Betriebssysteme

WS 1997/98

Prof. Dr. P.P. Spies

Vorlesungsskript

Vorläufige Fassung

Stand vom 13.5.1998

Inhaltsverzeichnis

1 Einführung **5**

1.1 Betriebssystem: das Management eines Rechensystems 5
1.2 Entwicklungsgeschichtliche Einordnung 7
1.3 Inhalt der Vorlesung 8

Teil I: Klassische Betriebssysteme — Grundlagen **9**

2 Grundlegende Eigenschaften einfacher paralleler Systeme **11**

2.1 Systeme mit atomaren Prozessen 11
2.1.1 Uninterpretierte Systeme 13
2.1.2 Interpretierte Systeme 15
2.2 Determinierte Systeme mit atomaren Prozessen 18
2.3 Systeme mit symmetrischer Konfliktvermeidung — Freier wechselseitiger Ausschluß 21
2.3.1 Serielle und serialisierbare Systeme 23
2.3.2 Relativ determinierte Systeme 24
2.4 Systeme mit antisymmetrischer Konfliktvermeidung — Geordnete Abhängigkeiten 27
2.5 Zusammenfassung und Einordnung 30

3 Einfache Systeme mit sequentiellen Prozessen **33**

3.1 Systeme mit sequentiellen Prozessen 33
3.2 Die Grundstruktur für Realisierungen 38
3.3 Koordinierungskonzepte und –mechanismen 41
3.3.1 Freier wechselseitiger Ausschluß 41
3.3.1.1 Koordinierungsaufgabe und Lösungsmöglichkeiten 42
3.3.1.2 Eine Lösung mit Speicherzellen als Basis 44
3.3.1.3 Zählende Semaphore 44
3.3.1.4 Wechselseitiger Ausschluß für das System $\mathcal{R}$ 48
3.3.1.5 Zentrale Lösungen 50
3.3.2 Synchronisation 53
3.3.2.1 Synchronisation gemäß der Vorrangrelation 53
3.3.2.2 Ein Nachrichtenpuffer 57
3.3.3 Koordinierungsverfahren und weitere Mechanismen 60

4 Betriebsmittel und Betriebsmittel–Management **63**

4.1 Betriebsmittel und Betriebsmittelarten 64
4.1.1 Betriebsmittelarten nach Lebenszeiten und Einsatzmöglichkeiten . . . 64
4.1.2 Betriebsmittelarten und Koordinierung 67

4.2 Systeme mit frei exklusiv benutzbaren Betriebsmitteln 72
4.2.1 Das Modellsystem 72
4.2.2 BM–Zulässigkeit und Koordinierung 73
4.2.3 Erweitertes BM–Management: Unterbrechbare BM 75
4.2.4 Deadlock–Zustände 78
4.2.5 Deadlock–Vermeidung 80
4.2.5.1 Kaskadenförmiger BM–Einsatz 81
4.2.5.2 Geordneter BM–Einsatz 82
4.2.6 Deadlock–Verhinderung 83
4.2.6.1 Das Bankier–Verfahren 84
4.2.6.2 Optimistische Deadlock–Verhinderung mit vorbeugenden Maßnahmen 84
4.2.7 Deadlock–Erkennung und –Behebung 86
4.3 Erzeuger–Verbraucher–Systeme 89
4.4 Zusammenfasssung und Einordnung 91

Teil II: Klassische Betriebssysteme — Realisierungen 93

5 Die Grundstruktur eines zentralen, parallelen Systems 95
5.1 Ein einfaches System mit Prozessen und einem Kern 96
5.1.1 Komponenten mit Rechen– und Speicherfähigkeiten 96
5.1.1.1 Eine einfache HK 96
5.1.1.2 Angepaßte Prozesse 97
5.1.1.3 Der Kern mit eingeordneter HK als gemeinsame Basis der Prozesse 98
5.1.2 Realisierung der Prozeß–Rechen– und Speicherfähigkeiten 100
5.1.2.1 Die Prozesse und die HK für Realisierungen 100
5.1.2.2 Realisierung der Prozeßrechenfähigkeiten: Das Zustandsdiagramm 104
5.1.2.3 Realisierung der Prozeßrechenfähigkeiten: Scheduling 109
5.1.2.4 Realisierung der Prozeß–Speicherfähigkeiten 119
5.2 Erweiterungen zu einem anwendungsgeeigneten Mehrbenutzersystem 122

6 Speicher– und EA–Management 125
6.1 Die statischen Speicherfähigkeiten — Das Dateisystem der persistenten Datenobjekte 126
6.1.1 Bytefolgen–Dateien 127
6.1.2 Verzeichnisdateien und der Dateienbaum 129
6.1.3 Der Baum der persistenten Datenobjekte 133
6.2 Prozesse mit virtuellen Adreßräumen und Speichern 138
6.2.1 Die prozeßprivaten Wirkungsbereiche 139
6.2.2 Realisierungen der *VMs* 142
6.2.3 Seitenbasierte *VM*–Realisierungen 144
6.3 Strategien seitenbasierter Speicherverwaltung 146
6.3.1 Prozeßlokale Einzelseiten–Anforderungsstrategien 148
6.3.2 Prozeßglobale, anpassungsfähige Strategien 152

Kapitel 1

Einführung

In diesem einführenden Kapitel werden folgende Fragen vorläufig beantwortet:

- Was sind Betriebssysteme?
- Warum befassen wir uns mit Betriebssystemen?
- Welche Themen werden in dieser Vorlesung behandelt?

1.1 Betriebssystem: das Management eines Rechensystems

Die Systeme, mit denen wir uns befassen, sind Rechensysteme. Das sind technische Systeme, die

- aus Hardware und aus Software bestehen,
- dazu fähig sind, Information zu speichern und zu verarbeiten, und
- Anwendern als leistungsfähige Hilfsmittel für Lösungen ihrer entsprechenden Aufgaben zur Verfügung gestellt werden und dienen sollen.

Das Betriebssystem ist der die Nutzungsmöglichkeiten bestimmende und prägende Teil eines Rechensystems. Seine Rolle und seine Aufgaben lassen sich zusammenfassend wie folgt charakterisieren: **Das Betriebssystem ist das Management eines Rechensystems**.
Die Bedeutung dieser Charakterisierung ergibt sich aus den allgemeinen Eigenschaften von Systemen und aus den spezifischen Eigenschaften von Rechensystemen.
Ein **System** ist ein für einen bestimmten Zweck gebildeter, zu einer abgegrenzten, vollständigen Einheit zusammengefaßter Ausschnitt aus einem Universum.
Dieser allgemeine Systembegriff lehrt, wie man systematisch mit komplizierten und komplexen Gegebenheiten — einem Universum — umgehen soll. Der erste Schritt besteht darin, einen vom jeweiligen Zweck bestimmten Ausschnitt zu bilden. Das Ergebnis ist eine Zerlegung des Universums in zwei abgegrenzte Teile, in das System $\mathcal{R}$ und die Umgebung $U(\mathcal{R})$. Mit dieser Zerlegung ergeben sich zwei einander ergänzende Sichten auf das System $\mathcal{R}$:

- Die **Black–Box–Sicht** von $\mathcal{R}$: Sie zeigt $\mathcal{R}$ als Einheit in der Umgebung $U(\mathcal{R})$ und definiert innen und außen für $\mathcal{R}$; diese **BB–Sicht** dient dazu, Abhängigkeiten zwischen $\mathcal{R}$ und $U(\mathcal{R})$ zu klären.

- Die **Glass–Box–Sicht** von $\mathcal{R}$: Sie zeigt die Zusammensetzung von $\mathcal{R}$ aus Komponenten und Abhängigkeiten zwischen diesen Komponenten; diese **GB–Sicht** dient dazu, die (inneren) Eigenschaften von $\mathcal{R}$ zu klären.

Die weiteren Schritte zum systematischen Umgang mit komplizierten und komplexen Gegebenheiten sind Verfeinerungsschritte für $\mathcal{R}$ und $U(\mathcal{R})$, welche die Rekursivität des Systembegriffs einsetzen und nutzen.

Mit diesen allgemeinen Eigenschaften sei jetzt $\mathcal{R}$ mit $U(\mathcal{R})$ ein **Rechensystem**. Dann ergibt sich daraus, daß die Fähigkeiten von $\mathcal{R}$ Anwendern zur Verfügung zu stellen sind:

- $\mathcal{R}$ ist ein **offenes** System mit Abhängigkeiten und Wechselwirkungen zwischen $\mathcal{R}$ und Anwendern in der Umgebung $U(\mathcal{R})$.

Für das offene System $\mathcal{R}$ ergibt sich, daß zum Rand von $\mathcal{R}$ wesentlich die $\mathcal{R}$–$U(\mathcal{R})$–**Schnittstelle** gehört; sie ist die Vermittlerin der Nutzungsmöglichkeiten von $\mathcal{R}$ und die Basis für Interaktionen zwischen $\mathcal{R}$ und Anwendern in $U(\mathcal{R})$.
Mit den spezifischen Fähigkeiten von $\mathcal{R}$, Information speichern und verarbeiten zu können, ergibt sich weiter:

- $\mathcal{R}$ ist ein **dynamisches System**, das dazu fähig ist, Berechnungen auszuführen; die Nutzung dieser Fähigkeiten erfolgt dadurch, daß $\mathcal{R}$ mit Berechnungen Information verarbeitet und gewinnt.

Für das dynamische System $\mathcal{R}$ ist das **Verhalten** ausschlaggebend. Dieses Verhalten wird durch die Berechnungen, die $\mathcal{R}$ ausführt, bestimmt. Die Zweckbestimmung von $\mathcal{R}$ wird in dem Maß erreicht, in dem das Verhalten entsprechend, also anwendungsgeeignet und auf die Lösungen von Anwendungsproblemen ausgerichtet, gestaltet wird. Für Anwender ist dabei das an der $\mathcal{R}$–$U(\mathcal{R})$–Schnittstelle beobachtbare Verhalten, das **EA–Verhalten** von $\mathcal{R}$, von primärem Interesse.
Für die GB–Sicht wurde schon festgestellt, daß $\mathcal{R}$ aus Hardware und aus Software besteht; sie zeigt das (dynamische) Netz mit den Komponenten der Hardware– und der Softwarekonfiguration. Für die Nutzbarkeit der Fähigkeiten von $\mathcal{R}$ spielen die Komponenten der Hardware– und der Softwarekonfiguration unterschiedliche, einander ergänzende Rollen: Die Hardwarekonfiguration ist notwendig dafür, daß Information tatsächlich gespeichert und Berechnungen tatsächlich ausgeführt werden können; ihre Komponenten sind jedoch wegen ihrer technisch bedingten Eigenschaften nicht zur unmittelbaren Nutzung für Anwender geeignet. Ihre Nutzbarkeit für Anwender muß dadurch erreicht werden, daß sie mit den Komponenten der Softwarekonfiguration geeignet gekapselt und damit mittelbar nutzbar werden. Diese Kapselung muß vielstufig so erfolgen, daß schließlich mit der $\mathcal{R}$–$U(\mathcal{R})$–Schnittstelle anwendungsgeeignete Eigenschaften und anwendungsgeeignetes EA–Verhalten erreicht wird.
Für die **GB–Sicht von** $\mathcal{R}$ ergibt sich damit ein **hierarchisches** (dynamisches) **Netz** mit Softwarekomponenten, welche die Hardwarekomponenten der Basis kapseln. Die Komponenten, die mit der $\mathcal{R}$–$U(\mathcal{R})$–Schnittstelle zur Verfügung gestellt werden, sind **abstrakte, rechenfähige Maschinen**, die der Hierarchie entsprechend mit Komponenten der Hardwarekonfiguration realisiert werden.
Damit ergibt sich eine erste Übersicht über die Eigenschaften des Rechensystems $\mathcal{R}$ und die **Aufgaben des Betriebssystems** von $\mathcal{R}$:

- Die Festlegung der Konzepte für die abstrakten Maschinen, die mit der $\mathcal{R}$–$U(\mathcal{R})$–Schnittstelle zur Verfügung gestellt werden sollen.

- Die Bereitstellungen der jeweils benötigten abstrakten Maschinen.

- Die Realisierung dieser Maschinen und die Gewährleistung der vollständigen, korrekten und effizienten Ausführung ihrer Berechnungen.

Diese grobe Übersicht erklärt die bestimmende und prägende Rolle, welche das Betriebssystem für die Nutzungsmöglichkeiten eines Rechensystems spielt. Sie liefert zugleich die Ausgangsbasis für die notwendigen Präzisierungen der Aufgaben eines Betriebssystems und für schrittweise Erklärungen der Vielfalt und der Vielgestaltigkeit dieser Aufgaben.
Orientierung für den Weg, der von einfachen zu komplizierteren Rechen– und Betriebssystemen führt, ergibt sich, indem sequentielle und parallele Berechnungen sowie Rechensysteme für einen oder für mehrere Benutzer unterschieden werden.
Der einfachste Fall, der sich ergibt, ist ein **sequentielles Einbenutzersystem**. Als abstrakte Maschine wird dabei ein sequentieller Prozeß zur Verfügung gestellt, mit dem der Benutzer sein System nutzen kann. Mit dem Konzept für sequentielle Prozesse kann der Übergang von einem sequentiellen zu einem parallelen System dadurch erfolgen, daß Prozesse erzeugt und aufgelöst werden, so daß mehrere Prozesse koexistieren können. Damit ergibt sich ein **paralleles Einbenutzersystem**; es bleibt das private System eines Benutzers; seine Nutzungsmöglichkeiten werden jedoch verallgemeinert.
Alternativ hierzu kann ein Rechensystem mit dem Konzept für sequentielle Prozesse mehreren Benutzern zur Verfügung gestellt werden, indem jeder Benutzer einen sequentiellen Prozeß als seine Benutzermaschine erhält. Damit ergibt sich ein **paralleles Mehrbenutzersystem**; das ist ein System, das gleichzeitig von mehreren Benutzern zunächst für sequentielle Berechnungen genutzt werden kann. Auch diese Systeme können weiter verallgemeinert werden.
Neben den beiden angegebenen Kriterien gibt es zahlreiche weitere; mit ihnen ergibt sich die Vielfalt der Rechen– und Betriebssysteme, die heute für zahlreiche Anwendungsbereiche im Einsatz sind.

1.2 Entwicklungsgeschichtliche Einordnung

Rechen– und Betriebssysteme haben eine über 40–jährige Entwicklungsgeschichte; die Entwicklungen erfolgen als evolutionärer Prozeß, der Impulse von den Fortschritten der Hardwaretechnologien, von den ständig wachsenden Anforderungen der Anwendungen, von den Fortschritten der Softwaretechnik sowie vom wachsenden Verständnis der Aufgabenstellungen, anwendungsgeeignete, leistungsfähige Rechen– und Betriebssysteme zu entwickeln, erhält. Ein Ende dieser Entwicklungen ist nicht abzusehen; der heute erreichte Stand der Hardware– und der Softwaretechnik eröffnet vielmehr weitreichende Perspektiven für die Zukunft.
Der evolutionäre Prozeß der Entwicklung von Rechen– und Betriebssystemen ist einerseits dadurch gekennzeichnet, daß sich mit dem jeweils erreichten Stand Herausforderungen und neue Möglichkeiten für Weiterentwicklungen ergeben. Er hat andererseits zur Folge, daß die Eigenschaften der jeweils verfügbaren Systeme Überlagerungen der Entwicklungsergebnisse im Laufe der Zeit sind; das liegt daran, daß die Entwicklungsergebnisse langlebige Produkte sind, die mit großem Aufwand und mit hohen Kosten entwickelt wurden. Das Vorhandensein und die Einordnung dieser Überlagerungen gehört wesentlich zum Verständnis von Rechen– und Betriebssystemen.
In der rund 40–jährigen Entwicklungsgeschichte von Rechen– und Betriebssystemen können grob vier Epochen unterschieden werden.

Die **erste Epoche** beginnt mit den ersten arbeitsfähigen Rechnern um 1950 und reicht bis etwa 1975. Die ersten Rechner, die zur Verfügung standen, lassen sich dadurch charakterisieren, daß jeweils ein Benutzer mit der zunächst leeren Maschine arbeitete; das sagt insbesondere, daß es keine Software gab. Von diesen Anfängen ausgehend entstanden in dieser Epoche schrittweise die leistungsfähigen Systeme, deren Nachfolger jetzt als mainframes bezeichnet werden. Um 1955 entstanden die ersten Betriebssysteme als Steuerprogramme für einfachen Stapelbetrieb; sie wurden zu Betriebssystemen für Time–Sharing– und für Mischbetrieb mit den bis heute wichtigen Konzepten weiterentwickelt. Charakteristisch für die ganze Epoche war die teure Hardware, die es möglichst effizient zu nutzen galt.
Die **zweite Epoche** ist gekennzeichnet durch die Einführung von Minirechnern um 1965. Diese Entwicklung wurde weitergeführt mit der Einführung der Mikrorechner um 1970, die kennzeichnend für die **dritte Epoche** sind. Mini– und Mikrorechner brachten den Durchbruch für die Verbreitung von Rechensystemen zunächst durch Hardware mit geringen Kosten. Für die Software verlief die Entwicklung zunächst analog zu der für die Rechner der ersten Epoche; dann wurden die Ergebnisse der Entwicklungen für die Rechner der ersten Epoche unter Anpassung an die Hardware auf die Minis und Mikros übertragen. Insbesondere entstand UNIX in der ersten Hälfte der 70er Jahre für den Minirechner PDP11 der Fa. DEC, der damals weit verbreitet war.
Für die Betriebssysteme, welche für die Rechner der drei angegebenen Epochen entwickelt wurden, ist kennzeichnend, daß einerseits bis heute wichtige Konzepte entstanden, daß diese jedoch überwiegend mit ad–hoc–Lösungen realisiert wurden. Das Verständnis für die Grundlagenprobleme, die bei diesen Entwicklungen zu lösen sind, wuchs wesentlich langsamer, und diese Lücke ist bis heute noch nicht geschlossen.
Die **vierte Epoche** ist gekennzeichnet durch die Vernetzung von Rechnern. Sie begann etwa 1975 und dauert mit dem Übergang von vernetzten zu verteilten Rechensystemen etwa ab 1980 bis heute an. Die Pionierarbeiten für Rechnernetze wurden Ende der 60er bis Anfang der 70er Jahre mit der Entwicklung des Arpanet geleistet. Von den Erfolgen dieses Netzes gingen die Impulse für die Entwicklungen der vernetzten Systeme aus, mit denen wir heute arbeiten.

1.3 Inhalt der Vorlesung

In der Vorlesung werden die klassischen Betriebssysteme eingehend behandelt. Das sind die zentralen, prozedurorientierten Systeme, wobei zentral die Verwendung zentraler Hardwarekonfigurationen als Basis bedeutet. Für diese Systeme werden die grundlegenden Konzepte paralleler Mehrbenutzersysteme, die Lösungen der Grundlagenprobleme und die Realisierungen erklärt.
Mit der Behandlung dieser Systeme werden ihre Stärken und ihre Schwächen erkennbar. Diese Schwächen können durch den Übergang zu prozeßorientierten Systemen (teilweise) überwunden werden. Für prozeßorientierte Systeme werden die grundlegenden Konzepte erklärt. Zudem wird der Übergang von prozeßorientierten Systemen zu vernetzten und verteilten Systemen in seinen Grundzügen behandelt.

Teil I
Klassische Betriebssysteme — Grundlagen

Klassische Betriebssysteme sind die Betriebssysteme paralleler Mehrbenutzersysteme. Es sind prozedurorientierte Betriebssysteme mit eingeschränkten Fähigkeiten für parallele Berechnungen und mit einer zentralen Hardwarekonfiguration als Realisierungsbasis. Die Konzepte, Mechanismen und Verfahren, die klassische Betriebssysteme verwenden, sind grundlegend für alle weiteren Arten von Betriebssystemen. Klassische Betriebssysteme sind (mit Überlagerungen) weit verbreitet; sie sind relativ einfach und sie sind gut dafür geeignet, Grundlagenprobleme paralleler Systeme und deren Lösungsmöglichkeiten zu erklären.
Ein klassisches Betriebssystem besteht i.w. aus drei Teilen, dem Prozeßmanagement, dem Dateimanagement und dem Betriebssystemkern. Das Prozeßmanagement ist in den Kern so integriert, daß Prozesse als abstrakte, rechenfähige Maschinen erzeugt und aufgelöst werden können; mit den jeweiligen Prozessen erhält ein System seine nutzbaren Rechenfähigkeiten. Das Dateimanagement ist ebenfalls in den Kern integriert; mit ihm werden Dateien als persistente Speicher des Systems mittelbar mit Kerndiensten nutzbar zur Verfügung gestellt. Abbildung 1.1 gibt einen Überblick über ein Rechensystem mit einem klassischen Betriebssystem.

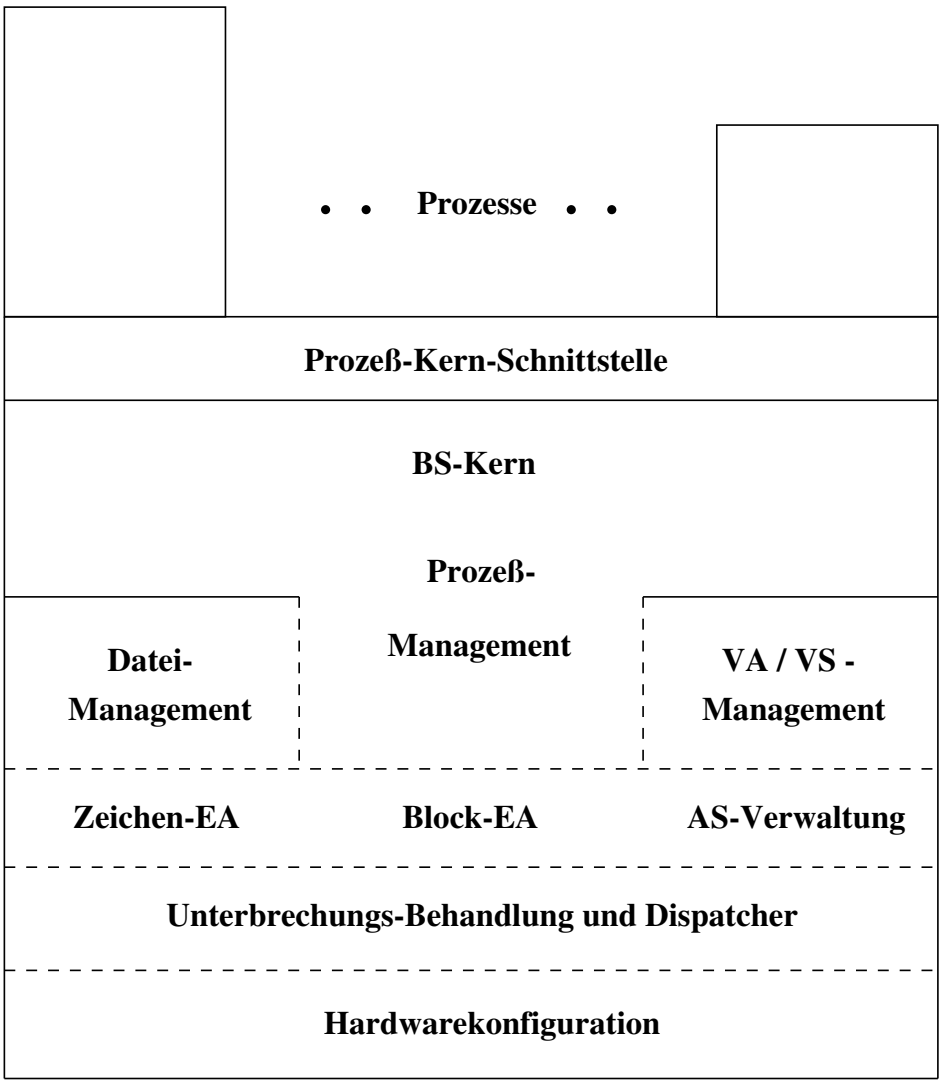


Abbildung 1.1: Ein System mit prozedurorientiertem Kern

Kapitel 2

Grundlegende Eigenschaften einfacher paralleler Systeme

Abbildung 1.1 zeigt einen Schnappschuß von einem Rechensystem mit einem klassischen Betriebssystem. Wenn man diesen Schnappschuß geeignet vergröbert, ergibt sich ein System, das aus einer Menge von Prozessen und dem Kern besteht; Abbildung 2.1 zeigt diesen vergröberten Schnappschuß.

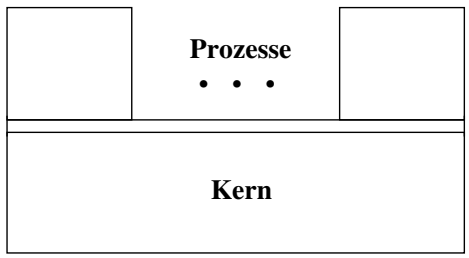


Abbildung 2.1: Ein Schnappschuß von einem parallelen System

Von dem Schnappschuß ausgehend, den die Abbildung zeigt, kann man einfache formale Modelle konstruieren, die dazu geeignet sind, Grundlagenprobleme paralleler Systeme zu formulieren und zu lösen. Diese Modelle liefern die Kriterien, nach denen Betriebssysteme zu konstruieren sind.

2.1 Systeme mit atomaren Prozessen

Die Modelle, die benutzt werden, erfassen das für parallele Systeme Wesentliche möglichst einfach und präzise so, daß die Eigenschaften eines Systems zunächst spezifiziert und dann verifiziert werden können.
Ein paralleles System wird konstruiert, indem Bausteine komponiert werden. Dazu sind

- die Bausteine mit ihren Eigenschaften festzulegen,
- die Komposition der Bausteine zum System festzulegen und die Eigenschaften des Systems zu spezifizieren und
- schließlich die Übereinstimmung der Systemeigenschaften mit den spezifizierten nachzuweisen.

Für die einfachen, hier betrachteten Systeme sind die Bausteine Prozesse als Berechnungsbausteine und Datenobjekte als Speicher– oder Wirkungsbereichsbausteine.
Für die einfachen **Datenobjekte** sind die Eigenschaften durch Zustandsräume und durch Lese– und Schreiboperationen festgelegt.
Die einfachen **Prozesse** werden als atomar, genauer als atomar in der Wirkung vorausgesetzt.
Die Eigenschaft der **Atomarität** ist eine wichtige, methodische Maßnahme, die zur Vereinfachung von Problemlösungen dient; sie schafft Einheiten durch Vergröberungen zu BBs, die dann mit ihren festgelegten Eigenschaften als Bausteine für weitere Konstruktionen durch Kompositionen benutzt werden können. Berechnungen haben Eigenschaften in den beiden Dimensionen Raum und Zeit:

- in der Dimension Raum werden Speicher als Wirkungsträger und die Wirkungen von Berechnungen beschrieben;

- in der Dimension Zeit wird die Ausführungsdauer von Berechnungen beschrieben.

Atomarität kann in beiden Dimensionen festgelegt werden. Atomarität in der Wirkung oder **w–Atomarität** bedeutet, daß von der Wirkung einer Berechnung allein die EA–Wirkung festgelegt und bekannt ist. Im nicht w–atomaren Fall sind zusätzlich Wirkungsdetails der Berechnungsausführung festgelegt. Atomarität in der Zeit oder **t–Atomarität** bedeutet, daß die Ausführungsdauer einer Berechnung 0 ist. Im nicht t–atomaren Fall ist die Ausführungsdauer > 0 (und endlich). Atomarität ist eine künstliche, methodische Maßnahme. Wenn mit ihr eine Konstruktion durchgeführt ist, und das konstruierte System dann realisiert werden soll, muß die Atomaritätsannahme für die Realisierung durchgesetzt werden.
Die Prozesse der Modelle für parallele Systeme, die wir verwenden, sind w–atomar; sie sind nicht t–atomar.
Die **Eigenschaften der w–atomaren Prozesse** werden wie folgt festgelegt:

- Ein Prozeß p hat einen Wirkungsbereich W_p; das ist die (vollständige) Menge der Datenobjekte, die zur Ausführung der p–Berechnung benötigt werden. W_p besteht aus einem Vorbereich V_p und einem Nachbereich N_p mit $W_p = V_p \cup N_p$.

- Die Wirkung der p–Berechnung wird mit einer Rechenfunktion R_p festgelegt; das ist eine Abbildung vom Zustandsraum der V_p in den Zustandsraum der N_p.

- Das Verhalten von p, also die Ausführung der p–Berechnung, wird mit zwei Ereignissen beschrieben,

  - dem Initiierungsereignis $\underline{p}$, mit dem die Berechnung beginnt, und
  - dem Terminierungsereignis $\overline{p}$, mit dem die Berechnung endet.

  Die Ereignisse $\underline{p}$ und $\overline{p}$ sind w– und t–atomar. Der Semantik der Ereignisse entsprechend muß gelten: $\underline{p}$ vor $\overline{p}$.

Nach diesen Vorbereitungen werden im folgenden die Systeme mit w–atomaren Prozessen definiert und erklärt; dabei werden uninterpretierte und interpretierte Systeme unterschieden.

2.1.1 Uninterpretierte Systeme

Für ein uninterpretiertes System werden die Datenobjekte der Wirkungsbereiche der Prozesse festgelegt; die Wirkungen der Berechnungen der Prozesse bleiben unbekannt.
Die Eigenschaften eines uninterpretierten Systems werden wie folgt festgelegt:

- durch die Menge D der Datenobjekte des Systems mit D nichtleer, endlich, linear geordnet;
- durch die Menge P der Prozesse des Systems mit P nichtleer und endlich;
- durch die linkstotale Abbildung $V : P \to POT(D) - \{\emptyset\}$, die für jedes p mit V_p die Datenobjekte des Vorbereichs der Berechnung von p festlegt;
- durch die linkstotale Abbildung $N : P \to POT(D) - \{\emptyset\}$, die für jedes p mit N_p die Datenobjekte des Nachbereichs der Berechnung von p festlegt;
- durch die 2–stellige Konfliktvermeidungsrelation $\kappa \subseteq P \times P$ des Systems.

Zusammengefaßt wird also ein **uninterpretiertes System mit einem 5–tupel**

$$\mathcal{A} = (D, P, \kappa, V, N)$$

definiert.
$\mathcal{A}$ definiert ein paralleles System als Kombination seiner Prozesse. Bei gegebenen Prozessen soll $\mathcal{A}$ Berechnungen ausführen, die Kombinationen der Berechnungen der $p \in P$ sind. Von $\mathcal{A}$ interessieren letztlich das Verhalten und die Wirkungen der Berechnungen; diese Wirkungen müssen spezifiziert werden. Das erfolgt zunächst dadurch, daß die Berechnungen der $p \in P$ kombiniert werden. Mit diesen Kombinationen können sich Konflikte ergeben, die **Interferenzen** der Wirkungen der Prozeßberechnungen zur Folge haben; wenn das der Fall ist, berechnet $\mathcal{A}$ Unsinn. Zur Vermeidung dieser Konflikte und Interferenzen können die Kombinationen der Prozesse mittels der **Konfliktvermeidungsrelation** κ eingeschränkt werden.
Vorerst wird angenommen, daß $\kappa = \emptyset$ gilt: $\mathcal{A}$ ist also eine einschränkungs–**freie Kombination** der $p \in P$.
Für das System $\mathcal{A} = (D, P, \emptyset, V, N)$ ist zu klären, wie sein **Verhalten**, also seine Dynamik bei Ausführung von Berechnungen, beschrieben werden soll. Das erfolgt auf der Grundlage des Verhaltens der Prozesse und der festgelegten, freien Kombination der Prozesse durch **Ereignisspuren** von $\mathcal{A}$. Das sind

> Folgen $\eta = \varepsilon^1 \ldots \varepsilon^l$ der Länge l mit $0 \leq l \leq 2|P|$,

für die gelten:

- die Folgenelemente sind die den $p \in P$ zugeordneten Ereignisse, wobei jedes dieser Ereignisse höchstens 1–mal auftritt;
- für alle $p \in P$ und $j \in \{1, \ldots, l\}$: $\varepsilon^j = \overline{p} \Rightarrow \exists i \in \{1, \ldots, l\}(i < j \wedge \varepsilon^i = \underline{p})$.

Sei $\eta = \varepsilon^1 \ldots \varepsilon^l$ eine Ereignisspur von $\mathcal{A}$. Wenn $l < 2|P|$ gilt, ist η eine **partielle** oder **unvollständige Ereignisspur**; sie beschreibt eine partiell ausgeführte Berechnung von $\mathcal{A}$, die noch fortzusetzen ist. Wenn $l = 2|P|$ gilt, ist η eine **vollständige Ereignisspur** von $\mathcal{A}$ mit den Berechnungen aller $p \in P$.
Für die **Beschreibung des Verhaltens** von $\mathcal{A}$ durch Ereignisspuren gilt folgendes:

- Eine (vollständige) Ereignisspur η von $\mathcal{A}$ beschreibt die Dynamik von $\mathcal{A}$ aus der Sicht eines Beobachters, der die eingetretenen Ereignisse in einer Folge sieht; die lineare Ordnung der Folge η entspricht einer **abstrakten Zeitskala des Beobachters**.

- Von dieser linearen Ordnung wohl zu unterscheiden sind die Gesetzmäßigkeiten, nach denen die Berechnungen von $\mathcal{A}$ ausgeführt werden. Für sie gelten die Gesetze $\underline{p}$ vor $\overline{p}$ für jedes $p \in P$ und die freie Kombination der Berechnungen der Prozesse; das ist eine **partielle Ordnung** über den Ereignissen von $\mathcal{A}$. Das Verhalten von $\mathcal{A}$ wird demnach durch die **Gesamtheit der vollständigen Ereignisspuren** H von $\mathcal{A}$ beschrieben. Die echt partielle Ordnung ist das Charakteristische eines parallelen Systems.

Beispiel

Für das System $\mathcal{A}$ sei die Prozeßmenge $P = \{p, q\}$.
Dann ergeben sich für $\mathcal{A}$ die folgenden vollständigen Ereignisspuren:

$$\begin{array}{lcllcl} \eta_1 & = & \underline{p}\,\overline{p}\,\underline{q}\,\overline{q}\,; & \eta_4 & = & \underline{q}\,\overline{q}\,\underline{p}\,\overline{p}\,; \\ \eta_2 & = & \underline{p}\,\underline{q}\,\overline{p}\,\overline{q}\,; & \eta_5 & = & \underline{q}\,\underline{p}\,\overline{q}\,\overline{p}\,; \\ \eta_3 & = & \underline{p}\,\underline{q}\,\overline{q}\,\overline{p}\,; & \eta_6 & = & \underline{q}\,\underline{p}\,\overline{p}\,\overline{q}\,; \end{array}$$

⌋

Ein System $\mathcal{A} = (D, P, \emptyset, V, N)$ ist ein paralleles System, wenn $|P| \neq 1$ gilt. Für das Verhalten von $\mathcal{A}$ gibt es viele Möglichkeiten; die **Verhaltenscharakteristika** von $\mathcal{A}$ lassen sich mit den Ereignisspuren präzise definieren:
Für $\mathcal{A}$ mit P und $\eta = \varepsilon^1 \ldots \varepsilon^l$:

- Seien $p \in P$ und $k \in \{1, \ldots, l\}$. Der Prozeß p **ist aktiv in** ε^k

  $\Leftrightarrow$ $\eta' = \varepsilon^1 \ldots \varepsilon^k$ enthält $\underline{p} \wedge \eta'$ enthält $\overline{p}$ nicht.

- η ist Ereignisspur einer **sequentiellen Berechnung** von $\mathcal{A}$

  $\Leftrightarrow$ für alle $k \in \{1, \ldots, l\}$ gilt: höchstens ein Prozeß p ist aktiv in ε^k.

- Seien $p, q \in P, p \neq q$. Die Prozesse p **und** q **sind parallel** oder **nebenläufig** (concurrent) oder **gleichzeitig in Ausführung** in η

  $\Leftrightarrow$ es gibt $k \in \{1, \ldots, l\}$ so, daß gilt: p und q sind aktiv in ε^k.

- η ist Ereignisspur einer **parallelen oder nebenläufigen Berechnung** von $\mathcal{A}$

  $\Leftrightarrow$ es gibt Prozesse $p, q \in P, p \neq q$, so daß gilt:
  p und q sind parallel oder nebenläufig in η.

Der Vielfalt der Verhaltenscharakteristika von $\mathcal{A}$ entspricht die Mächtigkeit von H. Für $\mathcal{A}$ mit $|P| = n$

- gibt es $n!$ verschiedene Ereignisspuren sequentieller Berechnungen und

- es gilt $|\mathrm{H}| = \frac{n(2n-1)!}{2^{n-1}}$.

Das ist eine mit n schnell anwachsende Anzahl:

$n = 1:$	1
$n = 2:$	6
$n = 3:$	90
$n = 4:$	2.520
$n = 5:$	113.400
$n = 6:$	7.484.400

Die große Anzahl dieser Verhaltensmöglichkeiten zeigt, daß es aussichtslos ist, das Verhalten eines parallelen Systems zu testen; man muß seine Eigenschaften beweisen!

2.1.2 Interpretierte Systeme

Für das uninterpretierte System $\mathcal{A} = (D, P, \emptyset, V, N)$ sind die Verhaltensmöglichkeiten wie oben erklärt festgelegt; für die Nutzung von $\mathcal{A}$ interessiert die Wirkung der Berechnungen von $\mathcal{A}$. Diese Wirkungen ergeben sich, wenn man eine Interpretation von $\mathcal{A}$ festlegt und $\mathcal{A}$ zu einem interpretierten System erweitert.
Eine **Interpretation** $\mathcal{I} = (Z, R)$ von $\mathcal{A}$ ist ein Paar, das

- mit Z den Zustandsraum der Datenobjekte von $\mathcal{A}$ und
- mit $R = (R_p \mid p \in P)$ die Rechenfunktionen der Prozesse von $\mathcal{A}$ festlegt.

Der Zustandsraum Z ergibt sich wie folgt:

- für jedes $d \in D$ ist $Z(d)$ der Zustandsraum von d;
- $Z = Z(D) = \bigtimes_{d \in D} Z(d)$.

Für $p \in P$ ist die Rechenfunktion R_p eine Abbildung

$$R_p : \bigtimes_{d \in V_p} Z(d) \to \bigtimes_{d \in N_p} Z(d) \ ,$$

die hier (zur Vereinfachung) als linkstotal vorausgesetzt wird.
Wenn $\mathcal{I} = (Z, R)$ eine Interpretation von $\mathcal{A}$ ist, ist $(\mathcal{A}, \mathcal{I})$ ein **interpretiertes System**.
Für das interpretierte System $(\mathcal{A}, \mathcal{I})$ können die Wirkungen der Berechnungen mit **Zustandsspuren** beschrieben werden. Sie ergeben sich, indem einem Anfangszustand $z^0 \in Z$ und einer Ereignisspur η von $\mathcal{A}$ eine Zustandsfolge zugeordnet wird: die Folge der Zustände, die $\mathcal{A}$ von z^0 ausgehend gemäß η berechnet.
Dabei wird jedem Ereignis von η ein Zustand zugeordnet, und zwar für $p \in P$

- dem Ereignis $\underline{p}$ der unveränderte Vorgängerzustand; das bedeutet: mit $\underline{p}$ liest p seine Eingabe,
- dem Ereignis $\overline{p}$ das mit der Eingabe und R_p berechnete Ergebnis; das bedeutet: mit $\overline{p}$ schreibt p seine Ausgabe.

Für das System $(\mathcal{A}, \mathcal{I})$ mit $\mathcal{I} = (Z, R)$ und die Ereignisspur $\eta = \varepsilon^1 \dots \varepsilon^k$ von $\mathcal{A}$ ist die $z^0 \in Z$ und η zugeordnete Zustandsspur $\zeta(z^0, \eta) = z^0 \dots z^k$ wie folgt definiert:

- $\zeta(z^0, \wedge) = z^0$;
- $\zeta(z^0, \eta) = \zeta((z^0, \varepsilon^1 \dots \varepsilon^{k-1}) z^k$ für $k \geq 1$ mit

- $z^k = z^{k-1}$, falls $\varepsilon^k = \underline{p}$ für $p \in P$,
- $z^k = (\pi_{\overline{N}_p}(z^{k-1}),\ R_p(\pi_{V_p}(z^i)))$,
  falls $\varepsilon^k = \overline{p}$ für $p \in P$ und $\varepsilon^i = \underline{p}$ mit $i < k$.

Abbildung 2.2 veranschaulicht diese Konstruktion.

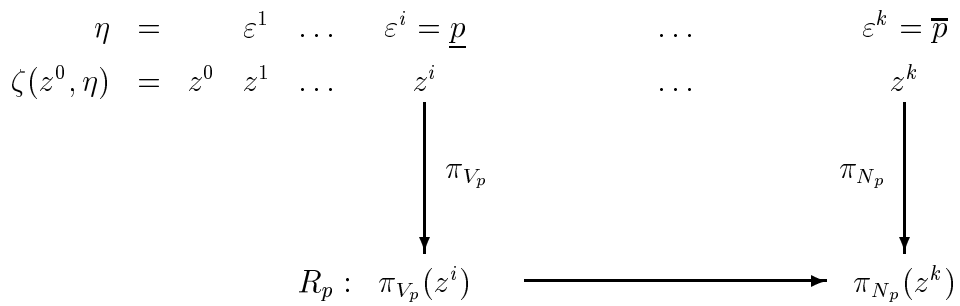


Abbildung 2.2: Konstruktion der z^0 und η zugeordneten Zustandsspur

Beispiel

Für das System $(\mathcal{A},\mathcal{I})$ mit $P = \{a, b\}$ seien die Rechenfunktionen der Prozesse

$$R_a : x := x + 1 \quad \text{und} \quad R_b : x := x + 2.$$

Im folgenden sind die Ereignisspuren von $\mathcal{A}$ und die ihnen zugeordneten Zustandsspuren angegeben.

	$\underline{a}$	$\overline{a}$	$\underline{b}$	$\overline{b}$	
x	x	$x+1$	$x+1$	$x+3$	$\Rightarrow \rho(x,\cdot) = x, x+1, x+3$
	$\underline{a}$	$\underline{b}$	$\overline{a}$	$\overline{b}$	
x	x	x	$x+1$	$x+2$	$\Rightarrow \rho(x,\cdot) = x, x+1, x+2$
	$\underline{a}$	$\underline{b}$	$\overline{b}$	$\overline{a}$	
x	x	x	$x+2$	$x+1$	$\Rightarrow \rho(x,\cdot) = x, x+2, x+1$
	$\underline{b}$	$\overline{b}$	$\underline{a}$	$\overline{a}$	
x	x	$x+2$	$x+2$	$x+3$	$\Rightarrow \rho(x,\cdot) = x, x+2, x+3$
	$\underline{b}$	$\underline{a}$	$\overline{b}$	$\overline{a}$	
x	x	x	$x+2$	$x+1$	$\Rightarrow \rho(x,\cdot) = x, x+2, x+1$
	$\underline{b}$	$\underline{a}$	$\overline{a}$	$\overline{b}$	
x	x	x	$x+1$	$x+2$	$\Rightarrow \rho(x,\cdot) = x, x+1, x+2$

Das Nebeneinander von uninterpretierten und interpretierten Systemen entspricht der Aufgabe, die Betriebssysteme haben: Es muß die **Berechnungen** eines parallelen Systems **koordinieren** und kennt dabei die Wirkungen der Prozeß–Berechnungen nicht. Als Anwender konstruiert man dagegen interpretierte Prozesse.
Eine Ereignisspur $\eta = \varepsilon^1 \dots \varepsilon^k$ und die ihr zugeordnete Zustandsspur $\zeta(z^0, \eta) = z^0 \dots z^k$ beschreiben den entsprechenden Ablauf einer Berechnung von $(\mathcal{A},\mathcal{I})$ vollständig. In vielen Fällen ist es zweckmäßig, diese Beschreibungen durch Beschränkungen auf das jeweils Interessierende und Relevante zu vereinfachen. Für diese Vereinfachungen sind insbesondere Projektionen nützlich. Im folgenden werden einige davon eingeführt und erklärt.

- Anstelle der vollständigen Menge D kann es zweckmäßig sein, einzelne Elemente $d \in D$ oder Teilmengen $D' \subseteq D$ zu betrachten; die Vereinfachung erfolgt dann mit einer Projektion von D auf d, π_d^D, oder von D auf D', $\pi_{D'}^D$. Diese Projektionen können auf Zustände oder auf Zustandsfolgen angewandt werden. Für $d \in D$ ergibt sich insbesondere mit $\zeta(d, z^0, \eta) = \pi_d^D(\zeta(z^0, \eta)) = z^0(d) \ldots z^k(d)$ die d**,** z^0 **und** η **zugeordnete Zustandsspur**.

- Aus der Ereignisspur η kann man die jeweils interessierenden Ereignisse durch Projektionen auswählen. Für eine Teilmenge $P' \subseteq P$ ist $\pi_{P'}(\eta)$ die Folge der aus η ausgewählten Ereignisse der Prozesse $p \in P'$. Statt der Auswahl nach Prozessen kann nach Initialisierungs– oder Terminierungsereignissen ausgewählt werden. $\pi_{\tau(P)}(\eta)$ ist die Folge der aus η ausgewählten Terminierungsereignisse der Prozesse $p \in P$. Für die so gebildeten Ereignisspuren können zugeordnete Zustandsspuren definiert werden. Mit $z^0 \in Z$ und $\pi_\tau(\eta)$ ist $\zeta(z^0, \pi_\tau(\eta))$ die z^0 **und** $\pi_\tau(\eta)$ **zugeordnete Zustandsspur**. Weil Veränderungen der Zustände der $d \in D$ höchstens mit Terminierungsereignissen erfolgen können, erfaßt $\zeta(z^0, \pi_\tau(\eta))$ von z^0 abgesehen alle potentiellen Zustandsänderungen der mit η beschriebenen Berechnung.

- Wenn η eine vollständige Ereignisspur ist, dann beschreibt für $z \in Z$ $\zeta(z, \eta)$ den vollständigen Ablauf der entsprechenden Berechnung. Mit dem ersten Element $\mathit{first}(\zeta(z, \eta))$ und dem letzten Element $\mathit{last}(\zeta(z, \eta))$ der Folge $\zeta(z, \eta)$ ergibt sich

  $$z = \mathit{first}(\zeta(z, \eta)) \xrightarrow{\eta} \mathit{last}(\zeta(z, \eta))$$

  das Paar der Zustände der **EA–Zuordnung für** D gemäß η. Dasselbe Paar ergibt sich mit der Ereignisspur $\pi_\tau(\eta)$

  $$\mathit{first}(\zeta(z, \pi_\tau(\eta))) \xrightarrow{\eta} \mathit{last}(\zeta(z, \pi_\tau(\eta))).$$

  Mit variierendem $z \in Z$ und festem η ergibt sich die **Funktion**

  $$\varphi_\eta : Z \to Z \text{ mit}$$
  $$\varphi_\eta(z) = \mathit{last}(\zeta(z, \pi_\tau(\eta))) \text{ für alle } z \in Z,$$

  die $\mathcal{A}$ gemäß η berechnet.

  Für einen **Prozeß** $p \in P$ ist mit der Rechenfunktion festgelegt, daß p

  $$R_p : Z(V_p) \to Z(N_p)$$

  berechnet. Mit η ist $\pi_p(\eta)$ die Projektion von η auf die p–Ereignisse, und für $z \in Z$ ist $\zeta(z, \pi_p(\eta))$ der Beitrag, den p zur Zustandsspur $\zeta(z, \eta)$ leistet. Die Ergebnisse, die p dabei berechnet, sind

  $$\pi_{N_p}^D(\mathit{last}(\zeta(z, \pi_p(\eta)))) \ .$$

- Wenn η eine vollständige Ereignisspur von $\mathcal{A}$ und $z \in Z$ ein Anfangszustand ist, dann beschreiben

  – $\zeta(z, \eta)$ den vollständigen Ablauf der entsprechenden Berechnung und
  – $\zeta(z, \pi_\tau(\eta))$ alle potentiellen Zustandsänderungen dieser Berechnung.

  Auch die Zustandsspur $\zeta(z, \pi_\tau(\eta))$ enthält noch redundante Zustände. Zustandsspuren, die keine Redundanzen enthalten, erhält man, indem

– für jedes $d \in D$ der Anfangszustand $z(d)$ und zusätzlich allein die Zustände von d erfaßt werden, die T–Ereignissen von Prozessen $p \in P$ mit $d \in N_p$ entsprechen.

Damit ergibt sich für jedes $d \in D$ die

d, z und η zugeordnete relevante Zustandsspur $\rho(d, z, \eta)$.

Die Längen dieser relevanten Zustandsspuren sind bei festem z und η für die einzelnen $d \in D$ verschieden. $\rho(d, z, \eta)$ hat die Länge 1, wenn d dem Nachbereich keines Prozesses von $\mathcal{A}$ angehört; wenn $n(d)$ die Anzahl der Prozesse $p \in P$ mit $d \in N_p$ ist, ist die Länge $n(d) + 1$.

Die relevanten Zustandsspuren der D–Elemente sind dazu geeignet, Unterschiede bei den Wirkungen der Berechnungen von $(\mathcal{A}, \mathcal{I})$ mit den Ereignisspuren von $\mathcal{A}$ kompakt und präzise zu erfassen.

Mit den jetzt eingeführten Systemen und mit dem Instrumentarium zur Beschreibung des Verhaltens dieser Systeme werden im folgenden Kriterien und Maßnahmen für die Konstruktion paralleler Systeme behandelt.

2.2 Determinierte Systeme mit atomaren Prozessen

Seien $\mathcal{A} = (D, P, \emptyset, V, N)$ ein uninterpretiertes System mit atomaren Prozessen und $\mathcal{I} = (Z, R)$ eine Interpretation von $\mathcal{A}$.
Für die Nutzung von $(\mathcal{A}, \mathcal{I})$ interessieren die Wirkungen der Berechnungen, die das System ausführt. Dazu ist schon bekannt, daß das Verhalten des Systems durch die Menge H der vollständigen Ereignisspuren von $\mathcal{A}$ bestimmt ist. Für $\eta \in H$ liefern

- $\zeta(z, \eta)$ die vollständige Zustandsspur und
- $\zeta(z, \pi_\tau(\eta))$ die Spur aller potentiellen Zustandsänderungen einer Berechnung von $(\mathcal{A}, \mathcal{I})$ mit dem Anfangszustand $z \in Z$, und
- $\varphi_\eta : Z \to Z$ ist die Funktion, die das System mit η berechnet.

Auf dieser Grundlage sind folgende Fragen zu beantworten:

- Welchen Einfluß haben die vielen Ereignisspuren von $\mathcal{A}$ auf die Wirkungen der Berechnungen von $(\mathcal{A}, \mathcal{I})$?
- Welche Wirkungen sollen die Berechnungen von $(\mathcal{A}, \mathcal{I})$, die sich mit den freien Kombinationen der Prozeßberechnungen ergeben, haben?
- Unter welchen Bedingungen haben die Berechnungen von $(\mathcal{A}, \mathcal{I})$ diese spezifizierten Wirkungen?

Zur Klärung des Einflusses der Ereignisspuren von $\mathcal{A}$ auf die Wirkungen der Berechnungen von $(\mathcal{A}, \mathcal{I})$ ist die folgende 2–stellige Relation auf H hilfreich: $\equiv \subseteq \mathrm{H} \times \mathrm{H}$ ist durch

$\eta \equiv \eta' \overset{\Delta}{\Leftrightarrow} \forall z \in Z, d \in D(\rho(d, z, \eta) = \rho(d, z, \eta'))$

für alle $\eta, \eta' \in \mathrm{H}$ definiert.

Zwei Ereignisspuren stehen also genau dann in der definierten Relation, wenn für alle $d \in D$ die diesen zugeordneten relevanten Zustandsspuren für alle Anfangszustände übereinstimmen.
Die Relation ist offenbar eine **Äquivalenzrelation** auf H. Sie ist eine starke Äquivalenz, die alle berechneten Zwischenergebnisse berücksichtigt, und für $\eta \equiv \eta'$ folgt offenbar $\varphi_\eta = \varphi_{\eta'}$.
Die Relation $\equiv$ definiert eine Zerlegung von H in Äquivalenzklassen. Für jede dieser Äquivalenzklassen $[\eta] \subseteq$ H gilt also:

> Mit allen $\eta' \in [\eta]$ berechnet $(\mathcal{A}, \mathcal{I})$ die Funktion φ_η.

Ein spezieller Fall liegt dann vor, wenn **alle** $\eta \in$ H **äquivalent** sind. Für ihn gilt: Die Reihenfolgen, mit denen die Ereignisse in den Spuren von $\mathcal{A}$ auftreten, haben keinen Einfluß auf die Wirkungen der Berechnungen der einzelnen Prozesse und folglich auch keinen Einfluß auf das EA–Verhalten des Systems $(\mathcal{A}, \mathcal{I})$.
In diesem Fall ist das System determiniert. Diese wichtige Eigenschaft ist wie folgt definiert:

- Das **interpretierte System** $(\mathcal{A}, \mathcal{I})$ ist (stark) **determiniert** $\Leftrightarrow$ alle $\eta \in$ H sind gemäß $\equiv$ äquivalent.

- Das **uninterpretierte System** $\mathcal{A}$ ist (stark) **determiniert** $\Leftrightarrow$ für jede Interpretation $\mathcal{I}$ von $\mathcal{A}$ ist $(\mathcal{A}, \mathcal{I})$ determiniert.

Determiniertheit ist eine wichtige Eigenschaft für ein paralleles Mehrbenutzersystem $\mathcal{R}$, dessen Benutzer unabhängig voneinander arbeiten wollen. Determiniertheit gewährleistet jedem Benutzer, daß $\mathcal{R}$ seine Berechnungen ohne Fremdeinwirkungen ausführt.
Nach dieser Klärung der praktischen Bedeutung ist es notwendig, einfach überprüfbare Bedingungen für Determiniertheit zur Verfügung zu haben. Diese Bedingungen müssen sicherstellen, daß zwischen den Prozessen von $\mathcal{A}$ keine Wechselwirkungen auftreten, welche die Wirkungen der einzelnen Prozesse verändern. Dabei ist klar, daß für ein interpretiertes System keine einfach nachprüfbaren Bedingungen erwartet werden können.
Einfach überprüfbar sind die Eigenschaften des uninterpretierten $\mathcal{A}$; die ausschlaggebende Bedingung ist die **Bernstein–Bedingung**:

- die Prozesse $p, q \in P, p \neq q$, erfüllen die Bernstein–Bedingung $B(p, q)$
  $\overset{\triangle}{\Leftrightarrow} V_p \cap N_q = V_q \cap N_p = N_p \cap N_q = \emptyset$;

- das System $\mathcal{A}$ erfüllt die Bernstein–Bedingung $B(\mathcal{A}) \overset{\triangle}{\Leftrightarrow}$ für alle $p, q \in P, p \neq q$, gilt $B(p, q)$.

Den Zusammenhang zwischen Determiniertheit und der Bernstein–Bedingung liefert der folgende

Satz: Das uninterpretierte System $\mathcal{A}$ ist determiniert $\Leftrightarrow B(\mathcal{A})$.

Den Beweis des Satzes führt man wie folgt:

- Die Aussage $\Leftarrow$ ist offensichtlich wahr.

- Die Aussage $\Rightarrow$ beweist man mit einem Widerspruch, den man mit passend gewählter Interpretation von $\mathcal{A}$ konstruiert.

Zum Beweis des Satzes muß man also die Interpretation von $\mathcal{A}$ wählen können; der Satz gilt nicht für ein interpretiertes System.
Für ein interpretiertes System $(\mathcal{A},\mathcal{I})$ mit $B(\mathcal{A})$ gilt: $(\mathcal{A},\mathcal{I})$ ist determiniert.
Die Bernstein–Bedingung für das System $\mathcal{A}$ gewährleistet **Interferenzfreiheit** der Wirkungen der Prozesse von $\mathcal{A}$. Sie ist eine wichtige, aber auch sehr restriktive Bedingung. Sie ist natürlich erfüllt, wenn die Wirkungsbereiche der Prozesse von $\mathcal{A}$ paarweise fremd sind. Darüber hinaus läßt sie für Prozesse als gemeinsam benutzbare Datenobjekte allein solche zu, die nicht verändert werden ($V_p \cap V_q \neq \emptyset$). Es kommen also i.w. **invariante Datenobjekte** — Konstanten — des Systems in Betracht.
Für die Konstruktion von Systemen ergibt sich folgendes:

- Man muß versuchen, die Bernstein–Bedingung möglichst weitgehend zu erfüllen und damit Interferenzfreiheit für die Prozesse zu gewährleisten.
- Die Konstruktion von parallelen Systemen allein nach dieser Regel ist höchstens in Ausnahmefällen möglich.

Determiniertheit und die Bernstein–Bedingung, die diese Eigenschaft gewährleisten, liefern auch die Antworten auf die weiteren, eingangs gestellten Fragen.
Wenn für das System $(\mathcal{A},\mathcal{I})$ die Wirkungen der Berechnungen der Prozesse und freie Kombinationen dieser Prozesse festgelegt sind, ergibt sich, daß $(\mathcal{A},\mathcal{I})$ Parallelkombinationen der Prozeßberechnungen berechnen muß. Das für $(\mathcal{A},\mathcal{I})$ **spezifizierte EA–Verhalten** ist also

$$R_{\mathcal{A}}^{soll} : Z \to Z \quad \text{mit} \quad R_{\mathcal{A}}^{soll} = \bigtimes_{p \in P} R_p.$$

Diese Formulierung des Soll–Verhaltens von $\mathcal{A}$ ist intuitiv klar; sie ist jedoch salopp. Ihre präzise Formulierung erfordert die Einordnung der Wirkungsbereiche der Prozesse in die Datenobjektmenge D, die Z zugrunde liegt.
Weil dieses „Einsortieren“ häufig benötigt wird, wird die erforderliche Abbildung hier angegeben.
Dazu sei $D' \subseteq D$. Das Einsortieren erfolgt mit der Abbildung

$$\amalg_D^{D'} : Z(D') \times Z(D) \to Z(D) \text{ mit}$$

$$\amalg_D^{D'}(z', z) \triangleq \bigtimes_{d \in D} z''(d) \text{ mit } z''(d) \triangleq \begin{cases} z'(d) & \text{, falls } d \in D' \\ z(d) & \text{, falls } d \in D. \end{cases}$$

für alle $d \in D$ und alle $(z', z) \in Z(D') \times Z(D)$.

$\amalg_D^{D'}$ ersetzt also die d–Werte für die $d \in D'$ und behält die d–Werte für die $d \in D - D'$ bei.
Die Abbildung $\amalg$ kann auf disjunkte Teilmengen $D_1, \ldots, D_n$ von D verallgemeinert werden.
Für die Spezifikation des EA–Verhaltens von $(\mathcal{A},\mathcal{I})$ mit $R_{\mathcal{A}}^{soll}$ ergeben sich damit:

- Notwendige Bedingung für die freie Kombination der R_p ist, daß die $(N_p \mid p \in P)$ paarweise disjunkt sind.
- Wenn diese Voraussetzung erfüllt ist, ist $R_{\mathcal{A}}^{soll} : Z \to Z$ durch

  $$\amalg_D^{(N_p \mid p \in P)} : \bigtimes_{p \in P} Z(N_p) \times Z(D) \text{ mit}$$

  $$R_{\mathcal{A}}^{soll}(z) = \amalg_D^{(N_p \mid p \in P)}(\bigtimes_{p \in P} R_p(\pi_D^{V_p}(z)), z) \text{ für alle } z \in Z \text{ definiert.}$$

Nach dieser Klärung kann $R_{\mathcal{A}}^{soll} = \bigtimes_{p \in P} R_p$ weiter als vereinfachte Notation benutzt werden.
Die Frage nach den Bedingungen, unter denen $(\mathcal{A}, \mathcal{I})$ das mit $R_{\mathcal{A}}^{soll}$ spezifizierte EA–Verhalten erfüllt, ist schon beantwortet: $B(\mathcal{A})$ ist eine hinreichende Bedingung.
Im weiteren ist wegen der restriktiven Ergebnisse, die jetzt vorliegen, zu klären, mit welchen Abschwächungen der bisher gestellten Forderungen parallele Systeme konstruiert werden können, die noch nutzungs–geeignet, aber praktikabler sind.

2.3 Systeme mit symmetrischer Konfliktvermeidung — Freier wechselseitiger Ausschluß

Für die Systeme mit atomaren Prozessen, die bisher betrachtet wurden, hat sich gezeigt, daß für die Prozesse Interferenzfreiheit gewährleistet werden muß und durch Beschränkungen der von den Prozessen gemeinsam benutzbaren Datenobjekte erreicht werden kann. Eine andere Möglichkeit, für die Prozesse eines parallelen Systems Interferenzfreiheit zu erreichen, besteht darin, die gemeinsame Benutzung von Datenobjekten zuzulassen, ihre gleichzeitige Benutzung jedoch auszuschließen. Die entsprechenden Datenobjekte müssen dann nacheinander, sequentiell oder seriell, benutzt werden, und der liberalste Fall hierfür liegt dann vor, wenn die Reihenfolge der seriellen Benutzung der Datenobjekte beliebig gewählt werden kann. Systeme mit atomaren Prozessen und diesen symmetrischen Reihenfolgerestriktionen werden im folgenden erklärt.
Ein **uninterpretiertes System** mit atomaren Prozessen und symmetrischer Konfliktvermeidung wird durch ein 5–tupel $\mathcal{A} = (D, P, \otimes, V, N)$ definiert. Dabei haben

- D, P, V und N die in Kapitel 2.1 angegebenen Eigenschaften und Bedeutungen, und
- $\otimes \subseteq P \times P$ ist eine irreflexive, symmetrische Konfliktvermeidungsrelation.

