology, 1986

[Gryczan & Züllighoven, 1992] GRYCZAN, Guido; ZÜLLIGHOVEN, Heinz:
Objektorientierte Systementwicklung: Leitbild und Entwicklungsdokumente.
In: Informatik-Spektrum, Jg. 15, 1992, S. 264–272

[Güsgen & Hertzberg, 1992] GÜSGEN, Hans Werner; HERTZBERG, Joachim:
A Perspective of Constraint-Based Reasoning.
Springer-Verlag, Reihe: Lecture Notes in Artificial Intelligence, Berlin Heidelberg, 1992

[Hailpern, 1986] HAILPERN, Brent:
Multiparadigm Laguages and Environments.
IEEE Software, Jg. 3, H. 1, 1986, S. 6–13

[Harel, 1992] HAREL, David:
Biting the Silver Bullet: Toward a Brighter Future for System Development.
In: Computer, Jg. 25, H. 1, 1992, S. 8–20

[Henderson-Sellers, 1992] HENDERSON-SELLERS, Brian:
A Book of Object-Oriented Knowledge: Object-Oriented Analysis, Design and Implementation, a New Approach to Software Engineering.
Prentice Hall, New Jersey, 1992

[Herrig, 1982] HERRIG, Dieter:
Design Theory for CAD-Systems and CAD-Objects.
In: Proceedings of IFIP WG 5.2 Working Conference on File Structures and Databases for CAD, North Holland, 1982, S. 245–257

[Hoare, 1965] HOARE, C. A. R.:
Record Handling.
In: IEEE Algol Bulletin, Jg. 21, Nov. 1965, S. 39–69

[Horebeek & Lewi, 1989] VAN HOREBEEK, Ivo; LEWI, Johan:
Algebraic Specifications in Software Engineering.
Springer-Verlag, Berlin Heidelberg, 1989

[Hübner, 1978] HÜBNER, Kurt:
Kritik der wissenschaftlichen Vernunft.
Verlag Karl Alber, Freiburg/München, 1978

[Hull & King, 1987] HULL, Richard; KING, Roger:
Semantic Database Modeling: Survey, Applications, and Research Issues.
In: ACM Computing Surveys, Bd. 19, H. 3, 1987

[IDEF, 1985] *Information Modeling Manual IDEF1 - Extended (IDEF1x).*
D. Appleton Company, Manhatten Beach, CA., 1985

[Informatik-Spektrum, 1992] Themenheft:
Projektmanagement für objektorientierte Software-Entwicklung.
In: Informatik-Spektrum, Jg. 15, S. 253–292, 1992

[Informatik-Spektrum, 1993] Themenheft:
Objektorientierte Datenbanksysteme.
In: Informatik-Spektrum, Jg. 16, S. 67–97, 1993

[Ishikawa *et al.*, 1991] ISHIKAWA, Hiroshi; IZUMIDA, Yoshio; KAWATO, Nobuaki:
An Object-Oriented Database: System and Applications.
In: IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, 1991, S. 288–291

[Johnson & Wirfs-Brock, 1990] JOHNSON, Ralph E.; WIRFS-BROCK, Rebecca J.:
Surveying Current Research in Object-Oriented Design.
In: CACM, Jg. 33, H. 9, 1990, S. 104–124

[Johnson *et al.*, 1992] JOHNSON, Ralph; LUCKHAM, David; PURTILO, James; SCHERLIS, William; WEGNER, Peter:
Object-Oriented Megaprogramming.
In: Proc. of OOPSLA '92, 1992, S. 392–396

[Katz *et al.*, 1986] KATZ, Randy H.; ANWARRUDIN, M.; CHANG, E.:
A Version Server for Computer-Aided Design Data.
In: Proc. of the 25th ACM/IEEE Design Automation Conf., 1986, S. 27–33

[Katz, 1990] KATZ, Randy H.
Toward a Unified Framework for Version Modeling in Engineering Databases.
In: ACM Computing Surveys, Jg. 22, H. 4, 1990, S. 375–408

[Kay, 1977] KAY, Alan C.:
Microelectronics and the Personal Computer.
In: Scientific American, Sept. 1977, S. 231–244

[Kay, 1993] KAY, Alan C.:
The Early History of Smalltalk.
In: SIGPLAN Notices, Jg. 28, H. 3, 1993, S. 69–95

[Kelly & Steinberg, 1982] KELLY, V. E.; STEINBERG, L. I.:
The Critter System: Analyzing Digital Circuits by Propagating Behaviors and Specifications.
Proc. of the Second National Conf. on Artificial Intelligence, Pittsburgh, PA, 1982, S. 284–289

[KI, 1994] *Zur Diskussion: Problemlösen ohne oder durch Suche?*
In: KI: Künstliche Intelligenz, Jg. 8, H. 1, 1994, S. 73–84

[Kim & Lochovsky (Hrsg.), 1989] KIM, Won; LOCHOVSKY, Frederick H. (Hrsg.):
Object-Oriented Concepts, Databases, and Applications.
Addison-Wesley Publishing Company, Reading, Mass., 1989

[Kim *et al.*, 1983] KIM, W.; LORIE, R.; MCNABB, D.; PFOUFFE, W.:
Nested Transactions for Engineering Databases.
In: IBM Research Reports, RJ 3934, 1983