Für $\mathcal{A}$ wird mit $\otimes$ festgelegt, daß die Berechnungen der $p, q \in P$ mit $(p, q) \in \otimes$ nicht gleichzeitig ausgeführt werden dürfen, also seriell oder wechselseitig ausgeschlossen auszuführen sind. Die Relation $\otimes$ legt also wechselseitigen Ausschluß für einen Teil der Prozesse von $\mathcal{A}$ fest, und entsprechend wird $\otimes$ als **wA–Relation** bezeichnet. Für die Berechnungen von $\mathcal{A}$ ist also festgelegt, daß sie freie Kombinationen der Berechnungen der $p \in P$ mit den zusätzlichen, durch $\otimes$ festgelegten wA–Restriktionen sind. wA–Restriktionen treten häufig zusammen mit weiteren Bedingungen auf; das ist hier nicht der Fall. $\mathcal{A}$ wird dem entsprechend als System mit **freiem wechselseitigem Ausschluß** bezeichnet.
Die Beschreibung des **Verhaltens** von $\mathcal{A}$ erfolgt mit den in Kapitel 2.1 erklärten Mitteln, also mit Ereignisspuren, die den für $\mathcal{A}$ festgelegten Beschränkungen entsprechen müssen. Grundlage der Ereignisspuren sind wieder die den Prozessen $p \in P$ zugeordneten Ereignisse $\underline{p}$ und $\overline{p}$, für die weiter $\underline{p}$ vor $\overline{p}$ gelten muß.
Eine Folge $\eta = \varepsilon^1 \ldots \varepsilon^k$ mit $0 \leq k \leq 2 \cdot |P|$ von Ereignissen der Prozesse von $\mathcal{A}$ ist eine **Ereignisspur von $\mathcal{A}$** $\triangleq$

- jedes der $2 \cdot |P|$ Prozeßereignisse tritt in η höchstens 1–mal auf;
- für alle $p \in P$ gilt: aus $\varepsilon^j = \overline{p}$ für $j \in \{1, \ldots, k\}$ folgt, daß es ein $i \in \{1, \ldots, k\}$ mit $i < j$ und $\varepsilon^i = \underline{p}$ in η gibt;
- für alle $p, q \in P$ und $i \in \{1, \ldots, k\}$ gilt: aus $(p, q) \in \otimes$ folgt, daß höchstens einer der Prozesse p und q in ε^i aktiv ist.

Wie für die in Kapitel 2.1 erklärten Systeme gilt hier für $\mathcal{A}$, daß die Gesetzmäßigkeiten der Berechnungen von $\mathcal{A}$ durch die partielle Ordnung auf der Menge der Prozeßereignisse festgelegt sind, so daß das Verhalten von $\mathcal{A}$ durch die **Menge** H **aller vollständigen Ereignisspuren** von $\mathcal{A}$ charakterisiert wird. Die Mächtigkeit von H ergibt sich aus $\otimes$. Ein Extremfall ergibt sich für $\otimes = \emptyset$; dann liegt die Situation von Kapitel 2.1 vor. Ein weiterer Extremfall liegt vor, wenn $(p,q) \in \otimes$ für alle $p,q \in P$, $p \neq q$, festgelegt ist; dann sind alle $\eta \in$ H Ereignisspuren sequentieller Berechnungen von $\mathcal{A}$ und es gilt $|\mathrm{H}| = |P|!$: Die Berechnungen von $\mathcal{A}$ sind **Permutationen** der Berechnungen der Prozesse von $\mathcal{A}$.
Ein **interpretiertes System** mit atomaren Prozessen und freiem wechselseitigem Ausschluß ergibt sich, indem für ein uninterpretiertes $\mathcal{A}$ eine Interpretation $\mathcal{I} = (Z, R)$ festgelegt wird; dabei gilt für $\mathcal{I}$ das in Kapitel 2.1 Gesagte entsprechend.
Das Verhalten des uninterpretierten Systems $(\mathcal{A}, \mathcal{I})$ wird wie früher erklärt beschrieben: Wenn η eine Ereignisspur von $\mathcal{A}$ und $z^0 \in Z$ ein Anfangszustand sind, ist die z^0 und η **zugeordnete Zustandsspur** $\zeta(z^0, \eta)$ analog zu Kapitel 2.1 definiert. $\zeta(z^0, \eta)$ beschreibt das Verhalten von $\mathcal{A}$ mit den Wirkungen der Berechnungsschritte, die gemäß η ausgeführt werden, vollständig.
Damit steht das Instrumentarium zur Beschreibung und zur Analyse des Verhaltens der hier betrachteten Systeme zur Verfügung.

Beispiel

Für das System $(\mathcal{A}, \mathcal{I})$ soll folgendes gelten:

- $D = \{x, y\}$; die D–Elemente sind Ganzzahl–Datenobjekte;
- $P = \{a, b, c\}$ mit $R_a : x := x + 1;\ R_b : x := x + 2;\ R_c : y := 2 * y;$
- $\otimes = \{(a,b), (b,a), (a,c), (c,a), (b,c), (c,b)\}$.

Für $\mathcal{A}$ ergibt sich mit $\otimes$, daß für alle Prozesse wA festgelegt ist; es gibt also 3! vollständige Ereignisspuren sequentieller Berechnungen von $\mathcal{A}$. Drei davon sind mit ihren Zustandsspuren im folgenden angegeben; die drei weiteren liefern nichts Neues.

	$\underline{a}$	$\overline{a}$	$\underline{b}$	$\overline{b}$	$\underline{c}$	$\overline{c}$	
x	x	$x+1$	$x+1$	$x+3$	$x+3$	$x+3$	$\rho(x,\cdot) = x, x+1, x+3$
y	y	y	y	y	y	$2*y$	$\rho(y,\cdot) = y, 2*y$
	$\underline{a}$	$\overline{a}$	$\underline{c}$	$\overline{c}$	$\underline{b}$	$\overline{b}$	
x	x	$x+1$	$x+1$	$x+1$	$x+1$	$x+3$	$\rho(x,\cdot) = x, x+1, x+3$
y	y	y	y	$2*y$	$2*y$	$2*y$	$\rho(y,\cdot) = y, 2*y$
	$\underline{b}$	$\overline{b}$	$\underline{c}$	$\overline{c}$	$\underline{a}$	$\overline{a}$	
x	x	$x+2$	$x+2$	$x+2$	$x+2$	$x+3$	$\rho(x,\cdot) = x, x+2, x+3$
y	y	y	y	$2*y$	$2*y$	$2*y$	$\rho(y,\cdot) = y, 2*y$

Von den drei Prozessen von $\mathcal{A}$ sind einerseits der Prozeß c und andererseits die Prozeßmenge $\{a, b\}$ von Interesse.
Für den Prozeß c mit $W_c = \{y\}$ gelten $B(a,c)$ und $B(b,c)$. Folglich gibt es keine Interferenzen zwischen c und den Prozessen von $\{a, b\}$; die für c mit $\otimes$ festgelegten Restriktionen haben keinen Einfluß auf die Wirkungen der Berechnungen von $\mathcal{A}$; sie sind i.d.S. überflüssig.
Für die Prozesse a und b mit $V_a = N_a = V_b = N_b = \{x\}$ legt $\otimes$ Sequentialisierung fest, und das ist zur Vermeidung von Interferenzen auch notwendig. Diese Prozesse berechnen also den beiden Permutationen ab und ba entsprechend eines der seriellen Produkte

$R_a \circ R_b$ oder $R_b \circ R_a$.

Mit den Rechenfunktionen von a und b ergibt sich, daß in beiden Fällen $x+3$ für alle x berechnet wird. Das ist eine Konsequenz der speziellen algebraischen Eigenschaften von R_a und R_b. Allgemein muß davon ausgegangen werden, daß verschiedene Permutationen verschiedene Ergebnisse liefern. Das ist ein Beispiel für die Notwendigkeit, zwischen uninterpretierten und interpretierten Systemen zu unterscheiden.

⌋

Das Beispiel lehrt, wie die wA–Relation als Ergänzung zu den Ergebnissen von Kapitel 2.2 zur Konstruktion paralleler Systeme eingesetzt werden kann. Die wA–Relation dient dazu, **Interferenzen** der Prozesse eines Systems zu **vermeiden**. Mit Kapitel 2.2 ergibt sich, daß interferenzvermeidende Maßnahmen immer dann erforderlich sind, wenn zwei Prozesse p und q eines Systems $B(p,q)$ nicht erfüllen. Interferenzvermeidung ist dann zunächst für die **Spezifikation des Verhaltens** des zu konstruierenden Systems erforderlich. Ein Instrument dafür ist die wA–Relation. Wenn das Verhalten des Systems entsprechend festgelegt ist, müssen die **festgelegten Restriktionen** mit der weiteren Konstruktion des Systems **durchgesetzt** werden. Das bedeutet nach dem bisher Erklärten, daß für die Dynamik des Systems die festgelegten Ereignisspuren gewährleistet werden müssen.
Wenn ein paralleles System wie oben erklärt konstruiert wird, ergibt sich in der Regel ein **Mischsystem**, also ein System mit Prozessen, die zum Teil die Bernstein–Bedingung erfüllen, und für die zum anderen Teil Interferenzen mit der wA–Relation vermieden werden. Für das Folgende interessiert der Einfluß der wA–Relation auf die Wirkungen der Berechnungen von Systemen und die Konstruktionsmöglichkeiten, die sich daraus ergeben.

2.3.1 Serielle und serialisierbare Systeme

In diesem Abschnitt sei $\mathcal{A} = (D, P, \otimes, V, N)$ ein System, für das gelten:

- für alle $p, q \in P$ mit $p \neq q$ gilt $\rceil B(p,q)$;
- $\otimes = \{(p,q) \in P \times P \mid p \neq q\}$.

Nach dem oben Erklärten gilt also, daß jedes Paar (p,q) von Prozessen von $\mathcal{A}$ gemeinsame Datenobjekte so benötigt, daß Interferenzen auftreten können, und daß diese Interferenzen durch $\otimes$ vermieden werden.
Die Menge H der vollständigen Ereignisspuren von $\mathcal{A}$ ist genau die Menge der Ereignisspuren der sequentiellen Berechnungen von $\mathcal{A}$, und die Menge der Permutationen von P entspricht bijektiv der Menge H.
Sind also $|P| = n$ und $((1), \ldots, (n))$ eine Permutation von P, so entspricht dieser die Ereignisspur $\eta = \underline{(1)}\overline{(1)} \ldots \underline{(n)}\overline{(n)}$. Sind $\mathcal{I} = (Z, R)$ eine Interpretation von $\mathcal{A}$ und $z^0 \in Z$ ein Anfangszustand, so ergibt sich für die z^0 und η zugeordnete Zustandsspur $\zeta(z^0, \eta) = z^0 \ldots z^{2n}$:

- $z^{2i+1} = z^{2i}$ für alle $i \in \{1, \ldots, n-1\}$ und
- $z^{2i} = \Pi_D^{N_{(i)}}(R_{(i)}(\pi^D_{V_{(i)}}(z_{2i-1})), z_{2i-1})$ für alle $i \in \{1, \ldots, n\}$

Dabei ist wieder (zur Vereinfachung) vorausgesetzt, daß alle Rechenfunktionen total definiert sind. Mit der Projektion $\pi_\tau(\eta)$ von η auf die Terminierungsereignisse beschreibt

- $\zeta(z^0, \pi_\tau(\eta)) = z^0 z^2 \ldots z^{2n}$

alle potentiellen Veränderungen der Zustände von D bei der Berechnung, die $(\mathcal{A},\mathcal{I})$ gemäß η ausführt. $(\mathcal{A},\mathcal{I})$ berechnet das **Serienprodukt** $R_{(1)} \circ \ldots \circ R_{(n)}(z^0)$ und allgemein die Funktion

- $\bigcirc_{i=1}^{n} R_{(i)} : Z \to Z$.

Entsprechendes gilt für alle Permutationen von P. Wie schon erklärt, muß im allgemeinen davon ausgegangen werden, daß verschiedene Permutationen verschiedene Ergebnisse liefern. Für einen Anfangszustand z^0 von $(\mathcal{A},\mathcal{I})$ sind also $n!$ verschiedene Endzustände von Berechnungen, die $(\mathcal{A},\mathcal{I})$ gemäß H ausführt, möglich.
Das Erklärte ist zunächst für die **Spezifikation des Verhaltens** des Systems $\mathcal{A}$ von Interesse. Die dem Erklärten entsprechende Spezifikation ist:

- $(\mathcal{A},\mathcal{I})$ berechnet das Serienprodukt einer beliebigen Permutation der Berechnungen der Prozesse von $\mathcal{A}$.

Diese Spezifikation für $(\mathcal{A},\mathcal{I})$ unterscheidet sich offenbar wesentlich von dem in Kapitel 2.2 spezifizierten Verhalten. Sie sagt, daß

- die einzelnen Prozesse die mit ihren Rechenfunktionen festgelegten Berechnungen ausführen und dabei
- die jeweiligen Zustände ihrer Vorbereiche als Eingaben benutzen sollen, wobei diese Zustände die Ergebnisse der Berechnungen anderer Prozesse sein können.

Diese Spezifikation des Verhaltens von $(\mathcal{A},\mathcal{I})$ geht davon aus, daß die Datenobjekte von $\mathcal{A}$ von den Prozessen gemeinsam, aber interferenzfrei benutzt werden. Das ist charakteristisch für parallele, konkurrierende Berechnungen auf varianten Datenobjekten; von diesen Datenobjekten sind die Zustände zulässige Eingaben, die Ergebnisse einer Folge abgeschlossener Berechnungen anderer Prozesse sind, wobei weitere Eigenschaften der Folge nicht festgelegt sind. Diese Verhaltensspezifikation ist charakteristisch und üblich für **Datenbanksysteme**. Die w–atomaren Prozesse der hier betrachteten Systeme werden dort Transaktionen genannt, und für Transaktionen wird w–Atomarität festgelegt.
Wenn für ein System $(\mathcal{A},\mathcal{I})$ das Verhalten wie erklärt spezifiziert ist, müssen die festgelegten Reihenfolgerestriktionen **konstruktiv durchgesetzt** werden. Das bedeutet, daß die mit der wa–Relation $\otimes$ für die Ereignisspuren festgelegten Beschränkungen zu gewährleisten sind. Für $\mathcal{A}$ mit w–atomaren Prozessen ergibt sich daraus die Beschränkung auf sequentielle Berechnungen. Wenn man dagegen von der BB–Sicht der Prozesse zur GB–Sicht übergeht, ergeben sich weniger restriktive und effizientere Möglichkeiten zur Koordinierung der Berechnungen des Systems: Die Berechnungen müssen nicht sequentiell ausgeführt werden; sie müssen jedoch die mit den Serienprodukten spezifizierten Wirkungen haben. Mit dieser Vorgehensweise für Realisierungen ergeben sich **serialisierbare Systeme**. Sie liefern typische Beispiele für die einander ergänzenden BB– und GB–Sichten, die für Rechen– und Betriebssysteme ausschlaggebend wichtig sind.

2.3.2 Relativ determinierte Systeme

Im folgenden werden weiter Systeme betrachtet, deren Prozesse Datenobjekte gemeinsam so benutzen, daß Interferenzen mit der wA–Relation vermieden werden müssen. Die Datenobjekte werden jedoch in wesentliche und unwesentliche aufgeteilt mit dem Ziel, für ein System Determiniertheit relativ zu den wesentlichen Datenobjekten zu erreichen. Diese relativ determinierten Systeme sind mit der Semantik ihrer Berechnungen die Alternative zu den oben

erklärten seriellen und serialisierbaren Systemen, und sie sind die Grundmodelle für parallele Mehrbenutzersysteme.
Im folgenden sei $\mathcal{A} = (D, P, \otimes, V, N)$ ein System mit:

- für alle $p, q \in P$ mit $p \neq q$ gilt $\rceil B(p, q)$;
- $\otimes = \{(p, q) \in P \times P \mid p \neq q\}$.

Für das Verhalten von $\mathcal{A}$ gilt also das im vorangehenden Kapitel Gesagte, und entsprechendes gilt für $(\mathcal{A}, \mathcal{I})$ mit einer Interpretation $\mathcal{I} = (Z, R)$ von $\mathcal{A}$.
Die angekündigte Aufteilung der Datenobjekte erfordert differenziertere Festlegungen der Eigenschaften der Prozesse von $\mathcal{A}$; das folgende Beispiel zeigt, worauf dabei zu achten ist.

Beispiel

Für das System $(\mathcal{A}, \mathcal{I})$ soll gelten:

- $D = \{u, v, w, h\}$ mit Ganzzahl–Datenobjekten;
- $P = \{p, q\}$ mit $R_p: \begin{array}{l} u := u^2 * (v - u) \\ h := v - u \end{array}$, $R_q: \begin{array}{l} w := 2 * v * (w - v^2) \\ h := 2 * v \end{array}$.

Es gilt also $\rceil B(p, q)$.
Interferenzen werden durch $\otimes = \{(p, q), (q, p)\}$ ausgeschlossen.
Für $\mathcal{A}$ gibt es 2 vollständige Ereignisspuren sequentieller Berechnungen:

$$\eta_1 = \underline{p}\overline{p}\underline{q}\overline{q} \text{ und } \eta_2 = \underline{q}\overline{q}\underline{p}\overline{p}.$$

Mit den Projektionen auf die T–Ereignisse ergeben sich:

u	$u^2 * (v - u)$	$u^2 * (v - u)$	u	$u^2 * (v - u)$
v	v	v	v	v
w	w	$2 * v * (w - v^2)$	$2 * v * (w - v^2)$	$2 * v * (w - v^2)$
h	$v - u$	$2 * v$	$2 * v$	$v - u$
z^0	$\zeta(z^0, \pi_\tau(\eta_1))$		$\zeta(z^0, \pi_\tau(\eta_2))$	

Das System ist determiniert, wenn allein die Datenobjekte $\{u, v, w\} \subset D$ berücksichtigt werden.
Beide Prozesse benutzen h mit beliebigem Anfangswert zum Speichern berechneter Ergebnisse, die entweder schließlich nicht mehr benötigte Zwischenergebnisse oder relevante Endergebnisse sein können.

⌋

Das Beispiel lehrt, daß eine Aufteilung der Datenobjekte von $\mathcal{A}$ in wesentliche und unwesentliche mit entsprechenden Spezifikationen für die Prozesse von $\mathcal{A}$ erfolgen muß: Zunächst müssen die Wirkungsbereiche der Prozesse mit den erforderlichen Differenzierungen festgelegt werden, und dann müssen die Rechenfunktionen der Prozesse für Interpretationen von $\mathcal{A}$ angepaßt werden.
Für den **Wirkungsbereich** W_p eines Prozesses p von $\mathcal{A}$ ist bisher $W_p = V_p \cup N_p$ festgelegt. Dabei sind:

- V_p der Vorbereich von p, was bedeutet, daß p die Zustände der $v \in V_p$ als Eingaben benutzt; sie müssen also als Vorbedingung für $\underline{p}$ definiert sein.

- N_p der Nachbereich von p, was bedeutet, daß p die $n \in N_p$ für Ausgaben berechneter Ergebnisse, die mit $\overline{p}$ vorliegen, benutzt.
- Für V_p und N_p ist $V_p \cap N_p \neq \emptyset$ zugelassen, was bedeutet, daß p die $d \in V_p \cap N_p$ für Eingaben und Ausgaben benutzt.

Die erforderlichen Differenzierungen werden dadurch erreicht, daß der Wirkungsbereich von p wie folgt in 4 disjunkte Teilmengen aufgeteilt wird:

- $$\begin{array}{rcccccccc} W_p & = & V_p \cup N_p & & & & & & \\ & = & (V_p - N_p) & + & (V_p \cap N_p) & + & (N_p - V_p) & & \\ & = & IN_p & + & IO_p & + & LH_p & + & OT_p. \end{array}$$

Dabei sind die Zustände der

- $d \in IN_p$ die für p konstanten **Eingaben**,
- $d \in IO_p$ die **Ein– und Ausgabevariablen**,
- $d \in LH_p$ die lokalen **Hilfsvariablen** und
- $d \in OT_p$ die **Ausgabevariablen**

der Berechnungen von p.
Mit diesen Differenzierungen wird ein **uninterpretiertes System** durch ein 7–tupel $\mathcal{A} = (D, P, \otimes, IN, IO, LH, OT)$ definiert. Dabei sind für einen Prozeß $p \in P$

- die Elemente von $W_p - LH_p$ die **wesentlichen Datenobjekte** und
- die Elemente von LH_p die unwesentlichen **Hilfsdatenobjekte**

der Berechnungen von p. Die Hilfsdatenobjekte sind Elemente von $N_p - V_p$; sie werden mit beliebigem Anfangszustand für die Berechnungen von p benötigt; sie werden jedoch von p allein für Zwischenergebnisse benutzt, und diese werden mit dem Abschluß der p–Berechnungen, also mit $\overline{p}$, irrelevant. Der Nachbereich von p zerfällt in $\overline{N_p} \triangleq N_p - LH_p$ und in LH_p; die für p relevanten Ergebnisse werden in den $d \in \overline{N_p}$ gespeichert.
Bei gegebenem uninterpretiertem $\mathcal{A}$ wird eine **Interpretation von $\mathcal{A}$** weiter mit einem Paar $\mathcal{I} = (Z, R)$ und mit Anpassungen an die differenzierteren Festlegungen definiert. Für $p \in P$ gilt zunächst

$$R_p : Z(V_p) \to Z(N_p).$$

Die wesentlichen Wirkungen der Berechnungen von p sind durch

$$\overline{R_p} : Z(V_p) \to Z(\overline{N_p})$$

festgelegt, und der Zusammenhang zwischen R_p und $\overline{R_p}$ ist

$$\overline{R_p} = \pi_{\frac{N_p}{N_p}}(R_p) \ ;$$

$\overline{R_p}$ ergibt sich also aus R_p, indem die unwesentlichen Zwischenergebnisse ignoriert werden.
Nach diesen Vorbereitungen ist klar, welche Eigenschaften relativ determinierte Systeme haben müssen und wie diese Systeme zu konstruieren sind.
Das System $\mathcal{A}$ oder $(\mathcal{A}, \mathcal{I})$ ist **im wesentlichen determiniert** $\overset{\triangle}{\Leftrightarrow}$ es ist bzgl. der wesentlichen Datenobjekte aller Prozesse von $\mathcal{A}$ determiniert.
Eine hinreichende Bedingung dafür, daß $(\mathcal{A}, \mathcal{I})$ im wesentlichen determiniert ist, ist

- alle $p, q \in P$ mit $p \neq q$ erfüllen die Bernstein–Bedingung bzgl. der V_p, $\overline{N_p}$, V_q und $\overline{N_q}$ und
- alle $p, q \in P$ mit $p \neq q$ sind mit $\otimes$ wechselseitig ausgeschlossen.

Wenn $(\mathcal{A}, \mathcal{I})$ im wesentlichen determiniert ist, dann berechnet $\mathcal{A}$ im wesentlichen

$$\overline{R_{\mathcal{A}}} = \bigtimes_{p \in P} \overline{R_p} \ .$$

Wie für die oben erklärten seriellen und serialisierbaren Systeme, so wird auch hier zwischen der für die Nutzung wichtigen BB–Sicht und der für die Realisierung wichtigen GB–Sicht auf ein System unterschieden.
Es sollte klar geworden sein, daß der erste Schritt zur Unterscheidung zwischen den beiden Sichten mit der Spezifikation des Verhaltens eines Systems erfolgt: Es muß klargestellt werden, welches Ziel erreicht werden soll. Darüber hinaus sollen Anforderungen explizit formuliert werden. Die angegebenen Differenzierungen für die i.w. determinierten Systeme zeigen die Ansätze für diese expliziten Formulierungen.

2.4 Systeme mit antisymmetrischer Konfliktvermeidung — Geordnete Abhängigkeiten

In den Kapiteln 2.2 und 2.3 sind Systeme mit frei kombinierten Prozessen betrachtet, und für diese Systeme sind die Möglichkeiten dafür aufgezeigt, Zusammenhänge zwischen diesen freien Kombinationen und dem Verhalten des entsprechenden Systems herzustellen. In diesem Kapitel werden Systeme betrachtet, deren Prozesse voneinander abhängig sind und zielgerichtet so kombiniert werden, daß sie zusammenwirken.
Ein **uninterpretiertes System** mit atomaren Prozessen und antisymmetrischer Konfliktvermeidung wird durch ein 5–tupel $\mathcal{A} = (D, P, \boxplus, V, N)$ definiert. Dabei haben

- D, P, V und N die in Kapitel 2.1 angegebenen Eigenschaften und Bedeutungen und
- $\boxplus \subseteq P \times P$ ist eine irreflexive Ordnungsrelation.

Für $\mathcal{A}$ wird mit $\boxplus$ festgelegt, daß die Berechnungen der Prozesse p und q mit $(p, q) \in \boxplus$ in der Reihenfolge erst p, dann q ausgeführt werden müssen. Wenn p und q Datenobjekte gemeinsam benutzen, ist damit festgelegt, daß die von p berechneten Ergebnisse vorliegen, wenn die q–Berechnung beginnt: p und q können mittels gemeinsam benutzter Datenobjekte **kooperieren**.
Für die Prozesse von $\mathcal{A}$ werden also mit $\boxplus$ gerichtete Abhängigkeiten festgelegt. Damit die Berechnungen aller Prozesse ausgeführt werden können, müssen diese Abhängigkeiten zyklenfrei sein; das wird mit $\boxplus$ sichergestellt: $\boxplus$ ist eine irreflexive Ordnung, also irreflexiv, antisymmetrisch und transitiv. $\boxplus$ ist die **Vorrangrelation** über der Prozeßmenge von $\mathcal{A}$.
Mit der Ordnungsrelation $\boxplus$ ergeben sich für Beschreibungen der für die Prozesse von $\mathcal{A}$ festgelegten Reihenfolgerestriktionen alle Möglichkeiten, die für geordnete Mengen gebräuchlich sind. Zur Veranschaulichung der Restriktionen bei einer kleinen Prozeßmenge ist insbesondere der Vorranggraph von $\mathcal{A}$, $\mathcal{G}_{\mathcal{A}}$, gut geeignet. Das ist der Graph mit der Knotenmenge P und der Kantenmenge gemäß $\boxplus$ unter Weglassung von Transitivitätskanten. Seien also

$$\boxplus^+ \triangleq \{(p, q) \in \boxplus \mid \exists r \in P((p, r) \in \boxplus \wedge (r, q) \in \boxplus)\} \text{ und}$$
$$\sqsubset \triangleq \boxplus - \boxplus^+ \ ,$$

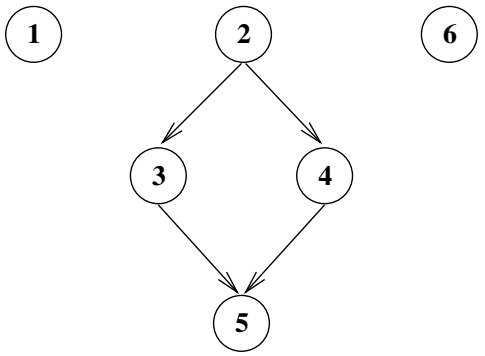


Abbildung 2.3: Ein Vorranggraph für ein System mit $P = \{1, 2, \ldots, 6\}$

so ist $\mathcal{G}_{\mathcal{A}} = (P, \sqsubset)$ der **Vorranggraph von** $\mathcal{A}$. Abbildung 2.3 zeigt den Vorranggraphen für ein System mit $P = \{1, 2, \ldots, 6\}$.
Die Beschreibung des **Verhaltens** von $\mathcal{A}$ erfolgt mit den in Kapitel 2.1 erklärten Mitteln, also mit Ereignisspuren, die an die mit $\boxtimes$ festgelegten Beschränkungen angepaßt sind.
Eine Folge $\eta = \varepsilon^1 \ldots \varepsilon^k$ mit $0 \leq k \leq 2 \cdot |P|$ von Ereignissen der Prozesse von $\mathcal{A}$ ist eine **Ereignisspur von** $\mathcal{A} \triangleq$

- jedes der $2 \cdot |P|$ Prozeßereignisse tritt in η höchstens 1–mal auf;
- für alle $p \in P$ gilt: aus $\varepsilon^j = \overline{p}$ für $j \in \{1, \ldots, k\}$ folgt, daß es ein $i \in \{1, \ldots, k\}$ mit $i < j$ und $\varepsilon^i = \underline{p}$ in η gibt;
- für alle $(p, q) \in \boxtimes$ gilt: aus $\varepsilon^j = \underline{q}$ für $j \in \{1, \ldots, k\}$ folgt, daß es ein $i \in \{1, \ldots, k\}$ mit $i < j$ und $\varepsilon^i = \overline{p}$ in η gibt.

Wie für die früher erklärten Systeme, so gilt auch hier, daß die Gesetzmäßigkeiten der Berechnungen von $\mathcal{A}$ durch die partielle Ordnung auf der Menge der Prozeßereignisse festgelegt sind: Das Verhalten von $\mathcal{A}$ wird durch die **Menge** H **der vollständigen Ereignisspuren von** $\mathcal{A}$ charakterisiert. Die Mächtigkeit von H ergibt sich insbesondere aus $\boxtimes$. Ein Extremfall ist $\boxtimes = \emptyset$; dann liegt eines der schon bekannten Systeme vor. Der Extremfall in entgegengesetzter Richtung liegt vor, wenn P durch $\boxtimes$ total oder linear geordnet ist, dann gibt es für $\mathcal{A}$ genau eine vollständige Ereignisspur.
Zu einem uninterpretierten $\mathcal{A}$ mit Vorrangrelation wird eine Interpretation $\mathcal{I} = (Z, R)$ wie früher erklärt definiert. Das Paar $(\mathcal{A}, \mathcal{I})$ mit einer Interpretation $\mathcal{I}$ von $\mathcal{A}$ ist ein **interpretiertes System mit Vorrangrelation**.
Das **Verhalten von** $(\mathcal{A}, \mathcal{I})$ wird wie früher erklärt beschrieben, also für $z \in Z$ und eine Ereignisspur η von $\mathcal{A}$ durch die z und η zugeordnete Zustandsspur $\zeta(z, \eta)$.
Bei der **Konstruktion eines Systems** $\mathcal{A}$ oder $(\mathcal{A}, \mathcal{I})$ mit Vorrangrelation und bei der Spezifikation der Wirkungen der Berechnungen des Systems ist die Vorgehensweise im Fall $\boxtimes = \emptyset$ bereits aus Kapitel 2.2 bekannt. Ebenso ist klar, daß für eine lineare Ordnung $\boxtimes$ ein spezieller Fall der in Kapitel 2.3 betrachteten Systeme vorliegt: Das System berechnet eine Serienkombination der Berechnungen seiner Prozesse, wobei die Reihenfolge für die Kombination mit $\boxtimes$ eindeutig festgelegt ist; es liegt ein sequentielles System vor. Von Interesse bleibt der Fall, in dem $\boxtimes$ eine nichtleere, echt partielle Ordnung ist. Im folgenden ist ein Beispiel hierfür angegeben.

Beispiel

Für das System $(\mathcal{A},\mathcal{I})$ soll folgendes gelten:

- $D = \{u, v, w\}$; die D–Elemente seien Boolesche Datenobjekte;
- $P = \{a, b, c\}$ mit $R_a : u := u \wedge v$; $R_b : w := v \wedge \neg v$; $R_c : w := \neg w$.
- $\boxplus = \{b, c)\}$.

Für $\mathcal{A}$ gibt es 15 vollständige Ereignisspuren. Zwei davon sind im folgenden mit ihren Zustandsspuren angegeben.

	$\underline{a}$	$\overline{a}$	$\underline{b}$	$\overline{b}$	$\underline{c}$	$\overline{c}$	
u	u	$u \wedge v$	$u \wedge v$	$u \wedge v$	$u \wedge v$	$u \wedge v$	$\rho(u,\cdot) = u, u \wedge v$
v	v	v	v	v	v	v	$\rho(v,\cdot) = v$
w	w	w	w	$w \wedge \neg v$	$w \wedge \neg v$	$\neg w \wedge v$	$\rho(w,\cdot) = w, w \wedge \neg v, \neg w \wedge v$

	$\underline{a}$	$\underline{b}$	$\overline{b}$	$\underline{c}$	$\overline{c}$	$\overline{a}$	
u	u	u	u	u	u	$u \wedge v$	$\rho(u,\cdot) = u, u \wedge v$
v	v	v	v	v	v	v	$\rho(v,\cdot) = v$
w	w	w	$w \wedge \neg v$	$w \wedge \neg v$	$\neg w \wedge v$	$\neg w \wedge v$	$\rho(w,\cdot) = w, w \wedge \neg v, \neg w \wedge v$

⌋

Das im Beispiel gezeigte System ist determiniert, wobei **Determiniertheit** für uninterpretierte und für interpretierte Systeme wie in Kapitel 2.2 angegeben definiert ist.
Das Beispiel lehrt auch, wie Determiniertheit für die hier betrachteten Systeme erreicht wird: Für die Prozesse b und c gilt zunächst $\rceil B(b,c)$; mit $(b,c) \in \boxplus$ sind jedoch Interferenzen von b und c ausgeschlossen. Die Bedingungen, mit denen Interferenzen vermieden werden, sind damit Kombinationen der Bernstein–Bedingungen mit der Vorrangrelation. Die Eigenschaft eines Systems, auf die es jetzt ankommt, ist Störungsfreiheit.
Sei $\mathcal{A}$ ein System mit der Prozeßmenge P und der Vorrangrelation $\boxplus$. Dann ist **Störungsfreiheit** von $\mathcal{A}$ wie folgt definiert:

- die Prozesse $p, q \in P, p \neq q$, sind störungsfrei $\overset{\triangle}{\Leftrightarrow}$ $p \boxplus q \vee q \boxplus p \vee B(p,q)$;
- $\mathcal{A}$ ist störungsfrei $\overset{\triangle}{\Leftrightarrow}$ alle $p, q \in P, p \neq q$, sind störungsfrei.

Den Zusammenhang zwischen Störungsfreiheit und Determiniertheit liefert der folgende

Satz: Das uninterpretierte System $\mathcal{A}$ ist determiniert $\Leftrightarrow$ $\mathcal{A}$ ist störungsfrei.

Für den Beweis dieses Satzes ist die Aussage $\Leftarrow$ evident. Die Aussage $\Rightarrow$ beweist man mit Induktion über die Anzahl der Prozesse durch Widerspruch, den man mit einer passend gewählten Interpretation konstruiert.

⌋

Der Satz gilt analog zu dem in Kapitel 2.2 angegebenen nicht für interpretierte Systeme. Ein störungsfreies, interpretiertes System ist determiniert.
Für ein störungsfreies, interpretiertes System mit Vorrangrelation $\boxplus$ ergibt sich die Rechenfunktion als aus $\boxplus$ abgeleitete Serien–Parallel–Kombination der Rechenfunktionen der Prozesse.

Beispiel

Für das System $\mathcal{A}$, dessen Vorranggraph in Abbildung 2.3 angegeben ist, ist die Rechenfunktion

$$R_{\mathcal{A}} = R_1 \times (R_2 \circ (R_3 \times R_4) \circ R_5) \times R_6$$

mit den Rechenfunktionen $R_i, i \in \{1, \ldots, 6\}$, der Prozesse unter der Voraussetzung, daß $\mathcal{A}$ störungsfrei ist.

⌋

Allgemein wird die Rechenfunktion eines Systems induktiv auf der Grundlage der Vorrangrelation definiert. Dazu sei $(\mathcal{A}, \mathcal{I})$ ein störungsfreies System mit der Prozeßmenge P, der Vorrangrelation $\boxtimes$ und der Familie $R = (R_p \mid p \in P)$ der Rechenfunktionen der Prozesse. Zunächst seien für $p \in P$

$$p^- \triangleq \{q \in P \mid (q,p) \in \boxtimes\} \text{ und}$$
$$p^+ \triangleq \{q \in P \mid (p,q) \in \boxtimes\}.$$

Für $p \in P$ können die Mengen p^- und p^+ leer sein, weil $\boxtimes$ irreflexiv ist.
Für $p \in P$ sei $Q(p) \triangleq \{p\} \cup p^+$.
Für ein $P' \subseteq P$ sei $\mathcal{A}(P')$ das System, das sich aus $\mathcal{A}$ durch Einschränkung auf die Prozeßmenge P' ergibt. Die Rechenfunktion von $\mathcal{A}(P')$ wird mit $R_{\mathcal{A}(P')}$ bezeichnet.
Sei $P_1 \triangleq \{p \in P \mid p^- = \emptyset\}$. Es gilt $P_1 \neq \emptyset$, weil $\boxtimes$ eine Ordnung ist. Weiter ist

$\{Q(p) \mid p \in P_1\}$ eine Zerlegung von P.

Damit ergibt sich für die **Rechenfunktion** $R_{\mathcal{A}}$ von $(\mathcal{A}, \mathcal{I})$

$$R_{\mathcal{A}} = \bigtimes_{p \in P_1} R_{\mathcal{A}(Q(p))} \text{ mit } R_{\mathcal{A}(Q(p))} = \begin{cases} R_p & \text{, falls } Q(p) = \{p\} \\ R_p \circ R_{\mathcal{A}(p^+)} & \text{, sonst} \end{cases} \text{ für alle } p \in P_1.$$

Anschließend ist darauf hinzuweisen, daß man die Kombination der Bernstein–Bedingungen mit der Vorrangrelation, die zu der für Determiniertheit ausschlaggebenden Störungsfreiheit führt, konstruktiv zur Parallelisierung von Systemen nutzen kann. Für ein gegebenes störungsfreies und damit determiniertes System kann man ein äquivalentes, **maximal paralleles System** dadurch konstruieren, daß die Vorrangrelation unter Gewährleistung der Störungsfreiheit abgeschwächt wird. Diese Maßnahmen gehören zum Standardrepertoire der Parallelisierung sequentieller Programme.

2.5 Zusammenfassung und Einordnung

In den Kapiteln 2.1 bis 2.4 sind mit einfachen Modellen die grundlegenden Eigenschaften paralleler Mehrbenutzersysteme erklärt. Dabei hat sich gezeigt, daß bei der Konstruktion eines Systems zunächst die Eigenschaften der Komponenten — die Datenobjekte und die Prozesse — und dann deren Komposition zum System festzulegen sind. Die einfachste Möglichkeit der Zusammenfassung vom Komponenten zu einem System ist die freie Komposition, also die Zusammenfassung ohne zusätzliche (einschränkende) Festlegungen. Diese freie Komposition hat zu dem für parallele Systeme charakteristischen Phänomen geführt: Es sind **Interferenzen** möglich, und Interferenzen haben zur Folge, daß die Berechnungen der Prozesse und die Berechnungen des Systems Wirkungen haben können, deren Gesetzmäßigkeiten unbekannt

sind. Daraus ergibt sich die grundlegende Forderung an die Konstruktion jedes parallelen Systems: Sie muß Interferenzfreiheit gewährleisten.
Die wichtigsten Maßnahmen, die zur Gewährleistung von Interferenzfreiheit geeignet sind, wurden erklärt. Dabei hat sich gezeigt, daß

- Interferenzvermeidung natürlich die Kenntnis potentieller Interferenzen — die Wirkungsbereiche der Prozesse — voraussetzt,
- die Maßnahmen zur Interferenzvermeidung auch für die Spezifikation des Systemverhaltens erforderlich sind, und
- unterschiedliche Maßnahmen zur Interferenzvermeidung zu Systemen mit wesentlich verschiedenen Verhalten führen.

Damit ergibt sich, daß am Anfang der Konstruktion eines Systems die **Spezifikation des Systemverhaltens**, also die Festlegung des System–Sollverhaltens auf der Grundlage des festgelegten Verhaltens der Prozesse, stehen muß. Die nächste Aufgabe besteht dann darin, dieses Sollverhalten konstruktiv durchzusetzen. Dazu sind Beschreibungen des Systemverhaltens erforderlich, und für diese Beschreibungen sind Ereignis– und Zustandsspuren geeignet.
Mit den Ereignisspuren, die zur Verhaltensbeschreibung dienen, hat sich ein weiteres Charakteristikum paralleler Systeme gezeigt: Das Wesentliche eines parallelen Systems ist die **echte partielle Ordnung** über den Prozessen des Systems. Diese partielle Ordnung hat zur Folge, daß es viele (vollständige) Ereignisspuren eines Systems gibt, und die Gesamtheit dieser Spuren ist erforderlich und geeignet, das Verhalten des Systems zu beschreiben.
Die Ereignisspuren eines Systems werden konstruiert, indem sie den festgelegten Regeln entsprechend hingeschrieben werden. Diese Konstruktion ist die (abstrakte) Lösung einer wesentlichen Aufgabe des Betriebssystems eines parallelen Systems, der Aufgabe, die **Berechnungen** der Prozesse so **zu koordinieren**, daß das Sollverhalten des Systems erreicht wird.
Nachdem die Lösungen dieser Koordinierungsaufgaben bekannt sind, ergibt sich, daß die Lösungsverfahren automatisiert werden müssen: Die Koordinierung der Berechnungen der Prozesse eines Systems muß mit Berechnungen des Betriebssystems erfolgen. Mit dieser Automatisierung werden wir uns im folgenden befassen.
Die Systeme, die bisher betrachtet wurden, sind einfache Modellsysteme. Rechen– und Betriebssysteme sind reale Systeme, die Berechnungen letztlich mit den Komponenten ihrer Hardwarekonfiguration tatsächlich ausführen. Der Übergang von einem Modellsystem zu einem realen System erfordert demnach (vielstufige) Übergänge vom Modell mit dessen Komponenten und Eigenschaften zur Hardwarekonfiguration und dem Einsatz der Eigenschaften deren Komponenten mit dem Ziel, das mit dem Modell spezifizierte Verhalten real zu erreichen.
Die Rolle, die Modelle bei diesem Konstruktionsprozeß spielen, sollte klar geworden sein. Für den Konstruktionsprozeß sind verschiedene, insbesondere allgemeinere Modelle erforderlich. Die benötigten Verallgemeinerung der bisher benutzten Modelle werden bei Bedarf für die jeweils zu lösenden Aufgaben vorgenommen und erklärt.

Kapitel 3

Einfache Systeme mit sequentiellen Prozessen

In diesem Kapitel werden zunächst Modellsysteme mit sequentiellen Prozessen eingeführt. Das sind einerseits Verallgemeinerungen der in Kapitel 2 erklärten Systeme, so daß — mit entsprechenden Anpassungen — die dort angegebenen Verhaltensbeschreibungen und die dort abgeleiteten Aussagen über parallele Systeme übertragen und weiter benutzt werden können. Andererseits sind Modellsysteme mit sequentiellen Prozessen dazu geeignet, schrittweise zu Rechen– und Betriebssystemen überzugehen; mit ihnen wird die Grundstruktur für Realisierungen paralleler Mehrbenutzersysteme begründet, die bereits wichtige Eigenschaften realer Systeme hat. Für die Systeme, die sich damit ergeben, wird schließlich erklärt, wie die Berechnungen ihrer Prozesse mit Betriebssystemberechnungen koordiniert werden können: Es werden Koordinierungskonzepte und –mechanismen behandelt.

3.1 Systeme mit sequentiellen Prozessen

Die Systeme mit sequentiellen Prozessen, die hier eingeführt und erklärt werden, sind Verallgemeinerungen der in Kapitel 2 behandelten Modellsysteme. Die Verallgemeinerung besteht darin, daß anstelle der atomaren Prozesse sequentielle Prozesse als Bausteine benutzt werden: Ein Prozeß führt jetzt Berechnungen aus, die aus einer Folge von w–atomaren Phasen bestehen.
Die Motivation für diese Verallgemeinerung ist klar: Ein Prozeß führt sequentielle Berechnungen aus, deren Fortschreiten den jeweiligen Anforderungen entsprechend differenziert beschrieben werden soll. Von den in Kapitel 2 behandelten Systemen ausgehend ergeben sich für die Verallgemeinerung zwei Ansätze. Der erste ergibt sich aus den in Kapitel 2.4 erklärten Systemen mit atomaren Prozessen und antisymmetrischer Konfliktvermeidung. Für ein entsprechendes System kann mit der Vorrangrelation eine lineare Ordnung der Prozesse festgelegt sein, so daß sich genau eine vollständige Ereignisspur einer sequentiellen Berechnung ergibt. Die Folge der Prozesse dieser Spur kann zu einem Prozeß vergröbert werden. Der zweite Ansatz geht von w–atomaren Prozessen als BBs aus und verfeinert diese durch Übergang zu GBs, die zusammengefaßt Folgen von w–atomaren Phasen sind. Im Gegensatz zum ersten Ansatz bleiben hierbei die Prozesse Einheiten, die in der GB–Sicht aus Phasenfolgen bestehen; die Vergröberungen des ersten Ansatzes werden damit als Spezialfälle miterfaßt.
Für die Systeme mit sequentiellen Prozessen sind wieder uninterpretierte und interpretierte zu unterscheiden. Die Erweiterung eines uninterpretierten Systems zu einem interpretier-

ten erfolgt formal so, wie es in Kapitel 2 erklärt ist. Im folgenden werden im wesentlichen uninterpretierte Systeme erklärt; die benötigten Interpretationen werden bei Bedarf ergänzt.

Ein **uninterpretiertes System mit sequentiellen Prozessen** wird durch 6–tupel $\mathcal{R} = (D, P, \alpha, \kappa, X, Y)$ definiert. Dabei sind:

- D die nichtleere, endliche, linear geordnete Menge der **Datenobjekte** des Systems;
- P die nichtleere, endliche Menge der **Prozesse**;
- $\alpha : P \to I\!N - \{0\}$ eine totale Abbildung, die **Phasenanzahlfunktion**;
- mit P und α sind
  - für $p \in P$ die linear geordnete Phasenmenge $A(p) = \{p_1, \ldots, p_{\alpha(p)}\}$ des Prozesses und
  - die **Phasenmenge des Systems** $PP = \bigcup_{p \in P} A_p$

  definiert;
- $\kappa \subseteq PP \times PP$ die **Konfliktvermeidungsrelation**;
- $X = (X_p \mid p \in P)$ die Familie der **Anforderungsfunktionen** der Prozesse mit einer totalen Abbildung $X_p : A(p) \to POT(D)$ für $p \in P$;
- $Y = (Y_p \mid p \in P)$ die Familie der **Freigabefunktionen** der Prozesse mit einer totalen Abbildung $Y_p : A(p) \to POT(D)$ für $p \in P$.

⌋

Im weiteren sei $\mathcal{R}$ ein System mit den angegebenen Bezeichnungen.
Für $\mathcal{R}$ ist festgelegt, daß seine Berechnungen die seiner Prozesse gemäß P und für jedes $p \in P$ die der Phasenfolge gemäß $A(p)$ sind; die einzelnen **Phasen** der Prozesse sind **w–atomar**. Alle weiteren für $\mathcal{R}$ festgelegten Eigenschaften sind auf diese 2–stufigkeit der Berechnungen von $\mathcal{R}$ ausgerichtet.
Für einen Prozeß $p \in P$ ist gemäß Kapitel 2 wesentlich der Wirkungsbereich W_p, also die Menge aller D–Elemente, die p für seine Berechnungen benötigt, festzulegen. Diese Wirkungsbereichsfestlegungen erfolgen der Phasenstruktur von p entsprechend so, daß die einzelnen Phasen verschiedene Wirkungsbereiche haben können; dazu dienen X_p und Y_p. X_p ermöglicht Erweiterungen und Y_p Reduktionen des Wirkungsbereichs; dabei sind $X_{p_i} = X_p(p_i)$ und $Y_{p_i} = Y_p(p_i)$ für $p_i \in A(p)$. D–Elemente können dem Wirkungsbereich von p für einzelne Phasen oder für mehrere aufeinanderfolgende Phasen angehören. Diesen Möglichkeiten entsprechend sind für den Prozeß p

- der **vergröberte Wirkungsbereich $\overline{W_p}$ der BB–Sicht** mit $\overline{W_p} = \bigcup_{i=1}^{\alpha(p)} X_{p_i}$ und
- die **Wirkungsbereichsfolge** $W_p = (W_{p_i} \mid p_i \in A(p))$ **der GB–Sicht** mit
  - $W_{p_1} = X_{p_1}$ und
  - $W_{p_{i+1}} = (W_{p_i} - Y_{p_i}) \cup X_{p_{i+1}}$ für $i \in \{1, \ldots, \alpha(p) - 1\}$

  zu unterscheiden.

Der Phasenstruktur der Prozesse von $\mathcal{R}$ entsprechend ist die **Konfliktvermeidungsrelation** κ auf der Phasenmenge PP definiert. κ enthält

- wenigstens die linearen Ordnungen über den $A(p)$, $\boxplus_p$, der $p \in P$ und
- weitere zur Interferenzvermeidung und/oder Vorrangregelung erforderliche

Reihenfolgerestriktionen für die Phasen der Prozesse von $\mathcal{R}$.
Das Verhalten des Systems $\mathcal{R}$ wird wie früher erklärt mit Ereignisspuren beschrieben; dazu werden den Prozessen von $\mathcal{R}$ Ereignismengen zugeordnet. Für den Prozeß $p \in P$ werden jeder Phase $p_i \in A(p)$ das Initiierungsereignis $\underline{p_i}$ und das Terminierungsereignis $\overline{p_i}$ zugeordnet; die Menge dieser Ereignisse ist die Ereignismenge E_p des Prozesses p. Mit der kausalen Ordnung der Phasenereignisse und der linearen Ordnung der Phasen von p ist E_p linear geordnet; diese Ordnung wird ebenfalls mit $\boxplus_p$ bezeichnet. Für $\mathcal{R}$ ergibt sich mit den Ereignismengen der Prozesse

- die **Ereignismenge von** $\mathcal{R}$ mit $E = \bigcup_{p \in P} E_p$.

Eine **Ereignisspur von** $\mathcal{R}$ ist eine Folge $\eta = \varepsilon^1 \ldots \varepsilon^k$ von Ereignissen aus E, für mit κ die früher erklärten Regeln gelten; dabei gilt $0 \leq k \leq 2 * |PP|$. Das Verhalten von $\mathcal{R}$ wird durch die Menge H der vollständigen Ereignisspuren von $\mathcal{R}$ beschrieben.
Bei gegebenem uninterpretiertem $\mathcal{R}$ ergibt sich ein **interpretiertes System** mit sequentiellen Prozessen durch Erweiterung um eine Interpretation. Für eine Interpretation von $\mathcal{R}$ sind

- der Zustandsraum Z von D,
- die Vor– und die Nachbereiche der Phasen der Prozesse von $\mathcal{R}$ mit einer Familie $V = (V_p \mid p \in P)$ von Vorbereichsfunktionen und einer Familie $N = (N_p \mid p \in P)$ von Nachbereichsfunktionen, die verträglich mit der Familie $(W_p \mid p \in P)$ der Wirkungsbereichsfolgen sein müssen, und
- die Familie $R = (R_p \mid p \in P)$ der Rechenfunktionen der Phasen der Prozesse von $\mathcal{R}$

festzulegen. Damit ist $\mathcal{I} = (Z, V, N, R)$ eine **Interpretation von** $\mathcal{R}$. Wie früher erklärt sind für die Wirkungsbereiche der Phasen der Prozesse von $\mathcal{R}$ weitere Differenzierungen möglich und ggf. zweckmäßig.
Das Verhalten des interpretierten Systems $(\mathcal{R}, \mathcal{I})$ wird wie früher erklärt beschrieben. Für $z \in Z$ und ein $\eta \in$ H wird die z und η **zugeordnete Zustandsspur** $\zeta(z, \eta)$ mit den Rechenfunktionen der Phasen der Prozesse von $\mathcal{R}$ definiert.
Im folgenden ist ein Beispiel für ein interpretiertes System angegeben, das einige Charakteristika der Systeme mit sequentiellen Prozessen zeigt.

Beispiel:

Für das System $(\mathcal{R}, \mathcal{I})$ soll gelten:

- $D = \{px, py, pz, qx, qy, qz, x, y\}$; die D–Elemente sind Ganzzahl–Datenobjekte.
- $P = \{p, q\}$ mit $\alpha(p) = \alpha(q) = 5$.

- Die Phasen der Prozesse haben folgende Rechenfunktionen:

$p:$	$p_1:$	$pz := px + py$	$q:$	$q_1:$	$qz := qx - qy$
	$p_2:$	$x := pz$		$q_2:$	$x := qz$
	$p_3:$	$x := f(x, y)$		$q_3:$	$x := g(x, y)$
	$p_4:$	$pz := x$		$q_4:$	$qz := x$
	$p_5:$	$pz := px * pz$		$q_5:$	$qz := qz - qy$

  dabei sind f und g definierte Ganzzahlfunktionen.

Mit den Rechenfunktionen der Phasen sind deren Wirkungsbereiche festgelegt. Mit diesen seien die folgenden Wirkungsbereiche der Prozesse definiert:

- $\overline{W_p} = D_p \cup \{x, y\}$ mit $D_p = \{px, py, pz\}$ und
- $W_p = (D_p, D_p \cup \{x\}, D_p \cup \{x, y\}, D_p \cup \{x\}, D_p)$.
- $\overline{W_q} = D_q \cup \{x, y\}$ mit $D_q = \{qx, qy, qz\}$ und
- $W_q = (D_q, D_q \cup \{x\}, D_q \cup \{x, y\}, D_q \cup \{x\}, D_q)$.

Für die Relation κ gilt zunächst $\boxplus_p, \boxplus_q \subseteq \kappa$.
Zur Interferenzvermeidung sind weitere Restriktionen erforderlich; sie werden mit der wA–Relation $\otimes$ für alle Phasen der Teilmenge $\{p_2, p_3, p_4, q_2, q_3, q_4\}$ von PP festgelegt.
Abbildung 3.1 veranschaulicht die für $\mathcal{R}$ durch $\kappa = \boxplus_p \cup \boxplus_q \cup \otimes$ festgelegten Restriktionen. Sie zeigt die Vorranggraphen der beiden Prozesse mit den Phasen als Knoten; zusätzlich ist $\otimes$ veranschaulicht.

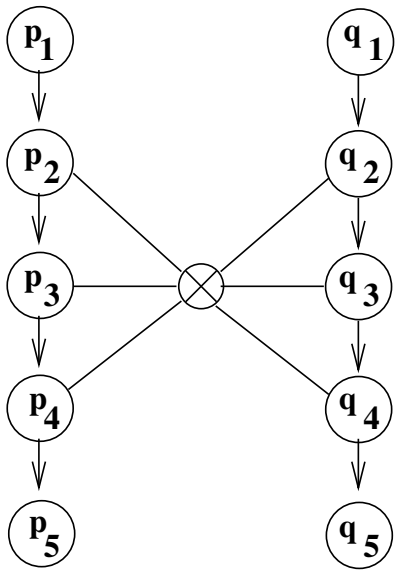


Abbildung 3.1: Ein System mit sequentiellen Prozessen und zusätzlichen Reihenfolgerestriktionen

Eine der vollständigen Ereignisspuren von $\mathcal{R}$ ist
$\eta = \underline{q_1}\ \underline{p_1}\ \overline{p_1}\ \underline{p_2}\ \overline{q_1}\ \overline{p_2}\ \underline{p_3}\ \overline{p_3}\ \underline{p_4}\ \overline{p_4}\ \underline{q_2}\ \underline{p_5}\ \overline{q_2}\ \underline{q_3}\ \overline{p_5}\ \overline{q_3}\ \underline{q_4}\ \overline{q_4}\ \underline{q_5}\ \overline{q_5}$
Man erkennt, wie die Kombination von Vorrang– und wA–Restriktionen sich auf die Ereignisspuren auswirken.

⌋

Die erklärten Modellsysteme mit sequentiellen Prozessen bieten vielfältige und flexible Möglichkeiten, Eigenschaften von parallelen Rechen– und Betriebssystemen zu beschreiben und zu analysieren. Diese Vielfalt und Flexibilität führt dazu, daß schnell Systeme entstehen, deren Beschreibungen und Analysen hohen Aufwand erfordern. Dabei gilt, daß der für Beschreibungen und Analysen zu leistende Aufwand proportional zur Anzahl der Komponenten eines Systems und der Komplexität ihrer Abhängigkeiten ist. Für die Konstruktion eines Systems ergibt sich damit die Notwendigkeit

- die Granularität der GB–Sicht und
- den Wert der nutzbaren Eigenschaften der BB–Sicht

abzuwägen und „zweckmäßig“ zu wählen. Für diese **Kosten/Nutzen–Analysen** sind Systemalternativen und Vergleiche zwischen diesen Alternativen erforderlich. Hierfür sind Modellsysteme unverzichtbar notwendig, und von den benutzten Modellen ist wichtig, daß mit ihnen einfach und flexibel „Experimente“ durchgeführt werden können.
Mit dem oben angegebenen Beispielsystem können einige **Experimente** durchgeführt werden, die zeigen, wie bei der Auswahl eines zweckmäßigen Systems vorzugehen ist.
Von dem als Beispiel angegebenen System $(\mathcal{R}, \mathcal{I})$ ausgehend stellt sich die Frage nach **zweckmäßigen Vereinfachungen**; sie ergeben sich durch Vergröberungen.
Eine Möglichkeit, $(\mathcal{R}, \mathcal{I})$ stark zu vergröbern, besteht darin, daß die beiden sequentiellen Prozesse zu zwei w–atomaren Prozesse vergröbert werden. Wenn diese beiden Prozesse wieder mit p und q bezeichnet werden, ist klar, wie die Eigenschaften von p und q aus den gegebenen berechnet werden. Zudem ist klar, daß $p \otimes q$ festgelegt werden muß. Als Charakteristikum für das Verhalten dieses vergröberten Systems ergeben sich 2 vollständige Ereignisspuren.
Eine weitere Möglichkeit, $(\mathcal{R}, \mathcal{I})$ zu vergröbern, besteht darin, die beiden sequentiellen Prozesse zu 3–phasigen Prozessen
$p \to a = (a_1, a_2, a_3)$ mit $a_1 \equiv p_1, a_2 \equiv (p_2, p_3, p_4)$ und $a_3 \equiv p_5$,
$q \to b = (b_1, b_2, b_3)$ mit $b_1 \equiv q_1, b_2 \equiv (q_2, q_3, q_4)$ und $b_3 \equiv q_5$
zusammenzufasssen. Auch hier ist klar, wie die Eigenschaften von a und b zu berechnen sind. Neben den linearen Ordnungen der Prozeßphasen ist $a_2 \otimes b_2$ festzulegen. Damit ergibt sich auch für dieses vergröberte System eine wesentliche Verringerung der Anzahl seiner vollständigen Ereignisspuren gegenüber $\mathcal{R}$.
Die beiden angegebenen Vergröberungen unterscheiden sich wesentlich dadurch, daß sie

- im ersten Fall zu einem System mit sequentiellen Berechnungen und
- im zweiten Fall zu einem System mit parallelen Berechnungen mit Sequentialisierung von a_2 und b_2, also der auch mit $\mathcal{R}$ sequentialisierten Teilberechnungen,

führen. Das kann zu wesentlich verschiedenen Ausführungszeiten der Berechnungen der Systeme führen. Diese Beispiele lehren, daß jeweils maximale Vergröberungen, welche die Parallelität erhalten, zweckmäßig sind.
Von den Modellen, die für diese Experimente eingesetzt werden, ist gefordert, daß ihr Beschreibungsinstrumentarium flexibel an die jeweiligen Anforderungen angepaßt werden kann.
Die Flexibilität der Modellsysteme mit sequentiellen Prozessen ergibt sich insbesondere daraus, daß ihre Grundstrukturen **einfache Serien–Parallel–Kombinationen** sind:

- die Prozesse sind Serienkombinationen ihrer Phasen;

- die Systeme sind Parallelkombinationen ihrer Prozesse;
- hinzu kommen Abhängigkeiten zwischen den Prozessen, die sich wesentlich daraus ergeben, daß die Phasen der Prozesse Datenobjekte der Systeme gemeinsam benutzen.

Wie diese Grundstruktur für Vergröberungen genutzt werden kann, haben die Beispiele oben gezeigt. Die Grundstruktur der Prozesse kann für Erweiterungen von Prozessen mit Serienkombinationen genutzt werden. Wenn p ein sequentieller Prozeß und $n \in \mathbb{N}$ sind, kann insbesondere

- der sequentielle Prozeß p^n als n–**fache Serienkombination von** p, d.h. von Phasen mit den Wirkungsbereichen der Phasen von p

konstruiert werden. Zudem kann für die Prozesse auf die bekannten Verfahren für Serienkombinationen zurückgegriffen werden: Prozesse führen sequentielle Berechnungen aus, und folglich können interpretierte Prozesse mit **sequentiellen Programmen** konstruiert werden.

3.2 Die Grundstruktur für Realisierungen

Sei $(\mathcal{R}, \mathcal{I})$ ein System mit sequentiellen Prozessen; dabei sei $\mathcal{R} = (D, P, \alpha, \kappa, X, Y)$.
Die in Kapitel 3.1 erklärten Eigenschaften von Systemen mit sequentiellen Prozessen werden im folgenden dazu genutzt, $(\mathcal{R}, \mathcal{I})$ so umzustrukturieren, daß sich eine geeignete Ausgangsbasis für Realisierungen dieses Systems und allgemein für parallele Mehrbenutzersysteme ergibt.
Wenn $(\mathcal{R}, \mathcal{I})$ Modell für ein paralleles Mehrbenutzersystem sein soll, müssen für die Berechnungen, die ausgeführt werden sollen, mit $(\mathcal{R}, \mathcal{I})$ zwei Forderungen erfüllt werden:

- die Soll–Wirkungen der Berechnungen sind spezifiziert;
- die Reihenfolgerestriktionen für die Phasen der Prozesse sind mit κ so festgelegt, daß die Soll–Wirkungen genau mit den vollständigen Ereignisspuren $\eta \in \mathrm{H}_{\mathcal{R}}$ erreicht werden.

Das sei für $(\mathcal{R}, \mathcal{I})$ der Fall. Für Realisierungen eines Systems mit den Eigenschaften von $(\mathcal{R}, \mathcal{I})$ müssen die Berechnungen der Prozesse so koordiniert werden, daß sich jeweils eine Ereignisspur aus $\mathrm{H}_{\mathcal{R}}$ ergibt. Diese **Koordinierung** muß mittels Berechnungen erfolgen, und dafür sind Koordinierungs–Datenobjekte und –Aktionen erforderlich. Damit ergeben sich für Realisierungen zwei zu lösende Teilaufgaben:

- Es werden geeignete Koordinierungsobjekte und –aktionen benötigt.
- Die Koordinierungsobjekte und –aktionen sind so in das System zu integrieren, daß die Prozesse des Systems ihre Berechnungen koordiniert mit dem angegebenen Ziel — es ergibt sich jeweils eine Ereignisspur aus $\mathrm{H}_{\mathcal{R}}$ — ausführen.

Geeignete Koordinierungsobjekte und –aktionen werden im folgenden Kapitel erklärt; hier soll deren Integration (vorläufig) geklärt werden.
Von $(\mathcal{R}, \mathcal{I})$ ausgehend ist naheliegend, wie diese Integration formal durchzuführen ist: $(\mathcal{R}, \mathcal{I})$ wird erweitert, und zwar dadurch, daß die benötigten Koordinierungsobjekte zu D hinzugenommen werden, und dadurch, daß die Phasenfolgen der Prozesse um Koordinierungsphasen erweitert werden.

Die wie skizziert konstruierte Erweiterung von $(\mathcal{R},\mathcal{I})$ sei das System $(\mathcal{R}',\mathcal{I}')$ mit $\mathcal{R}' = (D', P', \alpha', \kappa', X', Y')$. Mit der Erweiterung wächst die Komplexität, so daß sich, wie in Kapitel 3.1 erklärt, die Notwendigkeit zur **Vereinfachung** von $(\mathcal{R}',\mathcal{I}')$ ergibt. Diese Vereinfachung erfolgt durch maximale, die Parallelität erhaltende Vergröberungen und durch Strukturierung der Komponentenmenge, die sich ergibt, nach ihren Abhängigkeiten.
Die Aufgabe, die Granularität der Komponenten der GB–Sicht eines Systems geeignet festzulegen und die resultierende Komponentenmenge geeignet zu strukturieren, gehört wesentlich au der Aufgabe, die Berechnungen der Prozesse des Systems mittels Berechnungen zu koordinieren. Hier wird lediglich eine Grobstruktur angegeben, die im weiteren als Ausgangsbasis für Erklärungen von Lösungen der Koordinierungsaufgaben und für Verfeinerungen dient.
Die **Grundstruktur für Realisierungen** von $(\mathcal{R}',\mathcal{I}')$ geht von einer Vergröberung der zu lösenden Koordinierungsaufgabe aus: Sie muß gewährleisten, daß die Berechnungen der Prozesse des Systems vollständig und interferenzfrei ausgeführt werden.
Die Kriterien für Interferenzfreiheit sind bekannt. Mit diesen Kriterien kann für $(\mathcal{R}',\mathcal{I}')$ eine **Zerlegung der Menge** D' der Datenobjekte des Systems in $|P|+1$ Teilmengen konstruiert werden, so daß sich

$$D' = \{D'_p \mid p \in P'\} \cup D'_\kappa$$

ergibt. Dabei gelten:

- für $p \in P'$ ist D'_p die Menge der Datenobjekte, die allein für die p–Berechnungen benötigt und benutzt werden; D_p ist der p–**private Teil von** D' und D'_p ist Teil des Wirkungsbereichs von p;
- D'_κ ist die Restmenge von D' bzgl. $\{D_p \mid p \in P\}$; D'_κ enthält gemeinsam benutzbare Datenobjekte und D'_κ ist der potentiell **kritische Teil von** D'.

Abbildung 3.2 veranschaulicht die Zerlegung der Datenobjektmenge D' und die Grundstruktur für Realisierungen von $(\mathcal{R}',\mathcal{I}')$, die sich damit ergibt.

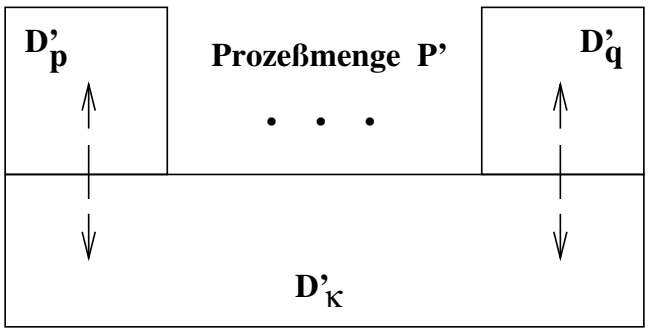


Abbildung 3.2: Die Zerlegung von D' und die Grundstruktur für Realisierungen von $(\mathcal{R}',\mathcal{I}')$

Die angegebene Zerlegung vereinfacht die Lösung der Koordinierungsaufgabe dadurch, daß sie die D'–Elemente in Klassen einteilt und zusammenfaßt. Es ergeben sich

- die D'_p der $p \in P'$, für die Interferenzfreiheit dadurch gewährleistet wird, daß die $d \in D'_p$ allein für die sequentiellen Berechnungen von p benutzt werden, und
- D'_κ als Zusammenfasssung der $d \in D'$, für die Interferenzfreiheit (und die weiteren Koordinierungsziele) mit zusätzlichen Maßnahmen erreicht werden müssen.

Für Realisierungen von $(\mathcal{R}', \mathcal{I}')$ ist wesentlich, daß die angegebenen Vereinfachungen nur dann erreicht werden, wenn die Elemente der angegebenen Zerlegung, also die D'_p für $p \in P'$ und D'_κ, durch **Grenzwälle** so getrennt werden, daß unkontrollierte und unzulässige Übergänge zwischen den Zerlegungselementen ausgeschlossen sind.
Die angegebene Zerlegung ist lediglich der erste Schritt zur Lösung der gestellten Koordinierungsaufgaben: Erreicht ist, daß die zur Koordinierung erforderlichen Maßnahmen auf D'_κ konzentriert und i.w. auf diese beschränkt werden können. Die **Komposition und die Strukturierung von** D'_κ ist der wesentliche Teil der Koordinierungsaufgaben, die noch zu lösen sind.
Für die **Berechnungen der Prozesse** von $(\mathcal{R}', \mathcal{I}')$ gilt nach wie vor, daß sie durch die Phasenstrukturen der Prozesse und durch die Wirkungsbereiche der Phasen bestimmt sind. Für den Prozeß $p \in P'$ ergibt sich, daß p seine Berechnungen teilweise mit Wirkungsbereichen, die zu D'_p gehören, und teilweise mit Wirkungsbereichen, die zu D'_κ gehören, ausführt. Mit D'_p und D'_κ als vergröberte Umgebungen für die Ausführung der Berechnungen von p ergibt sich, daß D'_p–D'_κ–**Umgebungswechsel**, also Übergänge von D'_p zu D'_κ und Übergänge von D'_κ zu D'_p, zu den bzgl. der mit der Zerlegung festgelegten Vergröberung wichtigen Ereignissen der Spuren der Berechnungen von p werden. Für Lösungen der Koordinierungsaufgaben sind die Teile der Ereignisspuren der Prozesse mit der Umgebung D'_κ ausschlaggebend und den Koordinierungszielen entsprechend zu konstruieren.
Für **Lösungen der Koordinierungsaufgaben** ist das System $(\mathcal{R}, \mathcal{I})$, wie eingangs erklärt, gegeben; das System $(\mathcal{R}', \mathcal{I}')$ wird als Erweiterung von $(\mathcal{R}, \mathcal{I})$ so konstruiert, daß die Berechnungen der Prozesse von $(\mathcal{R}, \mathcal{I})$ mit Berechnungen von $(\mathcal{R}', \mathcal{I}')$ koordiniert werden. Die für $(\mathcal{R}', \mathcal{I}')$ angegebene Zerlegung von D' impliziert eine entsprechende Zerlegung für D. Diese Zusammenhänge zwischen $(\mathcal{R}, \mathcal{I})$ und $(\mathcal{R}', \mathcal{I}')$ werden für Lösungen der Koordinierungsaufgaben in 2–facher Hinsicht genutzt:

- Für $(\mathcal{R}, \mathcal{I})$ können die Eigenschaften der Phasen der Prozesse $p \in P$ mit Wirkungsbereichen, die zu D_p gehören, vergröbert festgelegt werden; wesentlich sind die Eigenschaften der Phasen mit Wirkungsbereichen, die zu D_κ gehören. Für Festlegungen der benötigten, wesentlichen Eigenschaften ist es zweckmäßig und ausreichend, von einem **vergröberten, uninterpretierten** $\mathcal{R}$ auszugehen.

- Wenn $(\mathcal{R}', \mathcal{I}')$ als Erweiterung von $(\mathcal{R}, \mathcal{I})$ und als Lösung der Koordinierungsaufgaben von $(\mathcal{R}, \mathcal{I})$ konstruiert ist, ist der **Beweis** dafür, **daß** $(\mathcal{R}', \mathcal{I}')$ **Lösung ist**, dadurch zu führen, daß folgendes bewiesen wird: Wenn η' eine Ereignisspur von $\mathcal{R}'$ und η die Projektion von η' auf die Ereignisse der Prozesse von $\mathcal{R}$ ist, dann ist η Ereignisspur von $\mathcal{R}$.

  Mit den Objekten und Aktionen, um die $(\mathcal{R}, \mathcal{I})$ zu $(\mathcal{R}', \mathcal{I}')$ erweitert wird, sollen die Koordinierungsaufgaben für $(\mathcal{R}, \mathcal{I})$ durch Berechnungen gelöst werden. Für diese Aktionen sind entsprechend Interpretationen erforderlich. Wenn anstelle von $(\mathcal{R}, \mathcal{I})$ von einem vergröberten, uninterpretierten $\mathcal{R}$ für die Lösung der Koordinierungsaufgaben ausgegangen wird, wird als Lösung anstelle von $(\mathcal{R}', \mathcal{I}')$ ein **gemischt uninterpretiertes–interpretiertes System** konstruiert; es erfaßt die Eigenschaften, die für die oben angegebenen Beweise erforderlich sind.

Die angegebene Grundstruktur für Realisierungen paralleler Mehrbenutzersysteme und die erklärte Vorgehensweise für Lösungen der Koordinierungsaufgaben ermöglichen es, entsprechende interpretierte **Systeme mit höheren Programmiersprachen zu konstruieren**. Das erfolgt zunächst dadurch, daß Prozesse mit sequentiellen Programmen definiert und zu einem System zusammengefaßt werden. Mit den benötigten Koordinierungsobjekten und

–aktionen, die ebenfalls programmiersprachlich definiert werden können, wird dann das gegebene System so erweitert, daß sich ein programmiersprachlich definiertes, paralleles Mehrbenutzersystem mit den charakteristischen Abhängigkeiten zwischen den Prozessen ergibt.

3.3 Koordinierungskonzepte und –mechanismen

In diesem Kapitel wird die Vorgehensweise bei Lösungen der für Betriebssysteme grundlegenden Koordinierungsaufgaben behandelt.
Wie schon erklärt, werden zu einem gegebenen Modellsystem mit spezifiziertem Verhalten Koordinierungsobjekte und –aktionen benötigt, mit denen durch Erweiterung ein System so konstruiert werden kann, daß dieses seine Berechnungen ohne zusätzliche (manuelle) Maßnahmen ausführt und dabei das spezifizierte Verhalten gewährleistet. Die Anforderungen, die an die Koordinierung gestellt werden können, ergeben sich aus den in Kapitel 2 erklärten Möglichkeiten für Abhängigkeiten zwischen den Prozessen eines Systems; sie variieren also von Fall zu Fall. Für alle diese Fälle sind Hilfsmittel und Verfahren zur Konstruktion der jeweiligen Lösungen anzugeben.
Im weiteren sei $\mathcal{R} = (D, P, \alpha, \kappa, X, Y)$ das jeweils gegebene System mit spezifiziertem Verhalten. Für $p \in P$ sind $A(p)$ die Phasenmenge von p und $PP = \cup_{p \in P} A(p)$ ist die Phasenmenge von $\mathcal{R}$. Die Menge D sei, wie in Kapitel 3.2 erklärt, zerlegt: $D = \{D_p \mid p \in P\} \cup D_\kappa$. Der Zerlegung von D entspricht eine Zerlegung der Phasenmenge von $\mathcal{R}$ mit

- $AK(p) \subseteq A(p)$ ist die Menge der kritischen Phasen des Prozesses $p \in P$ und
- $PK \subseteq PP, PK = \cup_{p \in P} AK(p)$, ist die Menge der kritischen Phasen von $\mathcal{R}$.

In die Zerlegungen von $A(p)$ in $AK(p)$ und $A(p) - AK(p)$ sowie von PP in PK und $PP - PK$ gehen insbesondere die mit κ festgelegten Reihenfolgerestriktionen, die über die linearen Ordnungen der Prozeßphasen hinaus gehen, ein.

3.3.1 Freier wechselseitiger Ausschluß

Der einfachste Fall für die Koordinierung liegt vor, wenn für Phasen von $\mathcal{R}$ freier wechselseitiger Ausschluß spezifiziert und dieser durch ein kritisches Datenobjekt begründet ist. Dieser Fall ist gut dafür geeignet, die Koordinierungsaufgaben, die Vorgehensweise bei Lösungen und die Konstruktion von Koordinierungsobjekten zu erklären.
Sei also $x \in D_\kappa$ das **kritische Datenobjekt** von $\mathcal{R}$, und für $\mathcal{R}$ sei $\kappa = \cup_{p \in P} \boxplus_p \cup \otimes$ mit

$$\otimes = \{(a, b) \mid a, b \in PK \wedge a \neq b\}.$$

Das Verhalten von $\mathcal{R}$ ist spezifiziert sagt hier also, daß Interferenzfreiheit bzgl. x (und damit insgesamt) mit $\otimes$ erreicht wird.
Für $\mathcal{R}$ sei $\mathrm{H}_\mathcal{R}$ die Menge der vollständigen Ereignisspuren. Das Verhalten von $\mathcal{R}$ ist wie folgt charakterisiert:

- Ist $\eta = \varepsilon^1 \ldots \varepsilon^l$ eine (partielle oder vollständige) Ereignisspur von $\mathcal{R}$, so gilt für alle $i \in \{1, \ldots, l\}$: höchstens ein $a \in AK$ ist aktiv in ε^i.

Diese Charakteristik von $\mathcal{R}$ wird im weiteren als wA–Bedingung für $\mathcal{R}$ mit wA–$\mathcal{R}$ bezeichnet.

3.3.1.1 Koordinierungsaufgabe und Lösungsmöglichkeiten

Die Koordinierungsaufgabe, die für $\mathcal{R}$ gestellt ist, lautet: wA–$\mathcal{R}$ ist konstruktiv durchzusetzen.
Die Prozesse p von $\mathcal{R}$ konkurrieren mit ihren Phasen $p_i \in AK(p)$ um die Benutzung des kritischen Objekts x; sie fordern, daß sie x jeweils interferenzfrei, also **w–atomar** benutzen können; es muß also erreicht werden, daß jeweils höchstens einer der Prozesse x benutzt. Abbildung 3.3 veranschaulicht diese Konkurrenz um x.

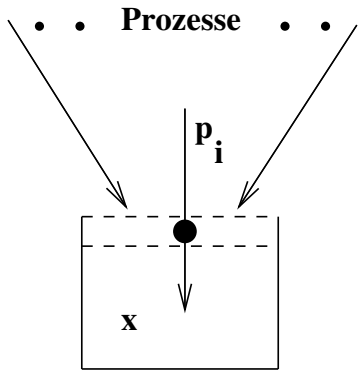


Abbildung 3.3: Die Prozesse von $\mathcal{R}$ konkurrieren um $x \in D_\kappa$

Für konstruktive Lösungen der gestellten Aufgabe ist offenbar erforderlich, **alle Zugriffe** von Prozessen zu x **zu kontrollieren** und den festgelegten Bedingungen entsprechend **zu reglementieren**. Dafür gibt es zwei alternative Möglichkeiten:

- Eine **dezentrale**, auf die Prozesse verteilte **Lösung**; dabei werden die Prozesse so erweitert, daß mit den Erweiterungen der Phasen $a \in AK$ die gestellten Bedingungen erfüllt werden.
- Eine **zentrale**, auf das kritische Objekt x zentralisierte **Lösung**; dabei wird x so konstruiert, daß für alle Zugriffe aller Prozesse zu x die gestellten Bedingungen erfüllt werden.

Von diesen Alternativen hat die dezentrale den offensichtlichen Nachteil, daß Erweiterungen aller Prozesse (mit Phasen aus PK) erforderlich sind; das ist fehleranfällig. Die zentrale Lösung, die hier vorzuziehen ist, führt bei Verallgemeinerungen auf Probleme, die noch erklärt werden.
Für beide Alternativen sind spezielle Hilfsmittel erforderlich. Zur systematischen Erfüllung der gestellten Bedingungen sind **Koordinierungsobjekte**, also Datenobjekte mit geeignet definierten Schnittstellenoperationen, nötig. Dabei werden zwei Schnittstellen– oder **Zugriffsoperationen** benötigt:

- eine Operation, mit der Prozesse die **Belegung** von x anfordern; diese Operation wird im weiteren mit `prolog` bezeichnet;
- eine Operation, mit der Prozesse die **Freigabe** des belegten x melden; diese Operation wird im weiteren mit `epilog` bezeichnet.

Für die Konstruktion eines Koordinierungsobjekts k ergibt sich, daß ein Datenobjekt r erforderlich ist, mit dessen Zustand $z(r)$ registriert wird, ob x belegt oder frei ist; Prozesse,

die `prolog` oder `epilog` auf k ausführen, müssen $z(r)$ lesen und verändern können. Damit folgt, daß für r w–Atomarität erforderlich ist. Als Fazit ergibt sich, daß zur Konstruktion des Koordinierungsobjekts k als Basis Hilfsobjekte, die w–atomar sind, benötigt werden, und k wird **relativ zur Basis** gegebener, w–atomarer Hilfsobjekte konstruiert.
Damit ist ein **Grundsatz der Vorgehensweise** zur Lösung der hier gestellten Koordinierungsaufgabe und der Koordinierungsaufgaben allgemein geklärt:

- Am Anfang der Lösung steht die **Auswahl der Basis** mit gegebenen, w–atomaren Objekten.
- Mit den Objekten der Basis werden **Koordinierungsobjekte** konstruiert, welche die zur Erfüllung der jeweils gestellten Bedingungen erforderlichen Eigenschaften haben.
- Mit den Koordinierungsobjekten können **Koordinierung– oder Nutzobjekte** mit weitergehenden Eigenschaften sukzessive konstruiert werden.

Für diese sukzessiven Konstruktionen, die zur Lösung von Koordinierungsaufgaben erforderlich sind, gilt weiter folgendes:

- Die Objekte, die zunächst als Basis für realisierte Lösungen der Koordinierungsaufgaben zur Verfügung stehen, sind die w–atomaren Objekte der jeweiligen **Hardwarekonfiguration** eines Systems; sie muß w–atomare Objekte zur Verfügung stellen, und mit den Eigenschaften dieser Objekte beginnt die schrittweise Konstruktion von Koordinierungsobjekten.
- Wenn Koordinierungsobjekte mit festgelegten Soll–Eigenschaften relativ zu einer Basis konstruiert werden, gehört der **Nachweis der Übereinstimmung von Ist– und Soll–Eigenschaften** wesentlich zur Lösung der Konstruktionsaufgaben. Der Aufwand für diese Nachweise wird wesentlich von der benutzten Basis bestimmt; dieser Aufwand ist das maßgebliche Kriterium für die Eignung der Basis.

Diese Zusammenhänge werden im weiteren noch deutlich werden.
Nach diesen Erklärungen zur Vorgehensweise werden im folgenden Lösungen für die hier gestellte Koordinierungsaufgabe behandelt. Zunächst, werden die **Anforderungen an Lösungen** explizit angegeben:

(1) wA–$\mathcal{R}$ ist konstruktiv durchzusetzen.

(2) Es sind keine Annahmen über die Reihenfolge, mit der die Prozesse von $\mathcal{R}$ das kritische Objekt x anfordern, zugelassen.

(3) Es sind keine Annahmen über die Ausführungszeiten der Berechnungen der Prozesse mit Ausnahme des Terminierens der Phasen zugelassen.

(4) Fairneß für alle Prozesse von $\mathcal{R}$.

Mit den Anforderungen (2) und (3) werden Lösungsversuche ausgeschlossen, die von speziellen Voraussetzungen über die Berechnungen der Prozesse ausgehen.
Die mit (4) geforderte **Fairneß** gehört zu den für parallele Systeme wichtigen Lebendigkeitseigenschaften (liveness property). Sie sagt, daß das **Verhungern** (starvation) von Prozessen verhindert werden muß: Es ist zu gewährleisten, daß ein Prozeß, der x anfordert, x nach endlicher Zeit erhält, so daß seine Berechnungen fortschreiten können. Die Forderung läßt noch zu, daß ein Prozeß von anderen überholt wird, so daß lange Wartezeiten möglich sind. Die

entsprechend schärfere Forderung ist **Fairneß mit linearem Warten**: Ein Prozeß, der x anfordert, erhält x, bevor ein anderer Prozeß x zum 2ten Mal erhält. Diese Fairneß–Forderung ist die praktisch wichtige, weil mit ihr Wartezeitschranken berechnet werden können. Sie wird noch verschärft mit der Forderung nach **FIFO–Fairneß**, die den Vorteil einfacher Realisierbarkeit hat.

3.3.1.2 Eine Lösung mit Speicherzellen als Basis

Für die gestellte Koordinierungsaufgabe wird im folgenden eine Lösung angegeben, die schwache Anforderungen an die Basis stellt: Als Basis werden **Speicherzellen mit w–atomaren Lese– und Schreiboperationen** vorausgesetzt.
Für die Lösung mit dieser Basis ist in Abbildung 3.4 ein Modul mit den beiden benötigten Zugriffsoperationen angegeben. Die Prozesse werden mit `pindex`–Werten identifiziert. Für die notwendigen Kontrollen werden das Feld `control` mit einer Komponente für jeden Prozeß und die `pindex`–Variable `k` benutzt. Die Werte der `control`–Komponenten werden von allen Prozessen gelesen; jeder Prozeß ändert allein den Wert seiner `control`–Komponente. Wenn ein Prozeß p das kritische Objekt x belegt, ist der Wert von `k` der `pindex`–Wert von p.
Der angegebene Modul macht den Zusammenhang deutlich, der zwischen den Eigenschaften der benutzten Basis und den Schwierigkeiten bei der Konstruktion eines Koordinierungsobjekts besteht. Diese Schwierigkeiten treten beim Nachweis der Erfüllung der Soll–Eigenschaften zu Tage; wegen der vielen Details, die beachtet werden müssen, ist dieser Nachweis hier aufwendig. Das angegebene Koordinierungsobjekt ist historisch interessant und wichtig; er ist das Ergebnis rd. 10–jähriger Bemühungen um die Präzisierung der Aufgabenstellung und ihre Lösung. E.W. Dijkstra veröffentlichte 1965 ein Programm, das Grundlage der hier angegebenen Lösung ist; es erfüllt jedoch nicht die Forderung (4). D.E. Knuth gab 1966 eine Lösung mit der Wartezeit $O(2^{n-1})$ mit der Anzahl n der Prozesse an. Diese Lösung wurde 1967 von N.G. de Bruijn auf die Wartezeit $O(n^2)$ verbessert. M.A. Eisenberg und M.R. McGuire gaben schließlich die hier wiedergegebene Lösung mit der Wartezeit $O(n)$ an.
Nachdem die Aufgabenstellung verstanden und das Wesentliche ihrer Lösung erkannt waren, konnten einfachere Koordinierungsobjekte konstruiert werden. Wesentliche Vereinfachungen ergeben sich mit FIFO–Warteräumen für Prozesse und mit entsprechenden w–atomaren Zugriffsoperationen der Koordinierungsobjekte.

3.3.1.3 Zählende Semaphore

Im folgenden werden zählende Semaphore als Koordinierungsobjekte eingeführt und erklärt. Sie verwenden FIFO–Warteräume für Prozesse und realisieren damit FIFO–Fairneß; zudem werden ihre Zugriffsoperationen als w–atomar vorausgesetzt. Zählende Semaphore sind geeignete, einfach zu handhabende Mechanismen zur Lösung der hier gestellten und weiterer Koordinierungsaufgaben. Neben zählenden gibt es **binäre Semaphore**; sie sind einfacher, aber sie erfüllen nicht die Anforderung (4).
Ein **zählendes Semaphor** ist ein Koordinierungsobjekt mit

- einer gekapselten Zählvariablen m vom Typ `integer`, die initial einen Wert ≥ 0 haben muß,
- einem gekapselten FIFO–Warteraum M für Prozeßidentifikatoren, der initial leer ist, und dessen Kapazität zur Speicherung aller Prozesse ausreichen muß, sowie
- den Zugriffsoperationen `prolog` und `epilog`, die w–atomar sind.

```
PACKAGE wa_mit_ls IS
   PROCEDURE prolog (i:IN pindex);
   PROCEDURE epilog (i:IN pindex);
END wa_mit_ls;

PACKAGE BODY wa_mit_ls IS
   SUBTYPE zustand IS integer RANGE 0..2;
   control: ARRAY (pindex) OF zustand := (0,...,0);
   k: pindex := pindex 'FIRST;
   PROCEDURE prolog (i:IN pindex) IS ... END prolog;
   PROCEDURE epilog (i:IN pindex) IS ... END epilog;
END wa_mit_ls;

PROCEDURE prolog (i:IN pindex) IS
   BEGIN
   L0: control(i):=1;
   L1: FOR j IN pindex RANGE k..pindex'LAST
         LOOP IF j=i THEN GOTO L3;
                 ELSIF control(j)≠0 THEN GOTO L1;
              END IF;
         END LOOP;
   L2: FOR j IN pindex
         LOOP IF j=i THEN GOTO L3;
                 ELSIF control(j)≠0 THEN GOTO L1;
              END IF;
         END LOOP;
   L3: control(i):=2;
   L4: FOR j IN pindex
         LOOP IF j≠i AND control(j)=2
                 THEN GOTO L0;
              END IF;
         END LOOP;
   L5: IF k≠i AND control(k)≠0 THEN GOTO L0; END IF;
   L6: k:=i;
END prolog;

PROCEDURE epilog (i:IN pindex) IS
   BEGIN
   L7:    FOR j IN pindex RANGE k..pindex'LAST
            LOOP IF j≠k AND control(j)≠0
                    THEN k:=j; GOTO L9;
                 END IF;
            END LOOP;
   L8:    FOR j IN pindex RANGE pindex'FIRST..k
            LOOP IF j≠k AND control(j)≠0
                    THEN k:=j; GOTO L9;
                 END IF;
            END LOOP;
   L9:    control(i):=0;
END epilog;
```

Abbildung 3.4: Ein Koordinierungsobjekt auf der Basis von Speicherzellen

Im folgenden wird für zählende Semaphore mit Abbildung 3.5 ein **generischer Modul** angegeben, der dazu dient, Semaphore als Koordinierungsmechanismen bei Bedarf zu erzeugen. Die programmiersprachliche Definition ist unvollständig; sie legt jedoch alle zum Verständnis und zur Nutzung der Semaphore nötigen Eigenschaften fest. Die Unvollständigkeit der Definition ergibt sich daraus, daß der FIFO–Warteraum, den ein Semaphor gekapselt enthält, als definiert vorausgesetzt und als solcher benutzt wird. Für die benötigten Warteräume ist in Abbildung 3.5 ebenfalls der Spezifikationsteil eines generischen Moduls festgelegt, so daß die Eigenschaften, die für ihre Nutzung erforderlich sind, bekannt sind; die weiteren Eigenschaften und die Gründe für die unvollständige Definition werden noch erklärt.
Der **generische Modul** `passives_warten` für FIFO–Warteräume ist mit dem generischen Parameter `kap` definiert; mit ihm wird bei Inkarnierung die Kapazität des Warteraums festgelegt. Für jeden Warteraum sind die Zugriffsoperationen `blockiere` und `entblockiere` definiert. Mit `blockiere` wird ein Prozeßidentifikator in den Warteraum eingefügt. Mit `entblockiere` wird ein Prozeßidentifikator aus dem Warteraum entnommen, wobei vorausgesetzt ist, daß der Warteraum nicht leer ist; diese Bedingung wird von dem Semaphor, das den Warteraum enthält, gewährleistet.

```
GENERIC
  kap:IN positive;
PACKAGE passives_warten IS
  PROCEDURE blockiere (x:IN pindex);
  PROCEDURE entblockiere;
END passives_warten;

GENERIC
  init:IN natural;
PACKAGE semaphor IS
  ATOMIC PROCEDURE prolog (i:IN pindex);
  ATOMIC PROCEDURE epilog;
END semaphor;

PACKAGE BODY semaphor IS
  m:integer := init;
PACKAGE M IS NEW passives_warten (pindex'LAST);
PROCEDURE prolog (i:IN pindex) IS
  BEGIN m := m-1;
        IF m<0 THEN M.blockiere(i); END IF;
  END prolog;
PROCEDURE epilog IS
  BEGIN m := m+1;
        IF m≤0 THEN M.entblockiere; END IF;
  END epilog;
END semaphor;
```

Abbildung 3.5: Ein generischer Modul für zählende Semaphore

Der generische Modul `semaphor` ist mit dem generischen Parameter `init` definiert; mit ihm wird bei Inkarnierung die Semaphor–Zählvariable initialisiert.
Sei jetzt *sem* **ein Semaphor** mit der Zählvariablen m und dem Warteraum M.

Jede Ausführung von `prolog` bewirkt die Dekrementierung von m. Darüber hinaus sind zwei Fälle möglich:

- die Ausführung mit `blockieren`, die bewirkt, daß der Identifikator des ausführenden Prozesses in M eingefügt, und der Prozeß blockiert wird;
- die Ausführung ohne `blockieren`, die keine weitere Wirkung hat.

Jede Ausführung von `epilog` bewirkt die Inkrementierung von m. Darüber hinaus sind zwei Fälle möglich:

- die Ausführung mit `entblockieren`, die mit $m \leq 0$ bedingt ist und bewirkt, daß der erste Prozeß gemäß FIFO aus M entnommen und entblockiert wird;
- die Ausführung ohne `entblockieren`, die keine weitere Wirkung hat.

Für diese Wirkungen der `sem`–Zugriffsoperationen gilt:

- Die Berechnungen eines Prozesses p, der sich mit `prolog` blockiert, schreiten nicht fort bis p von einem Prozeß $q \neq p$ mit `epilog` entblockiert wird. Das wird wie folgt formuliert: p ist im **Zustand blockiert oder wartend bzgl. sem**.

Dieser Teil der Festlegungen der Wirkungen der `sem`–Zugriffsoperationen liefert die Gründe dafür, daß Semaphore hier unvollständig definiert werden. Die Festlegungen lassen offen, ob Prozesse aktiv oder passiv warten. Für abstrakte Prozesse ist diese Unterscheidung irrelevant; wesentlich ist, daß die Berechnungen blockierter/wartender Prozesse nicht fortschreiten. Für realisierte Prozesse ist die Unterscheidung wichtig; passiv wartende Prozesse geben ihre Prozessoren (für andere Prozesse) frei.
Semaphore als Koordinierungsobjekte wurden von E.W. Dijkstra eingeführt. Ihre Eignung ergibt sich aus der w–Atomarität ihrer Zugriffsoperationen und daraus, daß die koordinierungsrelevanten Eigenschaften eines Systems mit Semaphoren durch die jeweiligen Werte der Zählvariablen und durch die jeweilige Anzahl der blockierten Prozesse beschrieben werden können. Als Reminiszenz für die Verdienste von E.W. Dijkstra werden die Zugriffsoperationen der Semaphore meist mit P (protekt für `prolog`) und V (vrej für `epilog`) bezeichnet.
Hier werden in weiteren die Zugriffsoperationen der Semaphore unter Bezugnahme auf den angegebenen Modul mit *pro* (für `prolog`) und mit *epi* (für `epilog`) bezeichnet; der Parameter von `prolog` (der Identifikator des jeweils ausführenden Prozesses) wird bei *pro* weggelassen.
Sei *sem* ein Semaphor mit der Zählvariablen m und dem Warteraum M. Zur Beschreibung der **Koordinierungscharakteristika** eines Systems mit *sem* werden folgende Bezeichnungen benutzt:

- *pro*/*epi*: eine Ausführung von `prolog` bzw. `epilog`;
- pro^+/pro^-: eine Ausführung von `prolog` mit/ohne `blockiere`;
- epi^+/epi^-: eine Ausführung von `epilog` mit/ohne `entblockiere`.

Die Anzahl der entsprechenden Ereignisse in Spuren wird mit # bezeichnet.
Sei η eine Ereignisspur eines Systems mit *sem*. Dann gelten folgende **Gleichungen für die Koordinierungscharakteristika**:

- $|M(\eta)| = |M(\wedge)| + \#pro^+(\eta) - \#epi^+(\eta)$

Dabei sind $|M(\wedge)|$ die initiale und $|M(\eta)|$ die dem letzten Ereignis von η entsprechende Anzahl der Prozesse in M. Mit der festgelegten Initialisierung von M ergibt sich

- $|M(\eta)| = \#pro^+(\eta) - \#epi^+(\eta)$.

Für die Zählvariable ergeben sich entsprechend:

- $\begin{aligned} m(\eta) &= m(\wedge) - \#pro(\eta) + \#epi(\eta) \\ &= m(\wedge) - (\#pro^+(\eta) + \#pro^-(\eta)) + \#epi^+(\eta) + \#epi^-(\eta) \end{aligned}$

Für den Zusammenhang zwischen m und M gilt:

- $m(\eta) = m(\wedge) - |M(\eta)| - (pro^-(\eta) - epi^-(\eta))$.

3.3.1.4 Wechselseitiger Ausschluß für das System $\mathcal{R}$

Für das System $\mathcal{R}$ mit dem kritischen Objekt $x \in D_\kappa$ und der wA–Relation $\otimes$ wird im folgenden eine **dezentrale Lösung der Koordinierungsaufgabe** mit einem zählenden Semaphor angegeben.
Dazu sei *sem* ein Semaphor mit der Zählvariablen m und dem Warteraum M. *sem* wird mit dem Initialwert $m = 1$ erzeugt; das sagt, daß x initial frei (unbenutzt) ist.
Mit $\mathcal{R}$ und *sem* wird das erweiterte System $\mathcal{R}'$ wie folgt konstruiert:

- D_κ wird um *sem* erweitert;
- die Phasenmenge von $\mathcal{R}$ wird erweitert, indem alle kritischen Phasen $a \in PK$ sequentiell (also als Phasenfolge von p für $a \in AK(p)$) geklammert zu

  sem.pro; a; *sem.epi*;

  erweitert werden.

Mit diesen Erweiterungen ergibt sich das System $\mathcal{R}'$. Für $\mathcal{R}'$ ist zu zeigen, daß es die für $\mathcal{R}$ gestellte Koordinierungsaufgabe löst und die Anforderungen (1) bis (4) erfüllt.
Die Anforderungen (2) und (3) sind durch Konstruktion von $\mathcal{R}'$ erfüllt.
Für das Folgende seien $n = |P| = |P'|$ und η eine Ereignisspur von $\mathcal{R}'$.
Mit den im vorangehenden Kapitel angegebenen Gleichungen und mit den Festlegungen für *sem* gelten:

(1) $|M(\eta)| = \#pro^+(\eta) - \#epi^+(\eta)$,

(2) $m(\eta) = 1 - \#pro(\eta) + \#epi(\eta)$ und

(3) $m(\eta) = 1 - |M(\eta)| - \#pro^-(\eta) + \#epi^-(\eta)$.

Daraus, daß die kritischen Phasen von $\mathcal{R}$ in $\mathcal{R}'$ mit *sem.pro* und *sem.epi* sequentiell geklammert sind, ergibt

$$0 \leq \#pro(\eta) - \#epi(\eta) \leq n;$$

daraus folgt zusammen mit (2)

(4) $-n + 1 \leq m(\eta) \leq 1$.

Die Ereignisspur η ist in folgendem Sinn regenerativ bzgl. der *sem*–Ereignisse:

- das erste *sem*–Ereignis ist ein pro^-–Ereignis; dem entspricht, daß initial $m = 1$ und $|M| = 0$ gelten;
- auf das erste pro^-–Ereignis folgen weitere *sem*–Ereignisse, zu denen jedoch kein pro^-–Ereignis gehört;
- dem nächsten pro^-–Ereignis muß ein epi^-–Ereignis vorausgehen; wenn dieses eintritt, gelten wieder $m = 1$ und $|M| = 0$;
- diese Gesetzmäßigkeit der Ereignisfolgen wiederholt sich; sie ergibt sich aus der Semantik von *pro* und *epi*; wenn ein epi^-–Ereignis eintritt, „erneuert“ sich *sem*; die Vergangenheit ist vergessen.

Aus diesem regenerativen Verhalten folgt:

(5) $0 \leq \#pro^-(\eta) - \#epi^-(\eta) \leq 1.$

Aus den Gleichungen bzw. Ungleichungen (1) bis (5) folgen alle weiteren Eigenschaften von $\mathcal{R}'$ und der Nachweis dafür, daß $\mathcal{R}'$ die Koordinierungsaufgabe für $\mathcal{R}$ löst.
Aus (3) ergibt sich zunächst

(6) $-|M(\eta)| = m(\eta) - 1 + \#pro^-(\eta) - \#epi^-(\eta).$

Aus (5) ergeben sich 2 Fälle. Der 1. davon entspricht

$$\#pro^-(\eta) - \#epi^-(\eta) = 0,$$

und nach (6) gilt $-|M(\eta)| = m(\eta) - 1$ mit

(7) $m(\eta) = 1$ und $|M(\eta)| = 0.$

Der 2. Fall entspricht

$$\#pro^-(\eta) - \#epi^-(\eta) = 1,$$

und es gelten nach (6)

(8) $m(\eta) \leq 0$ und $-|M(\eta)| = m(\eta)$, so daß mit (4)

(9) $-n + 1 \leq m(\eta) \leq 0$ und $0 \leq |M(\eta)| \leq n - 1$

folgen.
Sei nun $b(\eta)$ die Anzahl der Prozesse, die in dem Zustand, der dem letzten Ereignis von η entspricht, das kritische Objekt x benutzen. Dann gilt

$$b(\eta) = \#pro(\eta) - \#epi(\eta) - |M(\eta)|,$$

und daraus ergibt sich mit (2)

(10) $b(\eta) = 1 - m(\eta) - |M(\eta)|.$

Mit (7) und (8) folgt daraus

(11) $0 \leq b(\eta) \leq 1.$

Von den angegebenen Ergebnissen sagen (11), daß die kritischen Phasen $a \in PK$ wechselseitig ausgeschlossen sind, sowie (9), daß höchstens $n-1$ Prozesse bzgl. *sem* blockiert sind. Die konstruierte Lösung ist fair; die lineare Wartezeit ergibt sich aus dem FIFO–Warteraum.
Für die mit (1) bis (11) angegebenen Eigenschaften des Systems $\mathcal{R}'$ ist darauf hinzuweisen, daß sie sich

- aus den allgemeinen Eigenschaften der Semaphore,
- aus der Initialisierung der *sem*–Zählvariablen und
- aus der Klammerung der kritischen Phasen

ergeben. In Kapitel 3.3.1.3 ist festgelegt, daß die Semaphor–Zählvariablen mit nichtnegativen, ganzzahligen Werten initialisiert werden können; diesbezüglich sind also Anpassungen der Gleichungen/Ungleichungen erforderlich. Die Klammerung der kritischen Phasen ist durchweg bei Lösungen von Koordinierungsaufgaben mit Semaphoren üblich; die hier benutzten Argumente zur Herleitung der Gleichungen/Ungleichungen sind entsprechend übertragbar.

3.3.1.5 Zentrale Lösungen

In Kapitel 3.3.1.1 ist darauf hingewiesen, daß für die hier gestellte Koordinierungsaufgabe zentrale und dezentrale Lösungen möglich sind. Semaphore als Koordinierungsobjekte sind für beide Varianten geeignet.
Im folgenden wird zunächst ein Beispiel für eine zentrale Lösung angegeben.

Beispiel

Für das Beispiel soll das kritische Objekt des Systems $\mathcal{R}$ ein Zähler mit einer Ganzzahlvariablen sein, die inkrementiert, dekrementiert und auf 0 getestet werden kann. Diese Zähler–Zugriffsoperationen sollen (frei) wechselseitig ausgeschlossen werden.
In Abbildung 3.6 ist ein Modul für den Zähler angegeben, der den geforderten wechselseitigen Ausschluß zentral mit einem Semaphor *wa*, das im Zähler gekapselt ist, durchsetzt. Man sieht die zu Kapitel 3.3.1.4 analoge Vorgehensweise mit der Klammerung der kritischen „Phasen“ und die Vorteile gegenüber der dezentralen Lösung. In den Phasen der Prozesse, die den Zähler nutzen, treten jetzt Aufrufe der Zähler–Zugriffsoperationen auf, und diese Phasen müssen nicht mehr erweitert werden.

⌋

Das Beispiel ist in mehrfacher Hinsicht lehrreich.
Zunächst macht es deutlich, daß freier wechselseitiger Ausschluß eine schwache Bedingung ist: Für einen Prozeß, der teste mit positivem Ergebnis ausführt, folgt, daß der Zählerstand bei Ausführung des Tests 0 war; weitere Kenntnisse über den Zähler gewinnt er nicht.
Das Beispiel weist darüber hinaus darauf hin, daß die Maßnahmen, die zur Durchsetzung des wechselseitigen Ausschlusses für den Zähler erforderlich sind, schematisch durchführbar und damit leicht automatisierbar sind. Das erreicht man, indem man in eine Programmiersprache das Konzept **Einfacher Monitor** einführt: Das ist ein Datenobjekt, dessen Zugriffsoperationen frei wechselseitig ausgeschlossen sind, und zwar konzeptionell.
Für einen Zähler, der als einfacher Monitor definiert wird, ist der Spezifikationsteil

```
SMONITOR zaehler IS ... END zaehler;
```

```
PACKAGE zaehler IS
   PROCEDURE ink;
   PROCEDURE dek;
   PROCEDURE teste (x:OUT boolean);
END zaehler;
```

ink: $\mathbb{Z} \to \mathbb{Z}$, ink(z) $\triangleq$ z+1

dek: $\mathbb{Z} \to \mathbb{Z}$, dek(z) $\triangleq$ z-1

teste: $\mathbb{Z} \to \mathbb{B}$, teste(z) $\triangleq \begin{cases} \texttt{true} & \text{, falls } \texttt{z} = 0 \\ \texttt{false} & \text{, sonst} \end{cases}$

```
PACKAGE BODY zaehler IS
   z: integer := 0;
   PACKAGE wa IS NEW semaphor(1);
   PROCEDURE ink IS
      BEGIN wa.pro; z:=z+1; wa.epi; END ink;
   PROCEDURE dek IS
      BEGIN wa.pro; z:=z-1; wa.epi; END dek;
   PROCEDURE teste (x:OUT boolean) IS
      BEGIN wa.pro;
            x := z=0;
            wa.epi;
      END teste;
   END zaehler;
```

Abbildung 3.6: Eine zentrale Lösung für freien wechselseitigen Ausschluß

wobei ... für die früher angegebenen Prozedur–Spezifikationsteile steht. Der Körper von `zaehler` wird wie folgt definiert:

```
SMONITOR BODY zaehler IS
    z : integer := 0;
    PROCEDURE ink IS BEGIN z:=z+1; END ink;
    PROCEDURE dek IS BEGIN z:=z-1; END dek;
    PROCEDURE teste (x: OUT boolean) IS BEGIN x:=z=0; END teste;
END zaehler;
```

Von dieser `zaehler`–Definition ausgehend werden alle weiteren Maßnahmen zum wechselseitigen Ausschluß der Zugriffsoperationen, also die Hinzunahme des gekapselten Semaphors und die Klammerung der Anweisungsteile der Körper der Zugriffsoperationen, gesteuert von `smonitor` vom Übersetzer ausgeführt.
Im Hinblick auf notwendige Verallgemeinerungen ist darauf hinzuweisen, daß die für die Zugriffsoperationen eines einfachen Monitors festgelegten Restriktionen

- mit der Spezifikation explizit angegeben sind und
- für jede Zugriffsoperationen–Ausführung als Einheit gelten.

Das ist wichtig für die Nutzung des Monitors durch Prozesse von außen und es vereinfacht die Implementierung des Monitors. Die Zusammenhänge zwischen der BB– und der GB–Sicht auf den Monitor, die sich ergeben, werden mit dem Diagramm der Abbildung 3.7 veranschaulicht.

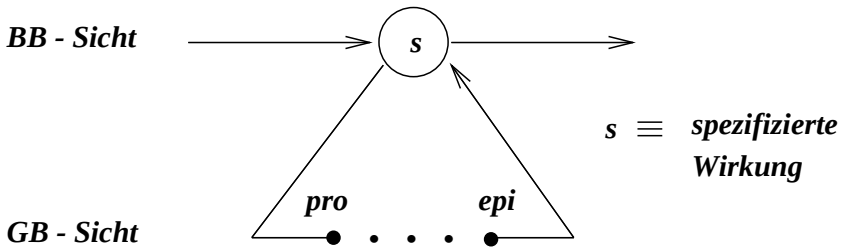


Abbildung 3.7: Die beiden Sichten auf die Ausführung einer Zugriffsoperation eines einfachen Monitors

Die im Zähler–Beispiel auftretenden Zugriffsoperationen sind sehr einfach; das ist jedoch ein für das Beispiel gewählter, spezieller Fall. Der Einsatz der Semaphore zur wA–Durchsetzung und einfache Monitore sind keineswegs auf solch einfache Fälle beschränkt. Man kann insbesondere Mengen von Datenobjekten, die als Module definiert sind, zusammenfassen und für die so gebildete Einheit wA mit einem Semaphor durchsetzen bzw. die Einheit als einfachen Monitor konstruieren.
Diese Vorgehensweise kann auf die in Kapitel 3.2 erklärte **Grundstruktur für Realisierungen** paralleler Systeme mit sequentiellen Prozessen angewandt werden. Dort ist die Menge D der Datenobjekte eines Systems in die Mengen der prozeßprivaten Datenobjekte und die Menge D_κ der kritischen Datenobjekte des Systems zerlegt. Wenn D_κ zu einer Einheit zusammengefaßt wird und die Zugriffe zu D_κ–Elementen mit einem Semaphor wechselseitig ausgeschlossen werden, ergibt sich aus D_κ ein (komplexes) Datenobjekt mit wA für die Zugriffe der Prozesse. Für einen Prozeß p sind die für seine Berechnungen charakteristischen

Umgebungswechsel $D_p \to D_\kappa$ und $D_\kappa \to D_p$ Übergänge zwischen Phasen von p, die parallel zu entsprechenden Phasen anderer Prozesse sind, und Phasen von p, die wechselseitig ausgeschlossen mit den D_κ–Phasen aller anderen Prozesse sind. Diese Vorgehensweise liefert (mit Vereinfachungen) das zusammengefaßte D_κ als **Kern eines** klassischen **Betriebssystems**, für den (vereinfacht) gilt: Berechnungen der Prozesse auf Kernobjekten sind wechselseitig ausgeschlossen. Diese Erklärung für den Kern eines Betriebssystems ist vereinfacht; sie führt auf Schwierigkeiten, die darin begründet sind, daß für Zugriffe zu Kernobjekten häufig weitere Vorbedingungen erfüllt werden müssen; das erfordert zusätzliche, bisher noch nicht erklärte Koordinierungsmaßnahmen.

3.3.2 Synchronisation

Neben der symmetrischen wA–Relation $\otimes$ wurde in Kapitel 2 die antisymmetrische Vorrangrelation $\boxplus$ als wichtige Reihenfolgerestriktion für die Berechnungen der Prozesse eines parallelen Systems erklärt. Im folgenden werden Lösungen der Koordinierungsaufgaben behandelt, die sich aus diesen antisymmetrischen Restriktionen ergeben.
Gerichtete Reihenfolgerestriktion für das System $\mathcal{R}$ sind in Situationen wie der folgenden erforderlich:

- Ein Prozeß $p \in P$ benötigt für die Phase $p_i \in A(p)$ das Datenobjekt $d \in D$; der Zustandsraum von d ist $Z(d)$; p_i setzt jedoch für d einen Zustand $z \in Z'(d)$ mit $Z'(d) \subseteq Z(d)$ und $Z'(d) \neq Z(d)$ voraus; $z \in Z'(d)$ ist dann Vorbedingung für das Ereignis $\underline{p_i}$.

- Wenn die Berechnungen von p vollständig ausführbar sein sollen, muß mit Berechnungen von $\mathcal{R}$ gewährleistet werden, daß das Datenobjekt d einen Wert $z \in Z'(d)$ erhält, mit dem $\underline{p_i}$ eintreten kann. Von den Möglichkeiten hierfür ist die von Interesse, für die gilt: Es gibt einen Prozeß $q \in P, q \neq p$, und eine Phase $q_j \in A(q)$ so, daß d mit den Berechnungen von q_j ein $z \in Z'(d)$ erhält. Dann muß gewährleistet werden, daß $\underline{p_i}$ mit dem von q_j berechneten $z \in Z'(d)$ eintreten kann: Die Phasen q_j und p_i sind mit $q_j \sqsubset p_i$ zu **synchronisieren**, so daß q_j und p_i das Objekt d mit $z \in Z'(d)$ nutzen; mit den beiden Phasen werden die Prozesse p und q synchronisiert.

Für das System $\mathcal{R}$ mit der festgelegten Zerlegung der Menge D seiner Datenobjekte ist das oben betrachtete $d \in D$ ein kritisches Datenobjekt $d \in D_\kappa$, und die bzgl. d gestellte Synchronisationsbedingung ist mit $q_j \sqsubset p_i$ formuliert. Dieser Fall, in dem die Synchronisationsbedingungen für $\mathcal{R}$ explizit mit der Vorrangrelation $\boxplus$ von $\mathcal{R}$ spezifiziert sind, wird im folgenden als erster behandelt. Allgemein muß davon ausgegangen werden, daß für ein System Synchronisationsbedingungen zunächst implizit auftreten. Dabei ist eine **Synchronisationsbedingung** eine Vorbedingung für das Eintreten eines Ereignisses eines Prozesses, die mit Berechnungen anderer Prozesse zu erfüllen sind; die Berechnungen der entsprechenden Prozesse müssen so synchronisiert werden, daß berechnete Werte gemeinsam zur Verfügung stehen.

3.3.2.1 Synchronisation gemäß der Vorrangrelation

Im folgenden sei für das System $\mathcal{R}$ die Relation $\kappa = \boxplus$ mit der Vorrangrelation über PP

$$\boxplus = \bigcup_{p \in P} \boxplus_p \cup \overset{\wedge}{\boxplus};$$

dabei sind die $\circledast_p$ für $p \in P$ die linearen Ordnungen auf den Phasen der Prozesse und $\overset{\wedge}{\circledast}$ die Vorrangrestriktionen für Phasen verschiedener Prozesse. Damit sei das Verhalten von $\mathcal{R}$ spezifiziert.
Für $\mathcal{R}$ sei $H_{\mathcal{R}}$ die Menge der vollständigen Ereignisspuren. Das Verhalten von $\mathcal{R}$ wird wie folgt charakterisiert:

- Ist $\eta = \varepsilon^1 \ldots \varepsilon^l$ eine (partielle oder vollständige) Ereignisspur von $\mathcal{R}$, so gilt für alle $b \in PP$: wenn es $a \in PP$ mit $(a, b) \in \circledast$ und $j \in \{1, \ldots, l\}$ mit $\varepsilon^j = \underline{b}$ gibt, dann gibt es $i \in \{1, \ldots, l\}$ mit $i < j$ und $\varepsilon^i = \overline{a}$.

Zu $\mathcal{R}$ ist durch Erweiterung das System $\mathcal{R}'$ so zu konstruieren, daß für jede Ereignisspur η' von $\mathcal{R}'$ gilt: die Projektion von η' auf die Ereignisse von $\mathcal{R}$ ist eine Ereignisspur von $\mathcal{R}$. Die Konstruktion von $\mathcal{R}'$ mit dieser Eigenschaft ist die hier gestellte Koordinierungsaufgabe: die **Synchronisation der Prozesse von $\mathcal{R}$ gemäß der Vorrangrelation**.
Zur Lösung dieser Koordinierungsaufgabe ist es zunächst hinreichend, die Restriktionen der Vorrangrelation ohne die Transitivitätspaare zu berücksichtigen. Wenn $\sqsubset$ diese Relation ist, gilt

$$\sqsubset = \bigcup_{p \in P} \sqsubset_p \cup \overset{\wedge}{\sqsubset} .$$

Von den Paaren in $\sqsubset$ erfordern allein die $(a, b) \in \overset{\wedge}{\sqsubset}$ Maßnahmen zur Synchronisation der entsprechenden Prozesse. Zur Synchronisation der entsprechenden Phasen der Prozesse können die in Kapitel 3.3.1.3 eingeführten zählenden **Semaphore als Synchronisationsmechanismen** eingesetzt werden.
Eine Möglichkeit zur Lösung der gestellten Koordinierungsaufgabe besteht darin, für jedes Paar von $\overset{\wedge}{\sqsubset}$ ein Semaphor zu verwenden und die Phasenmenge der Prozesse von $\mathcal{R}$ so zu erweitern, daß die erforderlichen Prolog– und Epilogoperationen auf den Semaphoren ausgeführt werden. Damit ergibt sich eine **dezentrale Lösung** der Koordinierungsaufgabe, die im folgenden erklärt wird.
Für die Konstruktion von $\mathcal{R}'$ sei zunächst

$$PS \triangleq \{s \in PP \mid (s, a) \in \overset{\wedge}{\sqsubset} \vee (a, s) \in \overset{\wedge}{\sqsubset}\}.$$

$PS \subseteq PP$ ist die Menge der Phasen von $\mathcal{R}$, für die Synchronisationsmaßnahmen erforderlich sind.
Für jedes Element von $\overset{\wedge}{\sqsubset}$ wird ein Semaphor eingesetzt. Sei also

$$\mathit{SEM} \triangleq \{\mathit{sem}(a, b) \mid (a, b) \in \overset{\wedge}{\sqsubset}\}.$$

Wenn $m(a, b)$ die Zählvariable von $\mathit{sem}(a, b) \in \mathit{SEM}$ ist, wird $\mathit{sem}(a, b)$ mit $m(a, b) = 0$ erzeugt; das bedeutet: $\mathit{sem}(a, b)$ wird zur Durchsetzung einer Bedingung eingesetzt, die initial nicht erfüllt ist.
$\mathcal{R}'$ ergibt sich aus $\mathcal{R}$ durch Erweiterung von D_κ um *SEM* und durch Erweiterung der Phasenmengen der Prozesse von $\mathcal{R}$ nach den im folgenden angegebenen Regeln.
Für jedes $s \in PS$ seien

$s^- \triangleq \{a \in PS \mid (a, s) \in \overset{\wedge}{\sqsubset}\} = \{a_1, \ldots, a_k\}$ mit $k = |s^-|$ und einer beliebigen Nummerierung der s^-–Elemente sowie

$s^+ \triangleq \{b \in PS \mid (s, b) \in \overset{\wedge}{\sqsubset}\} = \{b_1, \ldots, b_l\}$ mit $l = |s^+|$ und einer beliebigen Nummerierung der s^+–Elemente.

Die Phase s des Prozesses $p \in P$ mit $s \in A(p)$ wird wie folgt geklammert:

$$sem(a_1, s).pro; \ldots; sem(a_k, s).pro;\ s;\ sem(s, b_1).epi; \ldots; sem(s, b_l).epi;$$

Die Phasenfolge des Prozesses p wird also um $|s^-| + |s^+|$ Phasen erweitert.
Durch entsprechende Erweiterungen für alle PS–Elemente ergibt sich die Menge P' der Prozesse von $\mathcal{R}'$.
Für das wie angegeben konstruierte System $\mathcal{R}'$ gilt der folgende

Satz: Das System $\mathcal{R}'$ löst die für $\mathcal{R}$ gestellte Synchronisationsaufgabe.

Beweis: Zum Beweis des Satzes kann man sich offenbar auf die Phasen $s \in PS$ mit $s^- \neq \emptyset$ beschränken.

Seien also $\eta' = \varepsilon^1 \ldots \varepsilon^m$ eine Ereignisspur von $\mathcal{R}'$, η die Projektion von η' auf die Ereignisse von $\mathcal{R}$ und $s \in PS$ mit $s^- \neq \emptyset$; für ein $i \in \{1, \ldots, m\}$ sei $\varepsilon^i = \underline{s}$.

Dann enthält η ebenfalls das Ereignis $\underline{s}$.

Mit $s^- = \{a_1, \ldots, a_k\}$ folgt, daß η' die Ereignisse der $sem(a_j, s).pro, j \in \{1, \ldots, k\}$, vor $\underline{s}$ enthält. Aus der Semantik der $sem(a_j, s).pro$ folgt, daß η' auch die Ereignisse der $sem(a_j, s).epi, j \in \{1, \ldots, k\}$, vor $\underline{s}$ enthält. Nach Konstruktion der erweiterten Phasenfolgen ergibt sich, daß diesen Ereignissen die $\overline{a_j}$ für alle $j \in \{1, \ldots, k\}$ vorausgehen; sie sind also in η' enthalten, und zwar vor $\underline{s}$. Die $\overline{a_j}$ sind Ereignisse von $\mathcal{R}$, und damit treten sie auch in η vor $\underline{s}$ auf.

Es bleibt noch zu zeigen, daß es eine Ereignisspur von $\mathcal{R}'$ mit den für η' und s angenommenen Eigenschaften gibt. Das folgt mit $s^- = \{a_1, \ldots, a_k\}$ daraus, daß für alle $j \in \{1, \ldots, k\}$ gelten:

- initial ist $m(a_j, s) = 0$;
- auf $sem(a_j, s)$ werden
  - *pro* genau 1–mal ausgeführt, und zwar vor s, und
  - *epi* genau 1–mal ausgeführt, und zwar nach a_j;
- wenn *epi* vor *pro* ausgeführt wird, wird *pro* ohne *blockiere* ausgeführt;
- wenn *pro* vor *epi* ausgeführt wird, wird *pro* mit *blockiere* ausgeführt, und der entsprechende Prozeß wird mit *epi* entblockiert.

⌋

Beispiel:

Für das System $\mathcal{R}$ mit der Prozeßmenge $P = \{p, q\}$ seien die Phasen und die Reihenfolgerestriktionen durch den Vorranggraphen der Abbildung 3.8 festgelegt.
Zur Synchronisation der Prozesse von $\mathcal{R}$ werden die Semaphore $sem(p_1, q_2)$ und $sem(q_2, p_3)$ eingesetzt.
Abbildung 3.9 zeigt den Vorranggraphen des erweiterten Systems.

⌋

Nach dem oben angegebenen Verfahren sind zur Synchronisation des Systems $\mathcal{R}$ gemäß der Vorrangrelation $| \overset{\wedge}{\sqsubset} |$ Semaphore erforderlich, wenn $\overset{\wedge}{\sqsubset}$ die dort für $\mathcal{R}$ angegebene Bedeutung hat. Wenn wieder PS die Menge der Phasen ist, für die Synchronisationsmaßnahmen nötig

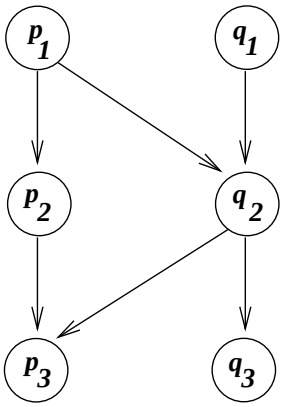


Abbildung 3.8: Vorranggraph des Systems $\mathcal{R}$

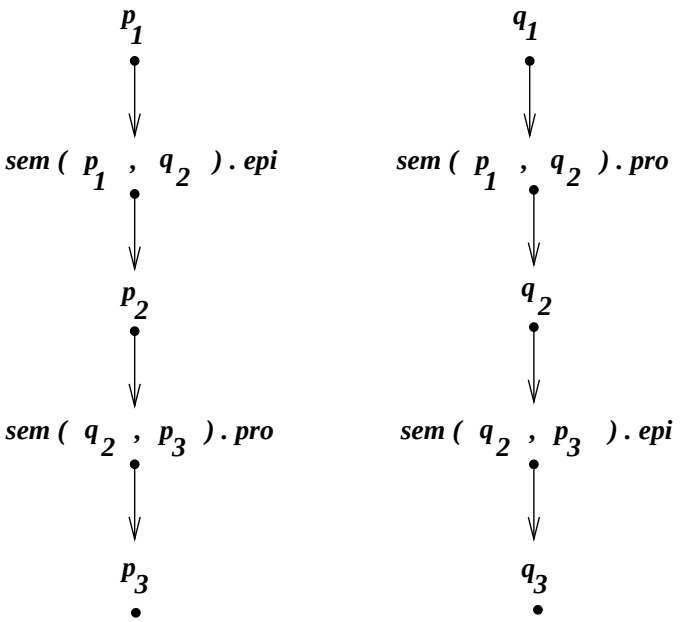


Abbildung 3.9: Vorranggraph des Systems $\mathcal{R}'$

sind, kann man die für $\mathcal{R}$ gestellte Synchronisationsaufgabe auch mit weniger Semaphoren lösen: Mit dem oben definierten s^- für $s \in PS$ kann die Aufgabe mit einem Semaphor $sem(s)$ für jedes $s \in PS$ mit $s^- \neq \emptyset$ gelöst werden, wobei die Zählvariablen der Semaphore wieder mit 0 initialisiert werden.
Mit $\overline{PS} = \{s \in PS \mid s^- \neq \emptyset\}$ sind dann die Semaphore

$$\overline{SEM} = \{sem(s) \mid s \in \overline{PS}\}$$

erforderlich. Für $s \in PS$ mit

$|s^-| = k$, $k \in I\!N_0$, und
$s^+ = \{s_1, \ldots, s_l\}$, $l = |s^+|$ und $s_1, \ldots, s_l \in \overline{PS}$ für $l \neq 0$

wird die Phase s des Prozesses p mit $s \in A(p)$ wie folgt geklammert:

$$(sem(s).pro;)^k \; s; \; sem(s_1).epi; \ldots; sem(s_l).epi;$$

Die Phasenfolge des Prozesses p wird also wieder um $|s^-| + |s^+|$ Phasen erweitert.
Durch Erweiterung von D_κ um $\overline{SEM}$ und durch Erweiterung aller PS–Elemente ergibt sich das System $\mathcal{R}'$, das die für $\mathcal{R}$ gestellte Synchronisationsaufgabe löst.