[King, 1989] KING, Roger:
My Cat is Object-Oriented.
In: [Kim & Lochovsky (Hrsg.), 1989], S. 23–30

[Klingsheim, 1990] KLINGSHEIM, Karl:
VLSI Design Theory: Concurrent Decision Cycles in VLSI Systems Design.
Dissertation, Universität Trondheim, Norwegen, 1990

[Klir, 1985] KLIR, George J.:
Complexity: Some General Observations.
In: Systems Research, Jg. 2, H. 2, 1985, S. 131–140

[Knolmayer & Myrach, 1990] KNOLMAYER, Gerhard; MYRACH, Thomas:
Anforderungen an Tools zur Darstellung und Analyse von Datenmodellen.
In: Handbuch der Modernen Datenverarbeitung, Jg. 27, H. 152, 1990, S. 90–102

[Knuth, 1974] KNUTH, Donald E.:
Computer Programming as an Art.
In: [ACM, 1987], S. 33–46

[Köhler, 1992] KÖHLER, Dagmar:
Konsistenzsicherung in graphischen Anwendungen: Modellierung und Sicherstellung graphischer Constraints mit dem CRUX-System.
Dissertation, THD Darmstadt, 1992

[Koepke, 1985] KOEPKE, David J.:
The Evolution of Software Design Ideas.
Master-of-Science-Arbeit, California State University, Northridge, 1985

[Koestler, 1967] KOESTLER, Arthur:
The Ghost in the Machine.
Hutchinson, London, 1967

[Koomen, 1978] KOOMEN, Cees J.:
Information Laws for System Design.
In: Proc. Intern. Conf. on Cybernetics and Society, Tokyo, Bd. II, 1978, S. 1013–1017

[Koomen, 1985] KOOMEN, Cees J.:
The Entropy of Design: A Study on the Meaning of Creativity.
In: IEEE Transactions on Systems, Man, and Cybernetics, Jg. 15, H. 1, 1985, S. 16–30

[Korson & McGregor, 1990] KORSON, Tim; MCGREGOR, John D.:
Understanding Object-Oriented: A Unifying Paradigm.
In: Communications of the ACM, Jg. 33, H. 9, 1990, S. 41–60

[Kotz, 1989] KOTZ, Angelika M.:
Triggermechanismen in Datenbanksystemen.
Springer-Verlag, Berlin Heidelberg, Informatik-Fachberichte 201, 1989

[Kuhn, 1989] KUHN, Thomas S.:
Die Struktur wissenschaftlicher Revolutionen.
Suhrkamp, 10. Aufl., 1989

[Küpfmüller, 1990] KÜPFMÜLLER, Karl:
Einführung in die Theoretische Elektrotechnik.
Springer-Verlag, Berlin, 13. Aufl., 1990

[Lang & Quibeldey-Cirkel (Hrsg.), 1992] LANG, Walter; QUIBELDEY-CIRKEL, Klaus (Hrsg.):
Reader OOKA: Objektorientierte Konzepte und Anwendungen.
Universität Siegen, 1992

[Lawson, 1990] LAWSON, Harold, W.:
Philosophies for Engineering Computer-Based Systems.
In: Computer, Jg. 23, H. 12, 1990, S. 52–63

[Leler, 1988] LELER. Wm:
Constraint Programming Languages: Their Specification and Generation.
Addison-Wesley Publishing Company, Reading, Mass., 1988

[Lieberman, 1986] LIEBERMAN, Henry:
Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems.
In: SIGPLAN Notices, Jg. 21, H. 9, 1986, S. 214–223

[Ling & Teo, 1992] LING, T. W.; TEO, P. K.:
On Rules and Integrity Constraints in Database systems.
In: Information and Software Technology, Jg. 34, H. 3, 1992, S. 147–158

[Lindsay & Norman, 1972] LINDSAY, P. H.; NORMAN, Donald A.:
Human Information Processing.
Academic Press, New York, 1972

[Lipeck, 1989] LIPECK, Udo W.:
Dynamische Integrität von Datenbanken.
Springer-Verlag, Berlin Heidelberg, Informatik-Fachberichte 209, 1989

[Liskov & Zilles, 1974] LISKOV, Barbara; ZILLES, Stephen N.:
Programming with Abstract Data Types.
In: SIGPLAN Notices, Jg. 9, H. 4, 1974, S. 50–59

[Lockemann, 1986] LOCKEMANN, Peter C.:
Konsistenz, Konkurrenz, Persistenz — Grundbegriffe der Informatik?.
In: Informatik-Spektrum, Jg. 9, 1986, S. 300–305

[Luft, 1988] LUFT, Alfred Lothar:
Informatik als Technik-Wissenschaft: Eine Orientierungshilfe für das Informatik-Studium.
BI-Wissenschaftsverlag, Mannheim, 1988

[Mackworth, 1977] MACKWORTH, Alan K.:
Consistency in Networks of Relations.
In: Artificial Intelligence, Jg. 8, 1977, S. 99–118

[Mackworth, 1987] MACKWORTH, Alan K.:
Constraint Satisfaction.
In: [Shapiro (Hrsg.), 1987], 205–211

[Maier, 1989] MAIER, David:
Making Database Systems Fast Enough for CAD Applications.
In: [Kim & Lochovsky (Hrsg.), 1989], S. 573–582

[March & Umphress, 1991] MARCH, Steven G.; UMPHRESS, David A.:
Object-Oriented Requirements Determination.
In: Journal of Object-Oriented Programming: Focus on Analysis & Design, Jg. 4, 1991, S. 35–40

[McIlroy, 1969] MCILROY, M. Doug:
Mass-Produced Software Components.
In: [Naur & Randell (Hrsg.), 1969], S. 138–150

[McLaren, 1992] MCLAREN, Harris:
PDES/STEP: The Critical Manufacturing Industry Standard of the Nineties.
Digital Equipment Corporation, Electronics/Aerospace IBU, 1992

[Meersman & Sernadas (Hrsg.), 1988] MEERSMAN, R. A.; SERNADAS A. C. (Hrsg.):
Data and Knowledge.
Proc. of the Second IFIP 2.6 Working Conference on Database Semantics, Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1988

[Mellor & Shlaer, 1992] MELLOR, Stephen J.;SHLAER, Sally:
Object Lifecycles: Modeling the World in States.
Prentice Hall, New Jersey, 1992

[Meyer, 1988] MEYER, Bertrand:
Object-Oriented Software Construction.
Prentice Hall, New Jersey, 1988

[Meyer, 1990] MEYER, Bertrand:
The New Culture of Software Development.
In: Journal of Object-Oriented Programming, Jg. 3, H. Nov./Dez., 1990, S. 76–81

[Miller, 1989] MILLER, Ann:
From Expert Assistant to Design Verification: Applications of AI to VLSI Design.
In: SOUTHEASTCON '89, Energy and Information Technologies in the Southeast, IEEE, New York, 1989, S. 406–410

[Miller, 1956] MILLER, George A.:
The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.
In: The Psychological Review, Jg. 63, H. 2, 1956, S. 81–97

[Minsky, 1981] MINSKY, Marvin Lee:
A Framework for Representing Knowledge.
In: [Brachman & Levesque (Hrsg.), 1985], S. 246–262

[Minsky, 1986] MINSKY, Marvin Lee:
The Society of Mind.
Simon and Schuster, New York, 1986

[Molzberger, 1984] MOLZBERGER, Peter:
Transcending the Basic Paradigm of Software Engineering.
Hochschule der Bundeswehr München, Bericht-Nr. 8405, 1984

[Monarchi & Puhr, 1992] MONARCHI, David E.; PUHR, Gretchen I.:
A Research Typology for Object-Oriented Analysis and Design.
In: Communications of the ACM, Jg. 35, H. 9, 1992, S. 35–47

[Montanari, 1974] MONTANARI, Ugo:
Networks of Constraints: Fundamental Properties and Applications to Picture Processing.
In: Information Sciences, Jg. 7, 1974, S. 95–132

[Moon, 1989] MOON, David A.:
The Common LISP Object-Oriented Programming Language Standard.
In: [Kim & Lochovsky (Hrsg.), 1989], S. 49–78

[Moore, 1975] MOORE, Gordon E.:
Progress in Digital Integrated Electronics.
In: Proc. of the IEEE Int. Electron Devices Meeting, Talk 1.3, Washington DC, 1975

[Moran, 1981] MORAN, Thomas P.:
The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems.
In: International Journal of Man-Maschine Systems, Jg. 15, H. 1, 1981, S. 3–50

[Mostow, 1985] MOSTOW, Jack:
Towards Better Models of the Design Process.
In: The AI Magazine, Jg. 2, H. 1, 1985, S. 44–57

[Motro, 1989] MOTRO, Amihai:
Integrity = Validity + Completeness.
In: ACM Transactions on Database Systems, Jg. 14, H. 4, 1989, S. 480–502

[Müller, 1993] MÜLLER, J.-A.:
Kommt die Entwicklung betrieblicher DV-Anwendungssysteme ohne Systems-Engineering aus?.
In: Informatik-Spektrum, Jg. 16, 1993, S. 167–169

[Müller, 1983] MÜLLER, Roland:
Zur Geschichte des Modelldenkens und des Modellbegriffs.
In: Modelle — Konstruktion der Wirklichkeit, Hrsg: Stachowiak, Herbert, Wilhelm-Fink-Verlag, München, 1983, S. 17–86

[Muroga, 1990] MUROGA, Eisuke:
Constraint Models for an Object-oriented Database.
Dissertation, University of Illinois at Urbana-Champaign, 1990

[Naclerio, 1989] NACLERIO, Nicholas J.:
Standards for Engineering Information Systems.
In: [EDIF, 1989], S. II-1–II-5

[Naur & Randell (Hrsg.), 1969] NAUR, Peter; RANDELL, Brian (Hrsg.):
Software Engineering: Report on a Conference, Garmisch 1968.
NATO Scientific Affairs Division, Brüssel, 1969

[Newell & Simon, 1972] NEWELL, Allen; SIMON, Herbert A:
Human Problem Solving.
Prentice Hall, New York, 1972

[Newell & Simon, 1975] NEWELL, Allen; SIMON, Herbert A:
Computer Science as Empirical Inquiry: Symbols and Search.
In: [ACM, 1987], S. 287–313