3.3.2.2 Ein Nachrichtenpuffer

Bei den einführenden Erklärungen der Synchronisationsaufgaben, die für ein paralleles System zu lösen sind, wurde darauf hingewiesen, daß die durchzusetzenden Synchronisationsbedingungen häufig implizit festgelegt sind. Das ist für das betrachtete System $\mathcal{R}$ insbesondere dann der Fall, wenn der kritische Teil D_κ der Datenobjekte von $\mathcal{R}$ Datenobjekte enthält, die mit Zugriffsoperationen definiert sind, deren Ausführung nur dann zulässig ist, wenn bestimmte Vorbedingungen erfüllt sind. Diese Vorbedingungen sind zur Gewährleistung der Konsistenz der Datenobjekte erforderlich.
Im folgenden wird ein Nachrichtenpuffer als wichtiges Beispiel für ein entsprechendes Datenobjekt als Element von D_κ erklärt. Wenn $s \in D_\kappa$ ein Nachrichtenpuffer ist, kann s von Prozessen von $\mathcal{R}$ zur mittelbaren Kooperation genutzt werden, indem die Prozesse mittels s Nachrichten austauschen. Hier interessiert die Konstruktion von s als ein zum Nachrichtenaustausch geeignetes, konsistentes Datenobjekt. Diese Konstruktion ist ein Beispiel für eine **zentrale Lösung einer Synchronisationsaufgabe**.
Der **Nachrichtenpuffer** s soll zum Speichern von Nachrichten eines festgelegten, hier jedoch nicht weiter interessierenden Typs geeignet sein. Hierfür enthält s gekapselt einen Speicher mit der **Kapazität** $kap(s) \in I\!N - \{0\}$. Die **Anzahl der** jeweils gespeicherten **Nachrichten** sei $in(s)$, so daß also stets

$$0 \leq in(s) \leq kap(s)$$

gilt. Initial sei $in(s) = 0$.
Als **Zugriffsoperationen** auf s sollen *ein* und *aus* jeweils für das Einfügen bzw. für die Entnahme einer Nachricht definiert werden.
Zur **Konsistenz** von s ist gefordert, daß Nachrichten allein mit *ein* in s eingefügt und mit *aus* aus s entnommen werden können; zudem sollen *ein* das Einfügen der Eingabe–Nachricht und *aus* die Entnahme einer Nachricht und deren Ausgabe korrekt bewirken. Die korrekte Ausführung der entsprechenden Operationen auf dem Speicher von s sei gewährleistet, wenn diese Operationen wechselseitig ausgeschlossen und damit interferenzfrei ausgeführt werden.
Für die Zugriffsoperationen auf s ergeben sich damit folgende Anforderungen:

(B1) wA für die Operationen auf dem Speicher von s.

Für *ein*

(B2) $in(s) < kap(s)$ als Vorbedingung und
$in(s) := in(s) + 1$ als Wirkung.

Für *aus*

(B3) $0 < in(s)$ als Vorbedingung und
$in(s) := in(s) - 1$ als Wirkung.

Die (B1) bis (B3) sind die **Synchronisationsbedingungen** für s, die mit der Konstruktion von s zentral durchgesetzt werden sollen.
Zur **Lösung der Synchronisationsaufgabe** werden hier wieder die in Kapitel 3.3.1.3 eingeführten zählenden Semaphore eingesetzt. Es sind drei Semaphore erforderlich.
Die Durchsetzung von (B1) ist eine der Aufgaben, die schon in Kapitel 3.3.1 gelöst wurden. Die Lösung erfolgt mit dem Semaphor *wa*, das mit dem Anfangswert der Zählvariablen $m(wa) = 1$ erzeugt wird.
Die Durchsetzung von (B2) erfolgt mit dem Semaphor *voll*. *voll* dient dazu, Prozesse die *s.ein* aufrufen, dann zu blockieren, wenn (B2) nicht erfüllt ist. *voll* wird mit dem Anfangswert der Zählvariablen $m(voll) = kap(s)$ erzeugt. Das entspricht der Festlegung, daß für s initial $in(s) = 0$ gilt, und sagt, daß *voll* zur Kontrolle einer Bedingung eingesetzt wird, die initial $kap(s)$–mal erfüllt ist. Prozesse, die sich bzgl. *voll* blockieren, können entblockiert werden, wenn Nachrichten aus s entnommen sind; das erfolgt mit *aus*–Ausführungen.
Die Durchsetzung von (B3) erfolgt mit dem Semaphor *leer*. *leer* dient dazu, Prozesse die *s.aus* aufrufen, dann zu blockieren, wenn (B3) nicht erfüllt ist. *leer* wird mit dem Anfangswert der Zählvariablen $m(leer) = 0$ erzeugt. Das entspricht der Festlegung, daß für s initial $in(s) = 0$ gilt, und sagt, daß *leer* zur Kontrolle einer Bedingung eingesetzt wird, die initial nicht erfüllt ist. Prozesse, die sich bzgl. *leer* blockieren, können entblockiert werden, wenn Nachrichten in s eingefügt sind; das erfolgt mit *ein*–Ausführungen.
Mit den drei angegebenen Semaphoren werden die Zugriffsoperationen von s wie folgt implementiert:

- **Anweisungsteil von** *ein*: *voll.pro*; *wa.pro*;
  geklammerte Anweisungen
  wa.epi; *leer.epi*;

- **Anweisungsteil von** *aus*: *leer.pro*; *wa.pro*;
  geklammerte Anweisungen
  wa.epi; *voll.epi*;

Damit ist erklärt, wie der Nachrichtenpuffer s konstruiert wird. Im weiteren wird angenommen, daß $s \in D_\kappa$ mit den angegebenen Eigenschaften zur Verfügung steht. s kann von den Prozessen von $\mathcal{R}$ benutzt werden; das erfolgt mit Aufrufen von *s.ein* bzw. *s.aus* durch die Prozesse. Für Berechnungen, die von Prozessen von $\mathcal{R}$ unter Verwendung des Nachrichtenpuffers s ausgeführt werden, ergeben sich zwei Fragen:

- Ist gewährleistet, daß für s stets $0 \leq in(s) \leq kap(s)$ gilt?

- Ist gewährleistet, daß die Prozesse von $\mathcal{R}$ ihre Berechnungen vollständig ausführen können?

Die erste dieser Fragen betrifft die Konsistenz des Nachrichtenpuffers s und die korrekte Implementierung der Zugriffsoperationen; sie wird im folgenden positiv beantwortet. Die zweite Frage betrifft die Lebendigkeit und Sicherheit des Systems $\mathcal{R}$; sie stellt Anforderungen an s und an die Prozesse von $\mathcal{R}$.
Zur Beantwortung der ersten Frage werden folgende Kenngrößen von $\mathcal{R}$ bzgl. s benötigt: Sei η eine Ereignisspur von $\mathcal{R}$. Dann seien

- $m(x, \eta)$ für $x \in \{voll, leer\}$

  der Wert der Zählvariablen von x beim letzten Ereignis von η,

- $\#ein(\eta)$ bzw. $\#aus(\eta)$ die Anzahl der Terminierungsereignisse von $s.ein$ bzw. $s.aus$, die in η auftreten, und

- $in(s, \eta)$ die Anzahl der in s gespeicherten Nachrichten beim letzten Ereignis von η.

Satz: Für jede Ereignisspur η von $\mathcal{R}$ gilt $0 \leq in(s, \eta) \leq kap(s)$.

Beweis: Sei η eine Ereignisspur von $\mathcal{R}$.

Nach Konstruktion von $s.aus$ enthält jede *aus*–Ausführung *leer.pro*. Mit der Semantik von *leer.pro* und aus $m(leer, \wedge) = 0$ folgt daraus

(1) $0 \leq \#aus(\eta) \leq \#ein(\eta)$.

Nach Konstruktion von $s.ein$ enthält jede *ein*–Ausführung *voll.pro*. Mit der Semantik von *voll.pro* und mit $m(voll, \wedge) = kap(s)$ folgt daraus

(2) $0 \leq \#ein(\eta) \leq kap(s) + \#aus(\eta)$.

Aus (1) und (2) folgt

(3) $0 \leq \#ein(\eta) - \#aus(\eta) \leq kap(s)$.

Nun gilt $in(s, \eta) = in(s, \wedge) + \#ein(\eta) - \#aus(\eta)$. Nach Konstruktion von s gilt $in(s, \wedge) = 0$, und damit ist (3) die Aussage des Satzes.

⌋

Die Situation, die sich mit diesem Satz für das System $\mathcal{R}$ ergibt, ist typisch für zentrale Lösungen von Synchronisationsaufgaben und für die Konstruktion von sog. **Erzeuger–Verbraucher–Systemen** mit Prozessen, die mittelbar kooperieren, indem sie mittels eines Puffers Nachrichten austauschen. Der Satz ist die positive Antwort auf die erste der oben gestellten Fragen. Er sagt, daß s als Puffer für Nachrichtenaustausch zwischen Prozessen geeignet ist. Das ist eine notwendige Bedingung für entsprechende Kooperation zwischen Prozessen und der erste Schritt zur Konstruktion eines Erzeuger–Verbraucher–Systems. Mit dem zweiten Schritt muß gewährleistet werden, daß alle Prozesse, die kooperieren sollen, ihre Beiträge — das sind hier ihre Operationen auf dem Puffer s — so leisten, daß das gemeinsame Kooperationsziel erreicht wird. Dazu muß die zweite der oben gestellten Fragen positiv beantwortet werden. Das Thema, zu dem diese Frage gehört, wird in Kapitel 4 behandelt.

3.3.3 Koordinierungsverfahren und weitere Mechanismen

Die in Kapitel 3.3.1.2 und Kapitel 3.3.1.3 eingeführten Koordinierungsmechanismen zeigen die Zusammenhänge, die zwischen der jeweils für einen Mechanismus benutzten Basis und dem Aufwand zum Nachweis von Eigenschaften, die mit einem Mechanismus durchgesetzt werden, besteht. Für die in Kapitel 3.3.1.3 eingeführten zählenden Semaphore hat sich gezeigt, daß sie sowohl zur Durchsetzung des freien wechselseitigen Ausschlusses als auch zur Durchsetzung komplizierterer Synchronisationsbedingungen gut geeignet sind. Die Nachweise der jeweiligen Soll–Eigenschaften sind mit der Fairneß, mit den Zählvariablen und mit den atomaren Prolog– und Epilog–Operationen relativ einfach zu führen.
Die durchgeführten Konstruktionen haben gezeigt, wie die Semaphore sowohl in dezentralen als auch in zentralen Koordinierungsverfahren eingesetzt werden können. Die **dezentralen Verfahren**, die in Kapitel 3.3.1.4 und in Kapitel 3.3.2.1 demonstriert sind, zeigen sowohl die Vorteile als auch die Nachteile. Sie sind flexibel nach Bedarf anwendbar; sie verwenden die Mechanismen, wenn sie benötigt werden. Sie führen jedoch auch schnell zu Fehlern: eine fehlende oder eine fehlerhaft eingeordnete Prolog– oder Epilog–Operation kann schwerwiegendes Fehlverhalten zur Folge haben. Die **zentralen Verfahren**, die in Kapitel 3.3.1.5 und in Kapitel 3.3.2.2 demonstriert sind, zeigen die Vorteile: Sie entsprechen dem Prinzip divide et impera; Teilprobleme werden lokal gelöst, und diese Konzentration hilft dabei, Fehler zu vermeiden. Diese gewichtigen Vorteile führen zu verringerter Flexibilität; das ist bei Systemen mit grobgranularen Objekten das Hindernis, das zentralen Lösungen entgegen steht.
Die Eignung der Semaphore als Koordinierungsmechanismen ergibt sich insbesondere aus der Atomarität ihrer Prolog– und Epilog–Operationen. Diese Eigenschaften haben zur Folge, daß zählende Semaphore in der **Abstraktionshierarchie der Koordinierungskonzepte** in eine Zwischenebene einzuordnen sind. Mit Semaphoren als Mechanismen können und müssen Konzepte mit höherem Abstraktionsniveau realisiert werden. Die einfachen Monitore, auf die in Kapitel 3.3.1.5 hingewiesen ist, sind ein Beispiel für Konzepte mit höherem Abstraktionsniveau; weiterreichende Konzepte werden noch erklärt werden.
Relativ zu den Konzepten, welche die Hardwarekonfiguration eines Rechensystems für Koordinierungsmechanismen zur Verfügung stellt, stehen die Semaphore auf hohem Abstraktionsniveau. Das demonstriert der Vergleich mit der in Kapitel 3.3.1.2 angegebenen wA–Lösung, und das wird deutlich, wenn die atomaren Prolog– und Epilog–Operationen mit Mitteln, welche die Hardware zur Verfügung stellt, realisiert werden müssen. Für Übergänge von der Hardware zu den zählenden Semaphoren sind daher einfachere (und schwächere) Koordinierungsmechanismen notwendig. Ereigniszähler und Sequenzierer sind geeignete Mechanismen in diesem Bereich; sie werden im folgenden kurz erklärt.
Ereigniszähler (eventcount) und Sequenzierer (sequencer) sind Koordinierungsmechanismen, die 1979 von D.P. Reed und R.K. Kanodia eingeführt wurden und häufig benutzt werden. Sie verwenden nichtnegative Ganzzahlen als Hilfsmittel, für die wesentlich ist, daß sie streng monoton wachsende Folgen bilden.
Ein **Ereigniszähler** E ist ein Datenobjekt mit einer Ganzzahlvariablen e und einem Prozeß–Warteraum. Weiter ist folgendes festgelegt:

- Der Anfangswert von e ist 0, und die Folge der e–Werte ist streng monoton wachsend. Der Warteraum ist initial leer.

- Auf E sind folgende Zugriffsoperationen definiert:

  - $E.read$: liefert den jeweiligen e–Wert als Ausgabe;

- $E.advance$: inkrementiert e, $e := e + 1$, w–atomar;
- $E.await(x)$ mit einer nichtnegativen Ganzzahl x als Eingabe: der ausführende Prozeß blockiert sich im Warteraum, falls $e < x$ gilt, und wird entblockiert, wenn $e \geq x$ wird.

Ein **Sequenzierer** S ist ein Datenobjekt mit einer Ganzzahlvariablen s und folgenden Festlegungen:

- Der Anfangswert von s ist 0, und die Folge der s–Werte ist streng monoton wachsend;
- auf S ist eine w–atomare Zugriffsoperation definiert:

  $S.ticket$: liefert den jeweiligen s–Wert als Ausgabe und inkrementiert s, $s := s+1$.

Es ist klar, wie Ereigniszähler und Sequenzierer einzusetzen sind. Die Ganzzahlvariablen mit ihren streng monotonen Werten sind als abstrakte Uhren anzusehen. Für Sequenzierer ist keine *blockiere*–Operation definiert; dem entsprechend werden Sequenzierer häufig in Kombination mit Ereigniszählern eingesetzt. Für Ereigniszähler ist zu beachten, daß gleichzeitig mehrere Prozesse entblockiert werden können.
Als sehr einfache, aber häufig benutzte Koordinierungsmechanismen sind schließlich **Sperren** (lock) zu nennen. Eine Sperre ist im einfachsten Fall ein Datenobjekt mit einer Booleschen Variablen, deren Wert w–atomar gelesen und verändert werden kann (*lock*– und *unlock*–Operationen). Darüber hinaus werden binäre Semaphore als Sperren eingesetzt.
Für den Einsatz aller Koordinierungsmechanismen, die in diesem Kapitel erklärt sind, sowie für die Anwendung sowohl der zentralen als auch der dezentralen Koordinierungsverfahren ist zu beachten, daß sie jeweils auf **ein Datenobjekt**, für das Bedingungen zu erfüllen sind, beschränkt betrachtet wurden. Die gestellten Anforderungen werden verschärft, wenn Prozesse — was die Regel ist — mehrere kritische Datenobjekte benötigen. Dann können zwar die hier erklärten Koordinierungsmechanismen weiter benutzt werden; es sind jedoch andere Koordinierungsverfahren erforderlich. Das ist das Thema des nächsten Kapitels.

Kapitel 4

Betriebsmittel und Betriebsmittel–Management

Die Modellsysteme mit sequentiellen Prozessen, die in Kapitel 3 zur Beschreibung des Verhaltens paralleler Mehrbenutzersysteme eingeführt wurden, und die von ihnen abgeleitete Grundstruktur für Realisierungen entsprechender Systeme waren die Ausgangsbasis für Formulierungen und Lösungen der einfachen, bisher behandelten Koordinierungsaufgaben. Diese Aufgaben ergeben sich aus den koexistierenden Prozessen und ihren Wirkungsbereichen: Den Prozessen werden für ihre Berechnungen dauerhaft private Datenobjekte zugeordnet, die jedoch i.d.R. nicht für alle Berechnungsphasen ausreichen. Dem entsprechend fordern Prozesse dynamisch weitere Datenobjekte, die sie benötigen, an und geben diese, wenn sie nicht mehr benötigt werden, wieder frei. Diese Datenobjekte sind potentiell kritisch; die Prozesse konkurrieren um sie und ihre Benutzung durch Prozesse erfordert spezifische Entscheidungen und Maßnahmen. Von den Gründen für die Aufteilung der Datenobjekte eines Systems in prozeßprivate und (potentiell) kritische wurden schon einige erklärt; weitere Gründe werden noch klar werden. Vorerst wird angenommen, daß die Aufteilung mit der Prozeßmenge eines Systems festgelegt ist und die Prozesse um die kritischen Datenobjekte konkurrieren.

In Kapitel 3.3, in dem es darum ging, grundlegende Aspekte der Koordinierungsaufgaben eines parallelen Systems zu erklären, wurden jeweils einzelne Datenobjekte und kritische Eigenschaften betrachtet. Diese unrealistische Beschränkung wird in diesem Kapitel aufgegeben: realistisch ist, daß der kritische Teil sehr viele und sehr vielfältige Datenobjekte enthält. Als Konsequenz ergibt sich, daß die Koordinierungsaufgaben mit wesentlich veränderten Anforderungen gestellt und für ihre Lösungen allgemeinere Verfahren erforderlich werden. Veränderte Anforderungen ergeben sich zunächst daraus, daß Prozesse häufig Datenobjekte zusätzlich benötigen oder freigeben können. Sie ergeben sich weiter daraus, daß der Aufwand für Koordinierungsmaßnahmen klein bleiben muß; es sind also flexibel einsetzbare, effiziente Koordinierungsverfahren nötig.

Diesen veränderten Rahmenbedingungen wird einerseits mit einer differenzierteren Sicht auf ein paralleles System Rechnung getragen: Für die Komponenten eines Systems sowie für die Abhängigkeiten zwischen Prozessen und Datenobjekten sind weitere Differenzierungen erforderlich. Mit diesen Differenzierungen wächst die Anzahl der möglichen Anforderungen an die Koordinierung weiter, so daß Maßnahmen zur Beherrschung dieser Quantitäten hinzukommen müssen; dazu sind Klassenbildungen und Strukturierung geeignet. Differenzierungen einerseits sowie Klassenbildung und Strukturierung andererseits sind iteriert und zielgerichtet anzuwenden. Der erste Schritt hierzu besteht darin, daß Komponenten eines Systems mit dem Begriff bezeichnet werden, der ihre relative Bedeutung zum Ausdruck bringt: Komponenten sind Hilfs– oder **Betriebsmittel** bzw. Ressourcen (resources) für Berechnungen; sie

müssen Prozessen für den jeweiligen Zweck zur Verfügung gestellt werden. Dazu gehört, daß existierende Betriebsmittel für die Prozesse verwaltet, also belegt und freigegeben werden. Dazu gehört weiter, daß Konzepte für geeignete Ressourcen festgelegt, entsprechende Ressourcen erzeugt, für Prozesse verwaltet und schließlich wieder aufgelöst werden. Das wird zusammengefaßt als **Management** der Betriebsmittel eines Systems bezeichnet: Es sind Management–Entscheidungen zu treffen und diese sind zu verwirklichen. Ein wichtiger Teil dieser Betriebsmittel–Managementaufgaben eines Betriebssystems wird in diesem Kapitel behandelt.
Im weiteren werden die Modellsysteme mit sequentiellen Prozessen weiter zur Beschreibung der gestellten Aufgaben benutzt. Für ein entsprechendes System $\mathcal{R}$ haben die Komponenten des 6–tupels die in Kapitel 3.1 angegebene Bedeutung im Zuge der differenzierteren Erfassung der Komponenteneigenschaften werden sie jedoch an die jeweiligen Gegebenheiten angepaßt.

4.1 Betriebsmittel und Betriebsmittelarten

Betriebsmittel (BM) als Komponenten eines Systems haben analog zu den bisher betrachteten Datenobjekten und Prozessen individuelle Eigenschaften. Wenn, wie hier angenommen, viele Komponenten existieren und erfaßt werden müssen, ist es zweckmäßig, Komponenten mit gleichartigen Eigenschaften zusammenzufasssen und die Individuen als Elemente der so gebildeten Mengen gleichartig zu behandeln. Für BM liefern die Zwecke, für die sie von Prozessen benötigt und eingesetzt werden, die Kriterien für diese Vorgehensweise. Mit entsprechenden Kriterien werden **BM–Arten** und mit verschärften Kriterien BM–Unterarten gebildet; damit ergibt sich eine hierarchische Struktur von Mengen, in welche die BM eingeordnet werden. Für die Elemente der Menge, die einer BM–Art entspricht, ist es möglich, daß sie für die Zwecke, für die sie von Prozessen eingesetzt werden, äquivalent sind. In diesem Fall ist die Menge zu einer Äquivalenzklasse zusammenfaßbar. Entsprechend gebildete **BM–Klassen** sind zweckmäßig für die weitere Strukturierung der BM.
Im folgenden werden BM–Arten nach groben Kriterien gebildet; mit ihnen wird die Vorgehensweise erklärt. Aus den Zusammenhängen zwischen den BM–Arten und der Koordinierung von Berechnungen ergeben sich die Aufgaben des BM–Managements, das wesentliche Beiträge zur Koordinierung der Berechnungen eines parallelen Systems zu leisten hat.

4.1.1 Betriebsmittelarten nach Lebenszeiten und Einsatzmöglichkeiten

Die BM sind die Komponenten eines Systems, welche für die Berechnungen der Prozesse benötigt und eingesetzt werden. Ein sequentieller Prozeß benötigt BM der Phasenstruktur seiner Berechnungen entsprechend: Für die Initiierungsereignisse der Phasen werden BM angefordert, und mit den Terminierungsereignissen der Phasen werden die nicht mehr benötigten BM freigegeben. BM–Arten sollen so definiert werden, daß die entsprechenden Maßnahmen effizient durchführbar sind. Wenn von einem Prozeß Anforderungen gestellt werden, ist zu klären, ob sie erfüllbar sind; wenn sie erfüllbar sind, sollen sie mit wenig Aufwand und mit kurzen Wartezeiten erfüllt werden.
Komponenten eines Systems können dauerhaft, d.h. für die gesamte Lebenszeit des Systems, oder temporär, d.h. in Zeitintervallen während der Lebenszeit des Systems, existieren. Mit der **Lebenszeit** der Komponenten als Kriterium ergeben sich die beiden BM–Arten

- DBM für dauerhaft existierende BM und

- TBM für temporär existierende BM.

Wenn davon ausgegangen wird, daß eine feste Hardwarekonfiguration als Basis für die Realisierung eines Systems benutzt wird, sind sämtliche Komponenten dieser Konfiguration DBM.

Aus der Dauerhaftigkeit ergibt sich als **charakteristisch für die DBM**, daß sie verwaltet werden müssen und über der Zeit für mehrere Berechnungen einsetzbar sind. Für ihre **Verwaltung** ist zunächst erforderlich, daß sämtliche DBM mit ihren Eigenschaften registriert werden. Für den Einsatz der DBM ist weiter zu registrieren, ob sie eingesetzt oder nicht eingesetzt sind. Das erfolgt zweckmäßig mit den **Einsatzzuständen** *frei* und *belegt* sowie mit entsprechenden Zustandsüberführungen. Auf dieser Grundlage sind für DBM (wenigstens) folgende Aktionen erforderlich:

- für Prozesse: anfordern und freigeben
- für die Verwaltung:
  - Anforderungen erfüllen mit Überführung in *belegt* und
  - Freigaben verwirklichen mit Überführung in *frei*.

Mit dem Erklärten ergibt sich der einfachste Fall eines Zustandsdiagramms für die Verwaltung eines DBM. Mit den entsprechenden Zustandsfolgen ergeben sich **wiederholt benutzbare** DBM — WDBM — was zum Ausdruck bringt, daß die BM über der Zeit für mehrere Berechnungen einsetzbar sind.
Charakteristisch für die TBM sind zunächst ihre Lebenszeiten. Die **Lebenszeit** eines TBM beginnt mit dem Abschluß seiner Erzeugung und endet mit der Initiierung seiner Auflösung. Dabei sind erzeugen und auflösen Aktionen von Berechnungen. Im Gegensatz zu den DBM, die mit ihren Eigenschaften gegeben sind, sind die Eigenschaften der TBM bei Erzeugung festzulegen. Damit ergibt sich einerseits die Möglichkeit, BM mit geeigneten Eigenschaften bei Bedarf zur Verfügung zu stellen. Andererseits ist klar, daß für die Erzeugung eines TBM schon existierende BM eingesetzt werden müssen; diese BM können DBM sein.
Typische Beispiele für TBM sind die Keller, die Prozesse für ihre Berechnungen benötigen. Der Keller *keller*(p) eines Prozesses p ist ein p–privates dynamisches Datenobjekt, das erzeugt wird, indem ein schon existierendes BM *kellerbereich*(p) für p belegt wird, und in dem p Kelleroperationen auf *kellerbereich*(p) zur Verfügung gestellt werden.
Für die existierenden TBM ist **Verwaltung** analog zu den DBM erforderlich; dabei sind jedoch die Lebenszeiten zu berücksichtigen: Die Lebenszeiten beschränken die Einsatzmöglichkeiten der TBM; es muß gewährleistet sein, daß die TBM konsistent eingesetzt und aufgelöst werden.
Ein Extremfall für Beschränkungen der Einsatzmöglichkeiten durch die Lebenszeiten liegt vor, wenn ein TBM mit seinem ersten Einsatz aufgelöst wird; das TBM ist in diesem Fall ein **einmal benutzbares BM** (consumable resource) — EBM. Ein TBM kann jedoch auch ein **wiederholt benutzbares BM** — WTBM — sein.
Ein Beispiel für ein EBM ist eine Nachricht, die von einem Sender erzeugt und von ihrem Empfänger aufgelöst wird.
Ein Beispiel für ein WTBM ist eine Datei, die erzeugt wird, dann für lange Zeit benutzt werden kann und schließlich aufgelöst wird.
Abbildung 4.1 zeigt die Struktur über den BM–Arten, die sich mit den bisher angegebenen Kriterien ergibt.
Die bisher eingeführten BM–Arten zeigen bereits das Wesentliche der Vorgehensweise:

- Die BM und die BM–Arten dienen dazu, die Koordinierung der Berechnungen eines Systems systematisch zu vereinfachen, indem die BM ihren Einsatzmöglichkeiten entsprechend bereitgestellt werden.

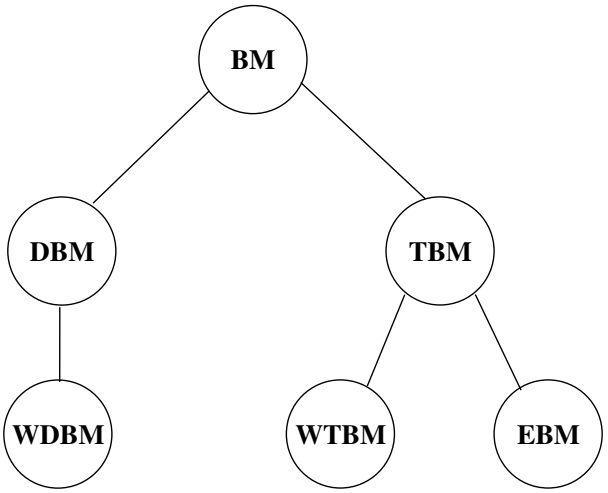


Abbildung 4.1: BM–Arten nach Lebenszeiten und lebenszeitbedingten Einsatzmöglichkeiten

- Zum BM–Management gehört einerseits die Verwaltung der jeweils existierenden BM und andererseits die Bereitstellung effektiver Konzepte für TBM, die nach Bedarf erzeugt und aufgelöst werden.
- Die BM erfassen alle Komponenten eines Systems, die für Berechnungen eingesetzt werden können, mit ihren Abhängigkeiten. Sie sind, von den primitiven Komponenten der Hardwarekonfiguration, die als DBM erfaßt werden, abgesehen, nach ihren Einsatzmöglichkeiten vergröberte Datenobjekte. Die BM und BM–Arten ermöglichen damit schrittweise Übergänge zwischen interpretierten und uninterpretierten Komponenten.

Die in Abbildung 4.1 zusammengefaßten BM–Arten ergeben sich mit groben Kriterien; sie sind Ausgangsbasis für notwendige weitere Verfeinerungen. Kriterien für diese Verfeinerungen werden im folgenden für die WBM angegeben. Die **wiederholt benutzbaren BM** — WBM — (reusable resources) ergeben sich durch Zusammenfasssung der WDBM und der WTBM, die in Zeitintervallen, in denen Auflösungen von WTBM ausgeschlossen sind, zulässig ist.
Für die WBM sind bisher die beiden Einsatzzustände *frei* und *belegt* eingeführt. Dabei ist unberücksichtigt geblieben, ob die BM gleichzeitig für mehrere Prozesse eingesetzt werden können. Diese wichtige Eigenschaft liefert die Kriterien für die im folgenden angegebenen WBM–Unterarten.
Ein BM, das ohne Einschränkungen gleichzeitig für beliebig viele Prozesse eingesetzt werden kann, ist ein **frei parallel benutzbares BM** — FPBM — und sonst ein bedingt parallel benutzbares BM — BPBM.
Typische Beispiele für FPBM sind invariante, definitionsgemäß wertkonstante Datenobjekte.

Die BPBM sind die Zusammenfasssung der WBM mit Einschränkungen für ihren parallelen Einsatz. Zu den BPBM gehören die **exklusiv benutzbaren BM** — XBM — die für jeweils höchstens einen Prozeß einsetzbar sind. Die XBM zerfallen in die **frei exklusiv benutzbaren BM** — FXBM — und die **bedingt exklusiv benutzbaren BM** — BXBM.

Der Zähler, der in Kapitel 3.3.1.5 als Beispiel angegeben ist, ist ein typisches Beispiel für ein FXBM. Allgemein sind die FXBM Vergröberungen von Komponenten mit Zugriffsoperationen, deren Ausführungen in beliebigen Permutationen zulässig sind.
Der Puffer, der in Kapitel 3.3.2.2 als Beispiel angegeben ist, ist ein typisches Beispiel für ein BXBM: zusätzlich zur exklusiven Benutzung (also zum wA) ist festgelegt, daß spezifische Bedingungen erfüllt sein müssen.
Abbildung 4.2 zeigt die vereinfachte Struktur über den eingeführten BM–Arten.

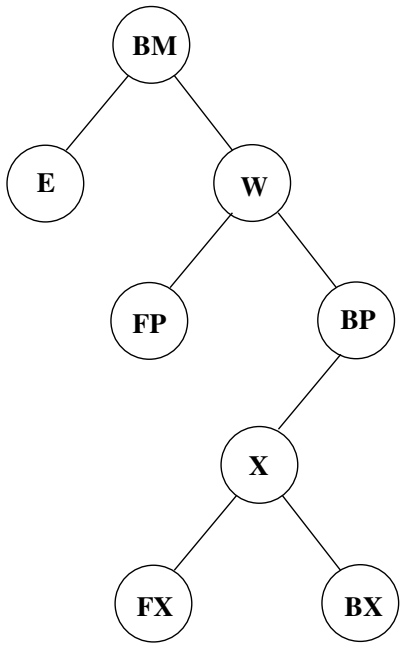


Abbildung 4.2: BM–Arten nach Lebenszeiten und Einsatzmöglichkeiten

4.1.2 Betriebsmittelarten und Koordinierung

BM und BM–Arten dienen dazu, die Koordinierung der Berechnungen eines parallelen Systems so vorzubereiten, daß effiziente Lösungen der Koordinierungsaufgaben möglich werden. Diese Zusammenhänge werden im folgenden mit den eingeführten BM–Arten erklärt.
Dazu sei $\mathcal{R} = (D, P, \alpha, \kappa, X, Y)$ ein System mit sequentiellen Prozessen. Die Komponenten des 6–tupels haben die in Kapitel 3.1 angegebene Bedeutung; D sei jedoch den eingeführten BM–Arten entsprechend strukturiert. Für $p \in P$ sind X_p und Y_p die Familien der Anforderungs– und Freigabefunktionen, mit denen sich die Wirkungsbereiche der Phasen von p wie in Kapitel 3.1 erklärt ergeben. Der Strukturierung von D entsprechend ergeben sich mit X_p und Y_p Anforderungen und Freigaben bzgl. der D–Elemente nach BM–Arten.
Für Beschreibungen des **Verhaltens von** $\mathcal{R}$ mit Ereignisspuren und zugeordneten Zustandsspuren sind jetzt Bedingungen zu erfüllen, die sich aus den Eigenschaften der BM ergeben, die den D–Elementen entsprechen:

- Wenn das System im Zustand z ist, und wenn für ein Ereignis ε Anforderungen als Vorbedingung gestellt sind, dann ist ε in z **zulässig**, wenn die Anforderungen für ε im Zustand z verträglich mit den Einsatzeigenschaften der BM erfüllbar sind.

Das Verhalten von $\mathcal{R}$ wird dem entsprechend mit **BM–zulässigen Ereignisspuren** und diesen zugeordneten Zustandsspuren beschrieben. Den Vergröberungen der D–Elemente zu BM entsprechend werden die Zustände, die zur Konstruktion von Ereignisspuren benutzt werden, ebenfalls vergröbert: Den Ereignissen werden Belegungs–Anforderungs–Zustände zugeordnet. Ein **Belegungs–Anforderungs–Zustand** z von $\mathcal{R}$ ist ein Paar $z = (B, A)$; dabei sind

- B die Beschreibung der BM–Belegungen und
- A die Beschreibung der BM–Anforderungen

der Prozesse von $\mathcal{R}$ im Zustand z.
Für die **Konstruktion** BM–zulässiger Ereignisspuren und zugeordneter Zustandsspuren gilt dann folgendes:

- Mit dem Anfangszustand $z^0 = (B^0, A^0)$ und $k \in I\!N - \{0\}$ seien $\eta = \varepsilon^1 \ldots \varepsilon^k = \eta' \varepsilon^k$ eine BM–zulässige Ereignisspur von $\mathcal{R}$ und $\zeta(z^0, \eta') = z^0 \ldots z^{k-1}$ die z^0 und η' zugeordnete Zustandsspur. Das nach Voraussetzung zulässige Ereignis ε^k ist Initiierungs– oder Terminierungsereignis einer Phase eines Prozesses $p \in P$.

- Wenn ε^k Initiierungsereignis ist, werden die für p gestellten Anforderungen mit ε^k erfüllt; $z^k = (B^k, A^k)$ wird mit z^{k-1} so definiert, daß z^k die für p erfüllten Anforderungen (zusätzliche Belegungen, keine Anforderungen) beschreibt.

- Wenn ε^k Terminierungsereignis ist, werden mit ε^k bis dahin von p belegte BM freigegeben und Anforderungen für die Folgephase von p gestellt; $z^k = (B^k, A^k)$ wird mit z^{k-1} so definiert, daß z^k die sich mit ε^k für p ergebenden, veränderten BM–Belegungen und –Anforderungen (Freigaben, Anforderungen für die Folgephase) beschreibt.

- $\zeta(z^0, \eta) = \zeta(z^0, \eta') z^k$ mit dem sich aus ε^k ergebenden z^k ist die z^0 und η zugeordnete Zustandsfolge.

⌋

Mit dem jetzt erklärten Instrumentarium zur Beschreibung des Verhaltens von $\mathcal{R}$ werden im folgenden die Zusammenhänge zwischen dem BM–Management und der Koordinierung der Berechnungen von $\mathcal{R}$ sowie die Einflüsse der verschiedenen BM–Arten auf die Koordinierung der Berechnungen von $\mathcal{R}$ erklärt. Die Erklärungen erfolgen, indem grundlegende Eigenschaften paralleler Systeme für $\mathcal{R}$ definiert werden.
Das Modellsystem $\mathcal{R}$ spezifiziert die Berechnungen, die ausgeführt werden sollen. Die globale Anforderung an die Koordinierung der Berechnungen von $\mathcal{R}$ sagt, daß die Berechnungen aller Prozesse von $\mathcal{R}$ vollständig ausgeführt werden sollen. Von $\mathcal{R}$ muß entsprechend gefordert werden, daß die mit $\mathcal{R}$ gestellte Koordinierungsaufgabe lösbar ist. Das Kriterium hierfür liefert die folgende Definition:

- Sei z^0 ein Belegungs–Anforderungs–Zustand von $\mathcal{R}$. Das System $\mathcal{R}$ mit z^0 als Anfangszustand ist **BM–zulässig** $\triangleq\!\Leftrightarrow$ es gibt mit z^0 eine vollständige BM–zulässige Ereignisspur von $\mathcal{R}$.

Offenbar ist BM–Zulässigkeit eine notwendige Bedingung dafür, daß die mit $\mathcal{R}$ spezifizierten Berechnungen ausführbar sind. BM–Zulässigkeit beschränkt sich auf die mit den BM–Arten

von $\mathcal{R}$ festgelegten Eigenschaften. Sie sind, wie in Kapitel 4.1.1 erklärt, durchweg Vergröberungen der Eigenschaften der Datenobjekte der Wirkungsbereiche der Prozesse. Von den Berechnungen von $\mathcal{R}$ interessieren letztlich die Wirkungen; sie sind mit den BM–Anforderungen und –Belegungen noch nicht erfaßt. BM–Zulässigkeit ist jedoch eine notwendige Bedingung dafür, daß die Berechnungen eines interpretierten $(\mathcal{R}, \mathcal{I})$ die spezifizierten Wirkungen haben.

Nach der angegebenen Definition erfordert die Überprüfung der BM–Zulässigkeit von $\mathcal{R}$ die Konstruktion einer BM–zulässigen Ereignisspur, also eine Lösung der Koordinierungsaufgabe für $\mathcal{R}$. Wenn man versucht, eine Aufgabe zu lösen, sollte vorab bekannt sein, ob sie lösbar ist. Dem entsprechend interessieren **hinreichende Bedingungen für die BM–Zulässigkeit** von $\mathcal{R}$, für die es nicht erforderlich ist, eine Lösung der Koordinierungsaufgabe zu konstruieren. Entsprechende Bedingungen können relativ einfach angegeben werden, wenn die BM von $\mathcal{R}$ sämtlich FPBM oder FXBM sind; sie werden im folgenden Kapitel angegeben. Wenn die Prozesse von $\mathcal{R}$ BM der weiteren in Kapitel 4.1.1 erklärten Arten benötigen, ist es schwieriger, entsprechende Bedingungen abzuleiten. Das ergibt sich daraus, daß in diesen Fällen die Abhängigkeiten zwischen Prozessen analysiert werden müssen. Das wird insbesondere in den Fällen nötig, in denen EBM auftreten, die von Prozessen erzeugt und von anderen Prozessen eingesetzt und aufgelöst werden.
Für das Weitere sei $\mathcal{R}$ mit dem Anfangszustand z^0 BM–zulässig.
Die im folgenden definierten Eigenschaften und Begriffe sind grundlegend für Lösungen der Koordinierungsaufgaben.
Seien η eine (partielle) BM–zulässige Ereignisspur von $\mathcal{R}$ und z der Endzustand der z^0 und η zugeordneten Zustandsspur.

- Der **Zustand z ist BM–sicher** $\triangleq$ es gibt eine vollständige, BM–zulässige Ereignisspur von $\mathcal{R}$ mit dem Präfix η.
- Der **Prozeß $p \in P$** ist **im Zustand z blockiert** $\triangleq$ p stellt BM–Anforderungen, die in z nicht erfüllbar sind.
- Der Zustand z ist **Deadlock–Zustand** $\triangleq$ es gibt einen Prozeß $p \in P$ so, daß gilt: sind η' eine BM–zulässige Ereignisspur von $\mathcal{R}$ mit dem Präfix η und z' der Endzustand von $\zeta(z^0, \eta')$, so ist p im Zustand z' blockiert.

⌋

Mit der ersten Definition werden die Belegungs–Anforderungs–Zustände von $\mathcal{R}$, die sich mit BM–zulässigen Ereignisspuren ergeben, in zwei Klassen, in BM–sichere und nicht BM–sichere, zerlegt. Für die Konstruktion von Lösungen der Koordinierungsaufgabe für $\mathcal{R}$ ist klar, daß mit jedem Schritt ein BM–sicherer Zustand erreicht werden muß, wenn sich eine Lösung ergeben soll.
Mit der zweiten Definition werden für Prozesse von $\mathcal{R}$ die Berechnungszustände blockiert und nicht blockiert in Belegungs–Anforderungs–Zuständen von $\mathcal{R}$ definiert. $p \in P$ ist in z blockiert sagt, daß die Berechnungen von p nicht fortschreiten können, weil von p benötigte BM in z nicht zugeteilt werden können. Wenn p in z blockiert ist, kann z ein BM–sicherer oder ein nicht BM–sicherer Zustand sein. Wenn z ein BM–sicherer Zustand ist, werden die von p in z gestellten Anforderungen im Gefolge weiterer Ereignisse erfüllt, so daß die Berechnungen von p fortschreiten können.
Die dritte Definition definiert Belegungs–Anforderungs–Zustände z von $\mathcal{R}$ als Deadlock–Zustände. Das sind Zustände, in denen wenigstens ein Prozeß von $\mathcal{R}$ blockiert ist und im Gefolge weiterer Ereignisse gemäß BM–zulässiger Ereignisspuren blockiert bleibt. Wenn z

Deadlock–Zustand ist, ist die Koordinierungsaufgabe für $\mathcal{R}$ nicht mehr lösbar; z ist also ein nicht BM–sicherer Zustand. Ein nicht BM–sicherer Zustand von $\mathcal{R}$ muß jedoch nicht Deadlock–Zustand sein; im Gefolge weiterer Ereignisse ergibt sich jedoch ein Deadlock–Zustand.
Abbildung 4.3 veranschaulicht die definierten Belegungs–Anforderungs–Zustände.

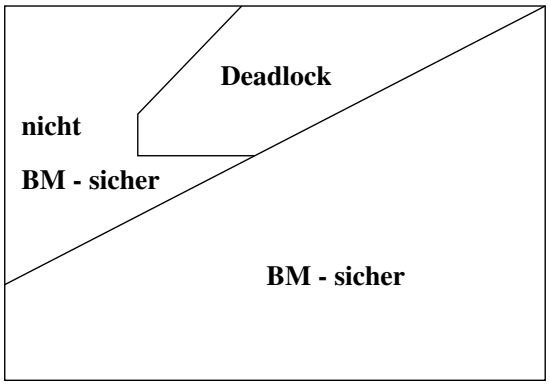


Abbildung 4.3: BM–sichere Zustände und Deadlock–Zustände

Mit den erklärten Zusammenhängen zwischen den BM–Arten und den Koordinierungsaufgaben ergibt sich als erste und unverzichtbare Forderung an ein paralleles System die seiner BM–Zulässigkeit. Wenn diese erfüllt ist, muß zusätzlich gewährleistet werden, daß das System seine Berechnungen vollständig ausführen kann.
Ein BM–zulässiges System $\mathcal{R}$ kann seine Berechnungen vollständig ausführen, wenn für $\mathcal{R}$ konstruktiv gewährleistet ist, daß keine Deadlock–Zustände auftreten können. Das wird mit Verfahren zur **Deadlock–Vermeidung** (deadlock prevention) erreicht: $\mathcal{R}$ wird so konstruiert, daß alle Belegungs–Anforderungs–Zustände, die sich mit BM–zulässigen Ereignisspuren ergeben, BM–sicher sind. Wenn das für $\mathcal{R}$ der Fall ist, bleibt für die Koordinierung der Berechnungen von $\mathcal{R}$ die Aufgabe, für die einzelnen Ereignisse die BM–Zulässigkeit zu gewährleisten; jedes Ereignis wird durch Abwarten schließlich BM–zulässig, und der sich mit einem Ereignis ergebende Zustand ist BM–sicher.
Aus der Definition BM–sicherer Zustände ergibt sich, daß zur Deadlock–Vermeidung für $\mathcal{R}$ i.d.R. detaillierte Kenntnisse über den Ablauf der Berechnungen erforderlich sind; die notwendigen Kenntnisse sind für die einzelnen BM–Arten unterschiedlich. Verfahren zur Deadlock–Vermeidung, die für die einzelnen BM–Arten anwendbar sind, werden in den folgenden Kapiteln noch behandelt. Auch in den Fällen, in denen diese Verfahren aufwendig sind, bleibt festzustellen, daß sich mit der Gewährleistung der Deadlock–Freiheit für ein System $\mathcal{R}$ eine erfolgversprechende Ausgangsbasis dafür ergibt, daß mit den Berechnungen von $\mathcal{R}$ auch die spezifizierten Wirkungen erreicht werden.
Wenn für ein BM–zulässiges System $\mathcal{R}$ nicht konstruktiv Deadlock–Freiheit im soeben erklärten Sinn gewährleistet ist, bleibt die Möglichkeit, durch Kontrollen der Ereignisse zu gewährleisten, daß kein Deadlock–Zustand auftritt. Das wird mit Verfahren zur **Deadlock–Verhinderung** (deadlock avoidance) erreicht: Für die einzelnen Ereignisse wird zusätzlich zu ihrer BM–Zulässigkeit kontrolliert, ob sie zu Deadlock–Zuständen führen; es wird gewährleistet, daß allein solche Ereignisse eintreten, die nicht zu Deadlock–Zuständen führen.
Verfahren zur Deadlock–Verhinderung erfordern Kontrollen für alle Ereignisse, für die zusätzliche BM angefordert werden; es muß gewährleistet werden, daß mit einem Ereignis und im Gefolge des Ereignisses kein Deadlock–Zustand auftritt; das ist partielle BM–Sicherheit. Der

Aufwand für diese Kontrollen kann hoch sein. Wie für Deadlock–Vermeidung, so ist für die Kontrollen erforderlich, daß das zukünftige Verhalten bekannt ist und analysiert wird; dabei kann die Analyse jedoch auf das vom kontrollierten Ereignis abhängige zukünftige Verhalten beschränkt werden. Das ist ein Vorteil von Deadlock–Verhinderung gegenüber –Vermeidung. Zukünftiges Verhalten ist häufig nur unvollständig bekannt. Verfahren zur Deadlock–Verhinderung können auch bei unvollständiger Kenntnis angewandt werden, wenn zunächst grobe Kenntnisse vorliegen, die mit dem Fortschreiten der Berechnungen im festgelegten Rahmen verfeinert werden. Entsprechende Verfahren, die wieder von den BM–Arten abhängig sind, werden noch behandelt. Der Aufwand, der zur Deadlock–Verhinderung zu leisten ist, kann hoch sein; es wird jedoch erreicht, daß die Berechnungen eines Systems vollständig ausgeführt werden können.
Wenn für ein BM–zulässiges System $\mathcal{R}$ weder Verfahren zur Deadlock–Vermeidung noch Verfahren zur Deadlock–Verhinderung angewandt werden, können Deadlock–Zustände auftreten. Nach Definition ist in jedem Deadlock–Zustand wenigstens ein Prozeß von $\mathcal{R}$ so blockiert, daß seine Berechnungen nicht weitergeführt werden können; es können mehrere oder alle Prozesse blockiert sein. Zur Gewährleistung der Betriebsfähigkeit sind für jedes parallele System ohne Deadlock–Vermeidung oder –Verhinderung wenigstens Notmaßnahmen erforderlich: Deadlock–Erkennung und –Behebung. Mit Verfahren zur **Deadlock–Erkennung** (deadlock detection) werden die Belegungs–Anforderungs–Zustände sporadisch oder systematisch auf Deadlock–Zustände analysiert. Wenn ein Deadlock–Zustand erkannt wird, ist ein Schaden eingetreten: die in ihm blockierten Prozesse können ihre Berechnungen nicht spezifikationsgemäß abschließen. Zudem belegen diese Prozesse BM, so daß weitere Prozesse Deadlock–gefährdet sind. In Kombination mit Deadlock–Erkennung sind deshalb Verfahren zur **Deadlock–Behebung** (deadlock recovery) erforderlich. Es sind Verfahren zur Schadensbegrenzung: Mit entsprechenden Maßnahmen wird $\mathcal{R}$ von einem erkannten Deadlock–Zustand in einen Nichtdeadlock–Zustand überführt; der schon entstandene Schaden kann damit nicht behoben werden; er kann begrenzt werden und weiterer Schaden kann vermieden werden. Verfahren zur Deadlock–Behebung können mehr oder weniger einfach und wirksam sein. Ein sehr einfaches Verfahren besteht darin, die Berechnungen der in einem Deadlock–Zustand blockierten Prozesse abzubrechen und die Prozesse aufzulösen; alles Weitere wird dann den Benutzern überlassen, für die Berechnungen ausgeführt werden sollten.
Die Möglichkeiten für Deadlock–Zustände sind wesentlich von den BM–Arten eines Systems abhängig; entsprechendes gilt für die Maßnahmen zur Deadlock–Vermeidung und –Verhinderung. Die Bereitstellung **geeigneter BM–Arten** und BM gehört entsprechend wesentlich zu den Aufgaben des BM–Managements eines Systems. Dazu werden noch BM–Unterarten eingeführt.
Deadlock–Zustände sind Belegungs–Anforderungs–Zustände eines Systems, in denen Prozesse so blockiert werden, daß ihre Berechnungen nicht mehr vollständig ausgeführt werden können. Sie sind nicht die alleinigen Gründe dafür, daß Prozesse ihre Berechnungen nicht vollständig ausführen können. **Nicht faire Koordinierung**, also nicht faire Koordinierungsmechanismen und nicht faire Scheduling–Strategien sind weitere Gründe; sie führen dazu, daß Prozesse ihre Berechnungen nicht fortführen können, weil sie von anderen überholt werden; sie führen zum **Verhungern** (starvation) der Prozesse. Weitere Gründe sind zu **aufwendige Verwaltungsmaßnahmen**, die dazu führen können, daß die Berechnungen der Prozesse nicht mehr fortschreiten, weil die verfügbaren BM für Verwaltungsmaßnahmen eingesetzt sind; das kann zum Verhungern der Prozesse und zum Stillstand ihrer Berechnungen wegen Überverwaltung (thrashing) führen.
Im folgenden werden die Zusammenhänge zwischen BM–Management und Koordinierung einschließlich einiger Verfahren zum Deadlock–Problem an Systemen mit spezifischen BM–

Arten genauer erklärt.

4.2 Systeme mit frei exklusiv benutzbaren Betriebsmitteln

Im folgenden werden Systeme betrachtet, für die allein WBM nach der in Kapitel 4.1.1 erklärten Zusammenfasssung zu verwalten sind. Von den WBM sind die FPBM für die Verwaltung unproblematisch, weil sie ohne Einschränkungen bei Bedarf zugeteilt und eingesetzt werden können. Von den BPBM ist die Verwaltung der FXBM problematisch; die Probleme, die sich ergeben, sind jedoch relativ einfach zu lösen. An Systemen, für die allein die FXBM die kritischen BM sind, lassen sich deshalb die Zusammenhänge zwischen BM–Management und Koordinierung gut erklären.
Charakteristisch für die FXBM ist, daß sie zwar exklusiv benutzbar, aber ohne weitere Restriktionen einsetzbar sind. Zu den FXBM gehören insbesondere die Komponenten der Hardwarekonfiguration eines parallelen Systems. Die Charakteristika der FXBM haben zur Folge, daß sie vielfältig benötigt werden und einsetzbar sind; sie haben weiter zur Folge, daß ihre Verwaltung relativ einfach ist.

4.2.1 Das Modellsystem

Zur Beschreibung der Aufgabe, die zu lösen ist, wird ein System $\mathcal{R}$ mit sequentiellen Prozessen benutzt, dessen hier relevante Datenobjekte und Komponenten sämtlich zu FXBM vergröbert sind. Die FXBM treten in großer Anzahl auf, und sie eignen sich für die Bildung von Äquivalenzklassen. Dabei sind zwei BM dann äquivalent, wenn sie für anfordernde Prozesse gleich einsetzbar sind.
Für das System $\mathcal{R}$ wird davon ausgegangen, daß seine BM in Äquivalenzklassen zerlegt und entsprechend beschrieben sind. Mit $l \in \mathbb{N}$ seien l Klassen gebildet; sie werden mit $1, 2, \ldots, l$ identifiziert. Für $i \in \{1, \ldots, l\}$ soll es $w_i \in \mathbb{N}$ Elemente der Klasse i geben. Dann werden die BM von $\mathcal{R}$ mit

- dem Vektor der **verfügbaren BM** $w = (w_1, \ldots, w_l)$ beschrieben.

Wenn P die Prozeßmenge und α die Phasenanzahlfunktion von $\mathcal{R}$ sind, werden mit $X = (X(p) \mid p \in P)$ und $Y = (Y(p) \mid p \in P)$ die BM–Anforderungen und –Freigaben für alle Phasen aller Prozesse beschrieben. Dabei sind

- $X(p)$ die Matrix der **Anforderungen** und
- $Y(p)$ die Matrix der **Freigaben**

des Prozesses p. $X(p)$ und $Y(p)$ sind $\alpha(p) \times l$–Matrizen. Für $i \in \{1, \ldots, l\}$ sind

- die i–te Zeile von $X(p)$ der Vektor $x(i,p) = (x_1(i,p), \ldots, x_l(i,p))$ mit der Anzahl $x_j(i,p)$ der BM der Klasse $j \in \{1, \ldots, l\}$, die für die i–te Phase von p angefordert werden, und
- die i–te Zeile von $Y(p)$ der Vektor $y(i,p) = (y_1(i,p), \ldots, y_l(i,p))$ mit der Anzahl $y_j(i,p)$ der BM der Klasse $j \in \{1, \ldots, l\}$, die mit dem Abschluß der i–ten Phase des Prozesses p freigegeben werden.

Die zu X und Y zusammengefaßten $(X(p))$ und $(Y(p))$ beschreiben für alle Prozesse detailliert die BM–Anforderungen und –Freigaben den Phasenstrukturen entsprechend. Diese detaillierten Kenntnisse liegen im voraus oft nicht vor. Diesen Gegebenheiten wird dadurch Rechnung getragen, daß (wenigstens) mit

- $H = (h(p) \mid p \in P)$ die **maximalen Anforderungen**

der Prozesse gegeben sind. Dabei ist für $p \in P$ mit H der Vektor $h(p) = (h_1(p), \ldots, h_l(p))$ der maximalen Anforderungen von p mit der Anzahl $h_j(p)$ der BM der Klasse $j \in \{1, \ldots, l\}$ über alle Phasen von p.
Als Beschreibung der Anforderungen an die Verwaltung der allein als relevant erfaßten FXBM und an die Koordinierung der Berechnungen ergibt sich damit das 6–tupel $\mathcal{R} = (w, P, \alpha, H, X, Y)$ mit den erklärten Bedeutungen der Komponenten. Reihenfolgerestriktionen für die Phasen der Prozesse sind implizit mit X und Y und den Charakteristika der FXBM festgelegt. Die maximalen Anforderungen sind mit H als bekannt vorausgesetzt. Die Anforderungen und Freigaben gemäß X und Y können partiell oder vollständig bekannt sein; sie müssen jedoch mit H verträglich sein.

4.2.2 BM–Zulässigkeit und Koordinierung

Die zu lösende BM–Verwaltungs– und Koordinierungsaufgabe sei mit $\mathcal{R} = (w, P, \alpha, H, X, Y)$ festgelegt.
Die Lösung der Aufgaben sind wesentlich bestimmt durch die FXBM–Charakteristika: Für sie muß zwar exklusive Benutzung gewährleistet werden; sie sind jedoch frei einsetzbar. Anforderungen der Prozesse können immer dann erfüllt werden, wenn freie BM in ausreichender Anzahl zur Verfügung stehen.
Als erstes ist die Lösbarkeit der Aufgaben, also die **BM–Zulässigkeit** von $\mathcal{R}$, zu klären. Dazu steht H zur Verfügung, und mit H kann sofort eine hinreichende Bedingung für BM–Zulässigkeit angegeben werden:

- $\mathcal{R}$ ist BM–zulässig, wenn $h(p) \leq w$ für alle $p \in P$ gilt.

Wenn diese Bedingung erfüllt ist, gibt es die geforderten, vollständigen, BM–zulässigen Ereignisspuren für $\mathcal{R}$, nämlich die $|P|!$ Spuren der sequentiellen Berechnungen der $p \in P$ gemäß einer Permutation von P.
Die angegebene Bedingung basiert auf der Verträglichkeit der Anforderungen und Freigaben für die Phasen der Prozesse gemäß X und Y mit H. Wenn X und Y a priori vollständig bekannt sind, sagt diese Verträglichkeit:

- Für alle Prozesse $p \in P$ gelten
  $0 \leq \sum_{j=1}^{i} x(j,p) - \sum_{j=1}^{i-1} y(j,p) \leq h(p)$ für alle $i \in \{1, \ldots, h(p)\}$ und
  $\sum_{j=1}^{\alpha(p)} x(j,p) - \sum_{j=1}^{\alpha(p)} y(j,p) = 0.$

Wenn X und Y vollständig bekannt sind, können die maximalen Anforderungen H berechnet werden. H und die Verträglichkeit der X und Y mit H sind dann relevant, wenn die Anforderungen und Freigaben sukzessive mit dem Fortschreiten der Berechnungen der Prozesse bekannt werden.
Im weiteren wird $\mathcal{R} = (w, P, \alpha, H, X, Y)$ als BM–zulässig vorausgesetzt.

Die Lösung der BM–Verwaltungs– und Koordinierungsaufgaben für $\mathcal{R}$ erfolgt, wie in Kapitel 4.1.2 allgemein erklärt, durch Konstruktion BM–zulässiger Ereignisspuren und der ihnen zugeordneten Zustandsspuren.
Der Raum der **Belegungs–Anforderungs–Zustände** wird mit Elementen $z = (B, A)$ festgelegt. Dabei sind

- $B = (b(p) \mid p \in P)$ die Matrix der **Belegungen** der BM für die Prozesse $p \in P$ im Zustand z mit dem Vektor $b(p) = (b_1(p), \ldots, b_l(p))$, der mit $b_j(p)$ die Anzahl der belegten BM der Klasse $j \in \{1, \ldots, l\}$ angibt, und
- $A = (a(p) \mid p \in P)$ die Matrix der **Anforderungen** an BM für die Prozesse $p \in P$ im Zustand z mit dem Vektor $a(p) = (a_1(p), \ldots, a_l(p))$, der mit $a_j(p)$ die Anzahl der angeforderten BM der Klasse $j \in \{1, \ldots, l\}$ angibt.

Für die Konstruktion BM–zulässiger Ereignisspuren von $\mathcal{R}$ wird der **Anfangszustand** $z^0 = (B^0, A^0)$ so festgelegt, daß mit ihm die Anforderungen für die 1–ten Phasen aller Prozesse gestellt und alle BM frei sind. Dem entsprechend sind

- $B^0 = (b^0(p) \mid p \in P)$ mit $b^0(p) = 0$ für alle $p \in P$ und
- $A^0 = (a^0(p) \mid p \in P)$ mit $a^0(p) = x(1, p)$ für alle $p \in P$.

Vom Anfangszustand z^0 ausgehend erfolgt die **Konstruktion BM–zulässiger Ereignisspuren** und der zugeordneten Zustandsspuren Schritt für Schritt; das wird im folgenden induktiv definiert. Bei jedem Schritt ist die BM–Zulässigkeit des eintretenden Ereignisses zu gewährleisten. Mit dem k–ten Schritt, $k \in \mathbb{N}$, wird der Zustand $z^k = (B^k, A^k)$ erreicht; dabei sind $B^k = (b^k(p) \mid p \in P)$ und $A^k = (a^k(p) \mid p \in P)$. Ein Ereignis ist in einem Zustand zulässig, wenn die für das Ereignis gestellten BM–Anforderungen mit den im Zustand freien BM erfüllt werden können.
Mit dem Anfangszustand z^0 und mit $k \in \mathbb{N}$ seien $\eta = \varepsilon^1 \ldots \varepsilon^k$ eine unvollständige, BM–zulässige Ereignisspur von $\mathcal{R}$ und $\zeta(z^0, \eta) = z^0 \ldots z^k$ die z^0 und η zugeordnete Zustandsspur. Das Ereignis ε sei Initiierungs– oder Terminierungsereignis der i–ten Phase des Prozesses p mit $i \in \{1, \ldots, \alpha(p)\}$; η enthalte genau die Ereignisse von p, die der Phasenstruktur von p entsprechend vor ε eintreten müssen.
Sei ε das **Initiierungsereignis der i–ten Phase von p**; dann sind für ε Anforderungen gestellt; sie sind mit $z^k = (B^k, A^k)$ durch $a^k(p)$ beschrieben; die Erfüllung dieser Anforderungen ist Bedingung für das Eintreten von ε. ε ist zulässig in z^k, wenn diese Anforderungen erfüllbar sind:

- ε ist **zulässig** in $z^k \Leftrightarrow \sum_{q \in P} b^k(q) + a^k(p) \leq w$.

Wenn ε in z^k zulässig ist, kann ε als Folgeereignis ε^{k+1} von η eintreten. Wenn das der Fall ist, wird $z^{k+1} = (B^{k+1}, A^{k+1})$ wie folgt berechnet:

- $b^{k+1}(p) := b^k(p) + a^k(p)$ und $b^{k+1}(q) := b^k(q)$ für alle $q \in P - \{p\}$;
- $a^{k+1}(p) := 0$ und $a^{k+1}(q) := a^k(q)$ für alle $q \in P - \{p\}$.

Sei ε das **Terminierungsereignis der i–ten Phase von p**; dann sind für ε keine Anforderungen gestellt; damit gilt

- ε ist **zulässig** in z^k.

Mit ε werden BM von p freigegeben und Anforderungen für die Folgephase von p gestellt. Wenn ε als Folgeereignis ε^{k+1} von η eintritt, wird $z^{k+1} = (B^{k+1}, A^{k+1})$ wie folgt berechnet:

- $b^{k+1}(p) := b^k(p) - y(i,p)$ und $b^{k+1}(q) := b^k(q)$ für alle $q \in P - \{p\}$.
- $a^{k+1}(p) := \begin{cases} x(i+1,p) & , \text{ falls } i < \alpha(p) \\ 0 & , \text{ falls } i = \alpha(p) \end{cases}$ und $a^{k+1}(q) := a^k(q)$ für alle $q \in P - \{p\}$.

⌋

Damit ist die Konstruktion BM–zulässiger Ereignisspuren für $\mathcal{R}$ und der ihnen zugeordneten Zustandsspuren erklärt. Wenn $z^k = (B^k, A^k)$ ein entsprechender Zustand ist, gilt

- $0 \leq \sum\limits_{p \in P} b(p) \leq w$

nach Definition der BM–Zulässigkeit. Im Zustand z^k ist

- $r^k \triangleq w - \sum\limits_{p \in P} b(p)$ der Vektor der in z^k **freien BM** oder der BM–Rest in z^k.

4.2.3 Erweitertes BM–Management: Unterbrechbare BM

Mit dem in Kapitel 4.2.2 Erklärten sind die Vorbereitungen zur Lösung der BM–Verwaltungs– und Koordinierungsaufgaben für das System $\mathcal{R}$ getroffen. Gemäß Kapitel 4.1.2 sind dazu für $\mathcal{R}$ BM–zulässige Ereignisspuren zu konstruieren, und dabei muß gewährleistet werden, daß keine Deadlock–Zustände auftreten.
Für Deadlock–Zustände von $\mathcal{R}$ ist, wie in Kapitel 4.1.2 erklärt, charakteristisch, daß es in ihnen Prozesse von $\mathcal{R}$ gibt, die ihre Berechnungen nicht fortsetzen und damit nicht vollständig ausführen können. Die Möglichkeiten für Deadlock–Zustände sind einerseits von den Prozessen und andererseits vom BM–Management bestimmt. Auf die Zusammenhänge zwischen möglichen Deadlock–Zuständen und den BM–Arten, die das BM–Management Prozessen zur Verfügung stellt, ist schon in Kapitel 4.1 hingewiesen. Für die weiteren Erklärungen von Lösungen der für $\mathcal{R}$ gestellten Aufgaben ist es geboten, diese wechselseitigen Abhängigkeiten zwischen den Prozessen und dem BM–Management von $\mathcal{R}$ genauer zu betrachten.
Die wechselseitigen **Abhängigkeiten zwischen den Prozessen und dem BM–Management von $\mathcal{R}$** sind charakteristisch für alle arbeitsteiligen, kooperativen Problemlösungen. Nach dem bisher Erklärten erfolgt die Verwaltung der FXBM von $\mathcal{R}$ und die Koordinierung der Berechnungen der Prozesse von $\mathcal{R}$ wie folgt:

- ein Prozeß $p \in P$ stellt für die Fortsetzung seiner Berechnungen mit einer Phase p_i BM–Anforderungen;
- diese Anforderungen müssen, damit p sein Ziel erreichen kann, vom BM–Management BM–zulässig und BM–sicher erfüllt werden.

Die wechselseitigen Abhängigkeiten treten hier wie folgt auf:

- Der Prozeß p stellt BM–Anforderungen für p_i; diese Anforderungen sind auf die vom BM–Management erfüllbaren eingeschränkt. Dem entspricht, daß der Prozeß p und das System BM–zulässig sein muß.
- Das BM–Management gewährleistet die Erfüllung der von p für p_i gestellten Anforderungen mit BM–zulässigen und BM–sicheren Zuteilungen von BM an den Prozeß p.

Die diesen Abhängigkeiten entsprechenden Maßnahmen sind **explizite Maßnahmen** auf der Grundlage festgelegter Verhaltensregeln für die kooperierenden Partner:

- Der Prozeß p stellt BM–Anforderungen für p_i und erwartet, daß diese erfüllt werden; dem entspricht, daß für das Eintreten des Initiierungsereignisses $\underline{p_i}$ vorausgesetzt wird, daß die Anforderungen erfüllt sind; die Berechnungen der Phase p_i setzen voraus, daß diese Bedingungen erfüllt sind.

- Das BM–Management erhält die BM–Anforderungen von p für p_i und hat die verbindliche Aufgabe, die Anforderungen BM–zulässig und BM–sicher zu erfüllen; dem entspricht, daß das Initiierungsereignis $\underline{p_i}$ mit der Erfüllung der gestellten BM–Anforderungen eintritt; der Prozeß p kann seine Berechnungen mit der Phase p_i fortsetzen.

Das Wesentliche dieser Verhaltensregeln ist die **Verbindlichkeit** der Beiträge der beiden Partner: Jeder Partner kann vom anderen das den festgelegten Regeln entsprechende Verhalten erwarten; auf dieser Grundlage werden die verfolgten Ziele gemeinsam kooperativ erreicht.
Auf der Grundlage der Verbindlichkeit der beiderseitigen Beiträge werden erweiterte, **implizite BM–Managementmaßnahmen** möglich. Implizit bezieht sich dabei auf die Sicht der Prozesse auf diese Maßnahmen: Die Prozesse leisten keine Beiträge zu diesen Maßnahmen; die Wirkungen der Maßnahmen sind für die Prozesse transparent (unsichtbar); die Maßnahmen sind relativ zu den Prozessen **funktional wirkungslos**. Für das BM–Management sind es spezielle Maßnahmen der BM–Verwaltung, die dazu dienen, die Effizienz der Berechnungen von $\mathcal{R}$ zu steigern.
Für die BM, auf welche die Maßnahmen angewandt werden, ergibt sich, daß sie wenigstens 2–stufig sind:

- Sie haben einerseits Eigenschaften, die für Prozesse relevant sind; bzgl. dieser sind die Maßnahmen funktional wirkungslos.

- Sie haben andererseits Eigenschaften, die für das BM–Management relevant sind; bzgl. dieser sind die Maßnahmen funktional wirksam und effizienzsteigernd.

Dieser 2–Stufigkeit entsprechend sind die BM–Managementmaßnahmen für BM von Interesse, die mit abstrakten Eigenschaften definiert und mittels weiterer BM realisiert sind. Die entsprechenden BM sind bzgl. ihres Einsatzes für Prozesse funktional wirkungsfrei unterbrechbare oder kurz **unterbrechbare BM** (preemptable resources). Die BM, mittels derer ein unterbrechbares BM realisiert ist, sind entziehbare BM; diese entziehbaren BM sind häufig auch äquivalent austauschbar.
Im folgenden wird ein Beispiel für das Erklärte angegeben.

Beispiel:

Sei d ein Datenobjekt mit dem Zustandsraum $Z(d)$, das zu einem FXBM vergröbert ist. Als FXBM kann d von einem Prozeß p angefordert, nach Zuteilung eingesetzt, benutzt und schließlich freigegeben werden.
Die Zustände von d sind einerseits mit d definiert und als solche werden sie mit Berechnungen von Prozessen, die d einsetzen, verändert. Andererseits müssen die Zustände mit geeigneten Repräsentationen als Containerinhalte gespeichert werden. Diese Container sind die BM, mittels derer d realisiert wird.

Sei jetzt d dem Prozeß p zugeteilt und von diesem eingesetzt; dann verändert p mit seinen Berechnungen die Zustände von d. Sei $z \in Z(d)$ ein entsprechend erreichter Zustand; c sei der Container, mit dem d realisiert ist; c enthält also z.
Wenn d ein funktional wirkungsfrei unterbrechbares BM mit c als entziehbarem BM ist, sind dazu folgende Maßnahmen des BM–Managements erforderlich:

- der Prozeß p wird blockiert;
- z wird als Inhalt eines geeigneten Containers c' gespeichert und c wird freigegeben;
- z wird — nach einer Wartezeit — als Inhalt eines mit c äquivalenten Containers c'' gespeichert und c' wird freigegeben;
- der Prozeß p wird entblockiert.

Für den Prozeß p gilt in dem Zeitintervall, in dem diese Maßnahmen erfolgen: p belegt d.
Im betrachteten Fall kann $c = c''$ gelten; es kann jedoch auch $c \neq c''$ gelten, wenn c äquivalent austauschbar ist. Für p sind vorraussetzungsgemäß die Realisierungen von d mit c und mit c'' äquivalent.
Die BM–Managementmaßnahmen sind für p funktional wirkungsfrei; sie sind für p temporal wirksam: während der Blockierung schreitet die Berechnung von p nicht fort.
Abbildung 4.4 veranschaulicht die Maßnahmen für die Sicht von p und für die Sicht des BM–Managements.

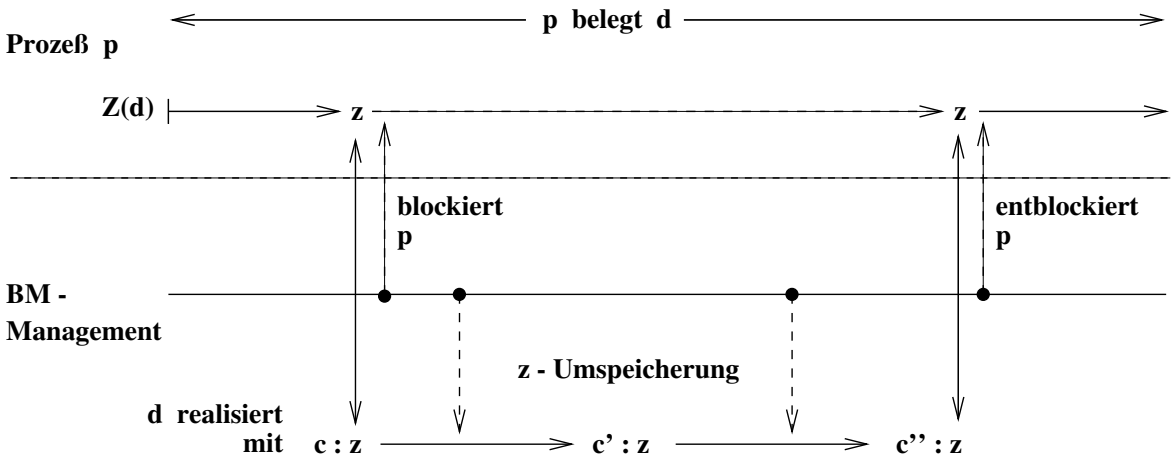


Abbildung 4.4: Funktional wirkungsfreie Unterbrechung des d–Einsatzes für den Prozeß p; Entzug und äquivalenter Austausch von c

⌋

Die erklärten Maßnahmen des BM–Managements, mit denen ein BM d zu einem unterbrechbaren BM wird, nutzen wesentlich die 2–Stufigkeit von d aus. Sie sind hier für ein FXBM d erklärt; sie sind jedoch auch auf andere BM–Arten anwendbar. Die BM, mit denen d realisiert wird, im Beispiel also die Container, müssen FXBM sein. Die Maßnahmen sind auf mehrstufige BM verallgemeinerbar.
Die erklärten Maßnahmen des BM–Managements sind wichtig für die Steigerung der **Effizienz** der Berechnungen des Systems. Das ergibt sich aus der Flexibilisierung des Einsatzes der Realisierungs–BM. Dieser Steigerung der Effizienz entsprechen verschärfte Anforderungen an die **Verbindlichkeit** der Verhaltensregeln für das BM–Management:

- Für einen Prozeß p, der ein unterbrechbares BM d anfordert, erhält und einsetzt, gilt, daß er d bis zur Freigabe belegt.

- Für die impliziten Maßnahmen des BM–Managements muß „p belegt d“ gewährleistet werden. Dazu gehört, daß p blockiert und nach endlicher Zeit mit unveränderten für p relevanten d–Eigenschaften entblockiert wird.

4.2.4 Deadlock–Zustände

Mit dieser Erklärung der erweiterten Fähigkeiten des BM–Managements, mit denen unterbrechbare BM zur Verfügung gestellt werden können, und mit denen sich verschärfte wechselseitige Abhängigkeiten zwischen den Prozessen und dem BM–Management ergeben, ergeben sich auch veränderte Rahmenbedingungen für Lösungen der für das System $\mathcal{R}$ gestellten BM–Verwaltungs– und Koordinierungsaufgaben: Die für $\mathcal{R}$ relevanten FXBM zerfallen in die unterbrechbaren und in die nicht unterbrechbaren FXBM. Für die unterbrechbaren BM gewährleistet das BM–Management, daß sie keine endlosen Prozeßblockierungen verursachen. Demnach **begründen unterbrechbare FXBM keine Deadlock–Zustände** von $\mathcal{R}$.
Diesen Gegebenheiten wird dadurch Rechnung getragen, daß für das Weitere angenommen wird: Die relevanten BM von $\mathcal{R}$ sind nicht unterbrechbare FXBM. Die in Kapitel 4.2.2 angegebenen Bedingungen für BM–Zulässigkeit gelten auch nach dieser Modifikation von $\mathcal{R}$. Ebenso werden BM–zulässige Ereignisspuren von $\mathcal{R}$ und zugeordnete Zustandsspuren wie dort angegeben konstruiert und beschrieben. Im folgenden werden die Charakteristika der Deadlock–Zustände von $\mathcal{R}$ erklärt.
Dazu seien $\eta = \varepsilon^1 \dots \varepsilon^k$ eine BM–zulässige Ereignisspur von $\mathcal{R}$ und $\zeta(z^0, \eta) = z^0 \dots z^k$ die dem Anfangszustand $z^0 = (B^0, A^0)$ und η zugeordnete Zustandsspur. Dann gilt:

- der Zustand $z^k = (B^k, A^k)$ ist **Deadlock–Zustand** von $\mathcal{R} \overset{\triangle}{\Longleftrightarrow}$

  es gibt $Q \subseteq P$ mit $Q \neq \emptyset$ so, daß für alle $q \in Q$

  $$a^k(q) \not\leq w - \sum_{p \in Q} b^k(p)$$

  oder äquivalent dazu

  $$a^k(q) \not\leq r^k + \sum_{p \in P-Q} b^k(p)$$

  gilt.

⌋

Diese Definition eines Deadlock–Zustandes z^k von $\mathcal{R}$ ergibt sich aus der in Kapitel 4.1.2 allgemein angegebenen durch Anpassung an die speziellen Gegebenheiten und Beschreibungen von $\mathcal{R}$. Sie sagt:
Wenn z^k Deadlock–Zustand ist, gibt es eine nichtleere Teilmenge Q der Prozesse von $\mathcal{R}$ so, daß diese Prozesse in z^k blockiert sind und, weil sie sich wechselseitig blockieren, endlos blockiert bleiben.
Die Prozesse, die Elemente von Q sind, sind die im Deadlock–Zustand z^k blockierten Prozesse. Dabei kann $Q = P$ gelten; wenn das der Fall ist, ist z^k ein totaler Deadlock–Zustand von $\mathcal{R}$.

Wenn $Q \neq P$ gilt, können für die Prozesse $p \in P - Q$ in z^k und in nachfolgenden Zuständen Ereignisse BM–zulässig eintreten. Die Prozesse können evtl. ihre Berechnungen vollständig ausführen und die von ihnen in z^k belegten BM freigeben; damit ändert sich nichts daran, daß die Prozesse $q \in Q$ in z^k blockiert sind und nach z^k blockiert bleiben.
Aus der angegebenen Definition folgen unmittelbar zwei charakteristische Eigenschaften von Deadlock–Zuständen des Systems $\mathcal{R}$. Dazu seien z^k Deadlock–Zustand von $\mathcal{R}$ und Q die Menge der in z^k blockierten Prozesse.
Dann folgt aus der BM–Zulässigkeit von $\mathcal{R}$:

(1) $|Q| \geq 2$.

Weiter folgen aus $0 \leq w - \sum_{p \in Q} b^k(p)$ und der BM–Zulässigkeit von $\mathcal{R}$:

(2) es gibt $Q' \subseteq Q$ mit $|Q'| \geq 2$ so, daß $a^k(q) \neq 0$ und $b^k(q) \neq 0$ für alle $q \in Q'$ gilt; die $q \in Q'$ sind die den Deadlock–Zustand z^k verursachenden Prozesse; wenn $Q - Q' \neq \emptyset$ gilt, sind die $q \in Q - Q'$ wegen der Prozesse gemäß Q' in z^k endlos blockiert.

Ein weiteres Charakteristikum der Deadlock–Zustände von $\mathcal{R}$ ergibt sich aus dem Belegungs–Anforderungs–Graphen, der für alle Zustände von $\mathcal{R}$ definiert ist.
Sei $z = (B, A)$ ein BM–zulässiger Belegungs–Anforderungs–Zustand von $\mathcal{R}$. Der Graph $\mathcal{G}_\mathcal{R}(z) = (K, L)$ ist der **Belegungs–Anforderungs–Graph** von $\mathcal{R}$ im Zustand z, wenn gelten:

- $K = \{1, \ldots, l\}$ ist die den BM–Klassen von $\mathcal{R}$ entsprechende Knotenmenge,
- für die Kantenmenge L gilt $L = \{(i, j) \in K \times K \mid \exists\, p \in P(b_i(p) \geq 1 \wedge a_j(p) \geq 1)\}$.

Im Belegungs–Anforderungs–Graphen $\mathcal{G}_\mathcal{R}(z) = (K, L)$ gibt es also für $i, j \in K$ eine Kante $i \to j$ genau dann, wenn es einen Prozeß $p \in P$ gibt, der in z wenigstens ein Element der BM–Klasse i belegt und wenigstens ein Element der BM–Klasse j anfordert. Es ist also zu beachten, daß der Graph $\mathcal{G}_\mathcal{R}(z)$ den Zustand z von $\mathcal{R}$ nur vergröbert beschreibt.

Beispiel:

Für das System $\mathcal{R}$ seien $P = \{p, q, u\}$ und $z = (B, A)$ mit

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

ein BM–zulässiger Belegungs–Anforderungs–Zustand. Abbildung 4.5 zeigt $\mathcal{G}_\mathcal{R}(z)$

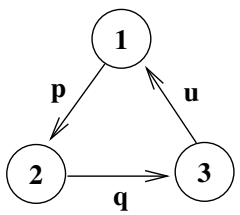


Abbildung 4.5: Der Belegungs–Anforderungs–Graph $\mathcal{G}_\mathcal{R}(z)$

Weil z BM–zulässig ist, muß für $\mathcal{R}$ $w \geq (1,1,1)$ gelten.
Wenn für $\mathcal{R}$ $w = (1,1,1)$ gilt, ist z totaler Deadlock–Zustand.
Wenn für $\mathcal{R}$ $w = (1,1,2)$ gilt, ist z kein Deadlock–Zustand.

⌋

Mit den Belegungs–Anforderungs–Graphen ergibt sich die folgende charakteristische Eigenschaft von Deadlock–Zuständen des Systems $\mathcal{R}$:

- Seien z^k ein BM–zulässiger Zustand und $\mathcal{G}_{\mathcal{R}}(z^k)$ der Belegungs–Anforderungs–Graph von $\mathcal{R}$ in z^k. Wenn z^k Deadlock–Zustand ist, enthält $\mathcal{G}_{\mathcal{R}}(z^k)$ einen Zyklus.

Der Beweis dieses Satzes ergibt sich wie folgt:
Seien z^k Deadlock–Zustand und Q die Menge der in z^k blockierten Prozesse. Sei $q \in Q$; dann gibt es wegen (2) eine Kante $i \to j$ in $\mathcal{G}_{\mathcal{R}}(z^k)$, die q begründet. Weil q in z^k blockiert ist, gibt es $q' \in Q$ und wegen (2) eine Kante $j \to m$ in $\mathcal{G}_{\mathcal{R}}(z^k)$, also einen Pfad $i \to j \to m$. Diese Argumentation kann fortgesetzt werden, so daß es einen beliebig langen Pfad $i \to j \to \ldots$ in $\mathcal{G}_{\mathcal{R}}(z^k)$ gibt. Weil die Knotenmenge von $\mathcal{G}_{\mathcal{R}}(z^k)$ endlich ist, folgt der Zyklus.

⌋

Zu den angegebenen Aussage ist zu beachten, daß ein Zyklus im Belegungs–Anforderungs–Graphen von $\mathcal{R}$ in z^k eine notwendige, jedoch keine hinreichende Bedingung dafür ist, daß z^k Deadlock–Zustand ist. Das oben angegebene Beispiel zeigt dies.
Mit dem bisher Erklärten sind die Deadlock–Zustände des Systems $\mathcal{R}$ charakterisiert. Die Eigenschaften von $\mathcal{R}$ wurden hier schrittweise so eingeschränkt, daß allein die zur Lösung der BM–Verwaltungs– und Koordinierungsaufgaben relevanten beibehalten wurden.
Für Systeme mit WBM kann man mit den bisher für $\mathcal{R}$ abgeleiteten Ergebnissen **notwendige Bedingungen für Deadlock–Zustände** angeben, die wichtige Hinweise für die Konstruktion entsprechender Systeme liefern. Es sind die vier Bedingungen, die bisher sukzessive erklärt wurden:

(ND1) Es gibt exklusiv benutzbare BM (mutual exclusion), also WBM der Art XBM.

(ND2) Es gibt nichtunterbrechbare BM (no preemption).

(ND3) Es gibt Prozesse, die BM belegen und weitere BM anfordern (hold and wait).

(ND4) Der Belegungs–Anforderungs–Graph eines BM–zulässigen Zustandes enthält einen Zyklus (circular wait).

Im weiteren wird angenommen, daß die BM des Systems $\mathcal{R}$, die erfaßt sind, sämtlich Deadlock–Zustände von $\mathcal{R}$ verursachen können; es sind also nichtunterbrechbare FXBM.

4.2.5 Deadlock–Vermeidung

Das System $\mathcal{R} = (w, P, \alpha, H, X, Y)$ sei BM–zulässig. Für $\mathcal{R}$ werden im folgenden Verfahren angegeben, mit denen erreicht wird, daß die BM–zulässigen Belegungs–Anforderungs–Zustände von $\mathcal{R}$ sämtlich BM–sicher sind.
Mit den in Kapitel 4.2.4 angegebenen notwendigen Bedingungen für Deadlock–Zustände von Systemen mit WBM ergeben sich Verfahren zur Deadlock–Vermeidung damit, daß Systeme konstruiert werden, für die stets wenigstens eine der vier Bedingungen nicht erfüllt ist. Für das System $\mathcal{R}$ sind die erfaßten BM sämtlich nichtunterbrechbare FXBM, so daß die Bedingungen ND1 und ND2 erfüllt sind; zur Deadlock–Vermeidung können also noch ND3 und ND4 genutzt werden.

4.2.5.1 Kaskadenförmiger BM–Einsatz

Deadlock–Vermeidung für das System $\mathcal{R}$ durch **Nutzung** der Notwendigkeit **der Bedingung ND3** bedeutet, die Prozesse von $\mathcal{R}$ mit Phasenstrukturen so zu konstruieren, daß bei Phasenübergängen die jeweils belegten BM freigegeben werden, wenn weitere BM erforderlich sind. Es ergeben sich Prozesse mit **kaskadenförmigem BM–Einsatz**:

- Für die Prozesse gilt: Sie stellen BM–Anforderungen höchstens in Zuständen, in denen sie keine BM belegen.
- Folgen von Phasen eines Prozesses werden zu Kaskaden zusammengefaßt, für die gelten:
  - BM–Anforderungen werden für die 1. Phase einer Kaskade gestellt;
  - mit den Terminierungsereignissen der Phasen werden BM freigegeben.

Für einen Prozeß p von $\mathcal{R}$, der entsprechend konstruiert ist, ergibt sich, daß die (kritischen) BM des Wirkungsbereichs von p in einer Folge von Kaskaden eingesetzt werden. Alle Phasen von p können zu einer Kaskade zusammengefaßt sein. Ein entarteter Fall liegt vor, wenn p einphasig ist. Abbildung 4.6 zeigt einen Prozeß mit kaskadenförmigem BM–Einsatz.

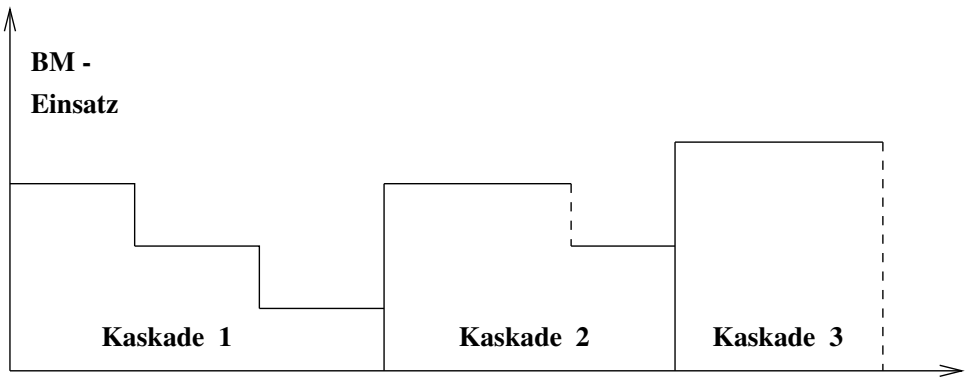


Abbildung 4.6: Kaskadenförmiger BM–Einsatz für einen Prozeß

Deadlock–Vermeidung für das System $\mathcal{R}$ mit kaskadenförmigem BM–Einsatz trägt den charakteristischen Eigenschaften der FXBM und den Einsatzmöglichkeiten der FXBM für die Berechnungen der Prozesse von $\mathcal{R}$ Rechnung:

- Für die Verwaltung der FXBM sind die Belegungszustände *frei* und *belegt* relevant: Ein BM x, das frei ist, kann jedem Prozeß, der x anfordert, zugeteilt werden; mit der Zuteilung von x an einen Prozeß p wird x belegt, und x ist exklusiv für die Berechnungen von p eingesetzt bis zur Freigabe von x durch p; mit der Überführung von x in *frei* steht x wieder für Prozesse zur Verfügung.
- In einem Zeitintervall, in dem das BM x für einen Prozeß p eingesetzt ist, bestimmt allein p die Zustände und Zustandsänderungen von x; x ist Teil des Wirkungsbereichs von p; Zustände von x sind Zwischenergebnisse der Berechnungen von p, die für p zur Verfügung stehen bis zur Freigabe von x durch p.
- Eine Kaskade k von p faßt die Folge der Phasen von p zusammen, in denen FXBM mit von p bestimmten Zuständen benötigt werden. Die benötigten BM stehen mit dem

Initiierungsereignis der 1. Phase von k zur Verfügung; die Zustände der BM bleiben bei Übergängen zwischen den Phasen von k erhalten, und das gilt bis zur Freigabe der BM. Das gilt nicht mehr für den Übergang von k zur Folgekaskade von p.

Kaskadenförmiger BM–Einsatz für die Prozesse von $\mathcal{R}$ erfordert, daß alle (kritischen) BM, die für eine Kaskade benötigt werden, vom Initiierungsereignis der 1. Phase der Kaskade an belegt sind; das gilt auch für die BM, die erst für Folgephasen der Kaskade benötigt werden. Deadlock–Vermeidung mit kaskadenförmigem BM–Einsatz kann folglich die Effizienz des Einsatzes der BM und damit die **Effizienz des Systems beeinträchtigen**. Das gilt insbesondere in den Fällen, in denen die Kaskaden entarten (die BM werden zusammen angefordert und zusammen freigegeben). Mit differenzierteren Verfahren zur Deadlock–Vermeidung können diese Effizienzbeeinträchtigungen abgeschwächt werden.

4.2.5.2 Geordneter BM–Einsatz

Deadlock–Vermeidung für das System $\mathcal{R}$ durch **Nutzung** der Notwendigkeit **der Bedingung ND4** kann dadurch erreicht werden, daß für die BM–Klasse von $\mathcal{R}$ eine lineare Ordnung festgelegt wird und die BM–Anforderungen und –Freigaben der Prozesse dieser Ordnung entsprechend reglementiert werden. Damit ergibt sich $\mathcal{R}$ als **System mit geordnetem BM–Einsatz** wie folgt:

- Die BM–Klassen werden mit $1, 2, \ldots, l$ linear geordnet.
- Für die BM–Anforderungen der Prozesse gelten: Wenn $p \in P$ BM der Klasse $j \in \{1, \ldots, l\}$ belegt, darf p zusätzlich BM der Klassen $i \in \{1, \ldots, l\}$ mit $i < j$ anfordern.

Wenn $\mathcal{R}$ den angegebenen Regeln entsprechend als System mit geordnetem BM–Einsatz konstruiert ist, ist damit offenbar gewährleistet, daß der Belegungs–Anforderungs–Graph $\mathcal{G}_{\mathcal{R}}(z)$ jedes BM–zulässigen Zustandes z von $\mathcal{R}$ keinen Zyklus enthält.
Die angegebenen Regeln erfordern, daß die BM–Klassen von $\mathcal{R}$ nach ihrer Wichtigkeit (von hoch zu niedrig) für die Effizienz des Systems geordnet werden; sie legen fest, daß die BM ihrem Gewicht entsprechend mit hoher bis niedriger Dynamik verwaltet werden. Abbildung 4.7 veranschaulicht diesen Zusammenhang zwischen den Gewichten und der Verwaltungsdynamik.

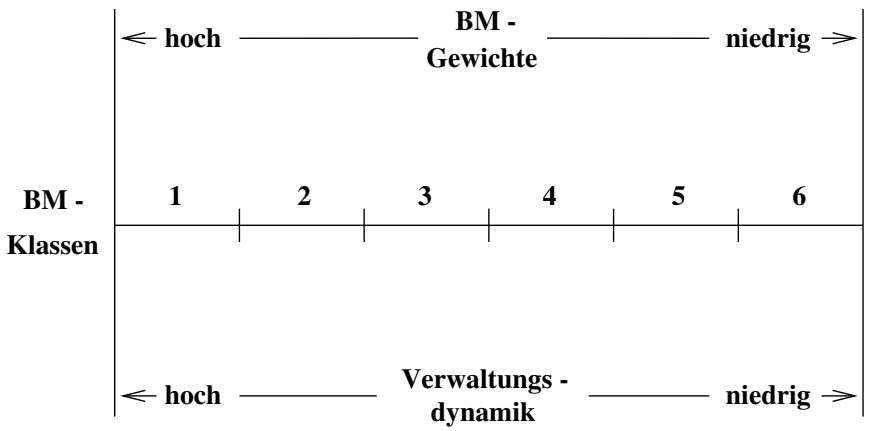


Abbildung 4.7: Gewichte und Verwaltungsdynamik eines Systems mit geordnetem BM–Einsatz

Für ein System mit geordnetem BM–Einsatz wird größere Effizienz des BM–Einsatzes durch mehr Flexibilität bei der Verwaltung der für alle Prozesse wichtigen BM erreicht. Die Charakteristika der Einsatzmöglichkeiten der FXBM für die Berechnungen der Prozesse, die in Kapitel 4.2.5.1 erklärt sind, haben natürlich auch hier ihre Gültigkeit; die Festlegung der BM–Gewichte muß dem Rechnung tragen.

4.2.6 Deadlock–Verhinderung

Das System $\mathcal{R} = (w, P, \alpha, H, X, Y)$ sei BM–zulässig. Für $\mathcal{R}$ seien jetzt BM–zulässige Belegungs–Anforderungs–Zustände nicht notwendig BM–sicher. Im folgenden werden Verfahren angegeben, mit denen Deadlock–Zustände für $\mathcal{R}$ verhindert werden.
Mit Verfahren zur **Deadlock–Verhinderung** für $\mathcal{R}$ wird durch Kontrollen sämtlicher BM–Zuteilungen an die Prozesse gewährleistet, daß alle tatsächlich auftretenden Belegungs–Anforderungs–Zustände BM–sicher sind. Nach Kapitel 4.2.2 ist der Anfangszustand z^0 des BM–zulässigen Systems $\mathcal{R}$ BM–sicher. Seien $\eta = \varepsilon^1 \ldots \varepsilon^k$ eine unvollständige, BM–zulässige Ereignisspur von $\mathcal{R}$ und $\zeta(z^0, \eta) = z^0 \ldots z^k$ die z^0 und η zugeordnete Zustandsspur. Zudem sei z^k BM–sicher. In dieser Situation bedeutet Deadlock–Verhinderung:

- Als Folgeereignis von η ist ein in z^k BM–zulässiges ε^{k+1} so auszuwählen, daß der zugeordnete Folgezustand z^{k+1} BM–sicher ist.

Nach der Definition BM–sicherer Zustände ist hierfür die Existenz einer vollständigen, BM–zulässigen Ereignisspur η' mit dem Präfix $\eta\varepsilon^{k+1}$ nachzuweisen.
Die Konstruktion einer entsprechenden Ereignisspur ist i.a. aufwendig. Der zu leistende **Aufwand** wird mit den beiden im folgenden angegebenen Aussagen reduziert.

- Wenn ε^{k+1} ein Terminierungsereignis ist, ist z^{k+1} BM–sicher.

Die Gültigkeit dieser Aussage ergibt sich daraus, daß mit Terminierungsereignissen keine zusätzlichen BM belegt werden. Wenn also r^k und r^{k+1} die Vektoren der freien BM in den Zuständen z^k und z^{k+1} sind, gilt $r^k \leq r^{k+1}$ und die BM–Sicherheit von z^{k+1} folgt daraus, daß z^k nach Voraussetzung sicher ist.
Kontrollen sind damit noch für alle Initiierungsereignisse notwendig. Es reicht jedoch aus, die Existenz einer vollständigen, BM–zulässigen Ereignisspur jeweils für den Prozeß nachzuweisen, für den ein Initiierungsereignis eintritt.

- ε^{k+1} sei Initiierungsereignis des Prozesses p. Wenn $\eta'' = \eta\varepsilon^{k+1} \ldots \varepsilon^l$ eine BM–zulässige Ereignisspur von $\mathcal{R}$ so ist, daß
  - die $\varepsilon^{k+1} \ldots \varepsilon^l$ Ereignisse des Prozesses p sind und
  - η'' eine vollständige Ereignisspur von p enthält,

  dann ist der ε^{k+1} zugeordnete Zustand z^{k+1} BM–sicher.

Die Gültigkeit dieser Aussage ergibt sich wie folgt: Sei $r^0 \ldots r^l$ die Folge der Vektoren der freien BM gemäß der Ereignisspur η''. Mit ε^l gibt der Prozeß p alle bis dahin belegten BM frei, so daß $r^k \leq r^l$ gilt. Weil z^k nach Voraussetzung BM–sicher ist, ist damit auch z^{k+1} sicher.
Auch mit den Reduktionen, die sich aus diesen Aussagen ergeben, bleiben die Kontrollen zur Deadlock–Verhinderung aufwendig: Für jedes Initiierungsereignis ist eine der angegebenen Ereignisspur–Konstruktionen erforderlich. Diese Konstruktionen setzen voraus, daß die BM–Anforderungen und –Freigaben der Prozesse im voraus bekannt sind. Mit dem im folgenden angegebenen Verfahren ist Deadlock–Verhinderung auch dann möglich, wenn für die Prozesse im voraus lediglich die maximalen BM–Anforderungen bekannt sind.

4.2.6.1 Das Bankier–Verfahren

Das im folgenden angegebene, seit langem bekannte und häufig angewandte Verfahren zur Deadlock–Verhinderung geht auf A.N. Habermann zurück. Es macht davon Gebrauch, daß für das System $\mathcal{R}$ mit $H = (h(p) \mid p \in P)$ wenigstens die maximalen BM–Anforderungen der Prozesse im voraus bekannt sind, und führt Deadlock–Verhinderung auf Deadlock–Erkennung für Pseudozustände zurück.
Zur Erklärung des Verfahrens wird die oben angegebene Situation mit η und $\zeta(z^0, \eta)$ betrachtet. Das Ereignis ε^{k+1} sei im Zustand z^k BM–zulässig und ε^{k+1} sei ein Ereignis des Prozesses p.
Mit dem Ereignis ε^{k+1} müssen die von p im Zustand $z^k = (B^k, A^k)$ mit $a^k(p)$ gestellten Anforderungen erfüllt werden. Auf dieser Grundlage wird der Pseudozustand $\widetilde{z}^{k+1} = (\widetilde{B}^{k+1}, \widetilde{A}^{k+1})$ so definiert, daß mit ihm alle Prozesse ihre maximalen Anforderungen stellen. Für $\widetilde{z}^{k+1}$ gelten:

- $\widetilde{b}^{k+1}(p) := b^k(p) + a^k(p)$ und $\widetilde{b}^{k+1}(q) := b^k(q)$ für alle $q \in P - \{p\}$,
- $\widetilde{a}^{k+1}(q) := \begin{cases} h(p) - \widetilde{b}^{k+1}(p) & , \text{ falls } q = p \\ h(q) - b^k(q) & , \text{ falls } q \neq p \text{ und } a^k(q) + b^k(q) \neq 0 \\ 0 & , \text{ falls } q \neq p \text{ und } a^k(q) + b^k(q) = 0 \end{cases}$

Im Pseudozustand $\widetilde{z}^{k+1}$ sind also für p die für ε^{k+1} gestellten Anforderungen erfüllt; die Prozesse $q \in P - \{p\}$ belegen weiter die in z^k belegten BM; alle Prozesse, für die η keine vollständige Ereignisspur enthält, stellen ihre Maximalanforderungen.
Deadlock–Verhinderung ergibt sich mit folgender Aussage:

- Wenn der Pseudozustand $\widetilde{z}^{k+1}$ kein Deadlock–Zustand ist, ist der dem Ereignis ε^{k+1} zugeordnete Zustand z^{k+1} BM–sicher.

Zum Nachweis der Gültigkeit dieser Aussage wird o.B.d.A. angenommen, daß η für keinen der Prozesse von $\mathcal{R}$ eine vollständige Ereignisspur enthält. $\widetilde{z}^{k+1}$ sei kein Deadlock–Zustand. Dann können von $\widetilde{z}^{k+1}$ ausgehend für alle Prozesse von $\mathcal{R}$ die in $\widetilde{z}^{k+1}$ gestellten Anforderungen sukzessive erfüllt werden. Mit $|P| \triangleq n$ seien die Prozesse von $\mathcal{R}$ nummeriert, und $1, 2, \ldots, n$ sei die Reihenfolge, in der die Anforderungen der Prozesse sukzessive erfüllt werden können. Mit den Suffixen $\eta_1, \ldots, \eta_n$ der Prozesse $1, 2, \ldots, n$ ist $\eta\varepsilon^{k+1}\eta_1, \ldots, \eta_n$ eine vollständige, BM–zulässige Ereignisspur von $\mathcal{R}$ mit dem Präfix $\eta\varepsilon^{k+1}$; damit ist der ε^{k+1} zugeordnete Zustand z^{k+1} BM–sicher.

⌋

Die Anwendung des Verfahrens erfordert die Analyse des Pseudozustands $\widetilde{z}^{k+1}$; das erfolgt mit Algorithmen zur Deadlock–Erkennung, die in Kapitel 4.2.7 behandelt werden. Diese Analyse ist zur Deadlock–Verhinderung immer dann erforderlich, wenn ein Initiierungsereignis mit BM–Belegungen eintreten soll. Der oben geführte Nachweis lehrt, daß dieser Analyse–Aufwand dann reduziert werden kann, wenn die konstruierte vollständige BM–zulässige Ereignisspur beibehalten wird. Das entspricht jedoch einer Sequentialisierung der weiteren Berechnungen der Prozesse von $\mathcal{R}$ und damit i.a. einer Effizienz–Minderung.

4.2.6.2 Optimistische Deadlock–Verhinderung mit vorbeugenden Maßnahmen

Systematische Deadlock–Verhinderung für das System $\mathcal{R}$ gewährleistet, daß die Prozesse von $\mathcal{R}$ ihre Berechnungen vollständig ausführen können. Dazu ist, wie sich gezeigt hat, ein hoher Aufwand zu leisten, der insbesondere dann fragwürdig ist, wenn die Gefährdung von

$\mathcal{R}$ durch Deadlock–Zustände klein ist. In diesem Fall sind Verfahren von Interesse, die erst dann hohen Aufwand verursachen, wenn Gefährdungen durch Deadlock–Zustände auftreten.

Die naheliegende Vorgehensweise, auf Maßnahmen zur Deadlock–Verhinderung zu verzichten, die Belegungs–Anforderungs–Zustände sporadisch auf Deadlock zu analysieren und bei auftretenden Deadlock–Zuständen die Berechnungen der in den Deadlock–Zuständen blockierten Prozesse abzubrechen, wurde schon in Kapitel 4.1.2 als unakzeptabel abgelehnt, weil sie Schaden für die Benutzer von $\mathcal{R}$ verursachen und die Konsequenzen mangelhafter Koordinierung auf die Benutzer abschieben. Die Vorgehensweise kann jedoch um vorbeugende Maßnahmen und um die Rücksetzung von Prozeßberechnungen im Deadlock–Fall so erweitert werden, daß sie dann geeignet ist, wenn die Deadlock–Gefährdung von $\mathcal{R}$ klein ist.

Für das Verständnis der optimistischen Verfahren zur Deadlock–Verhinderung mit vorbeugenden Maßnahmen ist wesentlich, daß für das System $\mathcal{R}$ zwischen **äußeren und inneren Eigenschaften** unterschieden werden kann und unterschieden wird. Die äußeren Eigenschaften sind die an der Schnittstelle von $\mathcal{R}$ beobachtbaren und damit von außen und für Benutzer sichtbaren Eigenschaften. Die inneren Eigenschaften sind die unter der Schnittstelle von $\mathcal{R}$ verborgenen und damit von außen und für Benutzer nicht sichtbaren Eigenschaften. Mit der Unterscheidung zwischen äußeren und inneren Eigenschaften von $\mathcal{R}$ wird es möglich, daß Deadlock–Zustände zwar auftreten, aber von außen unsichtbar und wirkungslos bleiben. Auf dieser Grundlage wird zur **Deadlock–Verhinderung** wie folgt verfahren:

- Die Anfangszustände der Prozesse werden gespeichert.
- Die Berechnungen der Prozesse werden versuchsweise und auf innere Wirkungen beschränkt ausgeführt; das wird mit versuchsweisen, BM–zulässigen Ereignisspuren von $\mathcal{R}$ und zugeordneten Zustandsspuren beschrieben. Die (versuchsweisen) Belegungs–Anforderungs–Zustände werden sporadisch auf Deadlocks analysiert.
- Für einen Prozeß, dessen (versuchsweise) Berechnungen vollständig ausgeführt werden können, werden diese (nachträglich) bestätigt; ihre Wirkungen werden — soweit festgelegt — nach außen sichtbar.
- Wenn ein Deadlock–Zustand erkannt wird und $Q \subseteq P$ die Menge der in diesem blockierten Prozesse ist, wird mit den $q \in Q$ wie folgt verfahren:
  - die q–Ereignisse werden annulliert und die von q belegten BM werden freigegeben;
  - q wird in den Anfangszustand zurückgesetzt und die q–Berechnungen werden nochmals (versuchsweise) ausgeführt.

⌋

Wenn die Berechnungen des Systems $\mathcal{R}$ mit diesem Verfahren koordiniert ausgeführt werden, wird das **Verhalten** von $\mathcal{R}$ mit BM–zulässigen Ereignisspuren und zugeordneten Zustandsspuren beschrieben. Die Ereignisse und Zustände sind mit versuchsweise sowie mit annulliert oder bestätigt zu markieren. Eine Ereignisspur beschreibt eine vollständige Ausführung der Berechnungen von $\mathcal{R}$, wenn sie für jeden Prozeß $p \in P$ eine vollständige, bestätigte Ereignisspur enthält. Die Projektion dieser Spur auf die bestätigten Ereignisse beschreibt das nach außen sichtbare Verhalten von $\mathcal{R}$; es entspricht der Ausführung der Berechnungen mit **Deadlock–Verhinderung**. Die Ereignisspur mit annullierten und bestätigten Ereignissen beschreibt das innere Verhalten von $\mathcal{R}$ mit evtl. aufgetretenen Deadlock–Zuständen, die jedoch nach außen unsichtbar und wirkungslos bleiben; das ist **Deadlock–Toleranz**. Das

Verfahren ist **optimistisch**, weil es davon ausgeht, daß Deadlock–Zustände selten auftreten und wiederholtes Versuchen schließlich zum Erfolg führt; dieser Erfolg ist jedoch nicht gewährleistet.
Das angegebene Verfahren nutzt wesentlich aus, daß die für $\mathcal{R}$ relevanten BM **sämtlich FXBM** sind. Aus ihren Charakteristika ergibt sich, daß die bei erkannten Deadlock–Zuständen notwendigen Ereignis–Annullierungen und BM–Freigaben mit geringem Aufwand durchführbar sind. Aus ihnen ergibt sich auch die Chance dafür, daß ein Prozeß durch Zurücksetzen in seinen Anfangszustand schließlich seine Berechnungen vollständig und mit der spezifizierten Wirkung ausführen kann. Wesentlich dabei ist, daß die Wirkungen bereits (versuchsweise) ausgeführter Berechnungen annulliert werden können.
Für das Verfahren ist festgelegt, daß die versuchsweisen Berechnungen auf **innere Wirkungen** beschränkt ausgeführt werden. Dazu ist erforderlich, daß mit diesen Berechnungen keine Ein– oder Ausgaben von oder zu der Umgebung von $\mathcal{R}$ erfolgen. Wenn Ein– oder Ausgaben notwendig sind, muß das Verfahren so erweitert werden, daß auch diese Wirkungen annulliert werden können.
Das angegebene Verfahren ist ein Musterbeispiel für die Nutzung der Unterscheidung zwischen äußeren und inneren Eigenschaften des Systems $\mathcal{R}$. Sie ist die Voraussetzung dafür, daß auftretende Deadlock–Zustände nach außen unsichtbar und wirkungslos bleiben. Entsprechend verallgemeinerte Verfahren können auch dazu genutzt werden, daß Fehler und Fehlverhalten nach außen unsichtbar und wirkungslos bleiben; das sind Verfahren zur **Fehlertoleranz**.

4.2.7 Deadlock–Erkennung und –Behebung

In den vorangehenden Kapiteln ergab sich wiederholt die Notwendigkeit, BM–zulässige Belegungs–Anforderungs–Zustände von $\mathcal{R}$ auf Deadlocks zu analysieren. Entsprechende Analysen sind als Notmaßnahmen dann erforderlich, wenn für $\mathcal{R}$ weder Verfahren zur Deadlock–Vermeidung noch Verfahren zur Deadlock–Verhinderung angewandt werden. Sie sind, wie sich gezeigt hat, jedoch auch bei Verfahren zur Deadlock–Verhinderung erforderlich.
Für das BM–zulässige System $\mathcal{R} = (w, P, \alpha, H, X, Y)$ wird im folgenden zunächst ein Algorithmus zur Deadlock–Erkennung angegeben; anschließend wird kurz auf die Behebung eines Deadlock–Zustandes eingegangen.
Für den Algorithmus zur **Deadlock–Erkennung** seien z^0 der Anfangszustand von $\mathcal{R}$, $\eta = \varepsilon^1 \dots \varepsilon^k$ eine BM–zulässige Ereignisspur von $\mathcal{R}$ und $\zeta(z^0, \eta) = z^0 \dots z^k$ die z^0 und η zugeordnete Zustandsspur. Der Zustand $z^k = (B^k, A^k)$ soll auf Deadlock analysiert werden. Das Verfahren hierfür geht von der in Kapitel 4.2.4 angegebenen Definition eines Deadlock–Zustandes mit

$$a^k(q) \not\leq r^k + \sum_{p \in P-Q} b^k(p)$$

für die $q \in Q$ einer nichtleeren Teilmenge $Q \subseteq P$ der Prozesse, die sich in z^k endlos blockieren, aus; dabei ist r^k der Vektor der freien BM in z^k. Diese Definition liefert die Bedingung für die erforderlichen Tests: r ist der Vektor der jeweils freien BM, wobei initial $r = r^k$ gilt.

- Wenn für einen Prozeß $p \in P$ die Bedingung $a^k(p) \leq r$ erfüllt ist, kann das Folgeereignis gemäß η für p eintreten; p ist nicht im Deadlock blockiert, und entsprechend tragen die von p belegten BM nicht zum Deadlock bei; für die weiteren Tests stehen demnach $r := r + b^k(p)$ BM frei zur Verfügung.

- Wenn für einen Prozeß $p \in P$ jedoch $a^k(p) \not\leq r$ gilt, kann p im Deadlock blockiert sein;

ein Deadlock–Zustand liegt dann vor, wenn dies für alle Prozesse einer nichtleeren Menge gilt.

In Abbildung 4.8 ist ein Programm zur Deadlock–Erkennung einschließlich der benötigten Typen und Hilfsfunktionen angegeben.
Mit $|P| = n$ sind die Prozesse mit Indizes aus $\{1, \ldots, n\}$ identifiziert. m ist die Anzahl der BM–Klassen. Für die Hilfsfunktionen gelten

- `kgl(x,y) = true` $\Leftrightarrow x \leq y$ und
- `vad(x,y) = x+y`.

Der mit `qd_erkennung` zu analysierende Zustand wird mit der Belegungsmatrix b, der Anforderungsmatrix a und dem Vektor r der freien BM eingegeben; der Eingabewert für `index` ist beliebig. Für die mit `qd_erkennung` berechneten Ergebnisse gelten:

- der analysierte Zustand ist Deadlock–Zustand $\Leftrightarrow d = \mathit{true}$;
- $Q = \{i \in \{1, \ldots, n\} \mid \texttt{index}(i) = \mathit{true}\}$ ist die Menge der im Deadlock–Zustand blockierten Prozesse.

Als Aufwand für die Analyse eines Zustandes ist die Anzahl der skalaren arithmetischen Vergleiche ausschlaggebend; die Zeitkomplexität im ungünstigsten Fall ist folglich $O(mn^2)$. Dieser mit der Anzahl der Prozesse quadratisch wachsende, hohe Aufwand kann auf in n lineare Komplexität reduziert werden, wenn die Anforderungen der Prozesse gemäß a geordnet werden. Auch in diesem Fall bleibt, daß der zur Deadlock–Erkennung zu leistende Aufwand hoch ist.
Sei jetzt z^k Deadlock–Zustand mit der Menge Q der in z^k blockierten Prozesse. Dann sind Maßnahmen zur **Deadlock–Behebung** erforderlich; mit diesen ist $\mathcal{R}$ von z^k in einen BM–sicheren Zustand $\tilde{z}^k$ zu überführen.
In Kapitel 4.1.2 wurde bereits darauf hingewiesen, daß mit jedem Deadlock–Zustand Schaden entstanden ist; Maßnahmen zur Deadlock–Behebung sind also Maßnahmen zur Schadensbegrenzung. Das gilt, nachdem festgelegt ist, daß die relevanten BM von $\mathcal{R}$ sämtlich nichtunterbrechbar sind, und nachdem in Kapitel 4.2.6.2 die vorbeugenden Maßnahmen optimistischer Deadlock–Verhinderung erklärt sind. Damit bleiben als Maßnahmen zur Deadlock–Behebung für $\mathcal{R}$ in z^k, der Abbruch der Berechnungen und die (Zwangs–)Freigabe der belegten BM der Prozesse von Q; das hat zur Folge, daß diese Prozesse ihre Berechnungen nicht vollständig ausführen können. Dem entsprechend sind zur Behebung des Deadlock–Zustandes z^k folgende Maßnahmen erforderlich:

- Berechnung der Teilmengen $Q' \subseteq Q$ für die gilt: durch Abbruch der Berechnungen und Freigabe der belegten BM der $q \in Q'$ kann $\mathcal{R}$ von z^k in einen BM–sicheren Zustand $\tilde{z}^k(Q')$ überführt werden.
- Auswahl einer Teilmenge $Q'' \subseteq Q$, mit der durch Abbruch der Berechnungen und BM–Freigabe die Überführung in einen BM–sicheren Zustand möglich und der dadurch entstehende Schaden möglichst klein ist.
- Mit dieser Menge Q'' die Überführung von $\mathcal{R}$ vom Deadlock–Zustand z^k in einen BM–sicheren Zustand $\tilde{z}^k(Q'')$.

```
TYPE menge IS ARRAY (1..n) OF boolean;    n=|P|
TYPE vektor IS ARRAY (1..m) OF natural;
TYPE matrix IS ARRAY (1..n) OF vektor;

FUNCTION kgl (x,y:IN vektor) RETURN boolean;
FUNCTION vad (x,y:IN vektor) RETURN vektor;

PROCEDURE qd_erkennung (b,a:IN matrix; r:IN vektor;
          d:OUT boolean; index:IN OUT menge) IS

    zaehler :natural :=index'LAST(1);
    rest    :vektor  :=r;
    weiter  :boolean :=true;

BEGIN  FOR i IN index'RANGE(1)
         LOOP index(i):=true; END LOOP;
       WHILE weiter
         LOOP weiter:=false;
              FOR i IN b'RANGE(1)
                LOOP IF index(i) AND kgl(a(i),rest)
                         THEN index(i):=false;
                              zaehler:=zaehler-1;
                              rest:=vad(rest,b(i));
                              weiter:=true;
                              EXIT;
                     END IF;
                END LOOP;
         END LOOP;
       d:=zaehler>0;
END qd_erkennung;
```

Abbildung 4.8: Ein Programm zur Deadlock–Erkennung

Eine Möglichkeit $\mathcal{R}$ von z^k in einen BM–sicheren Zustand $\tilde{z}^k$ zu überführen, besteht darin, die Berechnungen der $q \in Q$ abzubrechen und die BM dieser Prozesse freizugeben, wobei zusätzlich die BM–Sicherheit des resultierenden Zustandes $\tilde{z}^k(Q)$ zu gewährleisten ist. Es ist klar, daß diese Vorgehensweise sowohl Schaden als auch Aufwand verursacht. Damit wird erneut deutlich, daß Beschränkungen auf Deadlock–Erkennung und –Behebung Beschränkungen auf Notmaßnahmen sind.

4.3 Erzeuger–Verbraucher–Systeme

Parallele Systeme, deren kritische BM höchstens nichtunterbrechbare, frei exklusiv benutzbare BM sind, sind die Systeme, für welche die Aufgaben der BM–Verwaltung und der Koordinierung relativ einfach gelöst werden können. Für Systeme mit einmal benutzbaren BM oder mit wiederholt und bedingt benutzbaren BM sind die Lösungen dieser Aufgaben schwieriger, weil kompliziertere Abhängigkeiten zwischen den Prozessen der Systeme berücksichtigt werden müssen. Im folgenden werden als Beispiel für solche Systeme noch einfache Erzeuger–Verbraucher–Systeme behandelt. Das sind Systeme, deren Prozesse mittelbar kooperieren, indem sie über gemeinsam benutzte Puffer Nachrichten kommunizieren.
Das **Modellsystem** $\mathcal{R} = (D, P, \alpha, X, Y)$ für die zu behandelnde Aufgabe ist das in Kapitel 3.3.2.2 betrachtete mit dem dort konstruierten Nachrichtenpuffer s der Kapazität $kap(s)$ als kritischem BM. Der Puffer s ist in Kapitel 3.3.2.2 bereits so konstruiert, daß die Konsistenz bei Ausführung der auf s definierten Operationen *ein* zum Einfügen einer Nachricht und *aus* zum Entnehmen einer Nachricht gewährleistet ist. Der Puffer s ist jetzt für die Berechnungen der Prozesse von $\mathcal{R}$ ein bedingt exklusiv benutzbares BM, für das BM–Sicherheit zu gewährleisten ist.
Für das System $\mathcal{R}$ wird vorausgesetzt, daß seine Prozeßmenge P zwei nichtleere, disjunkte Teilmengen, nämlich

- die Menge *ERZ* der **Erzeuger** mit $erz = |ERZ|$ und
- die Menge *VEB* der **Verbraucher** mit $veb = |VEB|$

von Nachrichten enthält. Die Prozesse aus *ERZ* erzeugen Nachrichten: sie berechnen Nachrichten und fügen diese dann mit $s.ein$ in den Puffer s ein. Die Prozesse aus *VEB* verbrauchen Nachrichten: sie entnehmen mit $s.aus$ Nachrichten aus dem Puffer s und führen dann mit diesen ihre Berechnungen weiter. Mit diesen vereinfachenden Voraussetzungen ist gefordert, daß die Prozesse von $\mathcal{R}$ ihre Berechnungen vollständig ausführen können. Zur vollständigen Ausführung der Berechnungen gehört, daß die Nachrichten, die erzeugt werden, auch verbraucht werden. Das System $\mathcal{R}$ ist demnach **BM–zulässig**, wenn

- für ein $kom \in \mathbb{N}$ die Erzeuger *kom* Nachrichten erzeugen und die Verbraucher *kom* Nachrichten verbrauchen sowie
- die Prozesse von $\mathcal{R}$ ihre Berechnungen BM–zulässig bzgl. des Puffers s vollständig ausführen können.

Wenn die notwendige Bedingung, daß *kom* Nachrichten erzeugt und verbraucht werden, erfüllt ist, muß für $\mathcal{R}$ noch gezeigt werden, daß für die Erzeuger und die Verbraucher keine endlosen Blockierungen — Deadlock–Zustände — bzgl. s auftreten.
Der Nachweis der BM–Sicherheit für $\mathcal{R}$ erfolgt, indem die Eigenschaften ausgewertet werden, mit denen der Puffer s in Kapitel 3.3.2.2 konstruiert ist. Diese Eigenschaften sind i.w. mit

den Semaphoren *voll* und *leer* festgelegt; die interessierenden Eigenschaften von $\mathcal{R}$ ergeben sich demnach aus Gleichungen und Ungleichungen für die Kenngrößen dieser Semaphore.
Für die im folgenden abgeleiteten Gleichungen und Ungleichungen werden die in Kapitel 3.3.2.2 eingeführten Bezeichnungen benutzt. Die Aussagen gelten jeweils für den Zustand, der mit einer BM–zulässigen Ereignisspur η von $\mathcal{R}$ erreicht wird.
Die folgenden Gleichungen beschreiben die Werte der Zählvariablen $m(voll)$ und $m(leer)$ der Semaphore; sie ergeben sich aus den festgelegten Initialwerten und den Semaphor–Zugriffsoperationen.

(1) $m(voll) = kap(s) - \#voll.pro + \#voll.epi$;

(2) $m(leer) = -\#leer.pro + \#leer.epi$.

Durch Addition von (1) und (2) ergibt sich

(3) $m(voll) + m(leer) - kap(s) = -\#voll.pro + \#voll.epi - \#leer.pro + \#leer.epi$.

Die Operationen *ein* und *aus* auf dem Puffer s sind jeweils mit Prolog– und Epilog–Operationen geklammert.
Aus der Klammerung von *ein* und daraus, daß die Erzeuger *ein* ausführen, ergibt sich

$$0 \leq \#voll.pro \leq \#leer.epi + erz$$

und daraus folgt

(4) $0 \leq \#voll.pro - \#leer.epi \leq erz$.

Aus der Klammerung von *aus* und daraus, daß die Verbraucher *aus* ausführen, ergibt sich

$$0 \leq \#leer.pro \leq \#voll.epi + veb$$

und daraus folgt

(5) $0 \leq \#leer.pro - \#voll.epi \leq veb$.

Durch Multiplikation von (4) und (5) mit -1 und anschließende Addition ergibt sich

(6) $-erz - veb \leq -\#voll.pro + \#voll.epi - \#leer.pro + \#leer.epi \leq 0$.

Mit (3) und (6) ergibt sich für den **Wertebereich der Summe der Zählvariablen der Semaphore** *voll* **und** *leer*

(7) $kap(s) - erz - veb \leq m(voll) + m(leer) \leq kap(s)$.

Weil nach Voraussetzung $kap(s) \geq 1$ gilt, folgt aus (7)

(8) $-erz - veb < m(voll) + m(leer) \leq kap(s)$.

Die Ungleichung (8) sagt insbesondere, daß nicht alle Erzeuger und alle Verbraucher gleichzeitig bzgl. s blockiert sein können.
Für das Semaphor *voll*, mit dem Erzeuger blockiert werden können, gilt

(9) $-erz \leq m(voll) \leq kap(s)$.

$m(voll) = -erz$ sagt, daß alle Erzeuger bzgl. *voll* blockiert sind. Wenn das der Fall ist, gilt nach (8) mit $-veb < m(leer)$, daß es nicht bzgl. *leer* blockierte Verbraucher gibt.
Für das Semaphor *leer*, mit dem Verbraucher blockiert werden können, gilt

(10) $-veb \leq m(leer) \leq kap(s)$.

$m(leer) = -veb$ sagt, daß alle Verbraucher bzgl. *leer* blockiert sind. Wenn das der Fall ist, gilt nach (8) mit $-erz < m(voll)$, daß es nicht bzgl. *voll* blockierte Erzeuger gibt.
Aus Ungleichung (8) ergibt sich also, daß die BM–zulässigen Zustände von $\mathcal{R}$ auch **BM–sichere** Zustände sind. Wenn also, wie vorausgesetzt, die Prozesse gemäß *ERZ kom* Nachrichten erzeugen und die Prozesse gemäß *VEB kom* Nachrichten verbrauchen, können die Berechnungen der Prozesse von $\mathcal{R}$ vollständig ausgeführt werden.
Für die angegebenen Aussagen zur BM–Sicherheit von $\mathcal{R}$ fällt auf, daß sie i.w. die für den Puffer s passend festgelegten Eigenschaften nutzen. Das ist eine Konsequenz der vereinfachenden Voraussetzungen für das System $\mathcal{R}$: Mit den disjunkten Mengen *ERZ* und *VEB* ergibt sich, daß die Erzeuger gemeinsam die Nachrichten erzeugen, welche die Verbraucher gemeinsam verbrauchen; der Puffer s gewährleistet, daß die Nachrichten konsistent kommuniziert werden können; wenn also die Nachrichtenbilanz ausgeglichen ist, können die Prozesse ihre Berechnungen vollständig ausführen.
Die Situation ändert sich wesentlich, wenn die Erzeuger– und Verbraucher–Prozeßmengen nicht mehr disjunkt sind. Dann ist nicht mehr der Puffer s das ausschlaggebende BM; es wird vielmehr wesentlich, daß die Berechnungen der kooperierenden Prozesse beim Erzeugen und Verbrauchen von Nachrichten aufeinander abgestimmt sind. Das wird insbesondere dann relevant, wenn die Prozesse eines Systems direkt mittels Nachrichtenkommunikation kooperieren.

4.4 Zusammenfasssung und Einordnung

Die in Kapitel 4 behandelten Themen der Betriebsmittelverwaltung und der Berechnungskoordinierung sind von zentraler Bedeutung für jedes Betriebssystem; sie sind wesentlicher Bestandteil der Managementaufgaben, die ein Betriebssystem zu erfüllen hat.
Ausgangsbasis für die Behandlung dieser Aufgaben waren die Modellsysteme mit sequentiellen Prozessen und die von ihnen abgeleitete Grundstruktur für Realisierungen paralleler Systeme von Kapitel 3. Ein Modellsystem liefert die Spezifikation der Berechnungen, die ein System ausführen soll, und damit die grundlegende Anforderung an die Koordinierung, die sagt: Alle Prozesse des Systems müssen ihre Berechnungen vollständig ausführen können. Zur Ausführung ihrer Berechnungen müssen den Prozessen die benötigten Systemkomponenten zur Verfügung stehen. Der Grundstruktur für Realisierungen entsprechend sind diese Komponenten den Prozessen, die existieren, bereits zu einem Teil als private, d.h. exklusiv benutzbare Komponenten zugeordnet; zum anderen Teil sind sie potentiell kritische Komponenten, um die die Prozesse konkurrieren. Diese kritischen Komponenten liefern die wesentlichen Anforderungen, die mit der Koordinierungsaufgabe gestellt sind:

- Zum einen ergeben sich generelle Abhängigkeiten zwischen den Prozessen und den kritischen Komponenten eines Systems. Sie schränken die Eigenschaften, die Prozesse haben können, ein und sagen, daß die Anforderungen, die Prozesse stellen, erfüllbar sein müssen. Das stellt klar, daß die Prozesse den für ein System festgelegten Rahmenbedingungen entsprechen müssen.

- Zum anderen ergibt sich unter der Voraussetzung, daß alle Prozesse erfüllbare Anforderungen stellen, daß die jeweils gestellten Anforderungen erfüllt und allen Prozessen die Möglichkeit, ihre Berechnungen vollständig auszuführen, gewährleistet werden müssen.