[Nierstrasz, 1989] NIERSTRASZ, Oscar:
A Survey of Object-Oriented Concepts.
In: [Kim & Lochovsky (Hrsg.), 1989], S. 3–21

[Norman, 1986] NORMAN, Donald A.:
Cognitive Engineering.
In: User-Centred System Design: New Perspectives on Human-Computer Interaction, DRAPER, S.; NORMAN, D. A. (Hrsg.), Hillsdale, New Jersey, 1986

[Odawara (Hrsg.), 1989] ODAWARA, G.:
CAD Systems Using AI Techniques.
Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1989

[Odell, 1991] ODELL, James:
Object-Oriented Analysis and Design.
In: JOOP: Focus on Analysis and Design, 1991, S. 74–84

[Odell, 1992] ODELL, James:
More Than a Programming Language: Object Concepts Provide a General Philosophy for System Development and Design.
In: Object Magazine, Jg. 2, H. 4, 1992, S. 47–49

[Orman, 1991] ORMAN, L.:
Constraint Maintenance as a Data Model Design Criterion.
In: The Computer Journal, Jg. 34, H. 1, 1991, S. 73–79

[Parker & Williams, 1979] PARKER, K. P.; WILLIAMS, T. W.:
Testing Logic Networks and Design for Testability.
In: Computer, Jg. 12, H. 10, 1979, S. 9–21

[Parnas, 1972a] PARNAS, David L.:
On the Criteria to be Used in Decomposing Systems into Modules.
In: Communications of the ACM, Jg. 15, H. 12, 1972, S. 1053–1058

[Parnas, 1972b] PARNAS, David L.:
Information Distribution Aspects of Design Methodology.
In: Information Processing 71, North-Holland Publishing Company, 1972, S. 339–344

[Patel-Schneider, 1990] PATEL-SCHNEIDER, P. F.:
Practical, Object-Based Knowledge Representation for Knowledge-Based Systems.
In: Information Systems, Jg. 15, H. 1, 1990, S. 9–19

[Popper, 1963] POPPER, Karl R.:
Conjectures and Refutations.
London, 1963

[Quibeldey-Cirkel, 1987] QUIBELDEY-CIRKEL, Klaus:
Rechnerunterstützter Entwurf eines Wellendigitalfilters als CMOS-Gate-Array.
FernUniversität Hagen, 1987

[Quibeldey-Cirkel & Wahl, 1990a] QUIBELDEY-CIRKEL, Klaus; WAHL, Michael G.:
The Electronic Design Interchange Format EDIF: Capabilities and Evaluation.
In: Proc. of the Int. Conf. on the Factory of the Future to Realise CIM, Intertechno '90, Hrsg: Horváth, Imre; Lehotzky, József, Verlag: Gépipari Tudományos Egyesület, Budapest, 1990, S. 515–524

[Quibeldey-Cirkel & Wahl, 1990b] QUIBELDEY-CIRKEL, Klaus; WAHL, Michael G.:
What's on in the EDIF Test Are(n)a?.
In: [EDIF, 1990], S. 92–101

[Quibeldey-Cirkel & Wojtkowiak, 1991] QUIBELDEY-CIRKEL, Klaus; WOJTKOWIAK, Hans:
Modellierung des VLSI-Entwurfsprozesses: Rückschau und Ausblick.
In: Entwurf Integrierter Schaltungen, 5. E.I.S.-Workshop am 8. und 9. April 1991 an der Technischen Universität Dresden, Hrsg: Kaesser, Augustin W., Gesellschaft für Mathematik und Datenverarbeitung GMD, GMD-Studie Nr. 188, 1991, S. 23–34

[Quibeldey-Cirkel, 1993a] QUIBELDEY-CIRKEL, Klaus:
Verbundprojekt DASSY: Datentransfer und Schnittstellen in offenen integrierten VLSI-Entwurfssystemen.
BMFT-Forschungsbericht, Technische Informationsbibliothek, Hannover, 1993

[Quibeldey-Cirkel, 1993b] QUIBELDEY-CIRKEL, Klaus:
CAD-Frameworks: Die Probleme jenseits der Entwurfswerkzeuge.
In: mikroelektronik, Jg. 7, H. 2, 1993, S. 72–76

[Quibeldey-Cirkel, 1993c] QUIBELDEY-CIRKEL, Klaus:
CAD-Frameworks: Ergebnisse des DASSY-Projekts.
In: CAD-CAM Report, Jg. 12, H. 8, 1993, S. 16–23

[Quibeldey-Cirkel, 1994a] QUIBELDEY-CIRKEL, Klaus:
Paradigmenwechsel im Software-Engineering: Auf dem Weg zu objektorientierten Weltmodellen.
In: Softwaretechnik-Trends, Jg. 14, H. 1, 1994, S. 47–58

[Quibeldey-Cirkel, 1994b] QUIBELDEY-CIRKEL, Klaus:
Zur Psychologie der Objektorientierung: Warum sind objektorientierte Konzepte intuitiv?
In: [Quibeldey-Cirkel, 1994c]

[Quibeldey-Cirkel, 1994c] QUIBELDEY-CIRKEL, Klaus:
Das Objekt-Paradigma in der Informatik.
Teubner-Verlag, Stuttgart, 1994