Voraussetzung dafür, daß erfüllbare Anforderungen gestellt werden, und dafür, daß gestellte Anforderungen erfüllt werden können, sind differenzierte und qualifizierte Kenntnisse von den Systemkomponenten, die zur Verfügung stehen. Weil viele und vielfältige Komponenten zur Verfügung stehen, ergibt sich zunächst die Notwendigkeit von effizienten Verfahren zur Registrierung und zur Verwaltung dieser Komponenten. Dazu werden Komponenten zu BM vergröbert sowie zu BM–Arten und BM–Klassen zusammengefaßt; damit werden Registrierung und Verwaltung vereinfacht. Als wirksamere Maßnahme zur Steigerung der Effizienz bietet sich jedoch an, mit gegebenen Komponenten zielgerichtet geeignetere Komponenten zu konstruieren und diese den Prozessen als BM zur Verfügung zu stellen. Diese kreativen Maßnahmen können analog zur Konstruktion von Datenobjekten iteriert angewandt werden. Iterierte, systematische Anwendungen dieser Maßnahmen sind das ausschlaggebende Instrumentarium zur Erfüllung der Managementaufgaben, die ein Betriebssystem hat. Von diesem Management wird demnach folgendes gefordert:

- Die Festlegung von Konzepten für BM, die für Prozesse und für effiziente Ausführungen der Berechnungen der Prozesse geeignet sind.
- Die Bereitstellung von effektiv einsetzbaren BM, die diesen Konzepten entsprechen.
- Die Verwaltung der verfügbaren BM für die Prozesse so, daß diese ihre Berechnungen effizient und vollständig ausführen können.

In Kapitel 4.2 ist dieses Management einschließlich der Verfahren, die zur Lösung der gestellten Aufgaben anzuwenden sind, für die einfachen, aber wichtigen FXBM ausführlich erklärt. Die Verfahren sind charakteristisch und als Ausgangsbasis für Verallgemeinerungen geeignet. Die FXBM sind wichtig; sie sind jedoch, wie erklärt, lediglich eine Unterart der WBM, die für ein paralleles System benötigt werden. Für eine weitere Unterart der WBM, nämlich die BXBM, ist in Kapitel 4.3 lediglich an einem wichtigen Beispiel erklärt, welche Anforderungen an das Management gestellt werden, und mit welchen Maßnahmen diese Anforderungen erfüllt werden können. Charakteristisch für die dort betrachteten Erzeuger–Verbraucher–Systeme sind die Abhängigkeiten zwischen Prozessen, die dadurch entstehen, daß sie gemeinsam ein BXBM benötigen. Als Konsequenz dieser Abhängigkeiten waren detaillierte Analysen zur Gewährleistung der geforderten Eigenschaften notwendig. Ein Vergleich dieser Analysen mit den in Kapitel 4.2 erklärten Verfahren weist auf die wachsenden Anforderungen an das Management hin, die sich für ein paralleles System dann ergeben, wenn die Prozesse des Systems kooperativ Anwendungsprobleme lösen sollen.
Parallele Systeme mit diesen Fähigkeiten sind wichtige und notwendige Erweiterungen der bisher betrachteten Systeme. Sie werden jedoch erst im Anschluß an Teil II behandelt.

Teil II
Klassische Betriebssysteme — Realisierungen

In den vorangehenden Kapiteln des Teils I wurden die primär wichtigen Eigenschaften paralleler Systeme an Modellen erklärt. Dabei hat sich gezeigt, daß die Kriterien für diese Eigenschaften aus den Prozessen eines Systems, genauer aus der Anforderung, daß diese Prozesse Berechnungen mit spezifizierten Wirkungen ausführen sollen, abzuleiten sind. Aus den Modellsystemen mit sequentiellen Prozessen ergab sich die Grundstruktur für Realisierungen paralleler Systeme. Diese Grundstruktur hat sich als die eines Modellsystems, wie in Kapitel 4 gezeigt ist, bereits bewährt. In den folgenden Kapiteln liefert sie die nötige Orientierung für die zu erklärenden Realisierungen paralleler Systeme.

Der Schwerpunkt Realisierungen dieses Teils II sagt, daß im folgenden eine Hardwarekonfiguration als Basis eines Systems benötigt wird und so einzusetzen ist, daß alle Eigenschaften, die ein System hat, mit den Komponenten und Eigenschaften dieser Konfiguration hardwaretechnisch realisiert zur Verfügung gestellt werden müssen. Wie früher bereits festgestellt wurde, hat ein klassisches Betriebssystem eine zentrale Hardwarekonfiguration als Realisierungsbasis. Es ergibt sich also, daß eine zentrale Hardwarekonfiguration, die als gegeben angenommen wird, in die Grundstruktur einzuordnen und dann als Basis für Realisierungen aller Eigenschaften, die ein paralleles System haben soll, einzusetzen ist.

Die Notwendigkeit, alle Eigenschaften, die ein paralleles System haben soll, mit der verfügbaren Hardwarekonfiguration realisieren zu müssen, erfordert eine Umorientierung für die Vorgehensweise bei der Konstruktion und der Erklärung eines Systems. Während für ein abstraktes System Kreativität für Konzepte, die zur Lösung von Anwendungsproblemen geeignet sind, im Vordergrund steht, legen die technischen Eigenschaften der eingesetzten Hardwarekonfiguration für ein reales System Rahmenbedingungen fest, die zwingend sind. Hinzu kommt, daß die Eigenschaften der Hardwarekonfiguration realisierungstechnisch bedingt und damit nicht unmittelbar für Problemlösungen von Anwendern geeignet sind. Diese Gegebenheiten bestimmen wesentlich die Anforderungen, die an ein Betriebssystem gestellt werden: Es muß eine Brücke zwischen verschiedenen Welten bauen, der Welt, in der Anwendungsprobleme vorliegen und zu lösen sind, und der Welt der Hardwarekonfiguration mit ihren technischen Eigenschaften, die unbedingt notwendig, aber nicht unmittelbar geeignet sind.

Das Betriebssystem als Brücke zwischen zwei Welten charakterisiert die Aufgabe, die mit der Entwicklung eines Betriebssystems zu lösen ist. Die Lösung muß den Anforderungen beider Seiten entsprechen. Wegen des großen Unterschieds zwischen den beiden Welten sind dazu viele, mit Sorgfalt ausgewählt und systematisch ausgeführte Schritte erforderlich. Die dafür nötige Orientierung liefern Modellsysteme, welche die für Anwendungen wichtigen Eigenschaften erfassen. Die Modelle legen die Ziele fest, auf die die Schritte auszurichten sind.

Im folgenden Kapitel wird zunächst die schon bekannte Grundstruktur eines parallelen Systems an die zu lösende Aufgabe, mittels einer gegebenen, zentralen Hardwarekonfiguration ein System zu konstruieren, das tatsächlich in der Lage ist, Berechnungen auszuführen, angepaßt. Damit ergibt sich ein rudimentäres, paralleles System. Es liefert den Rahmen, der dann auszufüllen und zu ergänzen ist, bis sich schließlich ein anwendungsgeeignetes, paralleles Mehrbenutzersystem ergibt. Zum Ausfüllen des Rahmens gehört insbesondere, daß die Speicherfähigkeiten eines realen Systems auch über Betriebspausen hinweg erhalten werden müssen; das erfolgt mittels der persistent gespeicherten Datenobjekte des Dateisystems eines klassischen Betriebssystems. Zu den Ergänzungen und Erweiterungen des Rahmens gehört insbesondere, daß Komponenten eines Systems, und zwar Datenobjekte und Prozesse, dynamisch erzeugt und aufgelöst werden können.

Kapitel 5

Die Grundstruktur eines zentralen, parallelen Systems

Die aus dem Modell eines parallelen Systems mit sequentiellen Prozessen abgeleitete Grundstruktur legt gemäß Kapitel 3.2 fest, daß ein System aus einer Menge von Prozessen und aus einer gemeinsamen Basis besteht. Den einzelnen Prozessen sind private Wirkungsbereiche zugeordnet, und die sequentiellen Prozesse führen die Berechnungen des Systems aus. Die gemeinsame Basis faßt alle den Prozessen nicht dauerhaft zugeordneten, potentiell kritischen Komponenten des Systems zusammen, und die Prozesse können diese, mit geeigneten Mechanismen und Maßnahmen koordiniert, für ihre Berechnungen benutzen.
Für eine Realisierung ergibt sich, daß die eingesetzte, zentrale Hardwarekonfiguration — HK — in die gemeinsame Basis einzuordnen ist. Damit folgt, daß alle Komponenten des Systems einerseits Komponenten der HK und andererseits mittels der HK und mittels Software realisierte Komponenten sind. Damit ergeben sich jeweils zwei Sichten auf ein reales System:

- Die Sicht auf die eingesetzte HK; sie zeigt die hardwaretechnisch realisierten Eigenschaften des Systems.

- Die Sicht auf das mit der HK realisierte, abstrakte System; sie zeigt, der Grundstruktur entsprechend, die Menge der Prozesse und die gemeinsame Basis mit der in sie eingeordneten HK.

Das Nebeneinander dieser beiden verschiedenen Sichten auf ein System ist charakteristisch für die Realisierungsaufgabe, die zu lösen ist. Das Thema des Kapitels mit den zunächst widersprüchlich erscheinenden Prädikaten trägt diesen beiden Sichten Rechnung; beide Sichten müssen verträglich überführbar sein.
Von der Grundstruktur eines parallelen Systems ausgehend beginnt die Erarbeitung einer Realisierung mit den beiden folgenden Anpassungsschritten:

- Die eingesetzte, zentrale HK wird in die Basis eingeordnet; damit ergibt sich der Betriebssystemkern oder einfach der Kern des entsprechenden Systems.

- Die Prozesse sind mit den zur Verfügung stehenden Mitteln realisierte, abstrakte Maschinen, die auf dem Kern koexistieren und ihre Berechnungen ausführen.

Diese grobe Vorstellung wird im folgenden so an die Realisierungsgegebenheiten angepaßt, daß sich ein Ansatz für die weiteren Schritte zur Lösung der gestellten Konstruktionsaufgabe ergibt.

5.1 Ein einfaches System mit Prozessen und einem Kern

Der angegebenen, groben Vorstellung entsprechend führen die koexistierenden Prozesse eines Systems sequentielle Berechnungen aus. Den einzelnen Prozessen sind private Wirkungsbereiche zugeordnet, mit denen (relative) Interferenzfreiheit gewährleistet wird. Im einfachsten Fall sind die Wirkungen der Berechnungen der Prozesse auf deren private Bereiche beschränkt. Im ersten Schritt soll ein System mit diesen beschränkten Fähigkeiten realisiert werden.
Für diese Realisierung ist eine HK erforderlich. An diese HK werden folgende Anforderungen gestellt:

- Sie muß einerseits dafür geeignet sein, daß jeder einzelne Prozeß mit ihr und mit entsprechender Software die Fähigkeiten erhält, seine Berechnungen mit den angegebenen Beschränkungen tatsächlich auszuführen.
- Sie muß andererseits dafür geeignet sein, den Kern als gemeinsame Basis aller Prozesse und mit dem Kern die koexistierenden Prozesse mit ihren angegebenen Eigenschaften zu realisieren.

Zur Erfüllung dieser Anforderungen wird zunächst geklärt, wie Rechen– und Speicherfähigkeiten mit einer einfachen HK zur Verfügung gestellt werden. Von diesen Gegebenheiten ausgehend werden dann die Eigenschaften der Prozesse und die Aufgaben des Kerns präzisiert.

5.1.1 Komponenten mit Rechen– und Speicherfähigkeiten

Für das zu erklärende System sind Komponenten mit Rechen– und Speicherfähigkeiten erforderlich. Es sind zunächst Komponenten der eingesetzten HK, dann Prozesse und schließlich der Kern. Die aneinander angepaßten Eigenschaften dieser Komponenten werden im folgenden erklärt.

5.1.1.1 Eine einfache HK

Eine einfache HK, mit der Rechen– und Speicherfähigkeiten zur Verfügung gestellt werden, besteht aus einem Speicher M mit Containern für Nachrichten und einem Prozessor R, der Folgen von elementaren Operationen unter Nutzung von M ausführen kann.
Der **Speicher** M besteht aus einer Menge von Bytezellen mit der Struktur eines 1–dimensionalen Feldes. Die Zellen werden der Struktur entsprechend mit numerischen Bezeichnern, M–Adressen genannt, des Adreßraums $MA = \{0, 1, \ldots, M_kap - 1\}$ von M identifiziert. Die Zellen von M sind Container für geeignet repräsentierte Nachrichten; als solche sind sie die wesentlichen Hilfsmittel, die eine HK für Realisierungen von Objekten zur Verfügung stellt. Die realisierten Objekte sind Container mit ihren Inhalten; die auf M ausführbaren Operationen sind Lese– und Schreiboperationen für Containerinhalte.
Der **Prozessor** R besteht aus einer Menge von Binärfeldzellen, R–Register genannt, die Container für geeignet repräsentierte Nachrichten sind und mit R–Adressen des Adreßraums RA von R identifiziert werden, sowie aus Schaltwerken, die R dazu befähigen, festgelegte Elementaroperationen, die als R–Befehle spezifiziert und geeignet repräsentiert vorliegen, in Zusammenarbeit mit M auszuführen. Mit R–Befehlen können R– und M–Adressen spezifiziert sein; R ist fähig, Adressen gemäß RA und MA zu interpretieren, zu den damit identifizierten R–Registern und M–Zellen zuzugreifen sowie, als Bestandteile der R–Befehle, auf diesen Lese– und Schreiboperationen auszuführen.

Für die R–Befehle sind die Wirkungen festgelegt; zudem sind mit R für Ausführungen von R–Befehlen Kontrollstrukturen festgelegt. Daraus ergeben sich die durch R definierten Programmkonzepte. Auf dieser Grundlage, also auf der Grundlage der R–Befehle und der Kontrollstrukturfestlegungen, können R–**Programme** entwickelt werden. Sie sind den geltenden Festlegungen entsprechend in M zu speichern, und in M gespeicherte Programme können von R ausgeführt werden.
Für **Berechnungen**, die mit R und M ausgeführt werden sollen, müssen den Wirkungen und Wirkungsbereichen dieser Berechnungen entsprechende Objekte konstruiert und in M gespeichert werden. Das sind hardware– oder **maschinennahe Berechnungsobjekte**, die Programm– und Datenstrukturen den Wirkungsbereichen der Berechnungen entsprechend zusammenfassen. Die Eigenschaften dieser maschinennahen Berechnungsobjekte werden einerseits von den für R und M erklärten Festlegungen und andererseits von den Zwecken, für die sie eingesetzt werden, bestimmt. Ein Teil der Objekte ist für Anwendungsberechnungen bestimmt; ihre Eigenschaften werden nach Konzepten für Anwendungsprogramme festgelegt und an die für R und M geltenden Festlegungen angepaßt; für diese Objekte wird das Konzept der angepaßten Prozesse eingeführt. Der andere Teil der Objekte ist für die Aufgaben des Kerns bestimmt; ihre Eigenschaften ergeben sich entsprechend erst im Zuge der weiteren Erklärungen.
Die erklärten Eigenschaften der einfachen HK mit R und M sind lediglich die Charakteristika einer HK, die für das Weitere benötigt werden. Sie sind dazu geeignet, die Eigenschaften der Prozesse und die Aufgaben des Kerns im folgenden zu präzisieren.

5.1.1.2 Angepaßte Prozesse

Die Prozesse eines Modellsystems, die in Teil I betrachtet wurden, sind abstrakte Komponenten, die sequentielle Berechnungen ausführen. Diese Eigenschaften sind jetzt an die Realisierungsgegebenheiten anzupassen. Die Prozesse, die jetzt benötigt werden, bleiben einerseits abstrakte Systemkomponenten, die sequentielle Berechnungen ausführen; damit wird erreicht, daß die Bedeutung, die Prozesse eines Modellsystems haben, für die Sicht von außen, also von Anwendern, auf ein mit einer HK realisiertes, abstraktes System, erhalten bleibt. Andererseits sind die Eigenschaften der Prozesse jetzt so festzulegen, daß ihre Berechnungen mit den Mitteln, welche die eingesetzte HK zur Verfügung stellt, tatsächlich ausgeführt werden können.
Das wird mit dem folgenden Prozeßkonzept für Realisierungen erreicht:

- Ein **Prozeß** ist eine mit einem festgelegten Bezeichner identifizierbare, abstrakte Maschine mit einem Speicher und einem Prozessor, die an den Kern gebunden ist und sequentielle Berechnungen ausführt; der Speicher enthält das Programm, das die Berechnungen spezifiziert, und die zugehörigen Daten; der Prozessor führt das Programm unter Nutzung von Kerndiensten aus.

Das angegebene Konzept wird vom **Prozeßmanagement** eines Betriebssystems festgelegt, so daß alle Prozesse eines Systems einerseits die angegebenen, konzeptionellen Eigenschaften und andererseits individuelle Eigenschaften haben. Die konzeptionell festgelegten Eigenschaften bestimmen die Rahmenbedingungen für die individuellen Eigenschaften der einzelnen Prozesse, und sie bestimmen zugleich die Aufgaben, die der Kern für die einzelnen Prozesse übernimmt.
Die Eigenschaften, die ein **Prozeß** p nach den angegebenen Festlegungen hat, ergeben sich zu einem Teil daraus, daß p eine abstrakte Maschine mit einem Speicher und einem Prozessor ist; der Speicher und der Prozessor von p sind Abstraktionen der Komponenten der HK,

die in Kapitel 5.1.1.1 erklärt sind. Die weiteren Eigenschaften von p ergeben sich daraus, daß die abstrakte Maschine so an den Kern gebunden ist, daß für die Berechnungen von p Kerndienste genutzt werden können; diese Eigenschaften werden entsprechend von den Kerndiensten bestimmt, die mit dem Kern angeboten und von p in Anspruch genommen werden. Mit der abstrakten, an den Kern gebundenen Maschine erhält p die Fähigkeit, seine sequentiellen Berechnungen auszuführen. Der Speicher enthält das Programm, das die Berechnungen von p spezifiziert, und die zugehörigen Daten, also das maschinennahe **Berechnungsobjekt** von p. Der Festlegung, daß für die Berechnungen von p Kerndienste genutzt werden können, entsprechend ist das Programm des Berechnungsobjekts von p an den Kern gebunden, so daß mit den Berechnungen von p Kerndienstaufrufe möglich sind. Für die Ausführungen der Berechnungen von p ergeben sich damit die **Umgebungswechsel**, die schon in Kapitel 3.2 im Zusammenhang mit der Grundstruktur für Realisierungen paralleler Systeme erklärt wurden, nämlich Wechsel zwischen dem Berechnungsobjekt von p und dem Kern für die Nutzung von Kerndiensten sowie vom Kern zurück zum Berechnungsobjekt von p.
Zu den **Kerndiensten**, die angeboten werden müssen und von jedem Prozeß in Anspruch genommen werden, gehören die, welche für Realisierungen der konzeptionell festgelegten, abstrakten Eigenschaften der Prozesse notwendig sind, also Dienste, mit denen der abstrakte Speicher und der abstrakte Prozessor jedes einzelnen Prozesses realisiert werden. Weitere Kerndienste, die angeboten werden, können von den Prozessen bei Bedarf in Anspruch genommen werden. Diese Möglichkeiten, die darauf basieren, daß die abstrakten Maschinen der Prozesse an den Kern gebunden sind, gewährleisten die Vielfalt der Eigenschaften, die Prozesse als die Komponenten eines parallelen Systems, welche die Berechnungen des Systems für Anwendungen ausführen, haben müssen.
Das angegebene Prozeßkonzept für Realisierungen legt Rahmen für die Prozesse eines Systems fest. Diese Rahmen werden mit den Berechnungsobjekten der einzelnen Prozesse unter Bezugnahme auf den Kern des Systems ausgefüllt.

5.1.1.3 Der Kern mit eingeordneter HK als gemeinsame Basis der Prozesse

Wenn P der Grundstruktur entsprechend die Menge der Prozesse eines Systems ist, ergibt sich aus dem bisher Erklärten eine erste Übersicht über die Aufgaben des Kerns und darüber, wie diese Aufgaben zu lösen sind.
Ein wesentlicher Teil dieser Aufgabe ist bereits dadurch gelöst, daß das im vorangehenden Kapitel angegebene Prozeßkonzept für Realisierungen festgelegt ist. Es hat zur Folge, daß die $p \in P$ für den Kern festgelegte Eigenschaften haben, nämlich die konzeptionell festgelegten, abstrakten Eigenschaften der Prozesse. Sie sagen, daß für jedes $p \in P$ der abstrakte Prozessor $R(p)$ und der abstrakte Speicher $M(p)$ definiert sind. Die $R(p)$ und $M(p)$ spezifizieren Realisierungsaufgaben für den Kern, mit denen erreicht wird, daß die abstrakten Fähigkeiten der Prozesse, ihre Berechnungen auszuführen, verwirklicht werden. Mit den festgelegten, abstrakten Eigenschaften der Prozesse, mit der HK, die in den Kern eingeordnet zur Verfügung steht, und damit, daß die Eigenschaften der Prozesse an die HK angepaßt sind, lassen sich die Maßnahmen des Kerns für seine Realisierungsaufgaben ableiten.
Wie das Betriebssystem eines parallelen Systems die Aufgabe hat, eine Brücke zwischen den an der Schnittstelle des Systems zu seiner Umgebung wesentlichen Eigenschaften und der zur Realisierung eingesetzten HK zu schlagen, so hat das Prozeßkonzept für Realisierungen die Aufgabe, seinen Beitrag für die Konstruktion eines wichtigen Teils dieser Brücke zu leisten. Für die allgemeine Vorgehensweise ist folgendes wichtig:

- Mit der Festlegung eines Realisierungskonzepts — hier des Prozeßkonzepts für Realisie-

rungen — wird für eine Klasse von zu lösenden Teilproblemen ein Rahmen festgelegt, der zur Systematisierung und zur Vereinfachung der weiteren Maßnahmen dient und erforderlich ist.

- Ein Realisierungskonzept muß an die Gegebenheiten der beiden Seiten, zwischen denen eine Brücke konstruiert werden soll, angepaßt sein. Diese Anpassung ist notwendig für die Ableitung der noch erforderlichen Realisierungsmaßnahmen. Die Anpassung an die beiden Seiten ist einerseits durch Konkretisierungen und Einschränkungen sowie andererseits durch Abstraktionen und Verallgemeinerungen charakterisiert. Festlegungen abstrakter Eigenschaften mit einem Realisierungskonzept sind Spezifikationen von Anforderungen an Realisierungsmaßnahmen, die noch zu erfüllen sind.

- Realisierungskonzepte dienen dazu, entsprechende Realisierungskomponenten zu erzeugen; das sind hier Prozesse mit ihren konzeptionell festgelegten Eigenschaften. Für diese Systemkomponenten ergibt sich aus dem Beitrag, den sie zur Brücke leisten sollen, daß ihre Eigenschaften jeweils unter den beiden Sichten einzuordnen und zu beschreiben sind, die für ein realisiertes, abstraktes System allgemein charakteristisch sind. Die Eigenschaften der beiden Sichten unterscheiden sich ihrer Zweckbestimmung entsprechend wesentlich, so daß zu den Eigenschaften jeweils die Sicht–Einordnung gehört.

Die Rolle des Betriebssystems als **Management** eines Rechensystems wird an der hier behandelten Realisierungsaufgabe deutlich:

- Es sind Realisierungskonzepte als kreative Beiträge zur Lösung der jeweils gestellten Aufgabenklasse erforderlich.

- Mit den Realisierungskonzepten können Realisierungs– oder Managementobjekte erzeugt werden. Diese sind Rahmen, in welche die jeweils gestellten Realisierungsanforderungen eingepaßt und dann systematisch erfüllt werden.

Die **Prozesse als Managementobjekte** sind Paare mit abstraktem Prozessor und Speicher, die

- einerseits die sequentiellen Berechnungen der Prozesse ausführen und

- andererseits die Anforderung spezifizieren, die abstrakten Prozessoren und Speicher mit dem Prozessor und dem Speicher der eingesetzten HK so zu realisieren, daß die Berechnungen tatsächlich ausgeführt werden.

- Die beiden Sichten auf einen Prozeß zeigen

  - einerseits die Berechnungen, die ausgeführt werden, und die Wirkungen dieser Berechnungen sowie
  - andererseits die Beschreibungen der gestellten Anforderungen, die mit geeigneten Realisierungsmaßnahmen zu erfüllen sind.

An dem Paar aus Prozessor und Speicher eines Prozesses als Managementobjekt kann man ableiten, wie die Realisierungsanforderungen erfüllt werden können; Abbildung 5.1 zeigt schematisch die Vorgehensweise. Die notwendigen Realisierungsmaßnahmen erfordern jedoch differenziertere Anpassungen an die Rahmenbedingungen. Mit ihnen ergeben sich die Kerndienste, die benötigt werden.

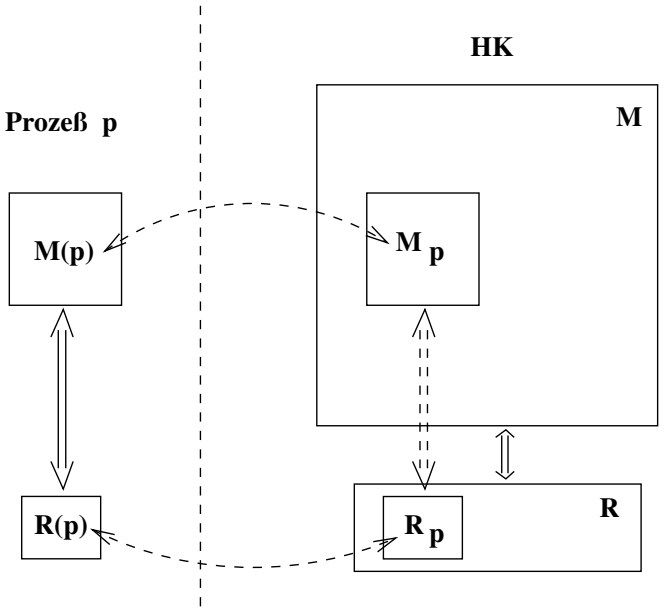


Abbildung 5.1: Schematische Realisierung eines Prozesses

5.1.2 Realisierung der Prozeß–Rechen– und Speicherfähigkeiten

Unter der vereinfachten Voraussetzung, daß Prozesse ihre Berechnungen allein auf ihren privaten Wirkungsbereichen ausführen, kann ein (abstrakter) Prozeß p seine Berechnungen mit seinem Prozessor $R(p)$ und seinem Speicher $M(p)$ ausführen. Das Paar $(R(p), M(p))$ stellt alles zur Verfügung, was hierzu erforderlich: den Wirkungsbereich der Berechnungen von p mit $M(p)$ und den $R(p)$–Registern, und die Fähigkeiten zur Ausführung der Berechnungen mit $R(p)$. Das in Abbildung 5.1 skizzierte, schematische Vorgehen für eine Realisierung von p mit der in Kapitel 5.1.1.1 eingeführten HK mit (R, M) sagt folgendes:

- Ein Teil von M ist als M_p für $M(p)$, und ein Teil von R ist als R_p für $R(p)$ einzusetzen.
- Wenn mit dieser Einsetzung erreicht wird, daß $(R(p), M(p))$ und (R_p, M_p) äquivalent sind, dann wird der Prozeß p mit (R_p, M_p) realisiert.

Dieses Schema charakterisiert einerseits das Realisierungsziel für den einen Prozeß p, und als solches ist es auch auf eine Prozeßmenge P verallgemeinerbar. Andererseits läßt das Schema wesentliche Rahmenbedingungen einer HK, die für Realisierungen eingesetzt werden soll, außer Acht. Im folgenden werden die Rahmenbedingungen der Aufgabe des Kerns, die Rechen– und Speicherfähigkeiten der Prozesse zu realisieren, differenzierter und realistischer erklärt; zudem werden Verfahren zur Lösung der Aufgabe angegeben.

5.1.2.1 Die Prozesse und die HK für Realisierungen

Für das Folgende sei P die Menge der zu realisierenden Prozesse. Für jeden Prozeß $p \in P$ sei $(R(p), M(p))$ festgelegt, und für Realisierungen der Prozesse soll die HK mit (R, M) und den in Kapitel 5.1.1.1 erklärten Eigenschaften zur Verfügung stehen.
Das oben schematisch erklärte Realisierungsziel ist für einen Prozeß p erreicht, wenn durch Einsetzung von M– und R–Teilen (R_p, M_p) so konstruiert werden kann, daß (R_p, M_p) und $(R(p), M(p))$ verhaltensäquivalent sind. Daraus ergibt sich die Frage, ob und wie dieses Ziel

erreicht werden kann. Die naheliegende Verallgemeinerung dieser Vorgehensweise für p auf die Menge P der koexistierenden Prozesse führt zu den koexistierenden Paaren (R_p, M_p) der $p \in P$ und damit zu der Frage, ob dieses Ziel erreichbar ist. Diese Frage stellt sich für die beiden Dimensionen Zeit und Raum, die für Berechnungen wesentlich sind. In der Dimension Raum ist, weil die Wirkungsbereiche der $p \in P$ gemäß $(R(p), M(p))$ nach Voraussetzung disjunkt sind, gefordert, daß paarweise disjunkte Teile von (M, R) eingesetzt werden. Das sind Anforderungen an die Speicherkapazitäten von M und der $M(p)$ sowie von R und der $R(p)$, die nicht erfüllt sind. In der Dimension Zeit ist festgelegt, daß R ein sequentieller Prozessor ist. Daraus ergibt sich, daß die Parallelität (Gleichzeitigkeit) der koexistierenden $p \in P$ durch Quasiparallelität, also durch Zeitmultiplex–Einsatz der Rechenkapazität von R, realisiert werden muß. Auch mit dieser realistischen Beschränkung von R auf einen sequentiellen Prozessor bleibt die oben diskutierte Forderung nach koexistierenden R_p–Registern als Teilen von R relevant; sie würde den Zeitmultiplex–Einsatz der Rechenkapazität von R vereinfachen; sie ist aber unrealistisch.
Zu den betrachteten, unmittelbaren Realisierungsanforderungen der $p \in P$ kommt hinzu, daß der Kern mit seinen Rechen– und Speicherfähigkeiten ebenfalls mit (R, M) zu realisieren ist. Dazu sind, dem bisherigen Ansatz folgend, ein weiterer R–Teil R_K mit Rechen– und Speicherkapazität sowie ein M–Teil M_K erforderlich; die Register gemäß R_K und M_K müssen wieder disjunkt zu allen R_p bzw. M_p sein.

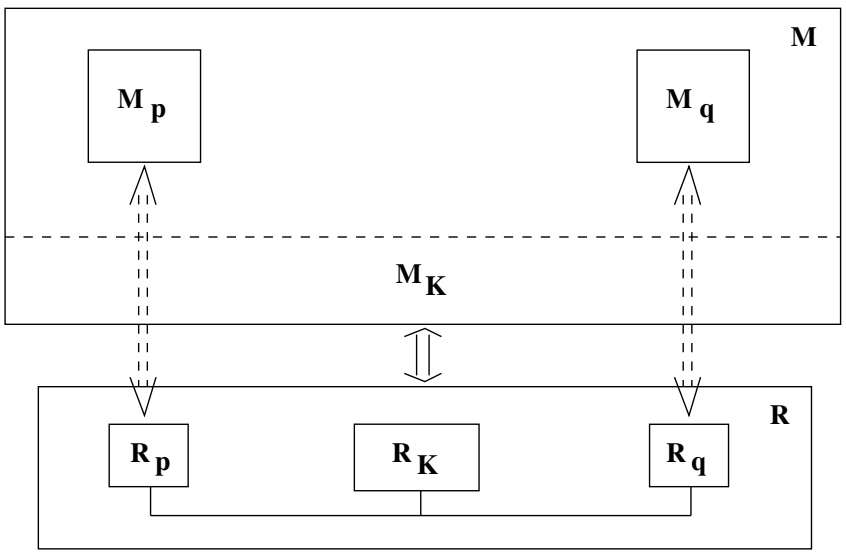


Abbildung 5.2: Ein Ansatz zur Realisierung eines Systems mit P

Mit diesem Ansatz ergibt sich die in Abbildung 5.2 skizzierte Realisierung des Systems mit der Prozeßmenge P. Sie zeigt die Prozesse mit ihren konzeptionell festgelegten Wirkungsbereichen. Die koexistierenden Prozesse führen ihre Berechnungen abstrakt parallel und realisiert quasiparallel aus. Dieser Unterschied ergibt sich daraus, daß die R–Rechenkapazität im Zeitmultiplex–Einsatz auf R_K und auf die $R_p, p \in P$, aufgeteilt wird. Wenn, wie hier vorausgesetzt ist, von den Prozessen keine weiteren Anforderungen gestellt werden, bleibt dem Kern die Aufgabe des „Umschaltens“ der R–Rechenkapazität zwischen den R_p; das ist einfache Koprogramm–Umschaltung. Der Ansatz liefert eine **hardwareunterstützte Koprogramm–Realisierung** für die Prozesse gemäß P. Für die Wirkungen der Berechnungen der Prozesse gilt:

- Die Wirkungen der realisierten, quasiparallelen Ausführungen der Berechnungen der $p \in P$ sind die der abstrakten, parallelen Ausführungen unter der Voraussetzung, daß R–Rechenkapazitäts–Umschaltungen für die Prozesse funktional wirkungslos sind; die realisierten und die abstrakten Ausführungen unterscheiden sich (lediglich) in den Ausführungszeiten.

Die Grundlage für diese wichtige Eigenschaft der Realisierung der Prozesse ist die Verhaltensäquivalenz der $(R(p), M(p))$ und (R_p, M_p) für die $p \in P$. Der Ansatz einer hardwareunterstützten Koprogramm–Realisierung der Prozesse gemäß P ist unrealistisch; er zeigt jedoch das Realisierungsziel, das mit realistischen Realisierungsgegebenheiten auf Umwegen erreicht werden muß. Abbildung 5.3 veranschaulicht die realisierten Prozesse für $P = \{a, b, c\}$. Die vertikalen Linien beschreiben Umschaltungen der R–Rechenkapazität.

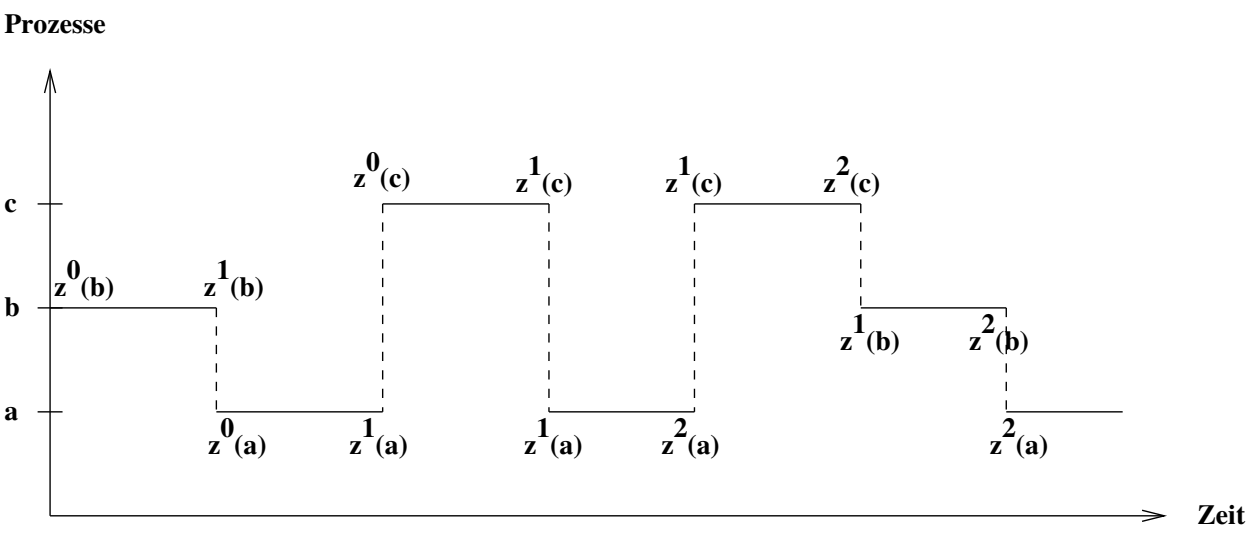


Abbildung 5.3: Koprogramm–Realisierung der Prozesse $P = \{a, b, c\}$

Die erklärte hardwareunterstützte Koprogramm–Realisierung ist **unrealistisch**, weil es keine HK gibt, welche die gestellten Anforderungen erfüllt. Von den Anforderungen, die mit P gestellt werden, ist die Diskrepanz in der Dimension Zeit mit dem Zeitmultiplex–Einsatz der R–Rechenkapazität bereits unter Gewährleistung der Wirkungen der Berechnungen der Prozesse überwunden. Es bleiben also noch die Anforderungen in der Dimension Raum, also die Anforderungen an die Kapazität von M und an die Speicherkapazität von R.
Die Anforderungen an die M–**Kapazität** sind unrealistisch; sie sind für variierende, große $|P|$ nicht erfüllbar. Die bestehende Diskrepanz kann jedoch analog zu der bzgl. der R–Rechenkapazität überwunden werden. Das erfolgt mit einer HK, die statt M eine Hierarchie von Speichern mit relativ zu R wachsenden Lese/Schreibzeiten und Kapazitäten zur Verfügung stellt, und mit Zeitmultiplex–Einsatz (oder effizienteren Einsatz–Verfahren) des Speichers der 1–ten Stufe.
Die Anforderungen an die R–**Speicherkapazität** sind unrealistisch. Prozessoren stellen vielmehr Register zur Verfügung, die auf die sequentielle Ausführung eines Programms (gemäß Prozessor–Programmkonzept) und darauf, daß der Prozessor einerseits für Prozeß– und andererseits für Kernberechnungen eingesetzt werden muß, zugeschnitten sind. Diese beschränkten Prozessor–Speicherfähigkeiten haben weitreichende Konsequenzen.
Bzgl. der erklärten hardwareunterstützten Koprogramm–Realisierung ergibt sich, daß nicht gleichzeitig R_K–und R_p–Zustände in Registern gespeichert werden können; dem entspre-

chend sind auch Koprogramm–Umschaltungen der Prozessor–Rechenkapazität nicht möglich. Statt dessen ist auch für die Register Zeitmultiplex–Einsatz erforderlich, und das „Umschalten“ erfordert die Ausführung von Befehlsfolgen. Das „Umschalten“ wird durch nichtdeterministische Kontrollstrukturkonzepte des Prozessor–Befehlszyklus, den sog. Programmausführungs–**Unterbrechungen**, unterstützt. Sie sind die Basis für den Zeitmultiplex–Einsatz der Rechen– und Speicherkapazitäten eines entsprechenden Prozessors und den Übergang von abstrakt parallelen zu realisiert quasiparallelen Berechnungs–Ausführungen. Bzgl. der erklärten hardwareunterstützten Koprogramm–Realisierung der Prozesse ergibt sich folgendes:

- Die Koprogramme sind jetzt dem Prozessor–Programmkonzept entsprechend zusammengefaßte Befehlsfolgen, und die Koprogramm–Umschaltungen erfolgen im Befehlszyklus als Unterbrechungen.

- Die Koprogramm–Realisierungen der Prozesse und die dafür erforderlichen „Umschaltungen“ müssen auf dieser Basis mit Befehlsfolgen vorbereitet und durchgeführt werden.

Die Koprogramm–Realisierungen der Prozesse müssen also verfeinert auf die Ebene der Prozessor–Befehle durchgeführt werden. Die Ausführungen dieser Befehle sind einfache Kernberechnungen; die entsprechenden Befehlsfolgen spezifizieren einfache Kerndienste, die Dienste des **Dispatchers**.
Die Register, mit denen ein Prozessor seine Speicherfähigkeiten erhält, sind die 0–te Stufe der Speicherhierarchie einer entsprechenden HK. Der notwendige Zeitmultiplex–Einsatz der Register wird mit dem Speicher der 1–ten Hierarchiestufe realisiert. Das ist der **Arbeitsspeicher AS** der entsprechenden HK; er ist dadurch ausgezeichnet, daß der Prozessor zu seinen Zellen (neben den Registern) direkt für Lese– und Schreiboperationen zugreifen kann. Zur Realisierung des Zeitmultiplex–Einsatzes der Register werden die R_K– und R_p–Zustände der $p \in P$ i.w. in AS gespeichert, und zwar als Datenobjekte des Kerns. Für diese sowie für die Programme und Daten des Kerns wird ein Teil von AS benötigt; er entspricht dem früher eingeführten M–Teil M_K und dient dazu, den Kern–Wirkungsbereich zu realisieren. Ebenso werden Teile von AS für Realisierungen der privaten Wirkungsbereiche der Prozesse benötigt, also für die früher eingeführten M–Teile M_p der $p \in P$. Für diese Teile von AS ist festgelegt, daß sie disjunkt sein müssen. Zudem müssen zwischen diesen Teilen, wie in Kapitel 3.2 erklärt, Grenzwälle gezogen werden, und zwar Grenzwälle zwischen den Teilen der privaten Wirkungsbereichen der einzelnen Prozesse sowie zwischen diesen Teilen und dem Teil des Kern–Wirkungsbereichs. Weil diese Grenzwälle für Berechnungs–Ausführungen wirksam sein müssen, müssen entsprechende Festlegungen für den Prozessor, der die AS–Zugriffe ausführt, getroffen werden; das erfolgt mittels spezieller Register. Die Wirksamkeit der **Grenzwälle** wird mit folgenden Maßnahmen erreicht:

- Für den Prozessor werden (wenigstens) zwei Betriebs–Modi festgelegt, der Kern– und der Benutzer–Modus.

- Die Menge der Prozessor–Befehle wird in privilegierte und nichtprivilegierte zerlegt, und es wird festgelegt, daß im Benutzer–Modus allein nichtprivilegierte, im Kern–Modus sämtliche Befehle zulässig sind.

Der Prozessor im Kern–Modus wird für Kernberechnungen eingesetzt; im Benutzer–Modus wird er für Berechnungen der Prozesse auf ihren privaten Wirkungsbereichen eingesetzt. Der

jeweilige Betriebs–Modus wird vom Inhalt eines speziellen Registers bestimmt; ebenso werden die Grenzlinien mit Registerinhalten gezogen. Schreiboperationen auf diesen Registern sind typische privilegierte Befehle.
Für einen Prozeß, der seine Berechnungen mit dem Prozessor im Benutzer–Modus ausführt, bewirkt ein Versuch, einen privilegierten Befehl auszuführen oder seinen privaten Wirkungsbereich zu überschreiten, eine **Unterbrechung** mit dem Übergang des Prozessors in den Kern–Modus und einem Kerndienst–Aufruf. Eine weitere Möglichkeit für Kerndienst–Aufrufe steht Prozessen mit einem speziellen, nichtprivilegierten Befehl, der ebenfalls zunächst eine Unterbrechung bewirkt, zur Verfügung. Mit diesem Instrumentarium und seinen Varianten sind implizite und explizite Kerndienst–Aufrufe für Prozesse sowie Umgebungswechsel zu realisieren, und mit den entsprechenden Kerndiensten werden die Prozesse wieder als Koprogramme realisiert.
Das Erklärte zeigt die Zusammenhänge auf, die zwischen den Eigenschaften eines Prozessors, seinen Speicherfähigkeiten durch Register und seinen Rechenfähigkeiten, und den Maßnahmen bestehen, die zur Realisierung der Prozesse erforderlich sind. Mit diesen Maßnahmen ergibt sich wieder, daß die Prozesse ihre Berechnungen abstrakt parallel und realisiert quasiparallel ausführen. Das notwendige „Umschalten“ erfolgt mit Kernberechnungen, und diese müssen wieder — relativ zu den Prozessen — wirkungslos sein.
Im folgenden wird die Aufgabe des Kerns, die Prozesse zu realisieren, genauer behandelt. Als Basis hierfür wird eine in den Kern eingeordnete HK eingesetzt, die vorerst

- aus einem Prozessor mit den oben erklärten Eigenschaften und
- aus einem Arbeitsspeicher AS

besteht.

5.1.2.2 Realisierung der Prozeßrechenfähigkeiten: Das Zustandsdiagramm

Im folgenden wird erklärt, wie die Rechenfähigkeiten der Prozesse, die sie abstrakt mit ihren Prozessoren haben, mit dem Prozessor der eingesetzten HK realisiert werden. Dabei wird von den Speicherfähigkeiten der Prozesse allein der Teil betrachtet, den sie mit den Registern ihrer Prozessoren erhalten.
Zunächst wird angenommen, daß ein Prozeß p außer der stets implizit gestellten Anforderung des Prozessors der HK, im weiteren realer Prozessor r genannt, keine weiteren Anforderungen an den Kern stellt.
Zur Erklärung der notwendigen Realisierungsmaßnahmen werden für den Prozeß p Zustände eingeführt, die das für das Fortschreiten der Berechnungen von p Wesentliche beschreiben. Die notwendigen Maßnahmen bestehen dann in Zustandsüberführungen.
Für den abstrakten Prozeß p sind die in Abbildung 5.4 gezeigten Zustände und Zustandsübergänge relevant; sie sind aus Teil I von den Verhaltensbeschreibungen der Prozesse mit Spuren bekannt.
Der Zustand v bedeutet, daß der Prozeß p existiert und zur Ausführung seiner Berechnungen vorbereitet ist. Der Zustand a bedeutet, daß p seine Berechnungen ausführt. p wird mit *starten* von v in a überführt; das entspricht dem Eintreten des Ereignisses $\underline{p}$. Der Zustand t bedeutet, daß p seine Berechnungen vollständig ausgeführt hat und terminiert ist. p wird mit *terminieren* von a in t überführt; das entspricht dem Eintreten des Ereignisses $\overline{p}$.
Zur Realisierung von p mit r ist r als FXBM einzusetzen; das entspricht dem schon erklärten Zeitmultiplex–Einsatz von r. Für die Beschreibung des Verhaltens von p ergibt sich, daß der Zustand a in zwei Zustände zu verfeinern ist:

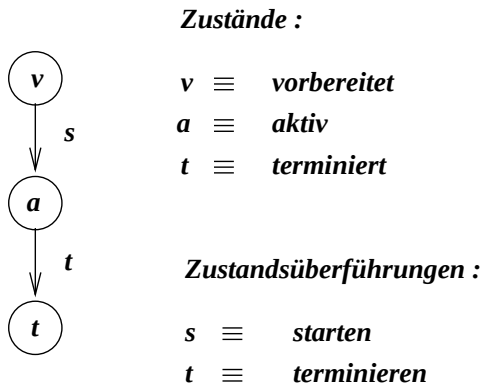


Abbildung 5.4: Das Zustandsdiagramm des abstrakten Prozesses p

- $rb \equiv$ *rechenbereit* bedeutet, daß p den Prozessor r angefordert hat und auf Zuteilung wartet;
- $rr \equiv$ *realrechnend* bedeutet, daß r für p eingesetzt ist, so daß die Berechnungen tatsächlich fortschreiten.

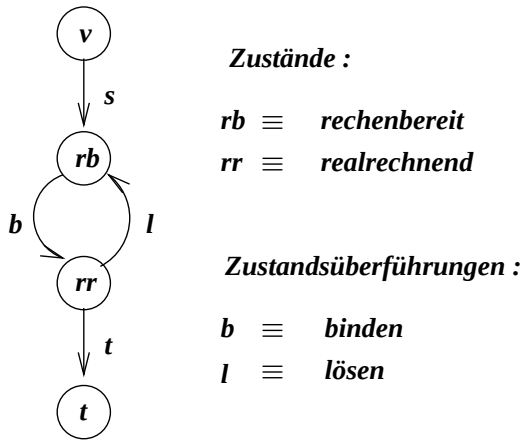


Abbildung 5.5: Das vereinfachte Zustandsdiagramm des realisierten Prozesses p

Der Prozeß p wird mit *starten* von v in rb überführt; das entspricht der Überführung des abstrakten p in a unter Berücksichtigung der Realisierungsgegebenheit, daß r benötigt und nicht notwendig frei ist. p wird mit *binden* von rb in rr überführt; p wird so an r gebunden, daß damit die Fähigkeiten von r für p eingesetzt sind. p kann mit *lösen* von rr in rb überführt werden; diese Überführung trägt der Konkurrenz der Prozesse um r Rechnung.
Wenn der Prozeß p keine weiteren Anforderungen stellt, kann sein Verhalten mit dem angegebenen Diagramm vollständig beschrieben werden. Zur Beschreibung der Fälle, in denen p weitere Anforderungen stellt, wird ein weiterer Zustand eingeführt

- $w \equiv$ *wartend* bedeutet, daß p auf die Erfüllung weiterer Anforderungen — nicht die von r — wartet und blockiert ist.

w ist ein Sammel–Wartezustand, der noch bzgl. der gestellten Anforderungen zu verfeinern ist.

p wird mit *blockieren* (*) von rr in w und mit *entblockieren* (**) von w in rb überführt. Abbildung 5.6 zeigt das Zustandsdiagramm, das sich mit dieser Erweiterung ergibt.

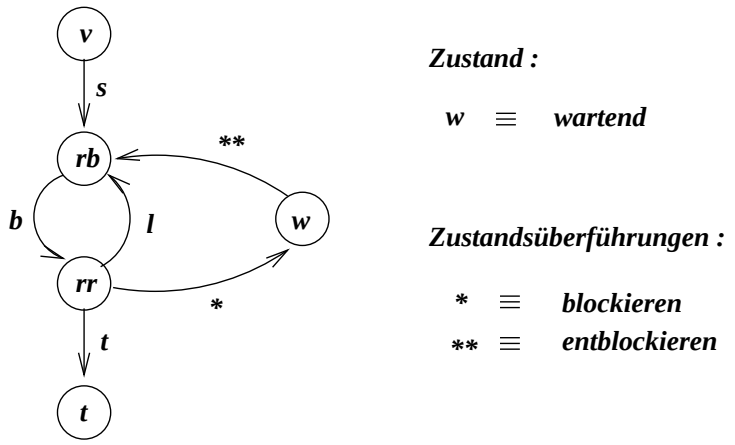


Abbildung 5.6: Das Zustandsdiagramm des realisierten Prozesses p

Mit dem angegebenen Zustandsdiagramm läßt sich das Verhalten des Prozesses p vergröbert auf das für das Fortschreiten seiner Berechnungen Wesentliche vollständig beschreiben. Die Zustände und Zustandsüberführungen sind wie folgt einzuordnen:

- Für den abstrakten Prozeß p, der seine Berechnungen vollständig im p–privaten Wirkungsbereich ausführt, sind die Zustände v, rb, rr und t mit den entsprechenden Überführungen relevant: Die Ausführung der Berechnungen beginnt mit *starten* ($\underline{p}$) und endet mit *terminieren* ($\overline{p}$); die Zustandsfolge $rb, rr, rb, \ldots, rr$ der Spur zwischen diesen Überführungen ergibt sich aus der Notwendigkeit, die Rechenfähigkeiten von p mit dem realen Prozessor r zu realisieren; die Überführungen müssen relativ zu p funktional wirkungslos sein; die Folge entspricht dem Zeitmultiplex–Einsatz von r für p. Die r–Anforderungen für p sind mit den konzeptionell festgelegten Eigenschaften von p (dem p–Prozessor) stets implizit gestellt.

- Für den abstrakten Prozeß p, der seine Berechnungen nicht vollständig im p–privaten Wirkungsbereich ausführt, ergibt sich, daß weitere Anforderungen explizit von p zu stellen und vom Kern zu erfüllen sind. Das wird auf der Grundlage des Zustandsdiagramms mit $rr \xrightarrow{*} w \xrightarrow{**} rb$ beschrieben: p stellt eine Anforderung und blockiert sich in w; wenn die Anforderung erfüllt ist, wird p entblockiert; mit dem Übergang von rr in w wird der reale Prozessor r freigegeben. Der Zustand w ist ein Sammel–Wartezustand, der bzgl. der jeweils gestellten Anforderungen verfeinert wird. Die Realisierung des bisher unberücksichtigten, konzeptionell festgelegten p–Speichers, die ebenfalls stets implizit angefordert ist, kann zu Wartephasen für p führen, die auch mit dem Zustand w beschrieben werden. In diesen Fällen, die noch erklärt werden, wird p mit $rr \xrightarrow{*} w$ blockiert und schließlich mit $w \xrightarrow{**} rb$ entblockiert.

Mit dem p–Zustandsdiagramm wird das Verhalten des Prozesses so beschrieben, wie es für die Wirkungen der Berechnungen von p relevant ist. Der Realisierungsaufgabe entsprechend, die vom Kern zu lösen ist, ist neben dieser Sicht auf p die **Realisierungs–Sicht** des Kerns auf p relevant. Für den Kern, der Dienste für die angegebenen Zustandsüberführungen zur Verfügung stellen und ausführen muß, ist der Prozeß p ein Datenobjekt, auf dem Kernberechnungen auszuführen sind.

Für die Realisierungsaufgaben des Kerns ist erforderlich, daß alle jeweils existierenden Prozesse mit allen ihren für den Kern relevanten Eigenschaften mit geeigneten Datenobjekten beschrieben werden. Dieses Datenobjekt ist die **Prozeßliste PL**. PL ist ein dynamisches Datenobjekt, dessen Elemente und Strukturen den vielfältigen Eigenschaften der Prozesse, die zu beschreiben sind, und den vielfältigen Zwecken, für die das Datenobjekt benötigt wird, geeignet festzulegen sind. Hier wird von den Elementen und Strukturen der PL lediglich ein kleiner Teil benötigt und erklärt; weitere Eigenschaften werden mit den entsprechenden Erklärungen ergänzt.
Für die Prozesse ist festgelegt, daß sie mit ihren Bezeichnern identifizierbar sind. Für die Bezeichner wird weiter die (dynamische) Menge P benutzt. Dem entsprechend müssen mit den $p \in P$ Zugriffe zu den die Prozesse beschreibenden Elementen von PL möglich sein. Diese Elemente von PL sind die **Prozeßdeskriptoren**, *deskriptor*(p) für $p \in P$. Das Datenobjekt *deskriptor*(p) beschreibt mit festgelegten Attributen die dauerhaften Eigenschaften des Prozesses und ist zudem die Wurzel für Datenobjekte, die weitere Eigenschaften von p beschreiben. *deskriptor*(p) und die weiteren Datenobjekte, mit denen Eigenschaften von p beschrieben werden, enthalten Zeiger für Einordnungen in dynamische Datenobjekte und für zugeordnete Datenobjekte; damit wird die für Prozeßbeschreibungen erforderliche Flexibilität erreicht.
Für $p \in P$ ist eines der von *deskriptor*(p) aus erreichbaren, weiteren Datenobjekte der **Deskriptor des Berechnungszustands** *berechnung*(p). Er beschreibt mit entsprechenden Attributen die für die hier behandelte Realisierungsaufgabe relevanten Eigenschaften des Prozesses p. Für die $p \in P$ werden die Deskriptoren *berechnung*(p) für die Zustandsüberführungen nach dem Zustandsdiagramm benötigt, also insbesondere für die Überführungen $rb \leftrightarrow rr$. Der Zustand rb ist ein spezieller Wartezustand, so daß von den Prozessen gemäß P i.a. mehrere in rb sind. Diese Prozesse haben gemeinsam, daß sie den realen Prozessor r anfordern; es ist deshalb zweckmäßig, die Prozesse mit einem geeigneten dynamischen Datenobjekt, der rb–**Warteliste**, zusammenzufasssen. Zusammenfasssungen der Beschreibungen der Prozesse einerseits in der Prozeßliste und andererseits in der rb–Warteliste sind typisch für die Flexibilität, die der Kern erfordert.
Die **Attribute eines Deskriptors des Berechnungszustandes** eines Prozesses werden den Rahmenbedingungen der Zustandsüberführungen des erklärten Diagramms entsprechend festgelegt. Sie werden von den Eigenschaften des realen Prozessors r und von den für Prozesse konzeptionell festgelegten Eigenschaften bestimmt; sie müssen wie schon erklärt, angepaßt sein. Die Attribute werden häufig unmittelbar von r übernommen und dann zusammengefaßt als **Prozeßkontrollblock** (PCB) bezeichnet. Für die Attribute ist jedoch stets die Zuordnung zu konzeptionell festgelegten bzw. realisierungsbedingten Eigenschaften zu unterscheiden.
Von den für einen Prozeß p konzeptionell festgelegten Eigenschaften sind primär die relevant, die den Registern des p–**Prozessors** entsprechen; zu diesen gehört wenigstens das Befehlsadreßregister (PC); sie sind Teil des p–privaten Wirkungsbereichs. Für diesen Teil soll folgendes gelten:

- *rp_register* seien die dem p–Prozessor entsprechenden Register von r;
- ein wohldefinierter Inhalt von *rp_register* wird mit *rp_zustand* bezeichnet;
- das Attribut von *berechnung*(p) für *rp_register* ist *prozessor*.

Zu den für den Prozeß p konzeptionell festgelegten Eigenschaften gehört weiter der p–**Speicher** als Teil des p–privaten Wirkungsbereichs. In Kapitel 5.1.2.1 ist erklärt, daß mit

dem p–Speicher Grenzwälle für den Arbeitsspeicher festgelegt werden, die mit dem realen Prozessor r auf der Grundlage von Inhalten spezieller Register kontrolliert werden. Die Inhalte dieser Register gehören nicht zum p–privaten Wirkungsbereich; sie werden jedoch für die kontrollierten Ausführungen der p–Berechnungen benötigt. Diesen Gegebenheiten entsprechend werden die oben angegebenen Festlegungen wie folgt erweitert:

- *rk_register* seien die Register *rp_register* zusammen mit den speziellen Registern von r, die zur Kontrolle der Grenzen des p–Speichers eingesetzt werden;
- ein wohldefinierter Inhalt von *rk_register* wird mit *rk_zustand* bezeichnet;
- das Attribut von *berechnung*(p) für *rk_register* ist *k_prozessor*.

Damit sind die Attribute von *berechnung*(p), soweit sie hier benötigt werden, festgelegt. Für die eingeführten Zustände zur Beschreibung des Fortschreitens der Berechnungen von p wird folgendes festgelegt:

- Wenn p in einem der Zustände *rb* oder *w* ist, ist der Wert von *prozessor*(p) der mit den Berechnungen von p erreichte Zustand des p–Prozessors als Teil des p–privaten Wirkungsbereichs.
- Wenn p im Zustand *rr* ist, ist der jeweilige *rp_zustand* der mit den Berechnungen von p erreichte Zustand des p–Prozessors als Teil des p–privaten Wirkungsbereichs.

Mit diesen Festlegungen werden die Soll–Wirkungen der Kerndienste für die **Zustandsüberführungen** *binden* **und** *lösen* bzgl. *rk_register* von r und bzgl. *k_prozessor* von *berechnung*(p) so spezifiziert, daß sie den p–Prozessor und die Grenzen des p–Speichers für Kontrollen erfassen:

- *binden* : $r.rk_register := berechnung(p).k_prozessor$;
- *lösen* : $berechnung(p).k_prozessor := r.rk_register$;

Diese Wirkungsfestlegungen sind für den Prozeß p die Grundlage für den Übergang von abstrakt parallel zu realisiert quasiparallel und für die mit den Zustandsüberführungen wesentlich zu erfüllende Forderung der funktionalen Wirkungslosigkeit relativ zu den p–Berechnungen.
Zur Implementierung der **Kerndienste mit r–Befehlsfolgen** stehen, wie in Kapitel 5.1.2.1 erklärt, die Programmausführungs–Unterbrechungen des r–Befehlszyklus und Programme zur Reaktion auf Unterbrechungen, Unterbrechungsreaktionsprogramme (interrupt handler), die mit Unterbrechungen aufgerufen werden, zur Verfügung. Die Unterbrechungen als gegebene Zeitmultiplex– und Programmaufruf–Basismechanismen „retten“ Inhalte von r–Registern und „laden“ diese für die Ausführung des aufgerufenen Programms. Zu diesen r–Registern gehört wenigstens das Befehlsadreßregister. Diese einzusetzenden Basismechanismen haben zur Folge, daß die Kernberechnungen der Zustandsüberführungen *binden* und *lösen* mit den festgelegten Wirkungen in 2 Ebenen auszuführen sind,

- in der Ebene des Befehlszyklus mit den Unterbrechungen und
- in der Ebene der Unterbrechungsreaktionsprogramme, in der Datenobjekte der Deskriptoren der Berechnungszustände der Prozesse den Festlegungen entsprechend zusammenzuführen und zu zerlegen sind.

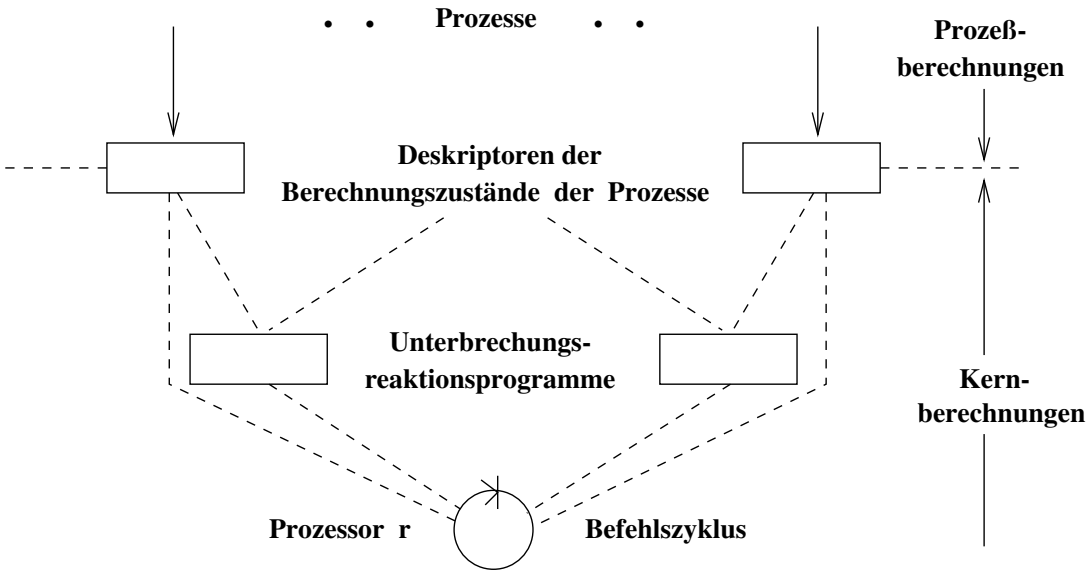


Abbildung 5.7: Ebenen der Realisierung der Prozeß–Rechenfähigkeiten

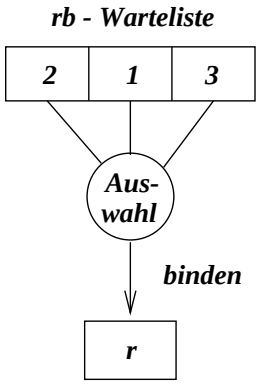
Abbildung 5.7 veranschaulicht diese Zusammenhänge.
Die Abbildung zeigt, eingeschränkt auf die Rechenfähigkeiten, die abstrakte Sicht auf die Prozesse mit ihren konzeptionell festgelegten Eigenschaften und die Realisierungs–Sicht mit den Eigenschaften der eingesetzten HK und den Kerndiensten, die beide Sichten verbinden und die Übergänge ermöglichen. Die abstrakte Sicht auf die Prozesse, die ihre Berechnungen in ihren privaten Wirkungsbereichen ausführen, endet für die Prozesse in einem der Zustände rb und w mit den Deskriptoren der Berechnungszustände und für den Prozeß im Zustand rr mit dem realen Prozessor r, der für den Prozeß eingesetzt ist. Das entspricht dem Übergang von abstrakt parallel zu realisiert quasiparallel. Für diesen Übergang sind die Kerndienste der Zustandsüberführungen erforderlich. Sie bewirken das Umschalten der Rechenkapazität von r zwischen den Prozessoren der Prozesse; sie bilden den **Dispatcher** als Teil des Kerns. Die Kerndienste sind auf die Basismechanismen von r und damit auf die Unterbrechungen einschließlich der für sie geltenden Festlegungen angewiesen; das führt zu den gezeigten Realisierungsebenen. Zudem werden die Kernberechnungen mit der Rechenkapazität von r ausgeführt.
Die Abbildung zeigt auch, daß das Nebeneinander der beiden Sichten das Ergebnis eines notwendigen, gezielten Abstraktionsprozesses ist: Alle Berechnungen können aus der Sicht des realen Prozessors r, der sie ausführt, gesehen werden. Dann sind sie Berechnungen, die mit einem Programm nach den durch r festgelegten Konzepten spezifiziert sind. Die Wirkungen dieser Berechnungen sollen für Anwendungen geeignet und für Anwender verständlich sein.

5.1.2.3 Realisierung der Prozeßrechenfähigkeiten: Scheduling

Für die Realisierung der Rechenfähigkeiten eines Prozesses p, der seine Berechnungen vollständig auf seinem privaten Wirkungsbereich ausführt, gilt das in Abbildung 5.5 gezeigte vereinfachte Zustandsdiagramm. Wenn das für p Gesagte für alle Elemente der Prozeßmenge P der Fall ist, sind von den Prozessen i.w. die Zustände rb und rr von Interesse. Von den erklärten Zustandsüberführungen *binden* und *lösen* wird nach den bisherigen Erklärungen *lösen* nicht dringend benötigt: Ein Prozeß, der im Zustand rb den realen Prozessor r anfordert, kann mit *binden* von rb in rr überführt werden, seine Berechnungen vollständig

ausführen und dann in den Zustand t übergehen. Auch mit dieser Vereinfachung bleibt mit den Ergebnissen von Kapitel 5.1.2.2 noch ein Teil der Aufgabe, die Rechenfähigkeiten der Prozesse zu realisieren, unbehandelt und ungelöst: Die Auswahl des Prozesses im Zustand rb, der in den Zustand rr überführt werden soll. Diese Auswahlentscheidungen, die mit dem Zeitmultiplex–Einsatz von r für die Prozesse erforderlich werden, haben wesentlichen Einfluß auf die Leistung eines Systems und müssen entsprechend nach geeigneten Kriterien und Verfahren getroffen werden. Diese Kriterien und Verfahren liefern Reihenfolgefestlegungen für das Fortschreiten der Berechnungen der Prozesse, Fahrpläne oder **Schedules** genannt, und dem entsprechend werden Scheduling–Strategien für die Prozesse und den Prozessor r benötigt. In diesem Kapitel werden einige einfache und grundlegende Fragen dieses wichtigen Themenbereichs behandelt.

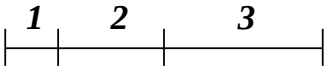
Abbildung 5.8 zeigt als Einführung die Skizze eines Modellsystems, das zur Beschreibung der gestellten Aufgabe geeignet ist und die Auswahlentscheidungen in den Zusammenhang der Realisierung der Rechenfähigkeiten der Prozesse einordnet. Sie zeigt zudem für ein Beispiel mit der Prozeßmenge $P = \{1, 2, 3\}$ zwei Schedules, die auf die Bedeutung geeigneter Strategien für die Leistung eines Systems hinweisen.

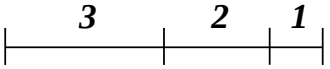


Prozesse $P = \{1, 2, 3\}$

mit Bedienzeiten $b_1 = 1, b_2 = 2, b_3 = 3$.

Reihenfolgen: $s_1 = (1, 2, 3), s_2 = (3, 2, 1)$

Schedule s_1: 

Schedule s_2: 

Bewertungen: $\overline{V}(s_1) = 10/3, \overline{V}(s_2) = 14/3$

Abbildung 5.8: Realisierung der Prozeßrechenfähigkeiten und Scheduling

Für präzise Formulierungen der gestellten Aufgaben sind **Bedienungsmodelle** geeignet. Ein Bedienungsmodell beschreibt Bediener, die Aufträge ausführen können, und Aufträge, die Anforderungen an die Bediener stellen und von diesen bedient oder ausgeführt werden. Hier sind Modelle mit einem **Bediener**, der den realen Prozessor r modelliert, von Interesse. Der Bediener kann in jedem Zeitpunkt höchstens einen Auftrag bedienen; dem entspricht, daß r ein XBM ist.

Die Aufträge stellen Anforderungen an den Bediener; sie modellieren die Prozesse, die r für ihre Berechnungen benötigen.
Als **Aufträge** eines Bedienungsmodells werden endliche oder unendliche Folgen betrachtet: $A_1, \ldots, A_n, \ldots$
Für die Aufträge werden Ankunfts– und Bedienzeiten festgelegt; dabei wird $\mathbb{R}_+$ als Zeitachse des Modells benutzt. Für A_i sind

- die **Ankunftszeit** $a_i \in \mathbb{R}_+$ der Zeitpunkt, zu dem A_i eintrifft, und
- die **Bedienzeit** $b_i \in \mathbb{R}_+$ die Dauer der Ausführung durch den Bediener, die A_i anfordert.

Die Ankunftszeiten der Aufträge werden zu $a = (a_1, a_2, \ldots)$ und die Bedienzeiten zu $b = (b_1, b_2, \ldots)$ zusammengefaßt.
Bei gegebenem Einbedienermodell mit a und b ist eine Scheduling–**Strategie** eine Vorschrift, die für jeden Zeitpunkt $\tau \in \mathbb{R}_+$ verträglich mit a und b festlegt, ob und ggf. welcher Auftrag in τ ausgeführt wird, und die gewährleistet, daß alle Aufträge vollständig ausgeführt werden.
Bei gegebenem a und b sei S eine Strategie für die Ausführung der Aufträge. Dann ist mit S

- die **Abgangszeit** $c_i(a, b, S) \in \mathbb{R}_+$ des Auftrags A_i, zu der die Bedienung vollständig abgeschlossen ist.

Mit den Aufträgen (A_i), die gemäß a eintreffen und gemäß b Anforderungen stellen, mit dem Bediener, der jeweils höchstens einen Auftrag ausführen kann, und mit einer Strategie S, die die Reihenfolge festlegt, in der die Aufträge bedient werden, ist das Verhalten eines Bedienungsmodells vollständig festlegt. Ein **Schedule** beschreibt die Belegung des Bedieners über der Zeit mit einer totalen Abbildung

$$s(a, b, S) : \mathbb{R}_+ \to \{\bot, A_1, A_2, \ldots\},$$

wobei $s(\tau) = \bot$ für $\tau \in \mathbb{R}_+$ sagt, daß der Bediener in τ nicht belegt/frei ist. Ein Auftrag A_i, der zur Zeit a_i ankommt, wird zur Zeit $\tau \geq a_i$ bedient, wenn $s(\tau) = A_i$ gilt, oder wartet, wenn $s(\tau) \neq A_i$ gilt. Für die jeweils wartenden Aufträge enthält das Bedienungsmodell einen Warteraum für Aufträge. Die Strategie S legt die Kriterien für die jeweils zur Ausführung auszuwählenden Aufträge fest und bestimmt damit das Verhalten des Systems wesentlich mit. Abbildung 5.9 zeigt das Bedienungsmodell, das sich ergibt.

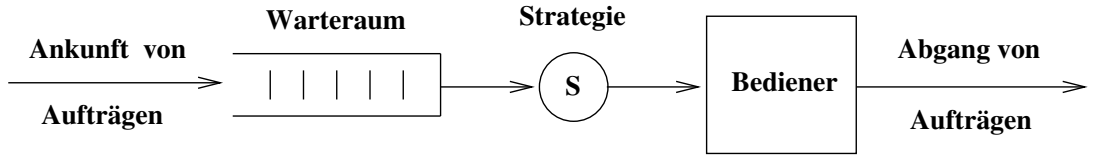


Abbildung 5.9: Ein Bedienungsmodell mit einem Bediener

Bedienungsmodelle dienen dazu, das Verhalten eines Systems bzgl. seiner **Leistung** und Leistungsfähigkeit quantitativ zu beschreiben. Die Strategien eines Systems bestimmen die Leistung wesentlich mit; daraus ergibt sich die gestellte Scheduling–Aufgabe: Die Entwicklung von Strategien, mit denen sich eine hohe Leistung oder Leistungsfähigkeit ergibt. Zur Lösung der Aufgabe sind Leistungskenngrößen erforderlich; mit diesen werden Zielfunktionen für die Leistung, die erreicht werden soll, festgelegt, und mit diesen werden geeignete Strategien als Lösungen von Optimierungsaufgaben entwickelt.

Auf der Grundlage der schon erklärten Eigenschaften des betrachteten Einbedienersystems werden im folgenden einige wichtige **Leistungskenngrößen** angegeben. Dabei werden die eingeführten Bezeichnungen weiter benutzt.
Die beiden folgenden Kenngrößen beschreiben die Arbeit, die ein System leisten kann, und die Arbeit, die von einem System gefordert und noch zu leisten ist.
Die Arbeit, die das betrachtete Einbedienersystem leisten kann, die (Arbeits–) **Kapazität** des Systems, ist die seines Bedieners; er kann in einem Zeitintervall der Länge T von den Bedienzeiten anstehender Aufträge T abarbeiten: Die Kapazität des Bedieners ist also 1. Der Bediener modelliert hier einen Prozessor mit seiner Rechenkapazität; die Kapazität des Bedieners entspricht dieser mit der hier festgelegten Zeitskala.
Die Arbeit, die von einem System gefordert und noch zu leisten ist, die (Arbeits–) **Last** des Systems, variiert mit der Zeit und wird von a, b und S bestimmt. Dazu sei

- $r_i(a,b,S;\tau)$ die **Restbedienzeit** des Auftrags A_i zur Zeit $\tau \in \mathbb{R}_+$.

Mit ihr ist die Last des Systems mit a, b und S zur Zeit $\tau \in \mathbb{R}_+$

- $L(a,b,S;\tau) \triangleq \sum\limits_{i \in \mathbb{N}: a_i \leq \tau} r_i(a,b,S;\tau)$.

Bei gegebenem a, b und S ist $L : \mathbb{R}_+ \to \mathbb{R}_+$ eine „Sägezahnfunktion“ mit Sprüngen in Ankunftszeiten und mit der Steigung -1 in Zeitintervallen, in denen Aufträge ausgeführt werden.
Die Kapazität und die Lastfunktion sind globale Leistungscharakteristika eines Systems. Die folgenden Kenngrößen sind für die Entwicklung von Strategien geeignet. Dabei sind anwendungs– und beriebsbezogene Leistungskenngrößen zu unterscheiden.
Als **anwendungsbezogene Kenngrößen** sind Verweil– und Wartezeiten von Aufträgen von Interesse; sie werden für die einzelnen Aufträge von a, b und S bestimmt.

- $v_i(a,b,S) \triangleq c_i(a,b,S) - a_i$ ist die **Verweilzeit** des Auftrags A_i;
- $w_i(a,b,S) \triangleq c_i(a,b,S) - a_i - b_i$ ist die **Wartezeit** des Auftrags A_i.

Mit den Verweilzeiten oder Antwortzeiten der einzelnen Aufträge können **mittlere Verweilzeiten** berechnet werden; sie sind die Leistungskenngrößen, die für Anwender zur Bewertung eines Systems geeignet sind. Entsprechendes gilt für die mittlere Wartezeit.
Als **betriebsbezogene Kenngröße** ist die Auslastung des Bedieners im zeitlichen Mittel von Interesse; sie wird von a, b und S bestimmt und auf der Grundlage der Schedule–Funktion $s(a,b,S) : \mathbb{R}_+ \to \{\perp, A_1, \ldots\}$ definiert. Dazu sei zunächst

- $\rho(a,b,S) : \mathbb{R}_+ \to \{0,1\}$ mit
  $\rho(a,b,S;\tau) \triangleq \begin{cases} 0 & , \text{ falls } s(a,b,S;\tau) = \perp \\ 1 & , \text{ falls } s(a,b,S;\tau) \neq \perp \end{cases}$ für alle $\tau \in \mathbb{R}_+$.

$\rho(a,b,S;\cdot)$ ist die Auslastungsfunktion des Bedieners. Mit ihr ist

- $\overline{\rho_T}(a,b,S) \triangleq \frac{1}{T} \int\limits_0^T \rho(a,b,S;\tau) d\tau$ für $T \in \mathbb{R}_+$ die **Auslastung des Bedieners** im Zeitintervall $[0,T]$.

Das bisher für Bedienungsmodelle Erklärte gibt einen Einblick in den wichtigen Themenbereich der Leistungsbewertung von Rechensystemen. Im weiteren werden einige einfache Aufgabenstellungen behandelt, die in direktem Zusammenhang mit der Realisierung der Rechenfähigkeiten der Prozesse stehen.

Permutationsstrategien

Für die Aufträge, mit denen die Anforderungen an ein Bedienungssystem gestellt werden, sind bisher endliche und unendliche Folgen zugelassen; dem entsprechen Systeme mit endlichem oder unendlichem Zeithorizont. Der einfachste Fall liegt vor, wenn die Anzahl der betrachteten Aufträge endlich ist und alle Aufträge zur Zeit 0 vorliegen.
Dieser Fall mit den Aufträgen $A_1, \ldots, A_n, n \in \mathbb{N}, a = 0$ und $b = (b_1, \ldots, b_n)$ wird im weiteren betrachtet.
Für die n Aufträge mit b ist die

- **mittlere Bedienzeit** eines Auftrags $\overline{b} = \frac{1}{n} \sum_{i=1}^{n} b_i$.

Zur Bewertung von Strategien zur Ausführung der n Aufträge mit b werden im folgenden anwendungsbezogene Leistungskenngrößen benutzt. Wenn S eine entsprechende Strategie ist, ist

- $\overline{V}(b, S) = \frac{1}{n} \sum_{i=1}^{n} v_i(b, S)$ die **mittlere Verweilzeit** eines Auftrags gemäß b und S.

Weiter ist

- $\overline{W}(b, S) = \frac{1}{n} \sum_{i=1}^{n} w_i(b, S)$ die **mittlere Wartezeit** eines Auftrags gemäß b und S.

Bei gegebenem b und S gilt

- $\overline{V}(b, S) = \overline{W}(b, S) + \overline{b}$.

Die mittlere Verweilzeit eines Auftrags gemäß b wird im folgenden als Optimierungs–Zielfunktion für die Konstruktion geeigneter Strategien benutzt. Die angegebene Gleichung sagt: Eine verweilzeitoptimale Strategie ist zugleich eine wartezeitoptimale Strategie.
Die einfachsten Strategien für Einbedienersysteme mit n gleichzeitig vorliegenden Aufträgen und Bedienzeiten gemäß b sind die **Permutationsstrategien**, für die folgendes gilt:

- Aus der Menge $\prod$ der Permutationen von $\{1, \ldots, n\}$ wird ein $\pi \in \prod$ mit $\pi = ((1), \ldots, (n))$ ausgewählt, und alle Aufträge werden in der durch π festgelegten Reihenfolge jeweils vollständig ausgeführt.

Für die Permutationsstrategie $\pi = ((1), \ldots, (n))$ kann die mittlere Verweilzeit eines Auftrags leicht berechnet werden:

- $\overline{V}(b, \pi) = \frac{1}{n} \sum_{i=1}^{n} (n + 1 - i) b_{(i)}$.

Mit dieser Formel kann ein Kriterium für verweilzeitoptimale Strategien angegeben werden; es ergibt sich wie folgt: Die Summe rechts ist ein Skalarprodukt von 2 Vektoren mit Komponenten aus $\mathbb{R}_+$, also $x \cdot y$; die Komponenten von x sind monoton fallend. Dann gilt, daß das Skalarprodukt dann minimal ist, wenn die Komponenten von y monoton wachsend sind.
Mit diesem Kriterium ergibt sich:

- Jedes $\pi = ((1), \ldots, (n))$ mit $b_{(1)} \leq b_{(2)} \leq \ldots \leq b_{(n)}$ definiert eine verweilzeitoptimale Permutationsstrategie.

Diese Strategien, die festlegen, daß jeweils der Auftrag mit der kürzesten Bedienzeit als nächster ausgeführt wird, werden als SPTF–Strategien (für Shortest Processing Time First) bezeichnet.
Neben diesem Ergebnis können mit Permutationsstrategien noch weitere wichtige Aussagen über das Verhalten von Einbedienersystemen abgeleitet werden, welche die Bedeutung und den Nutzen von Strategien für die mittlere Verweilzeit charakterisieren. Für diese Aussagen werden einerseits antithetische Permutationen und andererseits die Pseudopermutationsstrategie RANDOM betrachtet.
Zunächst ist für $\pi \in \prod$ mit $\pi = (i_1, \ldots, i_n)$ die Permutation $\pi^* = (i_n, i_{n-1}, \ldots, i_1)$ die zu π **antithetische**.
Die **Strategie RANDOM** (R) wählt die Aufträge „zufällig“ aus; das bedeutet, daß die Aufträge gemäß einer Permutation $\pi \in \prod$ ausgeführt werden, wobei π gemäß einer Gleichverteilung aus den $n!$ Permutationen ausgewählt wird. Diese Festlegung der Auswahl der Aufträge auf der Grundlage einer Wahrscheinlichkeitsverteilung hat zur Folge, daß die mittleren Verweilzeiten $V(b, \pi)$ jetzt Werte von Zufallsvariablen sind, mit denen der Erwartungswert $E[V(b, R)]$ der Verweilzeit eines Auftrags gemäß b und R berechnet wird.
Damit ergibt sich zunächst $E[V(b, R)] = \frac{1}{n!} \sum_{\pi \in \prod} V(b, \pi)$.
Zur Berechnung der Summe seien $i, k \in \{1, \ldots, n\}$. Für den Auftrag A_i mit b_i gibt es $(n-1)!$ Permutationen, in denen A_i die Position k hat; jede dieser Permutationen liefert den Beitrag $\frac{1}{n}(n + 1 - k)b_i$ zur Summe. Damit ergibt sich

$$E[V(b, R)] = \frac{(n-1)!}{n!\ n} \sum_{i=1}^{n} \sum_{k=1}^{n} (n + 1 - k)b_i.$$

Es gilt also

- $E[V(b, \mathrm{RANDOM})] = \frac{n+1}{2}\ \overline{b}$.

Die „zufällige“ Auswahl der Aufträge nach dieser Strategie bedeutet, daß die Strategie „blind“ für die Eigenschaften der Aufträge ist; sie nutzt die Information über die Aufträge (b) nicht und entspricht damit einer Auswahl, für die keine Information über die Aufträge zur Verfügung steht.
Die charakteristischen Aussagen über den Nutzen von Strategien erhält man aus dem Zusammenhang, der zwischen RANDOM und antithetischen Permutationsstrategien besteht.
Für **antithetische Permutationen** ergibt sich folgendes.
Sei π die Permutation $\pi = (1, \ldots, n)$; dann ist die zu π antithetische Permutation $\pi^* = (n, n-1, \ldots, 1)$. Für die Summe der mittleren Verweilzeiten eines Auftrags gemäß π und π^* gilt dann

$$\overline{V}(b, \pi) + \overline{V}(b, \pi^*) = (n + 1)\overline{b}.$$

Daraus ergibt sich

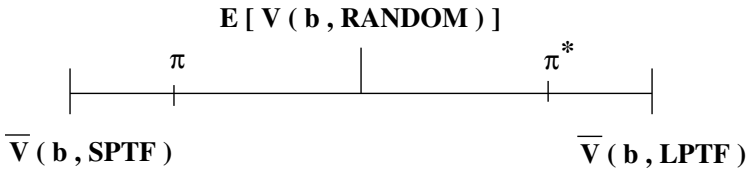
- $\overline{V}(b, \pi) + \overline{V}(b, \pi^*) = 2E[V(b, \mathrm{RANDOM})]$
  für alle Paare von antithetischen Permutationen $\pi, \pi^* \in \prod$.

Mit diesem Ergebnis können die Permutationsstrategien wie folgt eingeordnet und bewertet werden:

- Bei gegebenen Bedienzeiten b sind die mittleren Verweilzeiten eines Auftrags für jedes Paar $\pi, \pi^* \in \prod$ von antithetischen Permutationen die Endpunkte eines Intervalls, dessen Mittelpunkt $E[V(b, \mathrm{RANDOM})]$ ist.

- Die Strategien SPTF und LPTF (für Largest Processing Time First) sind antithetisch, und sie bestimmen die Endpunkte des längsten Verweilzeit–Intervalls.
- Eine Permutation $\pi \in \prod$ liefert eine Verweilzeit, die kleiner (größer) als $E[V(b, \mathrm{RANDOM})]$ ist, wenn ihre Reihenfolge positiv (negativ) mit der Monotonie der Bedienzeiten der Aufträge korreliert.

Abbildung 5.10 zeigt diese Zusammenhänge und ein Paar antithetischer Strategien für ein Beispiel.



Beispiel mit $n = 5$ und $b = (1, 3/2, 2, 4, 5)$:

$E[V(b, \mathrm{RANDOM})] = 8,1$,

$\overline{V}(b, \mathrm{SPTF}) = 6$

$\pi = ((1), (3), (4), (2), (5))$ mit $\overline{V}(b, \pi) = 6,6$

ist positiv korreliert mit den Bedienzeiten der Aufträge.

Abbildung 5.10: Das Verweilzeitintervall der Permutationsstrategien

Der erklärte Zusammenhang sagt, daß mit einer Strategie S für die Ausführung von n Aufträgen, welche die Reihenfolge nach anderen, nicht an den Bedienzeiten orientierten Kriterien festlegt, eine mittlere Verweilzeit erreicht werden kann, die größer als die bei zufälliger Auswahl ist. Das gilt insbesondere bei Reihenfolgefestlegung durch (externe) Prioritäten. Zur Verkürzung der mittleren Verweilzeit sind Kenntnisse über die Bedienzeiten der Aufträge nötig. Die Bedienzeiten müssen nicht exakt bekannt sein; die Kenntnisse müssen jedoch die geforderte positive Korrelation mit den Bedienzeiten ermöglichen.
Die Permutationsstrategien gehören zur Klasse der **nichtunterbrechenden Strategien**. Man kann leicht zeigen, daß für die hier betrachteten Systeme mit gleichen Ankunftszeiten für alle Aufträge mit unterbrechenden Strategien keine weitere Reduktion der mittleren Verweilzeit erreicht werden kann.
Die Minimierung der mittleren Verweilzeit ist jedoch nicht in allen Fällen das geeignete Optimierungsziel: SPTF bevorzugt die Aufträge mit kleinen Bedienzeiten zu Lasten der Aufträge mit großen Bedienzeiten. Es ist also geboten, neben kurzen Verweilzeiten auch Fairneß bzgl. aller Aufträge in das Optimierungsziel aufzunehmen. Mit diesen veränderten Optimierungszielen wird es notwendig, **unterbrechende Strategien** zu berücksichtigen.

Zeitscheiben–Strategien

Eine häufig benutzte Klasse von unterbrechenden Strategien sind die Zeitscheiben–Strategien. Für $q \in \mathbb{R}_+, q > 0$, legt die **Zeitscheiben–Strategie mit dem Quantum** q, S_q, folgendes fest:

- Die Aufträge werden in Zyklen bedient, und in jedem Zyklus werden alle Aufträge mit positiven Restbedienzeiten bedient;

- wenn die Restbedienzeit eines Auftrags zu Beginn eines Zyklus $r > 0$ ist, gilt für diesen Zyklus:
  - der Auftrag wird vollständig ausgeführt, falls $r \leq q$ gilt;
  - der Auftrag wird für die Dauer des Quantums q ausgeführt, falls $r > q$ gilt.

Für einen Zyklus mit den Aufträgen $A_1, \ldots, A_n$ und den Restbedienzeiten $r_1, \ldots, r_n$ gilt also mit der Strategie S_q:

- die Aufträge mit $r_i \leq q$ werden vollständig ausgeführt; sie werden „ausgesiebt“;
- die übrigen Aufträge werden für q bedient, so daß ihre Restbedienzeiten um q reduziert werden.

Zur Berechnung der mittleren Verweilzeit eines Auftrags mit S_q muß die Reihenfolge der Aufträge in jedem Zyklus berücksichtigt werden. Das ist nicht mehr der Fall, wenn ein „feines Sieb“ benutzt wird, also für $q \downarrow 0$ und die Strategie $S_{q\downarrow 0}$.
Zur Berechnung der mittleren Verweilzeit für $S_{q\downarrow 0}$ bei n Aufträgen mit den Bedienzeiten b sei o.B.d.A. die Reihenfolge der Aufträge $1, \ldots, n$ und es gelte $b_1 \leq b_2 \ldots \leq b_n$. Dann ergeben sich

$$v_1(b, S_{q\downarrow 0}) = nb_1 \ ,$$
$$v_2(b, S_{q\downarrow 0}) = nb_1 + (n-1)(b_2 - b_1) = b_1 + (n-1)b_2 \ ,$$
und allgemein für $i \in \{1, \ldots, n\}$
$$v_i(b, S_{q\downarrow 0}) = \sum_{k=1}^{i-1} b_k + (n+1-i)b_i \ .$$

Daraus folgt

$$\overline{V}(b, S_{q\downarrow 0}) = \frac{1}{n} \sum_{i=1}^{n} (n+1-i)b_i + \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i-1} b_k \ .$$

Der 1–te Summand rechts ist $\overline{V}(b, \mathrm{SPTF})$; vom 2–ten Summanden gilt $\sum_{k=1}^{i-1} b_k = w_i(b, \mathrm{SPTF})$.
Daraus folgt für die mittlere Verweilzeit bei der Strategie $S_{q\downarrow 0}$:

- $\overline{V}(b, S_{q\downarrow 0}) = \overline{V}(b, \mathrm{SPTF}) + \overline{W}(b, \mathrm{SPTF})$.

Mit dem bekannten Zusammenhang zwischen Warte– und Verweilzeiten kann man dieses Ergebnis auf die mittleren Wartezeiten übertragen; das liefert

- $\overline{W}(b, S_{q\downarrow 0}) = 2\overline{W}(b, \mathrm{SPTF})$.

Diese Ergebnisse quantifizieren die Zuwächse zur minimalen mittleren Verweilzeit oder Wartezeit, die sich durch zusätzliche Berücksichtigung der Fairneß ergeben.
Die Zeitscheiben–Strategien mit dem Quantum q werden als Round Robin–Strategien (RR–) und die mit $q \downarrow 0$ als Processor Sharing–Strategie (PS–) bezeichnet.
Für eine Zeitscheiben–Strategie S_q sind keine a priori–Kenntnisse über die Bedienzeiten der Aufträge erforderlich. S_q „siebt“ die Aufträge ihren Bedienzeiten entsprechend und erreicht damit einen Kompromiß zwischen kurzen Verweilzeiten und Fairneß. Die zur Reduktion der Verweilzeiten notwendige Orientierung an den Bedienzeiten ist perfekt für $q \downarrow 0$.
Die Kombination der beiden Ziele kurze Verweilzeiten und Fairneß, die der Optimierungs–Zielfunktion der Zeitscheiben–Strategien entspricht, ist typisch für die Anforderungen, die an

Scheduling–Zielfunktionen gestellt werden. Die Konsequenzen, die sich dann ergeben, wenn allein die Verweilzeiten minimiert werden, sind oben schon aufgezeigt. Entsprechendes ergibt sich, wenn allein nach Fairneß im Sinn von Gleichbehandlung aller Aufträge optimiert wird. Dann wird erreicht, daß die Verweilzeiten aller Aufträge gleich sind, nämlich $n \cdot \overline{b}$, und damit sind die Wartezeiten aller Aufträge maximal.

Scheduling und Dispatching

Wie am Anfang dieses Kapitels erklärt, sind die Scheduling–Strategien für die Dispatcher–Kerndienste zur Realisierung der Rechenfähigkeiten der Prozesse, also für *binden* und *lösen*, erforderlich. Die erklärten Zeitscheiben–Strategien sind Beispiele dafür, daß neben *binden* auch die Zustandsüberführung *lösen* für unterbrechende Strategien notwendig ist.
Bei der **Integration der Strategie– und Umschaltmaßnahmen** in die Kerndienste sind folgende Aspekte zu beachten:

- Die vorrangigen Anforderungen an die Umschaltdienste sind
  - die Gewährleistung der relativen funktionalen Wirkungsfreiheit für die Prozesse und
  - kurze Ausführungszeiten zur Gewährleistung des effizienten Einsatzes des realen Prozessors.
- Die vorrangige Anforderung an die Strategie für die Auswahlentscheidungen ist möglichst optimales Verhalten bzgl. eines festgelegten Leistungskriteriums, wobei zu berücksichtigen ist, daß die Auswahlentscheidungen ihrerseits, wie erklärt, Berechnungen erfordern.

Diese gegeneinander gerichteten Anforderungen müssen durch abgestufte und aufeinander abgestimmte Maßnahmen erfüllt werden. Die kurzen Ausführungszeiten für die Umschaltdienste werden dadurch erreicht, daß die Aufträge in der Reihenfolge, in der sie auszuführen sind, geordnet bereitgestellt werden; das erfolgt mittels der *rb*–**Warteliste**, welche die Prozesse im Zustand *rb* in der Reihenfolge enthält, in der sie auszuwählen sind. Die strategischen Entscheidungen über die Reihenfolgen werden, soweit möglich, von den Umschaltmaßnahmen abgetrennt vorab getroffen, und zwar mit zeitlichen Granularitäten, die dem Aufwand für die Entscheidungen entsprechen. Das führt zu einer mehrstufigen Hierarchie der Scheduling–Entscheidungen und der Datenobjekte, mit denen sie verwirklicht werden. Die Umschalt– und Scheduling–Maßnahmen müssen aufeinander abgestimmt sein, weil die Umschaltdienste das Repertoire der Scheduling–Maßnahmen mitbestimmen. Die erklärten Zeitscheiben–Strategien sind gute Beispiele für diese notwendige Abstimmung von Umschalt– und Scheduling–Maßnahmen.
Nach dem Erklärten sind die in Kapitel 5.1.2.2 erklärten Kerndienste für *binden* und *lösen* so zu erweitern, daß sie Scheduling–Maßnahmen, soweit erforderlich, einbeziehen. Das wird im folgenden für **Zeitscheiben–Strategien** erklärt: Mit dem Quantum q soll also die Strategie S_q mit den Umschaltdiensten kombiniert realisiert werden. Nach dem oben bereits Erklärten läßt sich die Lösung, die verwirklicht werden soll, gemäß Abbildung 5.11 veranschaulichen.
Für die notwendige Anpassung wird zunächst die *rb*–Warteliste wie oben erklärt für die Prozesse im Zustand *rb* eingeführt. Mit ihr ergibt sich, daß der für *binden* auszuwählende Prozeß jeweils der 1–te in *rb*–Warteliste ist. Ein Prozeß, dessen Berechnungsausführung unterbrochen wird, weil seine Zeitscheibe endet, wird als letzter mit *lösen* in die *rb*–Warteliste eingefügt. Es bleibt also noch, die Zeitscheiben zu realisieren.
Zur **Realisierung der Zeitscheiben** müssen die bisher betrachteten Fähigkeiten des Prozessors r um einen Wecker erweitert werden. Ein **Wecker** ist ein einfacher Spezialprozessor

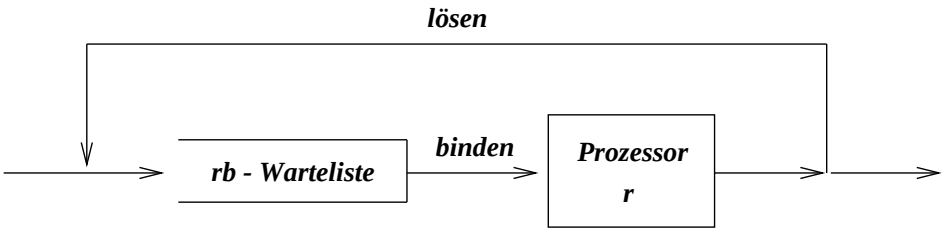


Abbildung 5.11: Realisierung der Strategie S_q

(Zähler), der fähig ist, die Operationen *starte* und *lies* parallel zu r auszuführen. Der Wecker führt, kontrolliert von r, Aufträge aus, die mit r erteilt werden. Die Operationen sollen folgende Wirkung haben:

- *wecker.starte*(t) mit einer positiven Ganzzahl t als Eingabe bewirkt, daß der Zähler mit t initialisiert und das Dekrementieren des Zählers gestartet wird; wenn der Zählerstand 0 erreicht ist, stoppt das Dekrementieren und der Wecker sendet ein Unterbrechungssignal an r.

- *wecker.lies* bewirkt, daß der Zählerstand gelesen (eine nichtnegative Ganzzahl als Ausgabe) und das Dekrementieren gestoppt wird.

Zur Realisierung der Zeitscheiben–Strategie mit dem Quantum q wird, wenn ein Prozeß p mit *binden* vom Zustand *rb* in *rr* überführt wird, der Wecker mit *starte*(q) gestartet. Das Unterbrechungssignal, das der Wecker bei Quantumsende erzeugt, wird von r im Befehlszyklus wahrgenommen und bewirkt eine Programmausführungs–Unterbrechung für r; das ist die Unterbrechung der Ausführung der Berechnungen des Prozesses p. Mit dem Programm der Wecker–Unterbrechungsreaktion wird p mit *lösen* in den Zustand *rb* überführt, also wie erklärt in die *rb*–Warteliste eingefügt. Die Alternative der Zeitscheiben–Strategie zu Quantumsende für den Prozeß p, also der Fall, in dem die Restbedienzeit von p kleiner q ist, tritt hier dann auf, wenn p im Zustandsdiagramm in t übergeht. In diesem Fall, in dem der Prozeß p terminiert, dient *wecker.lies* dazu, die Verweilzeit von p abzurechnen.
Oben ist erklärt, daß die Dispatching– und Scheduling–Maßnahmen aufeinander abzustimmen sind und eine mehrstufige Hierarchie der Scheduling–Entscheidungen notwendig ist. Das für die Zeitscheiben–Strategie Erklärte zeigt, wie die Abstimmung erreicht wird:

- Mit der *rb*–Warteliste wird erreicht, daß die Umschaltdienste effizient ausgeführt werden können.

- Mit der Erweiterung des Prozessors r um den Wecker werden unterbrechende Strategien realisierbar.

Eine Hierarchie der Scheduling–Entscheidungen ergibt sich, indem die erklärte Realisierung einer Zeitscheiben–Strategie zu einer mehrstufigen verallgemeinert wird; Abbildung 5.12 veranschaulicht das Ergebnis dieser Verallgemeinerung. Die Stufen der Hierarchie können dazu genutzt werden, den „Siebeffekt“ der Zeitscheiben–Strategien zu verallgemeinern und den Umschaltaufwand zu reduzieren, indem bei k Stufen die Quanten mit $q_1 < q_2 \ldots < q_k$ festgelegt werden. Die Hierarchie, die sich so ergibt, ist ein kleiner Teil der Scheduling–Hierarchie, die erforderlich ist, wenn zu dem realen Prozessor die weiteren Betriebsmittel, die Prozesse benötigen, erfaßt werden.

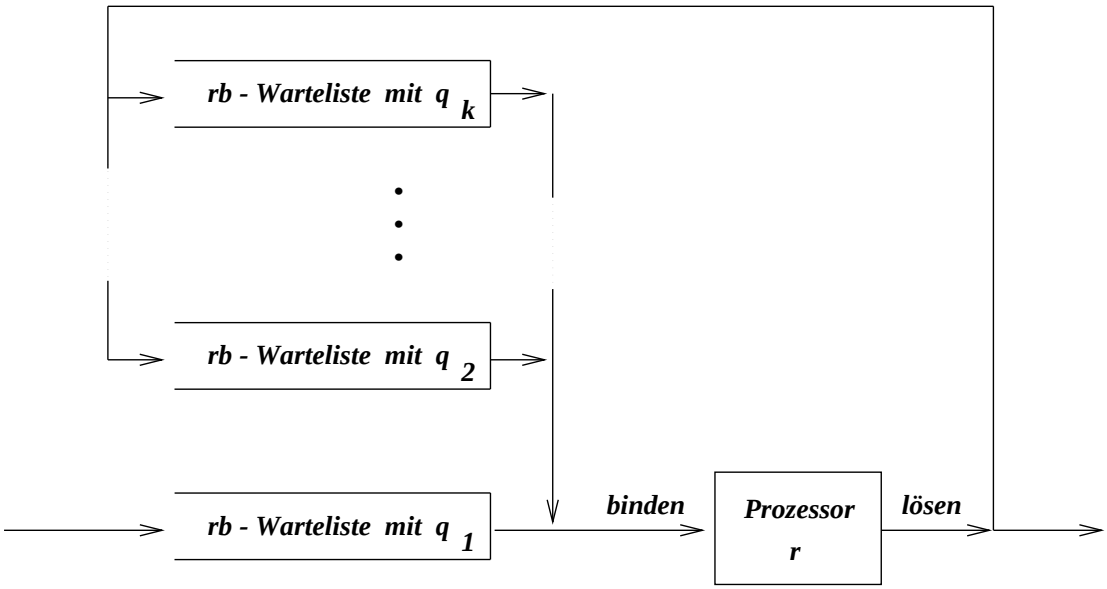


Abbildung 5.12: Realisierung der k–stufigen Zeitscheiben–Strategie

5.1.2.4 Realisierung der Prozeß–Speicherfähigkeiten

Von den für die Prozesse als Managementobjekte in Kapitel 5.1.1 konzeptionell festgelegten Eigenschaften werden bisher lediglich die Rechenfähigkeiten sowie die Speicherfähigkeiten der abstrakten Prozessoren berücksichtigt; zusätzlich verfügt jeder Prozeß über einen abstrakten Speicher. Für einen Prozeß p ist festgelegt, daß der Speicher das Programm, das die Berechnungen von p spezifiziert, und die zugehörigen Daten enthält. Damit und mit dem Prozessor von p ergibt sich der p–private Teil des Wirkungsbereichs der p–Berechnungen, der die Grundlage der Gewährleistung der Interferenzfreiheit für p ist. Der Prozeß p ist an den Kern gebunden, und die Berechnungen des Prozesses werden vom p–Prozessor unter Nutzung der Kerndienste ausgeführt. Für die Berechnungen des Prozesses p ergibt sich damit ein 2–geteilter Wirkungsbereich mit dem p–privaten Teil einerseits und dem (varianten) p–spezifischen Kernteil andererseits sowie mit den charakteristischen Umgebungswechseln, die Übergängen zwischen diesen beiden Teilen entsprechen. Diese Festlegungen bestimmen die Speicherfähigkeiten des Prozesses p, und sie liefern zugleich die Anforderungen an Realisierungen dieser Speicherfähigkeiten.
Dem Zweck der Managementobjekte entsprechend, systematische Übergänge zwischen konzeptionell festgelegten, abstrakten Eigenschaften und von der eingesetzten HK bestimmten Eigenschaften zu ermöglichen und zu vereinfachen, ergeben sich für die Aufgabe der Realisierung der Prozeß–Speicherfähigkeiten wieder zwei Sichten, die verträglich miteinander kombiniert werden müssen. Diese Aufgabe zerlegt man zweckmäßig in zwei weitgehend getrennt lösbare Teilaufgaben. Diese Aufteilung erfolgt für klassische Betriebssysteme mit dem Konzept der **prozeßspezifischen virtuellen Adreßräume**. Nach diesem Konzept wird jedem Prozeß p ein virtueller Adreßraum $VA(p)$ zugeordnet, wobei die Adreßräume aller Prozesse strukturell gleich sind. Für einen Prozeß p wird mit $VA(p)$ ein abstrakter Speicher $VM(p)$, der virtuelle Speicher des Prozesses p, assoziiert. Damit ergeben sich die beiden Teilaufgaben der Realisierung der Prozeß–Speicherfähigkeiten wie folgt:

- Für jeden Prozeß p ist der Inhalt des abstrakten Speichers $VM(p)$ geeignet, also den Anforderungen für die p–Berechnungen entsprechend, festzulegen.

- Für die Menge p der koexistierenden Prozesse und für den Kern als gemeinsame Basis aller Prozesse sind die abstrakten Speicher mit den Speichern der HK so zu realisieren, daß die Prozesse ihre Berechnungen mit ihren 2–geteilten Wirkungsbereichen tatsächlich ausführen können.

Diese beiden Teilaufgaben, für die es viele Lösungsmöglichkeiten und –verfahren gibt, werden in folgenden Kapiteln behandelt. Hier wird lediglich noch der grundlegende Beitrag des Konzepts der prozeßspezifischen virtuellen Adreßräume zur gestellten Realisierungsaufgabe erklärt.
Von dem in Kapitel 5.1.1 angegebenen Prozeßkonzept für Realisierungen ist wesentlich und charakteristisch, daß ein Konzept, das Rechen– und Speicherfähigkeiten zusammenfaßt, für sequentielle Berechnungen festgelegt ist. Dieser Zusammenfasssung entsprechend ist ein Prozeß eine Maschine mit Prozessor uns Speicher. Das Konzept der prozeßspezifischen virtuellen Adreßräume entspricht dieser Zusammenfasssung und ergänzt sie.
Das Konzept legt zunächst einen **virtuellen Adreßraum** VA für alle Prozesse eines Systems fest. Für VA gilt durchweg $VA = \{0, \ldots, 2^{32} - 1\}$. Das Konzept legt weiter fest, daß jedem Prozeß p dauerhaft der **prozeßspezifische virtuelle Adreßraum** $VA(p)$ zugeordnet wird. Diese Zuordnung sagt, daß p der Bedeutungskontext für die $va \in VA(p)$ ist. Wenn also p und q zwei koexistierende Prozesse und $va \in VA$ sind, dann sind (p, va) und (q, va) zwei numerische Bezeichner, die Bedeutungen haben können, und zwar (p, va) für p und (q, va) für q.
Für einen Prozeß p, dem $VA(p)$ zugeordnet ist, sind die $va \in VA(p)$ zunächst interpretationsgeeignete, bedeutungslose **numerische Bezeichner**. Interpretationsgeeignet sagt, daß der reale Prozessor r dazu fähig ist, die $va \in VA(p)$ in Standardrepräsentation zu interpretieren. Bedeutungslos sagt, daß den $va \in VA(p)$ zunächst keine Bedeutung zugeordnet ist; sie erhalten ihre Bedeutung durch entsprechende Zuordnungen, die noch erfolgen müssen. Die va ohne bzw. mit zugeordneter Bedeutung werden als undefinierte bzw. definierte $VA(p)$–Elemente bezeichnet.
Die Bedeutungen, die den $VA(p)$–Elementen zugeordnet werden können, sind Adressen von Bytezellen eines abstrakten Speichers; dem entspricht, daß $VA(p)$ als **virtueller Adreßraum** bezeichnet wird. Der Unterscheidung zwischen undefinierten und definierten $VA(p)$–Elementen wird mit dem abstrakten Speicher, der dem Prozeß p assoziiert wird, Rechnung getragen: Der **virtuelle Speicher** $VM(p)$, der dem Prozeß p assoziiert ist, ist der abstrakte Speicher, der dem definierten Teil von $VA(p)$ entspricht. Die Zellen von $VM(p)$ sind Container mit den Eigenschaften, die denen der Bytezellen des Arbeitsspeichers der HK entsprechen. Die Container dienen dazu, Zustandsobjekte zu realisieren, und diese Objekte werden, wie üblich, mit den (virtuellen) Adressen ihrer Container identifiziert.
Diese Festlegungen sind die Grundlage der Aufteilung der Realisierungsaufgabe der Prozeß–Speicherfähigkeiten in die beiden oben angegebenen Teilaufgaben. Die Teilaufgaben können weitgehend getrennt gelöst werden; die zur Lösung eingesetzten Konzepte und Maßnahmen müssen jedoch, weil abstrakte und reale Eigenschaften überbrückt werden sollen, aufeinander abgestimmt sein. Zu den Konzepten, die abzustimmen sind, gehören insbesondere Strukturierungskonzepte, die für beide Seiten geeignet festgelegt werden müssen.
Für einen Prozeß p mit $VA(p)$ besteht die erste Teilaufgabe darin, den Inhalt des dem definierten Teil von $VA(p)$ entsprechenden $VM(p)$ geeignet festzulegen. $VM(p)$ ist der Speicher des Wirkungsbereichs der p–Berechnungen, und dieser Wirkungsbereich ist, wie bekannt, zweigeteilt. Diese 2–teilung, die der Bindung des Prozesses p an den Kern entspricht, erfordert Ergänzungen zu dem bisher von $VA(p)$ und $VM(p)$ Erklärten. Dazu wird $VA(p)$ in die beiden Teile $VAP(p)$, den privaten Teil, und $VAK(p)$, den Kernteil, zerlegt. Diese Zer-

legung erfolgt für alle Prozesse gleich. Die wesentliche Ergänzung besteht darin, daß der Prozeß p $VAK(p)$ einschließlich $VMK(p)$ erbt, während $VAP(p)$ mit p und für p erzeugt wird. Das oben für Bedeutungszuordnungen Gesagte gilt entsprechend **eingeschränkt auf die** $VAP(p)$**–Elemente**. Die $VAK(p)$–Elemente sind dagegen dem Kern entsprechend undefiniert bzw. definiert, und p erbt $VMK(p)$. $VMK(p)$ ist der abstrakte Speicher des Kerns und sein Inhalt ist die Zusammenfasssung der Objekte des Kerns, der die gemeinsame Basis aller Prozesse ist. Damit ergibt sich der **abstrakte Speicher** VMK **des Kerns** mit dem Adreßraum VAK und der Kern, der als Inhalt dieses Speichers Inhalt der $VMK(p)$ aller koexistierenden Prozesse p ist, und dessen Objekte mit $VAK(p)$–Elementen identifiziert und damit benutzt werden können.
Damit sind die Rahmenbedingungen für Realisierungen der Prozeß–Speicherfähigkeiten erklärt; der 2–teilung der Prozeß–Wirkungsbereiche entsprechend beziehen sie den Kern mit ein. Abbildung 5.13 veranschaulicht das Erklärte zunächst aus der Sicht eines Prozesses und dann mit der Sicht auf ein System mit der Prozeßmenge P.

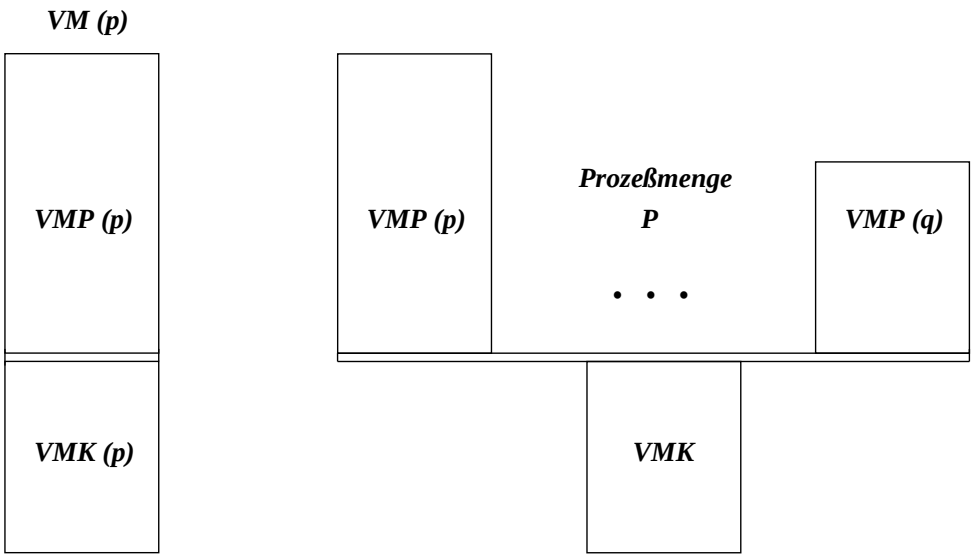


Abbildung 5.13: Die Prozeß–Speicherfähigkeiten mit dem integrierten Kern

Ein Prozeß p erhält mit dem prozeßspezifischen Adreßraum $VA(p)$ den Raum der Bezeichner, die ihm zur Identifikation von Objekten zur Verfügung stehen, und den assoziierten Speicher $VM(p)$, der die jeweils definierten Objekte einschließlich der des Kerns enthält. Die 2–teilung von $VM(p)$ in den p–privaten Teil $VMP(p)$ und den Kernteil VMK entspricht der 2–teilung des Wirkungsbereichs der p–Berechnungen und dem **Grenzwall** zwischen diesen beiden Teilen. Die Wirksamkeit dieses Grenzwalls wird, wie in Kapitel 5.1.2.1 erklärt, mit dem realen Prozessor r mittels der Betriebs–Modi und der festgelegten Modus–Wechsel für die Umgebungswechsel zwischen den beiden Wirkungsbereichsteilen der p–Berechnungen gewährleistet. Diese Kontrollmechanismen des Prozessors r bestimmen damit die **Prozeß–Kern–Schnittstelle** und die Regeln für das Binden des Prozesses p an den Kern sowie für die Nutzung von Kerndiensten durch p.
Die Prozeß–Kern–Schnittstelle ist jedoch, wie in Kapitel 5.1.2.2 erklärt, auch die Schnittstelle, mit der die wesentlich verschiedenen **Programmkonzepte**, die der Anwendungsprogramme und die der Prozessorprogramme, zusammentreffen und überbrückt werden müssen. Diese Überbrückung erfolgt einerseits dadurch, daß für die privaten Teile VAP der Adreßräume

der Prozesse und der assoziierten Speicher *VMP*, die für Anwendungsprogramme benötigt werden, einheitliche Strukturen für maschinennahe Berechnungsobjekte festgelegt werden. Die Überbrückung erfolgt andererseits dadurch, daß sowohl prozeßseitig im Rahmen der maschinennahen Berechnungsobjekte (mit Systemdienst–Standardbibliotheks–Modulen) als auch kernseitig Anpassungen vorgenommen werden. Die Kerndienste, die Prozesse nutzen, werden von den Prozessen, wie bereits erklärt, zum einen Teil implizit und zum anderen Teil explizit aufgerufen. Die genannten Überbrückungsanpassungen sind für die Kerndienste erforderlich, die von den Prozessen explizit genutzt und aufgerufen werden. Für diese Kerndienste, die für Anwendungen wichtig sind, weil alle potentiell kritischen Datenobjekte eines Systems dem Kern zugeordnet sind, ergibt sich damit die **prozedurale Prozeß–Kern–Schnittstelle**, die für die klassischen prozedurorientierten Betriebssysteme charakteristisch ist.

5.2 Erweiterungen zu einem anwendungsgeeigneten Mehrbenutzersystem

In Kapitel 5.1 sind die grundlegenden Konzepte für den Übergang von einem abstrakten zu einem realen parallelen System und die Rahmenstruktur des Systems, das sich damit ergibt, erklärt. Im weiteren ist dieser Rahmen so auszufüllen und zu ergänzen, daß sich ein anwendungsgeeignetes, paralleles Mehrbenutzersystem ergibt.
Diese notwendigen Erweiterungen können in zwei Bereiche aufgeteilt werden. **Der erste** dieser **Erweiterungsbereiche** ergibt sich daraus, daß von der zentralen HK, die als Realisierungsbasis eingesetzt wird, bisher lediglich der Prozessor und der Arbeitsspeicher mit den Eigenschaften berücksichtigt wurden, welche unmittelbar für Realisierungen der Prozeß–Rechenfähigkeiten und –Speicherfähigkeiten unter Einschluß des Kerns erforderlich sind. Die HK ist jedoch weit vielfältiger. Zu ihr gehören wesentlich die schon erwähnte Speicherhierarchie und die EA–Konfiguration. Diese Teile der HK werden wesentlich dazu benötigt, ein System mit den bisher erklärten Eigenschaften zu einem anwendungsgeeigneten System zu erweitern. Sie gehören zu den Themenbereichen Speicher– und EA–Management, die in den folgenden Kapiteln behandelt werden.
Mit dem Speicher– und dem EA–Management werden die Rechen– und die Speicherfähigkeiten eines Systems wesentlich erweitert. Zu den erweiterten Speicherfähigkeiten gehört insbesondere, daß Datenobjekte als Dateien langfristig und über Betriebspausen hinweg persistent gespeichert werden können. Dem entsprechend ist das Dateimanagement ein wesentlicher Bestandteil des Speichermanagements, und die Dateien sind die Datenobjekte, mit denen die Speicherfähigkeiten eines Systems Anwendern i.w. zur Verfügung gestellt werden.
Speicher– und EA–Management sind bei einem klassischen Betriebssystem in den Kern integriert. Das hat zur Folge, daß sowohl die Operationen auf Dateien als auch die EA–Operationen als Kerndienste zur Verfügung gestellt werden und genutzt werden müssen. Das führt für die in Kapitel 5.1 erklärte Rahmenstruktur, die nach wie vor Bestand hat, zu Schwerpunktverlagerungen. Die Kerndienste für Datei– und EA–Operationen werden zu gewichtigen Teilen der Prozeßberechnungen mit der Konsequenz der Zentralisierung dieser Berechnungen. Andererseits ergibt sich mit den in den Kern integrierten Dateien, die Programm–Module sein können, die Möglichkeit, Prozesse für die jeweils auszuführenden Berechnungen zu erzeugen und wieder aufzulösen, wenn sie nicht mehr benötigt werden.
Diese Erweiterungen und Schwerpunktverlagerungen werden in den folgenden Kapiteln behandelt. Sie liefern die Voraussetzungen für den **zweiten Erweiterungsbereich**, mit dem Anwendern geeignete Arbeitsmöglichkeiten für ihre Problemlösungen zur Verfügung gestellt

werden. Das erfolgt zunächst mit Prozessen als Pförtnern für Zugangskontrollen zu einem Mehrbenutzersystem und dann mit Prozessen, die jedem berechtigten Benutzer seine Maschine mit den benötigten Fähigkeiten und mit den erforderlichen Benutzungsschnittstellen bereitstellen.

Kapitel 6

Speicher– und EA–Management

Mit den Themen, die in diesem Kapitel behandelt werden, wird die in den vorangehenden Kapiteln entwickelte Rahmenstruktur eines einfachen parallelen Systems so ausgefüllt und erweitert, daß sich ein einsatzfähiges, anwendungsgeeignetes paralleles Mehrbenutzersystem ergibt.

Der erste, notwendige Schritt hierfür besteht darin, die als Basis für Realisierungen eingesetzte HK, von der bisher lediglich die unverzichtbaren und bestimmenden Komponenten Prozessor und Arbeitsspeicher berücksichtigt wurden, den Zielen entsprechend zu erweitern. Die **eingesetzte HK** besteht jetzt aus der RM–Konfiguration und aus der EA–Konfiguration sowie aus Verbindungen zwischen den Komponenten dieser Konfigurationen, die Nachrichtentransporte und Zusammenarbeit zwischen diesen ermöglichen. Die **RM–Konfiguration** ist die in Kapitel 5.1.1.1 erklärte; sie besteht i.w. aus dem universellen Rechenprozessor (R–Prozessor) und dem Arbeitsspeicher. Die **EA–Konfiguration** besteht aus den benötigten EA–Geräten sowie den für ihre Nutzung erforderlichen EA–Steuerwerken und EA–Prozessoren. Allen Geräten der EA–Konfiguration ist gemeinsam, daß der R–Prozessor nicht unmittelbar zu ihnen zugreifen kann; für ihre Nutzung ist Zusammenarbeit zwischen dem R–Prozessor und den Steuerwerken und/oder Prozessoren erforderlich. Bis auf diese Gemeinsamkeit sind die EA–Geräte wesentlich verschieden; sie überdecken das breite Spektrum der Komponenten, die zur Erweiterung des betrachteten einfachen Systems zu einem einsatzfähigen erforderlich sind. Ihren verschiedenen Eigenschaften entsprechend werden die Geräte in zwei Klassen, die EA–Speicher und die echten EA–Geräte, aufgeteilt. Zu den Geräten der Klasse der **EA–Speicher** gehören die elektromechanischen Speicher, also die (magnetischen oder optischen) Plattenspeicher und die Magnetbandspeicher. Mit ihnen wird die Speicherhierarchie des Systems gebildet. Die **echten EA–Geräte** dienen zum Nachrichtenaustausch zwischen dem System und seiner Umgebung. Zu ihnen gehören Terminals, Drucker, Scanner u.a. Dieser Klassenaufteilung der Geräte entspricht eine Aufteilung der EA–Konfiguration.

Die EA–Geräte sind dazu fähig, elementare Operationen auszuführen; d.s. i.w. Lese– und/oder Schreiboperationen für Bytes oder Bytefolgen. Diese Operationen stehen dem R–Prozessor nicht unmittelbar zur Verfügung; er muß vielmehr, wie schon angegeben, mit den entsprechenden EA–Steuerwerken und/oder EA–Prozessoren zusammenarbeiten. Der **R–Prozessor steuert und kontrolliert** in Zusammenarbeit mit den Steuerwerken und/oder Prozessoren **alle EA–Operationen**, die mit den Geräten ausgeführt werden, mit unterschiedlichen, an die einzelnen Geräte angepaßten Verfahren. Das können **Polling–Verfahren** mit Auftragserteilungen und periodischen Fertigmeldungsabfragen oder **unterbrechungsgesteuerte Verfahren** mit Auftragserteilungen, nebenläufig zu R–Berechnungen von den Steuerwerken und/oder Prozessoren gesteuerten und kontrollierten Auftragsausführungen und R unterbrechenden Fertigmeldungen sein. Die Wahl der Verfahren und der eingesetzten Kom-

ponenten der EA–Konfiguration erfolgt i.w. nach den Raten der zu transportierenden Bytes. Mit der Wahl ergeben sich **einfache oder mächtige EA–Operationen**, die dem R–Prozessor zur Verfügung stehen und von diesem wie erklärt zu steuern und zu kontrollieren sind. Mit diesen EA–Operationen werden Bytefolgen zwischen dem Arbeitsspeicher und den EA–Geräten transportiert. Für das Weitere wird angenommen, daß unterbrechungsgesteuerte Verfahren für alle EA–Operationen angewandt werden. Weiter wird angenommen, daß mit den EA–Operationen für echte EA–Geräte Einzelbyte–Folgen transportiert werden, während mit den EA–Operationen für EA–Speicher Bytefolgen fester oder variabler Längen, Blöcke genannt, transportiert und EA–Prozessoren eingesetzt werden. Für den R–Prozessor ergeben sich damit zwei Klassen von EA–Operationen, die als **Zeichen–EA** und als **Block–EA** bezeichnet werden.
Von der jetzt erklärten HK bilden die Speicher der RM–Konfiguration und die EA–Speicher mit den entsprechenden Komponenten der EA–Konfiguration die **Speicherhierarchie** des Systems. Die Speicher dieser Hierarchie zerfallen bzgl. des R–Prozessors zunächst in zwei Klassen, in die Speicher, zu denen der R–Prozessor direkt zugreifen kann, und in die EA–Speicher. Die Speicher der ersten Klasse bilden mit den Registern des R–Prozessors, mit dem R–Prozessor zugeordneten Zwischenspeichern — Caches genannt — und mit dem Arbeitsspeicher eine 3–stufige Hierarchie, von der im weiteren — abgesehen von den schon in Kapitel 5 erklärten Registern — allein der **Arbeitsspeicher AS** wichtig ist und berücksichtigt wird. Die EA–Speicher bilden ihrerseits eine mehrstufige Hierarchie; die Speicher werden zu einer 2–stufigen Hierarchie bestehend aus dem Hintergrundspeicher HS und dem Archivspeicher zusammengefaßt. Der **Hintergrundspeicher HS** besteht aus einer Menge H von Plattenspeichern, wobei für jedes $h \in H$ gilt: h besteht aus Blockbereichen zur Speicherung von Blöcken fester Länge; die Blockbereiche werden mit Adressen identifiziert und sind für Lese– und Schreiboperationen wahlfrei zugreifbar. Der **Archivspeicher** besteht aus weiteren Speichern mit insgesamt großer Kapazität; er wird jedoch im weiteren nicht berücksichtigt.

Mit dieser im groben erklärten, erweiterten HK als Realisierungsbasis eines Systems werden im folgenden das Speicher– und das EA–Management erklärt. Das Speichermanagement hat eine zweigeteilte Aufgabe: Es stellt persistente Datenobjekte zur Verfügung, mit denen ein System seine langfristigen, Betriebspausen überdauernden Speicherfähigkeiten erhält; das ist für klassische Betriebssysteme eine Aufgabe des Dateimanagements. Der zweite Teil ist das Speichermanagement für die Ausführung von Berechnungen; das ist mit den in Kapitel 5.1.2.4 erklärten Konzepten für die Speicherfähigkeiten der Prozesse die Aufgabe des *VA/VM*–Managements, das so zu erweitern ist, daß die Prozesse auch persistente Datenobjekte nutzen und verändern können. Mit dem EA–Management wird erreicht, daß Prozesse mit ihren Berechnungen auch EA–Operationen ausführen können. Damit wird der Austausch von Nachrichten zwischen dem System und seiner Umgebung möglich. Die Prozesse können mit den nötigen Schnittstellen für Interaktionen mit Benutzern des Systems ausgestattet und als Benutzermaschinen zur Verfügung gestellt werden.