[Quillian, 1967] QUILLIAN, M. Ross:
Word Concepts: A Theory and Simulation of Some Basic Semantic Capabilities.
In: [Brachman & Levesque (Hrsg.), 1985], S. 98–118

[Rammig, 1985] RAMMIG, Franz J.:
A Multilevel Cybernetic Model of the Design Process.
In: Methodologies for Computer System Design, Hrsg.: Giloi, W. K.; Shriver, B. D., Elsevier Science Publishers B. V. (North-Holland), IFIP, 1985, S. 87–103

[Rammig (Hrsg.), 1987] RAMMIG, Franz J. (Hrsg.):
Tool Integration and Design Environments.
Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1987

[Rammig, 1989] RAMMIG, Franz J.:
Systematischer Entwurf digitaler Systeme.
Teubner-Verlag, Stuttgart, 1989

[Rammig & Steinmüller, 1992] RAMMIG, Franz J.; STEINMÜLLER, Bernd:
Frameworks und Entwurfsumgebungen.
In: Informatik-Spektrum, Jg. 15, 1992, S. 33–43

[Rechenberg, 1991] RECHENBERG, P.:
Übersetzungen von Informatik-Literatur bekümmert betrachtet.
In: Informatik-Spektrum, Jg. 14, 1991, S. 28–33

[Reimer, 1991] REIMER, Ulrich:
Einführung in die Wissensrepräsentation.
Teubner-Verlag, Stuttgart, 1991

[Reitman, 1965] REITMAN, W. R.:
Cognition and Thought: An Information-Processing Approach.
John Wiley, New York, 1965

[Rhyne, 1989] RHYNE, Tom:
The MCC CAD Framework Laboratory.
In: [EDIF, 1989]

[Rösch, 1993] RÖSCH, Martin:
Object Request Broker: Funktionsweise und erste Erfahrungen.
In: unix/mail, Jg. 11, H. 1, 1993, S. 18–21

[Ropohl, 1980] ROPOHL, Günter:
Modelle im Technikunterricht.
In: Modelle und Modelldenken im Unterricht: Anwendung der Allgemeinen Modelltheorie auf die Unterrichtspraxis, Hrsg: Stachowiak, Herbert, Verlag Julius Klinkhardt, Bad Heilbrunn/Obb., 1980, S. 123–143

[Rosen, 1977] ROSEN, Robert:
Complexity as a System Property.
In: International Journal of General Systems, Jg. 3, 1977, S. 227–232

[Rumbaugh, 1993] RUMBAUGH, James:
Disinherited! Examples of Misuse of Inheritance.
In: Journal of Object-Oriented Programming, Jg. 5, H. 9, 1993, S. 22–24

[Sachsse, 1971] SACHSSE, Hans:
Einführung in die Kybernetik.
F. Vieweg, Braunschweig, 1971

[Saraswat, 1993] SARAWAT, Vijay A.:
Concurrent Constraint Programming.
MIT Press, Cambridge, Mass., 1993

[Schauber & Tauber (Hrsg.), 1982] SCHAUER, Helmut; TAUBER, Michael J. (Hrsg.):
Informatik und Psychologie.
R.-Oldenbourg-Verlag, München, 1982

[Schlageter & Stucky, 1983] SCHLAGETER, Gunter; STUCKY, Wolffried:
Datenbanksysteme: Konzepte und Modelle.
Teubner-Verlag, Stuttgart, 1983

[Schenk, 1990] SCHENK, Douglas:
EXPRESS Language Reference Manual.
McDonnell Aircraft Company, St. Louis, MO., 1990

[Schneider, 1991] SCHNEIDER, Birger:
Testing Tomorrow's Technology.
ESPRIT 2318, Elektronik Centralen, Hørsholm, 1991

[Schneider, 1988] SCHNEIDER, Wolf:
Deutsch für Kenner: Die neue Stilkunde.
Gruner und Jahr, Hamburg, 3. Aufl., 1988

[Schnupp, 1983] SCHNUPP, Peter:
Softwaretechnologie für den kommerziellen Anwender: Bringen die 80er Jahre einen Paradigmenwechsel?.
In: Psychologische Aspekte der Software-Entwicklung, Hrsg: Schelle, H.; Molzberger, Peter, Oldenburg, München-Wien, 1983, S. 156–171

[Schröder, 1992] SCHRÖDER, Uwe:
Erzeugung testerorientierter Signalbeschreibungen aus ereignisbezogenen Daten.
Dissertation, Universität Siegen, 1992

[Schulz, 1991] SCHULZ, Steven E.:
Frameworks: Debunking the Myths.
In: Electronic Design, Jg. 39, H. 16, 1991, S. 71–80

[Schweizer & Thomé, 1993] SCHWEIZER, G.; THOMÉ, Bernhard:
Informatiksystemtechnik (CBSE): Gedanken zu einer Disziplin.
In: Informatik-Spektrum, Jg. 16, 1993, S. 215–221

[Seifart, 1988] SEIFART, Manfred:
Digitale Schaltungen.
Hüthig-Verlag, Heidelberg, 3. Aufl., 1988

[Séquin, 1983] SÉQUIN, Carlo H.:
Managing VLSI Complexity: An Outlook.
In: Proceedings of the IEEE, Jg. 71, H. 1, 1983, S. 149–166

[Shannon & Weaver, 1949] SHANNON, Claude E.; WEAVER, Warren:
A Mathematical Theory of Communication.
University of Illinois Press, Urbana, 1949;
Deutsche Übersetzung von Dreßler, Helmut:
Mathematische Grundlagen der Informationstheorie.
R.-Oldenbourg-Verlag, München, 1976

[Shapiro (Hrsg.), 1987] SHAPIRO, S. C. (Hrsg.):
Encyclopedia of Artificial Intelligence.
John Wiley & Sons, Chichester, England, 1987

[Shaw, 1984] SHAW, Mary:
Abstraction Techniques in Modern Programming Languages.
IEEE Software, Jg. 1, H. 4, 1984, S. 10–26

[Shneiderman, 1980] SHNEIDERMAN, B.:
Software Psychology.
Winthrop Publishers, Cambridge, Mass., 1980

[Siepmann, 1991] SIEPMANN, Ernst:
Entwurfstheorie und Entwurfsdatenmodellierung für CAD-Frameworks.
Dissertation, Universität Kaiserslautern, 1991

[Simon, 1962] SIMON, Herbert A.:
The Architecture of Complexity.
Proc. of The American Philosophical Society, Jg. 106, H. 6, 1962, S. 467–482;
Nachdruck in: General Systems, Jg. 10, 1965, S. 63–76

[Simon, 1976] SIMON, Herbert A.:
Administrative Behavior.
Macmillan, New York, 1976

[Simon, 1982] SIMON, Herbert A.:
The Sciences of the Artificial.
MIT Press, Cambridge, Mass., 2. Aufl., 1982

[Smith & Smith, 1977] SMITH, J. M.; SMITH, D. C. P.:
Database Abstractions: Aggregation and Generalization.
In: ACM Transactions on Database Systems, Jg. 2, H. 2, 1977, S. 105–133

[Snyder, 1986] SNYDER, Alan:
Encapsulation and Inheritance in Object-Oriented Programming Languages.
In: SIGPLAN Notices, Jg. 21, H. 9, 1986, S. 38–45

[Snyder, 1993] SNYDER, Alan:
The Essence of Objects: Concepts and Terms.
In: IEEE Software, Jg. 10, H. 1, 1993, S. 31–42

[Software, 1993] Themenheft:
Making O-O Work.
In: Software, Jg. 10, H. 1, 1993

[Soley (Hrsg.), 1992] OBJECT MANAGEMENT GROUP:
Common Object Request Broker Architecture and Specification.
Hrsg.: SOLEY, R., Framingham, Mass., 1992

[Sommerville, 1990] SOMMERVILLE, Ian:
Software Engineering.
Addison-Wesley Publishing Company, Reading, Mass., 1990

[Sapossnek, 1989] SAPOSSNEK, Mark:
Research on Constraint-Based Design Systems.
Proc. of the Fourth Int. Conf. on the Applications of Artificial Intelligence in Engineering, Comput. Mech. Publications, Southampton, UK, 1989, S. 385–403

[Spinner, 1987] SPINNER, Helmut F.:
Problemlösungsprozesse.
Studienskript der FernUniversität Hagen, 1987

[Stachowiak, 1973] STACHOWIAK, Herbert:
Allgemeine Modelltheorie.
Springer-Verlag, Wien New York, 1973

[Stachowiak, 1983] STACHOWIAK, Herbert:
Erkenntnisstufen zum Systematischen Neopragmatismus und zur Allgemeinen Modelltheorie.
In: Modelle — Konstruktion der Wirklichkeit, Hrsg: Stachowiak, Herbert, Wilhelm-Fink-Verlag, München, 1983, S. 87–146

[Stallman & Sussman, 1977] STALLMAN, R. M.; SUSSMAN, Gerald J.:
Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis.
In: Artificial Intelligence, Jg. 9, H. 2, 1977, S. 135–196

[Steele & Sussman, 1980] STEELE, Guy Lewis; SUSSMAN, Gerald Jay:
CONSTRAINTS: A Language for Expressing Almost-Hierarchical Descriptions.
In: Artificial Intelligence, Jg. 14, 1980, S. 1–39

[Stefik, 1981] STEFIK, Mark:
Planning with Constraints: (MOLGEN: Part1).
In: Artificial Intelligence, Jg. 16, H. 2, 1977, S. 111–139

[Stein *et al.*, 1989] STEIN, Lynn Andrea; LIEBERMAN, Henry; UNGAR, David:
A Shared View of Sharing: The Treaty of Orlando.
In: [Kim & Lochovsky (Hrsg.), 1989], S. 31–48

[Stein, 1993] STEIN, Wolfgang:
Objektorientierte Analysemethoden: ein Vergleich.
In: Informatik-Spektrum, Jg. 16, 1993, S. 317–332

[Steinbauer & Wedekind, 1985] STEINBAUER, Dieter; WEDEKIND, Hartmut:
Integritätsaspekte in Datenbanksystemen.
In: Informatik-Spektrum, Jg. 8, 1985, S. 60–68

[Stroustrup, 1986] STROUSTRUP, Bjarne:
The C++ Programming Language.
Addison-Wesley Publishing Company, Reading, Mass., 1986

[Struß, 1991] STRUSS, Peter (Hrsg.):
Wissensrepräsentation.
R.-Oldenbourg-Verlag, München, 1991

[Sutherland, 1963] SUTHERLAND, Ivan:
Sketchpad: A Man-Machine Graphical Communication System.
In: Proc. of the AFIPS Spring Joint Computer Conference, 1963, S. 329–346

[Thiele, 1991] THIELE, Albert:
Überzeugend Präsentieren: Präsentationstechnik für Fach- und Führungskräfte.
VDI-Verlag, Düsseldorf, 1991

[Thomas & Walker, 1985] THOMAS, Donald E.; WALKER, Robert A.:
A Model of Design Representation and Synthesis.
In: Proc. of the 22nd ACM/IEEE Design Automation Conf., 1985, S. 453–459

[Thomé, 1993] THOMÉ, Bernhard (Hrsg.):
Systems Engineering: Principles and Practice of Computer-Based Systems Engineering.
Wiley, New York, 1993

[Thompson, 1979] THOMPSON, C. D.:
Area-Time Complexity for VLSI.
In: Proc. of the 11th Annual ACM Symposium on the Theory of Computing, Atlanta, Ga, 1979, S. 81–88

[Touretzky, 1986] TOURETZKY, David S.:
The Mathematics of Inheritance Systems.
Morgan Kaufmann Publishers, Inc., Los Altos, Kalifornien, 1986

[Tsang, 1990] TSANG, Jean Patrick:
Constraint Propagation Issues in Automated Design.
In: [Gottlob & Nejdl (Hrsg.), 1990], S. 135–151

[Ullman, 1984] ULLMAN, Jeffrey D.:
Computational Aspects of VLSI.
Computer Science Press, Rockville, Maryland, 1984

[Ungar & Smith, 1987] UNGAR, David.; SMITH, R. B.:
Self: The Power of Simplicity.
In: SIGPLAN Notices, Jg. 22, H. 10, 1987, S. 227–242

[Wahl (Hrsg.), 1992] WAHL, Michael G. (Hrsg.):
EDIF Test View, EDIF V 2 0 301.
Universität Siegen, 1992

[Walker, 1992] WALKER, Ian:
Requirements of an Object-Oriented Design Method.
In: Software Engineering Journal, März 1992, S. 102–113

[Waltz, 1972] WALTZ, David L.:
Generating Semantic Descriptions from Drawings of Scenes with Shadows.
Technical report AI-TR-271, Massachusetts Institute of Technology, Cambridge, Mass., 1972

[Wand, 1989a] WAND, Yair:
A Proposal for a Formal Model of Objects.
In: [Kim & Lochovsky (Hrsg.), 1989], S. 537–559

[Wand, 1989b] WAND, Yair:
An Ontological Foundation for Information Systems Design Theory.
In: Office Information Systems: The Design Process, Hrsg.: Pernici, B.; Verrijn-Stuart, A. A., Elsevier Science Publishers B. V. (North-Holland), IFIP, 1989, S. 201–221

[Wand & Weber, 1989] WAND, Yair; WEBER, Ron:
An Ontological Evaluation of Systems Analysis and Design Methods.
In: Information System Concepts: An In-Depth Analysis, Hrsg.: Falkenberg, E. D.; Lindgreen, P., Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1989, S. 79–107

[Weaver, 1948] WEAVER, Warren:
Science and Complexity.
American Scientist, Jg. 36, 1948, S. 536–544

[Weber, 1985] WEBER, Konrad:
Eine Konzeption zur Automatisierung stratifizierter Entwürfe digitaler Systeme basierend auf sukzessiver Detaillierung objektorientierter Modelle.
Dissertation, TU München, 1985

[Wedekind, 1990] WEDEKIND, Hartmut:
Objektorientierung und Vererbung.
In: Informationstechnik, Jg. 32, H. 2, 1990, S. 79–86

[Wedekind, 1992] WEDEKIND, Hartmut:
Objektorientierte Schemaentwicklung: Ein kategorialer Ansatz für Datenbanken und Programmierung.
BI-Wissenschaftsverlag, Mannheim, 1992

[Wegner, 1987] WEGNER, Peter:
Dimensions of Object-Based Language Design.
In: SIGPLAN Notices, Jg. 22, H. 10, 1987, S. 168–182

[Wegner, 1989] WEGNER, Peter:
Capital-intensive Software Technology.
In: [Biggerstaff & Perlis, 1989], S. 43–97

[Weinberg, 1971] WEINBERG, Gerald M.:
The Psychology of Computer Programming.
Van Nostrand Reinhold Company, New York, 1971

[Weiser Friedman, 1992] WEISER FRIEDMAN, Linda:
From Babbage to Babel and Beyond: A Brief History of Programming Languages.
In: Computer Languages, Jg. 17, H. 1, 1992, S. 1–17

[Wendt, 1993] WENDT, S.:
Defizite im Software-Engineering.
In: Informatik-Spektrum, Jg. 16, 1993, S. 34–38

[White, 1993] WHITE, Stephanie:
Systems Engineering of Computer-Based Systems.
In: Computer, Jg. 26, H. 11, 1993, S. 54–65

[Whitesitt, 1973] WHITESITT, J. Eldon:
Boolesche Algebra und ihre Anwendungen: Logik und Grundlagen der Mathematik.
Vieweg-Verlag, Braunschweig, 2. Aufl., 1973

[Whorf, 1974] WHORF, Benjamin Lee:
Language, Thought and Reality.
deutsche Übersetzung von KRAUSSER, Peter (Hrsg.):
Sprache, Denken, Wirklichkeit: Beiträge zur Metalinguistik und Sprachphilosophie/Benjamin Lee Whorf.
Rowohlt, 1974

[Wiener *et al.*, 1990] WIENER, L.; WILKERSON, B.; WIRFS-BROCK, Rebecca J.:
Designing Object-Oriented Software.
Prentice Hall, New Jersey, 1990

[Wiener, 1948] WIENER, Norbert:
Cybernetics: Or Control and Communication in the Animal and the Machine.
MIT Press, Cambridge, Mass., 1948

[Wilkerson & Wirfs-Brock, 1989] WILKERSON, B.; WIRFS-BROCK, Rebecca J.:
A Responsibility-Driven Approach.
In: SIGPLAN Notices, Jg. 24, H. 10, 1989, S. 71–76

[Wilkes, 1988] WILKES, Wolfgang:
Instance Inheritance Mechanisms for Object-Oriented Databases.
In: Int. Workshop on Object-Oriented Databases, Lecture Notes in Computer Science, Springer-Verlag, Bd. 334, 1988

[Wilkes (Hrsg.), 1992] WILKES, Wolfgang (Hrsg.):
IMG Information Model of EDIF Version 2 0 0, Version 1.0.
FernUniversität Hagen, 1992

[Winblad *et al.*, 1990] WINBLAD, Ann L.; EDWARDS, Samuel D.; KING, David R.:
Object-Oriented Software.
Addison-Wesley Publishing Company, Reading, Mass., 1990

[Winograd, 1975a] WINOGRAD, Terry:
Breaking the Complexity Barriere, Again.
In: ACM SIGPLAN Notices, Jg. 10, H. 1, S. 13–30

[Winograd, 1975b] WINOGRAD, Terry:
Frame Representations and the Declarative/Procedural Controversy.
In: [Brachman & Levesque (Hrsg.), 1985], S. 358–370

[Winston, 1987] WINSTON, Patrick Henry:
Künstliche Intelligenz.
Addison-Wesley-Verlag, Bonn, 1987

[Wirth, 1971] WIRTH, Niklas:
Program Development by Stepwise Refinement.
In: Communications of the ACM, Jg. 14, H. 4, 1971

[Wirth, 1983] WIRTH, Niklaus:
Algorithmen und Datenstrukturen.
Teubner-Verlag, Stuttgart, 1983

[Wirth, 1985] WIRTH, Niklaus:
Programming in Modula-2.
Springer-Verlag, Berlin Heidelberg, 1985

[Wittgenstein, 1973] WITTGENSTEIN, Ludwig:
Tractatus logico-philosophicus: Logisch-philosophische Abhandlungen.
Suhrkamp, Frankfurt a. M., 9. Aufl., 1973

[Wojtkowiak, 1988] WOJTKOWIAK, Hans:
Test und Testbarkeit digitaler Schaltungen.
Teubner-Verlag, Stuttgart, 1988

[Wojtkowiak, 1989] WOJTKOWIAK, Hans:
Entwurfsmethodik für komplexe digitale Systeme.
Vorlesungsmanuskript, Universität Siegen, 1989

[Wojtkowiak, 1990] WOJTKOWIAK, Hans:
Von 0 und 1 zur Künstlichen Intelligenz.
In: Diagonal, Jg. 1, H. 1, 1990, S. 107–114

[Wunderlich, 1991] WUNDERLICH, Hans-Joachim:
Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test.
Springer-Verlag, Berlin Heidelberg, 1991

[Yokoyama, 1990] YOKOYAMA, Takanori:
An Object-Oriented and Constraint-Based Knowledge Representation System for Design Object Modeling.
In: Proc. of the Sixth Conf. on Artificial Intelligence Applications, IEEE, 1990, S. 146–152

[Young, 1988] YOUNG, Jeffrey S:
Steve Jobs: The Journey is the Reward.
Scott, Foresman and Company, 1988

[Yourdon, 1988] YOURDON, Edward:
Managing the Structured Techniques.
4. Auflage, Prentice Hall, New Jersey, 1988

[Zadeh, 1965] ZADEH, Lotfi A.:
Fuzzy Sets.
In: Information and Control, Jg. 8, 1965, S. 338–353

[Zemanek, 1991] ZEMANEK, Heinz:
Weltmacht Computer, Weltreich der Information.
Bechtle-Verlag, Esslingen München, 1991

[Zemanek, 1992] ZEMANEK, Heinz:
Das geistige Umfeld der Informationstechnik.
Springer-Verlag, Berlin Heidelberg, 1992

[Zuse, 1993] ZUSE, Konrad:
Der Computer: Mein Lebenswerk.
Springer-Verlag, Berlin, 3. Aufl., 1993