6.1 Die statischen Speicherfähigkeiten — Das Dateisystem der persistenten Datenobjekte

Im folgenden werden die statischen Speicherfähigkeiten eines Systems erklärt. Im Gegensatz zu allem bisher für Rechen– und Betriebssysteme Erklärten werden hier keine Berechnungen betrachtet; es geht um die Eigenschaften, die ein Rechensystem hat, wenn keine Berechnungen ausgeführt werden, also in Betriebspausen und in Phasen, in denen das System nicht

nutzbar ist. In diesen Phasen liegt ein statisches System vor, und dieses System soll Speicherfähigkeiten haben. Die Sicht auf das statische, abstrakte System zeigt Datenobjekte, die gespeichert sind und für die gefordert ist, daß sie unverändert erhalten bleiben. Das sind die **persistenten Datenobjekte** des statischen Systems, das Speicherfähigkeiten hat.
Für klassische Betriebssysteme sind die persistenten Datenobjekte die Dateien des Dateisystems, das vom **Dateimanagement** für diesen Zweck und für andere Zwecke zur Verfügung steht. Die Persistenzeigenschaften, die gewährleistet werden, sind schwach. Sie sagen, daß die Menge der persistenten Datenobjekte vollständig und konsistent auf Speichern, deren Inhalt bei Stromausfall erhalten bleibt, gespeichert sein soll. Als Speicher mit diesen Eigenschaften stehen hier die Platten von HS zur Verfügung. Die angegebene Einordnung der persistenten Datenobjekte in das Dateimanagement hat zur Folge, daß die hier zu erklärenden statischen Speicherfähigkeiten ein Teil der Eigenschaften des Dateisystems sind.

6.1.1 Bytefolgen–Dateien

Die Dateien klassischer Betriebssysteme sind einfache, universell nutzbare, hardwarenahe Datenobjekte. Es sind einfache Managementobjekte zur Überbrückung der Eigenschaften, die einfache, universelle Datenobjekte als Elemente von Berechnungs–Wirkungsbereichen haben sollen, und der Eigenschaften, welche die Speicher der HK, hier einerseits der AS und andererseits die Speicher von HS, für Datenobjekte zur Verfügung stellen. Diese Datenobjekte sind Bytefolgen.
Dem entsprechend ist eine Bytefolgen–Datei (BFD) oder einfach eine **Datei** ein Datenobjekt, das mit einem festgelegten Bezeichner identifiziert wird, und dessen Wert jeweils eine Folge von Bytes ist, deren Länge in festgelegten Grenzen variieren kann.
Das Dateimanagement legt das Konzept für BFD (oder allgemeiner Dateikonzepte) fest. Dazu gehören Festlegungen der Eigenschaften der abstrakten Datenobjekte mit Diensten für Dateien und Realisierungsfestlegungen. Die **Dienste für Dateien** sind Dienste zur Erzeugung neuer Dateien sowie Dienste zur Nutzung und zur Auflösung existierender Dateien. Mit der Erzeugung einer neuen Datei ist ihr Bezeichner zur Identifikation eindeutig festzulegen. Die Dienste zur Nutzung einer existierenden Datei sind Dienste zum Lesen und Schreiben von Bytefolgen relativ zu einer Startposition, Dienste zur Festlegung einer Startposition und der Länge sowie Dienste zum Öffnen und Schließen der Datei mit der Festlegung, daß die Aufrufe der sonstigen Dienste zur Nutzung mit Öffnen und Schließen der Datei zu klammern sind. Für **Realisierungen** wird festgelegt, daß Dateien als persistente Datenobjekte vollständig auf HS zu speichern sind.
Sei d eine BFD. Dann sind mit dem oben Erklärten die Eigenschaften der beiden Sichten auf d festgelegt. Ein Schnappschuß von d zeigt in der **abstrakten Sicht** das Datenobjekt, also eine Bytefolge der Länge l_d; als solches kann d in AS gespeichert und mit den oben angegebenen Diensten benutzt werden. Ein Schnappschuß von d zeigt in der **Realisierungssicht** das persistente Objekt als Menge von Blöcken, die auf HS gespeichert sind. Abbildung 6.1 zeigt d mit diesen beiden Sichten.
Von den beiden Sichten auf die BFD interessiert hier vorrangig die einer **Realisierung auf einem Speicher** $h \in H$. Das Wesentliche hierfür wird im folgenden erklärt.
Das statische Datenobjekt d ist eine Bytefolge der Länge l_d. Die Elemente der Folge werden ihrer linearen Ordnung entsprechend mit Indizes aus der Indexmenge

$$I(d) \triangleq \{0, 1, \ldots, l_d - 1\}$$

identifiziert; $I(d)$ ist die **Byteindexmenge** von d.

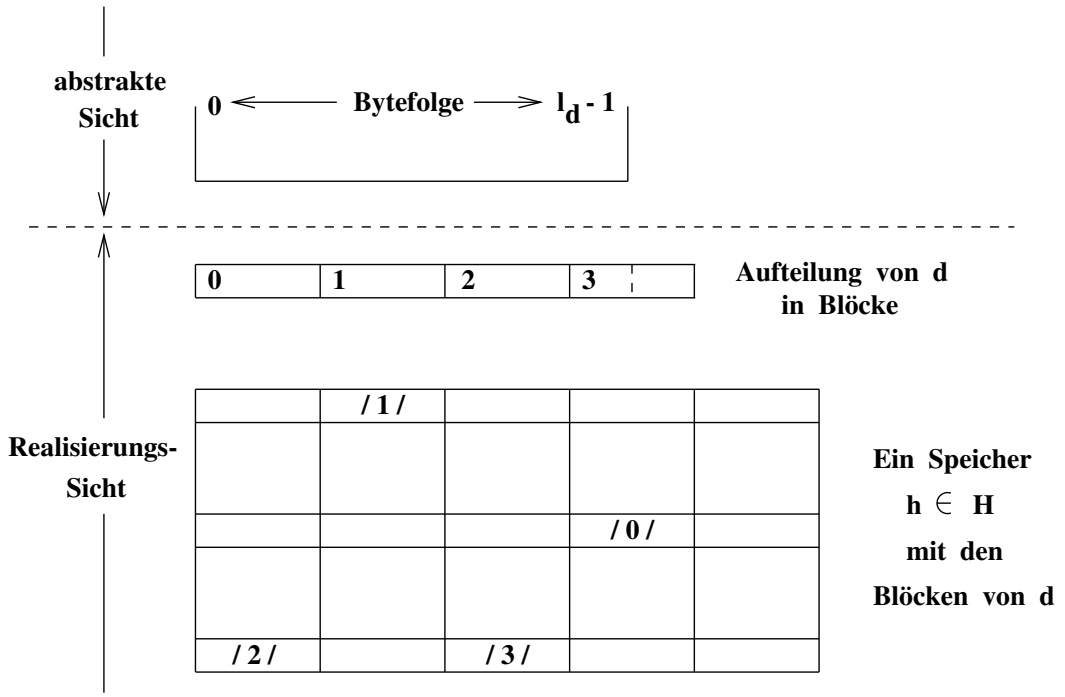


Abbildung 6.1: Die beiden Sichten auf eine Datei d

Für die Speicherung von d auf HS sei $h \in H$. h ist ein Plattenspeicher, der durch Adressen identifizierte Sektoren zur Speicherung von Blöcken fester Länge zur Verfügung stellt. Diese Sektoren werden im weiteren h–**Blockbereiche** (h–BB) genannt. Sei $BA(h)$ der h–**BB–Adreßraum**. In jedem h–BB können Bitfolgen gespeichert werden, an die von h bestimmte Anforderungen gestellt werden. Zur Speicherung der Datei d auf h wird davon ausgegangen, daß in einem h–BB ein h–**Block** der Länge bl Bytes gespeichert werden kann. Wenn b ein h–Block ist, dann wird angenommen, daß b mit Schreiboperationen auf h den von h gestellten Anforderungen entsprechend transformiert wird; entsprechendes soll für Leseoperationen auf h gelten. In diesem Sinn gilt, daß auf h h–Blöcke gespeichert sind, die als solche gelesen und geschrieben werden können.
Zur Speicherung der Datei d auf h ist erforderlich, daß d in h–Blöcke aufgeteilt wird; diese h–Blöcke werden dann in h–BBs gespeichert.
Zur **Aufteilung von** d **in** h–**Blöcke** sei $b_d \triangleq \lceil l_d/bl \rceil$, und mit b_d sei

$$BI(d) \triangleq \{0, 1, \ldots, b_d - 1\}.$$

Die Aufteilung von d erfolgt, indem b_d aufeinanderfolgende Bytes von d zu einem h–Block zusammengefaßt und aus d eine h–Blockfolge gebildet wird. $BI(d)$ ist die h–**Blockindexmenge** von d. d wird in b_d h–Blöcke aufgeteilt. Weil nicht notwendig $l_d \equiv 0(bl)$ gilt, ist der h–Block mit dem Index $b_d - 1$ ggf. partiell undefiniert. Das muß bei Übergängen zwischen der Bytefolge d und der h–Blockfolge für d beachtet werden. Die Speicherung von d auf h erfolgt durch Speicherung der h–Blöcke von d auf h. Die hier auftretende Möglichkeit eines partiell undefinierten Blockes ist typisch für die angewandte Vorgehensweise, mit der einem Datenobjekt mit 1–dimensionaler Struktur eine 2–dimensionale aufgeprägt wird. Als Konsequenz des partiell undefinierten Blockes ergibt sich, daß ein entsprechender h–BB nicht vollständig genutzt wird; das wird als **interne Fragmentierung** bezeichnet.

Mit der Aufteilung von d in h–Blöcke und der Blockindexmenge $BI(d)$ ergibt sich, daß zur **vollständigen Speicherung von d auf h** der Mächtigkeit von $BI(d)$ entsprechend b_d h–BBs erforderlich sich sind. Diese h–BBs müssen zunächst frei sein und dann für d belegt werden. Zudem muß beschrieben werden, in welchen h–BBs die Blöcke von d gespeichert werden und sind. Diese Zuordnung wird mit einer **Speicherfunktion**

$$\sigma_d^h : BI(d) \to BA(h),$$

die bei vollständiger Speicherung von d auf h linkstotal und injektiv sein muß, beschrieben.
Für das **Datenobjekt d und seine Realisierung auf h** ergibt sich mit dem bisher Erklärten:

- d ist vollständig auf h gespeichert, wenn mit $BI(d)$ und σ_d^h die h–BBs $\sigma_d^h(BI(d))$ die h–Blöcke von d enthalten.
- Wenn d persistent auf h gespeichert sein soll, dann muß mit d auch $\sigma_d^h(BI(d))$ vollständig auf h gespeichert sein.

Die Verallgemeinerung der hier für d angegebenen Forderung auf eine Menge von Datenobjekten liefert die Persistenzeigenschaft, die von der statischen Speicherfähigkeit gefordert ist; sie muß im weiteren noch erfüllt werden.
Die BFDs sind einfache, universell nutzbare, hardwarenahe Datenobjekte; ihre konzeptionell festgelegten Eigenschaften sind schwach. Die BFDs sind daher als Ausgangsbasis für die Konstruktion weiterer Datenobjekte mit differenzierter festgelegten, stärkeren Eigenschaften zu benutzen. Entsprechende Konstruktionen sind im Dateimanagement klassischer Betriebssysteme lediglich in Ansätzen vorhanden.
Die Vorgehensweise zur Konstruktion weiterer Datenobjekte ist klar: Es müssen weitere **Dateitypen** so definiert werden, daß Datenobjekte als Inkarnationen bzgl. dieser Typen erzeugt und mit BFDs realisiert werden können. Nach dieser Vorgehensweise werden die auch für BFDs vordringlich benötigten Verzeichnisdateien zur Beschreibung und zur Strukturierung der Mengen der jeweils existierenden Dateien konstruiert; sie werden im folgenden Kapitel behandelt. Für weitere Datenobjekte gibt es in klassischen Dateisystemen lediglich nützliche Hilfskonstruktionen. Sie werden als „Dateitypen“ bezeichnet; Abbildung 6.2 gibt eine Übersicht über gebräuchliche „Dateitypen“. Von diesen sind insbesondere die als Object bezeichneten Dateien von Interesse; sie enthalten Programme in Binder–Sprache, die zur Erzeugung von Prozessen benötigt werden.

6.1.2 Verzeichnisdateien und der Dateienbaum

Mit den Dateien erhält ein Rechensystem seine statischen Speicherfähigkeiten und darüber hinaus einen wesentlichen Teil seiner gesamten Speicherfähigkeiten. Die BFDs sind, wie erklärt, dazu geeignet, diese Speicherfähigkeiten als einfache Datenobjekte, die von den technisch bedingten Eigenschaften der Speicher der HK abstrahieren, zur Verfügung zu stellen. Die BFDs können bei Bedarf erzeugt und, wenn sie nicht mehr erforderlich sind, aufgelöst werden. Damit ergibt sich die Notwendigkeit, die Mengen der jeweils existierenden Dateien geeignet zu beschreiben. An diese **Beschreibungen der jeweils existierenden Dateien** werden folgende Anforderungen gestellt:

- Für jede existierende Datei sind die jeweils **festgelegten und für ihre Nutzung erforderlichen Eigenschaften** zu beschreiben; das sind auf Lebenszeit der Datei festgelegte und den jeweiligen Zustand einschließlich der Realisierung beschreibende Eigenschaften.

File type	Usual extension	Function
Executable	exe, com, bin or none	ready–to–run machine–language program
Object	obj, o	compiled, machine language, not linked
Source code	c, p, pas, f77, asm, a	source code in various languages
Batch	bat, sh	commands to the command interpreter
Text	txt, doc	textual data, documents
Word processor	wp, tex, rrf, etc	various word–processor formats
Library	lib, a	libraries of routines for programmers
Print or view	ps, dvi, gif	ASCII or binary file in a format for printing or viewing
Archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage

Abbildung 6.2: Gebräuchliche „Dateitypen“

- Für die Dateien gilt, daß sie mit dem für sie festgelegten Bezeichner identifiziert werden; dem entsprechend ist von den Bezeichnern Eindeutigkeit, genauer **globale Eineindeutigkeit**, gefordert. Mit den Beschreibungen der Dateien muß diese Forderung durchgesetzt werden; die Beschreibungen können und sollen dazu beitragen, die Gewährleistung dieser Eindeutigkeit auch bei großen Dateimengen zu vereinfachen.

- Die Dateien sind **dynamische Datenobjekte** mit individuellen, beschränkten Lebenszeiten. Wie bei allen dynamischen Datenobjekten muß also bei Dateien versucht werden, Inkonsistenzen der Lebenszeiten und dem anwendenden Auftreten von Bezeichnern zu vermeiden. Durch systematische Strukturierung der Mengen der jeweils existierenden Dateien mit ihren Beschreibungen kann die Vermeidung dieser Inkonsistenzen wesentlich vereinfacht werden.

Klassische Dateisysteme tragen diesen Anforderungen teilweise Rechnung, indem sie für Dateibeschreibungen Verzeichnisdateien verwenden und mit diesen für die Mengen der jeweils existierenden Dateien Baumstrukturen realisieren. Diese Vorgehensweise wird im folgenden erklärt.
Dazu werden **zwei Dateitypen** festgelegt, die hier als D–Dateien und als V–Dateien bezeichnet werden. Die **D–Dateien** sind die bereits erklärten BFDs. Die **V–Dateien** sind die hinzukommenden Verzeichnisdateien (directories), die noch erklärt werden. D– und V–Dateien werden, wenn es auf die Unterscheidung nicht ankommt, als Dateien bezeichnet.
Jede existierende Datei wird mit einem **Dateideskriptor** beschrieben. Wenn d eine Datei ist, wird das entsprechende Datenobjekt als *descriptor*(d) bezeichnet. *descriptor*(d) beschreibt die Eigenschaften von d mit festgelegten Attributen, wobei Attribute für die auf Lebenszeit von d festgelegten Eigenschaften und Attribute für die den jeweiligen Zustand von d beschreibenden Eigenschaften zu unterscheiden sind. Zu den Attributen für die auf Lebenszeit festgelegten Eigenschaften gehören hier wesentlich der Bezeichner und der Typ; hinzu kommen der Eigner/Benutzer, festgelegte Benutzungsrechte u.a. Zu den Attributen für die den

jeweiligen Zustand beschreibenden Eigenschaften gehören hier wesentlich die Länge und die Realisierung. Für diese gilt das in Kapitel 5.1.2.2 für Prozeßbeschreibungen Erklärte entsprechend: Die erforderliche Flexibilität wird dadurch erreicht, daß *descriptor*(d) als Wurzel der Beschreibungen der Eigenschaften von d benutzt wird, welche die erklärten Attribute und Zeiger auf Datenobjekte, mit denen die weiteren Eigenschaften von d beschrieben werden, enthält. Entsprechende Datenobjekte werden für die in Kapitel 6.1.1 erklärten Speicherfunktionen der Realisierungen benutzt.
Die jetzt erklärten Dateideskriptoren sind die Komponenten der Verzeichnisdateien. Eine **V–Datei** ist ein Datenobjekt, das mit einem festgelegten Bezeichner identifiziert wird, und das jeweils aus einer Menge von Dateideskriptoren besteht, deren Mächtigkeit in festgelegten Grenzen variieren kann. Analog zu den D–Dateien legt das Dateimanagement das Konzept für V–Dateien einschließlich der Dienste zur Nutzung und der Realisierungseigenschaften fest.
Mit den D– und den V–Dateien wird für die jeweilige Menge aller existierenden Dateien als Struktur die eines Baumes festgelegt. Mit dieser Festlegung ergibt sich jeweils ein **Dateisystem–Baum**. Das ist ein spezieller, azyklischer Graph, der sich nach folgenden Regeln ergibt:

- Die jeweils existierenden D– und V–Dateien sind die **Knoten** des Graphen; zu diesen gehört als ausgezeichnete V–Datei stets die **Wurzel** des Baumes.

- Die **Kanten** des Graphen sind gerichtete Knotenpaare. Wenn x und y zwei Knoten sind, sagt die Kante (x, y) : x ist V–Datei und enthält den Deskriptor von y. x ist **direkter Vorgänger** von y.

- Zu jedem Knoten y des Graphen, der nicht die Wurzel ist, gilt: Es gibt genau einen Knoten $x \neq y$ so, daß (x, y) Kante des Graphen ist. Die Wurzel hat keinen Vorgänger, und für alle weiteren Knoten gilt **Eindeutigkeit des direkten Vorgängers**.

Mit diesen Strukturierungsregeln ergibt sich die erweiterte Aufgabe des **Dateimanagements**: Es legt die Konzepte für ein baumstrukturiertes Dateisystem mit D– und V–Dateien fest; es stellt die Dienste zur Verfügung, mit denen Dateien erzeugt, genutzt und aufgelöst werden können, mit denen die jeweils existierenden Dateien realisiert werden, und mit denen die festgelegten Strukturierungsregeln durchgesetzt werden. Die für die Dateien geforderte **Bezeichner–Eineindeutigkeit** wird mittels der V–Dateien durchgesetzt und vereinfacht. Das wird wie folgt erreicht:

- Dateien werden jeweils bzgl. der V–Dateien, die ihre Deskriptoren aufnehmen bzw. enthalten, erzeugt und aufgelöst.

- Für jede V–Datei wird festgelegt, daß die Bezeichner der Dateien, deren Deskriptoren sie aufnimmt, verschieden sein müssen.

Mit diesen Festlegungen wird globale Bezeichner–Eineindeutigkeit mit V–lokaler Eineindeutigkeit erreicht. Zugleich ergibt sich, daß die Dateien ihrer Einordnung in den Baum entsprechend mit **Pfadnamen** identifiziert werden können. Dabei sind vollständige oder **absolute Pfadnamen**, die von der Wurzel des Baumes ausgehen, und **relative Pfadnamen**, die von einem beliebigen Knoten des Baumes ausgehen und die Eindeutigkeit der direkten Vorgänger nutzen, möglich. Für die Einordnung der Dateien in den Baum ergibt sich aus dem Zusammenhang zwischen den Deskriptoren in den V–Dateien und den Kanten des Baumes, daß jeweils sämtliche D–Dateien **Blätter** des Baumes sind.

Für die **D–Dateien** als Knoten des jeweiligen Dateienbaums gilt das in Kapitel 6.1.1 Erklärte bis auf ihre Erzeugung und Auflösung bzgl. V–Dateien weiter.
Für die **V–Dateien** sind noch ihre Eigenschaften als abstrakte Datenobjekte und ihre Realisierungen zu erklären. Nach den angegebenen Strukturierungsregeln sind die V–Dateien **Strukturierungsobjekte** für die Dateienbäume. Dem entsprechend sind die Dienste für die Nutzung Dienste zum Erzeugen von Dateien, deren Deskriptoren sie aufnehmen, Dienste zum Auflösen von Dateien, deren Deskriptoren sie enthalten, sowie Dienste zum Lesen von Deskriptoren und Deskriptor–Attributen, die insbesondere zur Erarbeitung von Pfadnamen benötigt werden. Mit dieser Rolle der V–Dateien als Strukturierungsobjekte für die Dateienbäume ergibt sich, daß die Werte der Deskriptoren einer V–Datei lediglich mittelbar verändert werden, nämlich dann, wenn die Eigenschaften der Dateien, die mit den Deskriptor–Attributen beschrieben sind, verändert werden. Weiter ergibt sich, daß mit einer V–Datei der Unterbaum, dessen Wurzel sie ist, aufgelöst wird. Diese Bedeutung der V–Dateien als Strukturierungsobjekte steht in engem Zusammenhang damit, daß die Dateienbäume dynamische Datenobjekte sind. Die Baumstrukturen vereinfachen die Vermeidung von Lebenszeit–Inkonsistenzen; sie reichen jedoch nicht aus zur Vermeidung; dazu sind weitere konzeptionelle Regelungen erforderlich.
Für **Realisierungen der V–Dateien** sind die in Kapitel 6.1.1 erklärten BFD als Hilfsmittel einzusetzen. Für eine V–Datei v ist festgelegt, daß sie ein dynamisches Datenobjekt mit Dateideskriptoren *deskriptor*(d) als Komponenten ist; dabei ist $d \in D$ mit einer Dateimenge D. Für Realisierungen von v sind damit folgende Maßnahmen erforderlich:

- Für die Deskriptoren *deskriptor*$(\cdot)$ ist eine geeignete Repräsentation als Bytefolge festzulegen.
- Die *deskriptor*(d) der $d \in D$ sind auf eine BFD $b(v)$ abzubilden; dabei ist zu beachten, daß D lediglich einem Schnappschuß von v entspricht.
- Schließlich ist die BFD $b(v)$ wie früher erklärt zu realisieren.

Realisierungen von v erfolgen damit 2–stufig als Realisierung von v mit $b(v)$ und als Realisierung von $b(v)$. Für Realisierungen von v mit $b(v)$ sind die für Realisierungen von dynamischen Datenobjekten auf linearen Datenobjekten (Speichern) zur Verfügung stehenden Verfahren anzuwenden, wobei ein Standardproblem und zwei spezifische Teilprobleme zu lösen sind.
Das Standardproblem besteht darin, für *deskriptor*$(\cdot)$ Bytefolgen–Repräsentationen festzulegen; dabei kommen Folgen fester Länge, Bytefelder, und Folgen variabler Längen in Betracht; insbesondere kommen Folgen variabler Längen als Paare $(kopf, körper)$ mit einem *körper* wählbarer Länge und einem *kopf* fester Länge, der die Länge von *körper* beschreibt, in Betracht.
Das erste spezifische Teilproblem ergibt sich daraus, daß V–Dateien als Strukturierungsobjekte insbesondere zur Erarbeitung von Pfadnamen benötigt werden. Für den betrachteten Schnappschuß von v ist demnach wichtig, daß mit den $d \in D$ effiziente Zugriffe zu den *deskriptor*(d) möglich sind.
Das zweite spezifische Teilproblem ergibt sich daraus, daß jeder Deskriptor *deskriptor*(d) lediglich die Wurzel der Beschreibung der Eigenschaften der Datei d ist. Dem wird, wie schon erklärt, dadurch Rechnung getragen, daß *deskriptor*(d) Zeiger auf Datenobjekte enthält, die weitere Eigenschaften von d beschreiben. Zu diesen weiteren Eigenschaften gehören die Realisierungseigenschaften. Das ist, wenn d eine D–Datei ist, die in Kapitel 6.1.1 erklärte Speicherfunktion. Wenn d eine V–Datei ist, ist es die Speicherfunktion der BFD, mit der d realisiert ist.

Aus dem Erklärten ergibt sich, daß sowohl für die D–Dateien als auch für die V–Dateien die Speicherfunktionen wesentlich zu den zu beschreibenden Eigenschaften gehören. Sie erfordern insbesondere für die persistenten Datenobjekte weitere Realisierungsmaßnahmen.

6.1.3 Der Baum der persistenten Datenobjekte

Mit Kapitel 6.1 sollen die statischen Speicherfähigkeiten eines Systems erklärt werden. In der Einführung sind die dabei gestellten Anforderungen angegeben; zudem ist auf den Zusammenhang mit dem Dateimanagement hingewiesen. In den Kapiteln 6.1.1 und 6.1.2 sind wesentliche Eigenschaften des Dateimanagements erklärt; dabei hat sich ergeben, daß die Mengen der jeweils existierenden Dateien in Dateibäumen strukturiert zusammengefaßt werden. Für die statischen Speicherfähigkeiten ergibt sich, daß ein vollständiger, konsistenter Dateienbaum, der auf den Platten von HS gespeichert ist, dazu geeignet ist, die Speicherfähigkeiten eines Systems über Betriebspausen hinweg zu erhalten. Für das statische System ergibt sich damit jeweils ein Baum mit persistenten Datenobjekten.
Von einem **Baum mit persistenten Datenobjekten**, der dazu geeignet ist, die Speicherfähigkeiten eines Systems über Betriebspausen hinweg zu erhalten, sind Vollständigkeit, Konsistenz und Persistenz gefordert. Diese Eigenschaften betreffen die einzelnen Dateien und die gesamte, strukturierte Menge der Dateien, also den jeweiligen Dateienbaum. **Konsistenz** für eine Datei ist ein Prädikat für die Eigenschaften, welche für die Datei spezifiziert sind. Das ist für eine D–Datei d: Die Länge l_d von d liegt fest, und die Bytefolge der Länge l_d liegt gespeichert vor. Mit der Festlegung, daß die Operationen auf Dateien mit *öffnen* und *schließen* zu klammern sind, ist d konsistent, wenn d „geschlossen“ ist; die Soll–Wirkung für *schließen* auf d ist die Erfüllung des angegebenen Konsistenzprädikats für d. Für eine V–Datei v, die für einen Schnappschuß die Deskriptoren der Dateimenge D enthält, sind **Konsistenz und Vollständigkeit** entsprechend für alle $d \in D$ und die vollständigen Beschreibungen der $d \in D$ festgelegt, also für $deskriptor(d)$ und für die Datenobjekte, die weitere d–Eigenschaften beschreiben. Das oben Gesagte gilt entsprechend für *schließen* auf v und „v ist geschlossen“ sagt: Für alle $d \in D$ liegen die vollständigen Beschreibungen gespeichert vor. Das impliziert: Wenn v geschlossen ist, sind alle $d \in D$ geschlossen. Damit sind Konsistenz für D–Dateien sowie Konsistenz und Vollständigkeit für V–Dateien definiert. Konsistenz und Vollständigkeit für einen Dateienbaum sind induktiv von den Blättern des Baumes ausgehend bis zur Wurzel definiert. Wenn die Wurzel des Baumes konsistent und vollständig ist, ist der Dateienbaum konsistent und vollständig. **Persistenz** für eine Datei gilt, wenn ihr Wert (für D–Dateien) bzw. die vollständigen Beschreibungen der Dateien, deren Deskriptoren sie enthält (für V–Dateien), persistent gespeichert (hier auf HS–Elementen) vorliegen. Ein konsistenter, vollständiger Dateienbaum ist persistent, wenn seine sämtlichen Dateien persistent sind. Ein konsistenter, vollständiger, persistenter Dateienbaum ist ein Baum mit persistenten Datenobjekten. Er ist dazu geeignet, die Speicherfähigkeiten eines Systems über Betriebspausen hinweg zu erhalten, und mit ihm erhält ein System seine statischen Speicherfähigkeiten.
Sei $\mathcal{B}$ ein Baum mit persistenten Datenobjekten. Für jede Datei von $\mathcal{B}$ ist gefordert, daß die vollständige Beschreibung ihrer Eigenschaften und ihr Wert auf einem HS–Element gespeichert sind. Damit wird erreicht, daß ein System, das am Ende einer Betriebspause für die nächste Betriebsphase vorbereitet wird, mit $\mathcal{B}$ für diese seine mit Berechnungen nutzbaren Speicherfähigkeiten erhält. Zu den an $\mathcal{B}$ gestellten Forderungen gehört insbesondere, daß die Beschreibungen der Realisierungen der Dateien von $\mathcal{B}$ auf HS–Elementen gespeichert sind.
Die Aufgaben, die zur Speicherung der **Beschreibungen der Realisierungen** der Dateien von $\mathcal{B}$ auf HS–Elementen zu lösen sind, werden im folgenden für eine Datei d von $\mathcal{B}$, die auf

dem Speicher $h \in H$ realisiert wird, behandelt. Nach dem für D– und V–Dateien Erklärten, kann angenommen werden, daß d für die Realisierungsaufgaben als BFD (unmittelbar für eine D–Datei d oder mittelbar für eine V–Datei d) angesehen wird. Für das Weitere wird entsprechend auf die für d und für h in Kapitel 6.1.1 erklärten Eigenschaften Bezug genommen.
Nach Kapitel 6.1.1 sind die hier relevanten Eigenschaften der Datei d mit ihrer Länge l_d und der sich daraus ergebenden Byteindexmenge $I(d)$ festgelegt. Für den Speicher $h \in H$ sind die h–Blöcke der Länge bl und der h–BB–Adreßraum $BA(h)$ festgelegt. Mit den h–Blöcken ergibt sich für d eine 2–dimensionale Struktur mit der h–Blockindexmenge von d

$$BI(d) = \{0, 1, \ldots, b_d - 1\} \ ,$$

und mit dieser ist Speicherfunktion der Realisierung von d auf h

$$\sigma_d^h : BI(d) \rightarrow BA(h)$$

definiert. Die **Beschreibung dieser Realisierung von** d erfordert demnach, daß die Bildmenge $\sigma_d^h(BI(d)) \subseteq BA(h)$ gespeichert zur Verfügung steht. Damit ergibt sich die Notwendigkeit $\sigma_d^h(BI(d))$ auf h oder einem anderen Speicher von H zu speichern.
Die Aufgabe, eine geeignete Beschreibung dieser Realisierung von d auf h festzulegen, wird im folgenden unter der Annahme behandelt, daß auch diese Beschreibung auf h gespeichert werden soll. Bei der Lösung dieser Aufgabe sind insbesondere die Geometrie des Plattenspeichers h und die Variabilität der Länge l_d der Datei d zu beachten.
Die **Geometrie des Speichers** h muß wegen ihres bestimmenden Einflusses auf die Zugriffszeiten zu den h–BBs beachtet werden. Für diese Zugriffszeiten sind die Positionierungszeiten der L/S–Köpfe auf Spuren (seek time) und dann die Rotationswartezeiten auf Sektoren (latency time, rotational dalay) die dominierenden Faktoren. Diesen Gegebenheiten kann dadurch Rechnung getragen werden, daß zusammengehörige h–Blöcke in den h–BBs eines Zylinders oder benachbarter Zylinder gespeichert werden.
Der **Variabilität der Längen** der Dateien muß auch für den hier betrachteten Baum $\mathcal{B}$ Rechnung getragen werden, weil $\mathcal{B}$ für die folgende Betriebsphase benötigt wird. Feste Längen kommen also lediglich in den Fällen in Betracht, in denen diese für Dateien festgelegt sind.
Im folgenden wird für die Datei d angenommen, daß ihre Länge im Bereich $0 \leq l_d \leq 2^{30}$ variieren kann. Mit l_d und den h–Blöcken ergibt sich die Mächtigkeit der Bildmenge $\sigma_d^h(BI(d))$ die zu speichern ist. Für h sollen die Blocklänge $bl = 2^{10}$ und $|BA(h)| = 2^{21}$ gelten. Zur Speicherung von $\sigma_d^h(BI(d))$ sind die Repräsentationen der h–BB–Adressen relevant; deren Länge sei 2^2 (in Bytes). Damit ergibt sich, daß ein h–BB zum Speichern von bis zu 2^8 $BA(h)$–Elementen geeignet ist.
Bei maximaler Länge von d ergibt sich damit $|BI(d)| = 2^{20}$. Bei voller Nutzung der h–BBs für die Speicherung von $\sigma_d^h(BI(d))$ sind folglich 2^{12} h–BBs erforderlich. Es ergibt sich, daß für die vollständige Speicherung von d und für die vollständige Speicherung der Realisierungsbeschreibung in diesem Fall wenigstens

- $2^{20} + 2^{12}$ h–BBs oder $50,2\%$ der h–Kapazität

erforderlich sind. Bei minimaler Länge $l_d = 0$ von d ist andererseits lediglich *deskriptor*(d) auf h zu speichern.
Mit den betrachteten Extremfällen für die Datei d wird deutlich, wie breit das Spektrum der Anforderungen für vollständige Speicherung ist, das sich aus der Variabilität der Dateilängen ergibt. Dieser Variabilität muß Rechnung getragen werden.

Zusätzlich zur Anpaßbarkeit an die Variabilität der Länge der Datei d muß die Speicherfunktion σ_d^h, welche die d–Realisierung auf h beschreibt, für Zugriffe zu den h–Blöcken von d **effizient berechnet** werden können: für $i \in BI(d)$ erfordert ein Zugriff zum Block von d mit dem Index i einen Zugriff zum h–BB mit der Adresse $\sigma_d^h(i)$. Die Definition der Speicherfunktion

$$\sigma_d^h : BI(d) \to BA(h)$$

legt hierfür die Speicherung von $\sigma_d^h(BI(d))$ als Tabelle und die Definition von σ_d^h als 1–stufig lineare Funktion nahe: mit a_0 als Index des Tabellenanfangs wird $\sigma_d^h(i) = a_0 + i$ berechnet, und das i–te Element der Tabelle ist $\sigma_d^h(i) \in BA(h)$. Diese Vorgehensweise ist jedoch höchstens in Ausnahmefällen geeignet, weil $\sigma_d^h(BI(d))$ auf h natürlich in h–BBs gespeichert werden muß. Dem entsprechend sind für $\sigma_d^h(BI(d))$ geeignete h–Blöcke erforderlich. Geeignet bedeutet, daß den gegebenen h–Blöcken eine Struktur aufzuprägen ist, die der Notwendigkeit, $BA(h)$–Elemente zu speichern, entspricht. Die entsprechenden Blöcke werden als *hba*–Blöcke bezeichnet. *hba*–Blöcke können in h–BBs gespeichert werden, und ein *hba*–Block ist für eine Tabelle mit höchstens 2^8 $BA(h)$–Elementen geeignet, wenn die oben für h angegebenen Längen zugrunde gelegt werden.
Mit den *hba*–Blöcken kann sowohl Anpaßbarkeit an die Variabilität der Längen der Dateien als auch Effizienz für die Berechnungen der Speicherfunktionen erreicht werden. Dafür sind gemischt 1– und mehrstufig lineare Speicherfunktionen geeignet. Diese Speicherfunktionen, die hier und für ähnliche Aufgabenstellungen nützlich sind, werden im folgenden für die Datei d und den Speicher h auf der Grundlage der oben getroffenen Festlegungen erklärt.
Eine **gemischt 1– und mehrstufige Speicherfunktion** für die Datei d und den Speicher h wird von der Blockindexmenge $BI(d)$ und der Speicherfunktion $\sigma_d^h : BI(d) \to BA(h)$ sowie von *hba*–Blöcken ausgehend wie folgt konstruiert:

- Gemäß σ_d^h sei $BAT(d)$ eine fiktive h–BB–Adreßtabelle mit Indizes gemäß $BI(d)$ und den Elementen $\sigma_d^h(i)$ für $i \in BI(d)$.

- Zum Speichern von $\sigma_d^h(BI(d))$ wird zunächst ein *hba*–Block als Tabelle mit h–BB–Adressen als Elementen benutzt; $BAT_0(d)$ sei diese Tabelle. Sie besteht aus 2^8 Elementen und ihre Indexmenge ist $\Im = \{0, 1, \ldots, 2^8 - 1\}$.

- $BAT_0(d)$ wird der maximalen Stufenanzahl entsprechend in einen Anfangsteil der Länge l_1 für die 1–te Stufe, einen Folgeteil der Länge l_2 für die 2–te Stufe usw. zerlegt. Hier wird eine 1– bis 3–stufige Speicherfunktion konstruiert. Die Längen der Teile sind $l_1 = 144, l_2 = 96$ und $l_3 = 16$; die entsprechenden Teile von $BAT_0(d)$ werden mit $BAT_0^1(d)$, $BAT_0^2(d)$ und $BAT_0^3(d)$ bezeichnet. Mit $BAT_0(d)$ und diesen Teilen wird die gemischte Speicherfunktion für d konstruiert.

- Die Konstruktion erfolgt, indem h–BB–Adreßtabellen festgelegt und als *hba*–Blöcke auf h gespeichert werden. Die h–BB–Adreßtabellen werden in Abhängigkeit von der Länge l_d von d konstruiert. Das sind Tabellen, deren Elemente i.w. die $\sigma_d^h(BI(d))$ als Werte haben. Weil diese Tabellen als *hba*–Blöcke auf h gespeichert werden, werden sie den Stufen der Speicherfunktion entsprechend konstruiert, indem $BAT(d)$ aufgeteilt wird, und indem $BAT(d)$ eine 2–dimensionale Struktur gemäß den *hba*–Blöcken aufgeprägt wird. Diese 2–dimensionale Struktur erfordert wieder eine Speicherfunktion α_d^h, deren Bilder zu speichern sind. Bei entsprechender Länge von d muß diese Konstruktion wiederholt werden. Dabei wird eine weitere Speicherfunktion β_d^h für die fiktive h–BB–Adreßtabelle erforderlich, deren Elemente die Bilder von α_d^h sind. Der jeweiligen Länge

l_d von d und der daraus abgeleiteten Mächtigkeit p_d von $BI(d)$ entsprechend ergeben sich die Fälle

- $0 \leq p_d \leq l_1$ mit einer 1–stufigen Speicherfunktion,
- $l_1 < p_d \leq l_1 + l_2 \cdot 2^8$ mit einer gemischt 1– und 2–stufigen Speicherfunktion und
- $l_1 + l_2 \cdot 2^8 < p_d \leq 2^{20}$ mit einer gemischt 1– bis 3–stufigen Speicherfunktion.

Für den **Fall** $0 \leq p_d \leq l_1$ ist $BAT_0^1(d)$ die h–BB–Adreßtabelle mit den Elementen $\sigma_d^h(BI(d))$. Genauer gelten:

- die Elemente von $BAT_0^1(d)$ mit den Indizes $i, 0 \leq i \leq p_d - 1$, haben die Werte $\sigma_d^h(i)$;
- die Elemente von $BAT_0^1(d)$ mit den Indizes $i, p_d \leq i \leq l_1 - 1$, und alle Elemente von $BAT_0^2(d)$ und $BAT_0^3(d)$ sind $\perp$.

Für den **Fall** $l_1 < p_d$ sei $BAT(d)|_{l_1}$ der Teil von $BAT(d)$ mit den Indizes $l_1, l_1 + 1, \ldots, p_d - 1$. Dann wird $BAT(d)|_{l_1}$ eine 2–dimensionale *hba*–Blockstruktur aufgeprägt. Mit ihr ergibt sich die Blockindexmenge $BI^1(d) = \{0, 1, \ldots, p_d^1 - 1\}$ mit $p_d^1 \triangleq \lceil (p_d - l_1)/2^8 \rceil$. Die Elemente von $BAT(d)|_{l_1}$ werden in den h–BB–Adreßtabellen

$$BAT_1^0(d), \ldots, BAT_1^{p_d^1 - 1}(d)$$

als *hba*–Blöcke auf h gespeichert. Die Elemente dieser Tabellen sind also die $\sigma_d^h(i)$ für $l_1 \leq i \leq p_d - 1$. Für diese Tabellen ist eine Speicherfunktion erforderlich; sie sei

- $\alpha_d^h : BI^1(d) \to BA(h)$.

Für den **Fall** $l_1 < p_d \leq l_1 + l_2 \cdot 2^8$ sind $BAT_0^1(d)$ die h–BB–Adreßtabelle mit den Elementen $\sigma_d^h(i)$ für $0 \leq i \leq l_1 - 1$ und $BAT_0^2(d)$ die h–BB–Adreßtabelle mit den $\alpha_d^h(BI^1(d))$. Genauer gelten:

- die Elemente von $BAT_0^2(d)$ mit den Indizes $l_1 + i, 0 \leq i \leq p_d^1 - 1$, haben die Werte $\alpha_d^h(i)$;
- die Elemente von $BAT_0^2(d)$ mit den Indizes $l_1 + i, p_d \leq i \leq l_2 - 1$ und alle Elemente von $BAT_0^3(d)$ sind $\perp$.

Für den **Fall** $l_1 + l_2 \cdot 2^8 < p_d \leq 2^{20}$ sei $BAT^1(d)$ eine fiktive h–BB–Adreßtabelle mit Indizes gemäß $BI^1(d)$ und den Elementen $\alpha_d^h(i)$ für $i \in BI^1(d)$. $BAT^1(d)|_{l_2}$ sei der Teil von $BAT^1(d)$ mit den Indizes $l_2, l_2 + 1, \ldots, p_d^1 - 1$. Dann wird wieder $BAT^1(d)|_{l_2}$ eine 2–dimensionale *hba*–Blockstruktur aufgeprägt. Mit ihr ergibt sich die Blockindexmenge $BI^2(d) = \{0, 1, \ldots, p_d^2 - 1\}$ mit $p_d^2 \triangleq \lceil (p_d^1 - l_2)/2^8 \rceil$. Die Elemente von $BAT^2(d)|_{l_2}$ werden in den h–BB–Adreßtabellen

$$BAT_2^0(d), \ldots, BAT_2^{p_d^2 - 1}(d)$$

als *hba*–Blöcke auf h gespeichert. Die Elemente dieser Tabellen sind also die $\alpha_d^h(i)$ für $l_2 \leq i \leq p_d^1 - 1$. Für diese Tabellen wird eine Speicherfunktion benötigt; sie sei

- $\beta_d^h : BI^2(d) \to BA(h)$.

In diesem Fall sind $BAT_0^1(d)$ die h–BB–Adreßtabelle mit den Elementen $\sigma_d^h(i)$ für $0 \leq i \leq l_1 - 1$, $BAT_0^2(d)$ die h–BB–Adreßtabelle mit den Elementen $\alpha_d^h(i)$ für $0 \leq i \leq l_2 - 1$ und $BAT_0^3(d)$ die h–BB–Adreßtabelle mit den $\beta_d^h(BI^2(d))$. Genauer gelten:

- die Elemente von $BAT_0^3(d)$ mit den Indizes $l_1 + l_2 + i, 0 \leq i \leq p_d^2 - 1$ haben die Werte $\beta_d^h(i)$;
- die Elemente von $BAT_0^3(d)$ mit den Indizes $l_1 + l_2 + i, p_d^2 \leq i$, sind $\perp$.

Damit sind die gemischt 1– bis 3–stufigen Speicherfunktionen erklärt. Abbildung 6.3 veranschaulicht die für d erforderlichen h–BB–Adreßtabellen und die h–Blöcke, in denen d gespeichert ist.
Die Vorgehensweise ermöglicht Anpassungen an dynamisch variierende Längen der Datei d. Der Aufwand, der für Berechnungen der Speicherfunktion zu leisten ist, läßt sich mit dem Erwartungswert der Anzahl der Zugriffe zu den Adreßtabellen bei einem Zugriff zu d quantifizieren; dazu muß die Verteilung der Längen von d bekannt sein. Mit der Verteilung (p_1, p_2, p_3) für die Längen im betrachteten Fall ergibt sich für den Erwartungswert der Anzahl der Tabellenzugriffe: $3 - 2p_1 - p_2$. Bei maximaler Länge von d ergibt sich für die gemischte Speicherfunktion ein geringer Zuwachs an benötigten *hba*–Blöcken: Hier sind für die vollständige Speicherung auf h $50,19612\%$ gegenüber $50,19531\%$ der h–Kapazität erforderlich.

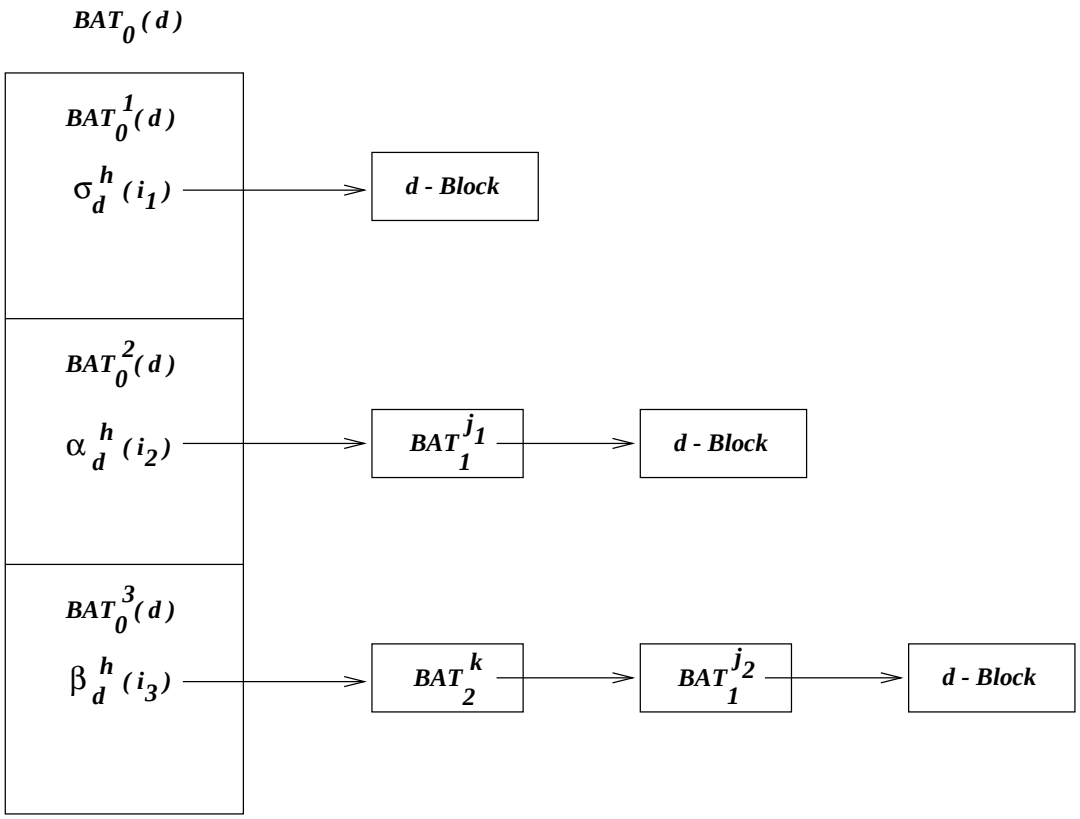


Abbildung 6.3: Eine gemischt 1– bis 3–stufige Speicherfunktion

Der **Baum** $\mathcal{B}$ kann vollständig auf den Speichern gemäß H gespeichert werden, indem die für die Datei d und den Speicher $h \in H$ erklärte Vorgehensweise entsprechend angewandt wird. Dabei sind noch einige weitere Aspekte zu berücksichtigen, auf die im folgenden hingewiesen wird.
Für jeden Speicher $h \in H$ muß zusätzlich zu den auf h gespeicherten Dateien auch die **Beschreibung der freien/belegten** h–**BBs** auf h gespeichert werden; diese Beschreibung

gehört zu den statischen Speicherfähigkeiten. Es ist naheliegend, die freien h–BBs als Pseudodateien in $\mathcal{B}$ aufzunehmen.

Zur Speicherung von $\mathcal{B}$ stehen i.a. gemäß H **mehrere Speicher** zur Verfügung, die unterschiedliche Eigenschaften haben können. Diese Eigenschaften sind einerseits die primär interessierenden BBs und Blöcke; es sind andererseits die EA–Steuerwerke und EA–Prozessoren, die für Zugriffe zu den Speichern benötigt werden. Wenn für zwei Speicher $h, h' \in H$ aus technischen Gründen die h–Blöcke und die h'–Blöcke verschieden sind, muß dem einerseits mit entsprechenden Festlegungen Rechnung getragen werden. Daraus ergibt sich, daß es zweckmäßig ist, jeweils vollständige Teilbäume von $\mathcal{B}$ auf den einzelnen Speichern zu realisieren. Andererseits bleibt, daß $\mathcal{B}$ die Einheit der statischen Speicherfähigkeiten eines Systems ist. Dem entsprechend ist es zweckmäßig, für $\mathcal{B}$ abstrakte BBs und Blöcke festzulegen, und diese dann mit den BBs und Blöcken der einzelnen $h \in H$ zu realisieren. Auf diesen Aspekt wird im Zusammenhang mit den Wirkungsbereichen der Prozesse noch eingegangen.

Für den Baum $\mathcal{B}$ mit persistenten Datenobjekten spielt die **Wurzel** eine ausgezeichnete Rolle: Es gibt keine V–Datei von $\mathcal{B}$, die den Deskriptor der Wurzel enthält. Dieser Sonderrolle der Wurzel wird dadurch Rechnung getragen, daß sie als ausgezeichnete V–Datei mit festgelegten BB–Adressen auf einem festgelegten Speicher $h \in H$ gespeichert wird.

Die jetzt mit dem Baum $\mathcal{B}$ der persistenten Datenobjekte erklärten statischen Speicherfähigkeiten eines Systems stehen auch in Betriebsphasen zur Verfügung. In den folgenden Kapiteln wird ein System wieder in Betriebsphasen betrachtet, so daß Berechnungen ausgeführt und die Speicherfähigkeiten genutzt werden können. Mit den Berechnungen ergeben sich Veränderungen der Datenobjekte, und damit erhalten die Dateien ihre Doppelrolle: Sie sind persistente Datenobjekte, und sie werden mit den Berechnungen verändert.

6.2 Prozesse mit virtuellen Adreßräumen und Speichern

In Kapitel 5.1.2.4 ist erklärt, daß die Prozesse eines Systems ihre konzeptionell festgelegten, abstrakten Speicherfähigkeiten i.w. mit prozeßspezifischen virtuellen Adreßräumen (VAs) und mit diesen zugeordneten virtuellen Speichern (VMs) erhalten. Dazu wird mit dem Prozeßkonzept eines Systems zunächst der für alle Prozesse strukturell gleiche Adreßraum festgelegt: $VA = \{0, \ldots, 2^{31} + 2^{30} - 1\}$. Im nächsten Schritt wird VA in zwei Teile zerlegt, den prozeßprivaten Teil VAP und den Kernteil VAK; dabei sind $VAP = \{0, \ldots, 2^{31} - 1\}$ und $VAK = \{2^{31}, \ldots, 2^{31} + 2^{30} - 1\}$ üblich. Wenn P die Menge der koexistierenden Prozesse des Systems ist, erhält jeder Prozeß $p \in P$ auf Lebenszeit den Adreßraum $VA(p)$. Mit der angegebenen Aufteilung erhält p damit den p–privaten Teil $VAP(p)$, mit dem $VMP(p)$ als p–privater, abstrakter Speicher gebildet wird. Zudem erhält p mit allen Prozessen gemeinsam VAK als Teil von $VA(p)$, und damit erbt p den abstrakten Speicher VMK des Kerns. Von diesen Festlegungen ausgehend werden in diesem Kapitel die beiden folgenden Teilaufgaben des Speichermanagements behandelt:

- Die prozeßprivaten Wirkungsbereiche, die für $p \in P$ i.w. mit $VAP(p)$ und mit $VMP(p)$ gebildet werden.

- Die Realisierungen der abstrakten Speicher der Prozesse und des Kerns mit den Speichern, die mit der eingesetzten HK zur Verfügung stehen.

6.2.1 Die prozeßprivaten Wirkungsbereiche

Ein Prozeß p erhält den p–privaten Wirkungsbereich, wie schon erklärt, mit den Registern seines Prozessors und mit seinem abstrakten Speicher *VMP*(p). p führt sequentielle Berechnungen aus, die mit einem entsprechenden Programm spezifiziert sind; dieses Programm muß mit *VMP*(p) zur Verfügung gestellt werden. Das erfolgt mit einem maschinennahen Berechnungsobjekt, das aus dem p–Progamm und den zugehörigen dynamischen Datenobjekten besteht. Mit diesem Berechnungsobjekt muß, wie in Kapitel 5.1.2.2 erklärt, die Brücke zwischen anwendungsgeeigneten Programmkonzepten höherer Programmiersprachen und den Prozessor–Programmkonzepten der HK geschlagen werden. Für die maschinennahen Berechnungsobjekte muß zudem berücksichtigt werden, daß es möglich sein soll, Prozeß–Programme aus Modulen verschiedener Quellsprachen zu komponieren.
Für die hier betrachteten Systeme mit prozeßspezifischen *VAs* werden diese Anforderungen wie folgt erfüllt:

- Die maschinennahen Berechnungsobjekte der Prozesse werden von Modulen einer geeignet festgelegten Zwischensprache, der Bindersprache, ausgehend gebildet.
- Für die maschinennahen Berechnungsobjekte werden Strukturen festgelegt, und die *VAPs* der Prozesse werden einer Vergröberung dieser Strukturen entsprechend zweigeteilt; auf dieser Grundlage werden die Berechnungsobjekte der Prozesse mit den *VMPs* bereitgestellt.

Die **Strukturen der maschinennahen Berechnungsobjekte** und die Vergröberungen für diese sind für alle Prozesse festgelegt. Für einen Prozeß p besteht das Berechnungsobjekt aus folgenden Teilen:

- *code*(p): dieser Teil ist der statisch definierte Code des ausführbereiten p–Progamms.
- *daten*(p): dieser Teil ist das strukturierte, statisch definierte Datenobjekt.
- *halde*(p): dieser Teil ist das strukturierte, dynamische Halden–Datenobjekt; es ist Teil von *haldenbereich*(p).
- *keller*(p): dieser Teil ist das strukturierte, dynamische Keller–Datenobjekt; es ist Teil von *kellerbereich*(p).

Die **vergröberte Struktur** des Berechnungsobjekts von p ergibt sich dadurch, daß mit den angegebenen Bestandteilen zwei lineare, dynamische Datenobjekte gebildet werden:

- *code*(p), *daten*(p) und *haldenbereich*(p) mit *halde*(p) werden zum **Code–Daten–Halden–Segment** von p, *cdh_seg*(p), zusammengefaßt;
- *kellerbereich*(p) mit *keller*(p) ist das **Keller–Segment** von p, *k_seg*(p).

Mit dieser Vergröberung der Berechnungsobjekte wird eine **Aufteilung der prozeßprivaten Adreßraumteile** festgelegt:

- *VAP* wird in *VAP0* und *VAP1* zerlegt; dabei sind $VAP0 = \{0, \ldots, 2^{30}-1\}$ und $VAP1 = \{2^{30}, \ldots, 2^{31}-1\}$.

Zudem wird für die **Segmente eines Prozesses** p festgelegt:

- das Segment *cdh_seg*(p) wird in dem *VAP0*(p) zugeordneten *VMP0*(p) und

- das Segment *k_seg*(p) wird in dem *VAP1*(p) zugeordneten *VMP1*(p)

gespeichert.
Damit sind die Rahmenbedingungen für die prozeßprivaten Wirkungsbereiche der Prozesse geklärt: Ein Prozeß p erhält den p–privaten Wirkungsbereich mit den beiden linearen, dynamischen Datenobjekten *cdh_seg*(p) und *k_seg*(p) sowie mit den Registern des p–Prozessors. Die oben angegebenen Festlegungen für die Adreßräume und die Segmente im virtuellen Speicher sind in Abbildung 6.4 veranschaulicht.
Den Festlegungen für die Berechnungsobjekte der Prozesse wird auch der Wirkungsbereich des Kerns festgelegt. Der Wirkungsbereich des Kerns ist das lineare, dynamische Datenobjekt **Kern–Segment** *kern_seg*; es ist in dem *VAK* zugeordneten *VMK* gespeichert. *kern_seg* hat eine Feinstruktur, die vom Kern festgelegt wird.

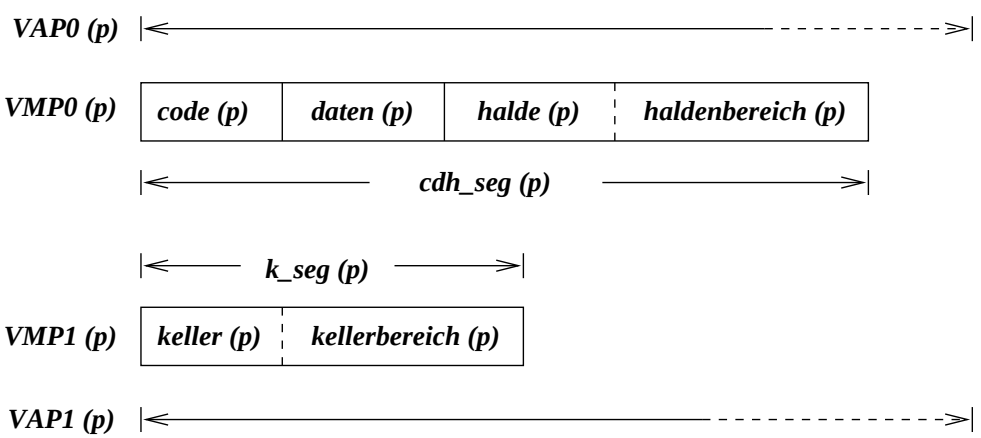


Abbildung 6.4: Die Struktur *VAP*(p) und das p–Berechnungsobjekt in *VMP*(p) für einen Prozeß p

Mit den maschinennahen Berechnungsobjekten und den für sie getroffenen Festlegungen ergeben sich die Rahmenbedingungen dafür, **Prozesse eines Systems** bei Bedarf **zu erzeugen** und, wenn sie nicht mehr benötigt werden, wieder aufzulösen. Die Maßnahmen, die für Prozeßerzeugungen erforderlich sind, werden im folgenden für den Prozeß p erklärt.
Der Prozeß p wird erzeugt, indem i.w. das Berechnungsobjekt von p erzeugt wird. Zudem muß der p–Prozessor erzeugt und initialisiert werden; damit existiert p im Zustand v gemäß Kapitel 5.1.2.2, und p kann gestartet werden.
Nach dem oben für die Berechnungsobjekte der Prozesse Erklärten, werden diese in den abstrakten Speichern der Prozesse gespeichert. Dem entspricht, daß bei Erzeugung von p das initiale Berechnungsobjekt von p in *VMP*(p) „geladen“ wird. Diese von realen Speichern übernommenen Begriffe werden im folgenden vermieden. Die für *VAP*(p) und *VMP*(p) erforderlichen Maßnahmen sind, wie bereits in Kapitel 5.1.2.2 erklärt, die folgenden:

- Bisher undefinierten *VAP*(p)–Elementen werden Bedeutungen zugeordnet; sie werden damit definiert.

- Indem *VAP*(p)–Elemente definiert werden, entstehen *VMP*(p)–Elemente; das sind Container für entsprechende Zustandsvariablen, die definierte oder undefinierte Werte haben können. In jedem Fall entstehen Container als *VMP*(p)–Elemente, die Inhalte haben.

- Diese Maßnahmen werden als **Prägen** von *VMP*(p) bezeichnet; das Ergebnis dieses Prägens ist ein strukturiertes Datenobjekt in *VMP*(p).

Wenn der Prozeß p erzeugt werden soll, wird das Berechnungsobjekt von p von Modulen in **Bindersprache** ausgehend erzeugt und *VMP*(p) wird mit diesem Objekt geprägt. Module in Bindersprache sind Werte von Dateien des „Typs“ *object*, auf die in Kapitel 6.1.1 hingewiesen wurde. Ein Modul m in Bindersprache

- definiert m–spezifische Bezeichner von Objekten und Objekte, die allein m–lokale Bedeutungen oder als m–Exporte nicht–m–lokale Bedeutungen haben können, und

- enthält anwendend Bezeichner, die m–lokale Bedeutungen haben und damit m–definiert sind, oder als m–Importe relativ zu m undefiniert sind.

Eine Object–Datei für m enthält den Modul m in Bindersprache, die m–Exporte und die m–Importe sowie alles, was für die Berechnung eines ausführfähigen Maschinenprogramms mit m erforderlich ist.
Das Berechnungsobjekt für den Prozeß p wird von einem für p gewählten Modul $m_0(p)$ in Bindersprache ausgehend berechnet, indem *code*(p) und *daten*(p) berechnet werden. *code*(p) und *daten*(p) werden berechnet, indem der Codeteil und der Datenteil von $m_0(p)$ kopiert und transformiert sowie um die entsprechenden, transformierten Teile der transitiven Hülle der $m_0(p)$–Importe erweitert werden. Die wesentlichen Transformationen, die dabei erforderlich sind, sind Ersetzungen der definierend und anwendend in den Modulen auftretenden Bezeichner durch die für *code*(p) und *daten*(p) festgelegten *VAP0*(p)–Elemente; dabei werden insbesondere alle relativ zu den Modulen undefiniert auftretenden Bezeichner durch definierte *VAP0*(p)–Elemente ersetzt. Diese Transformationen mit den transitiv abgeschlossenen Bezeichner–Ersetzungen bezeichnet man als **Binden** (linkage). *code*(p) und *daten*(p) ergeben sich also durch bzgl. der $m_0(d)$–Importe transitiv abgeschlossenes Kopieren und Binden der entsprechenden Modulteile.
Mit diesen Maßnahmen zur Erzeugung des Berechnungsobjekts für den Prozeß p werden die Anfangsteile von *VMP0*(p) mit *code*(p) und mit *daten*(p) geprägt; durch Erweiterung um *haldenbereich*(p) mit *halde*(p) ergibt sich die initiale Prägung von *VMP0*(p) mit *cdh_seg*(p). Zudem wird der Anfangsteil von *VMP1*(p) mit *kellerbereich*(p) und *keller*(p) geprägt; damit ergibt sich die initiale Prägung von *VMP1*(p) mit *k_seg*(p). Damit steht das maschinennahe **Berechnungsobjekt von** p mit dem Segmentpaar (*cdh_seg*, *k_seg*) zur Verfügung.
Zur Erzeugung des Prozesses p gehört zusätzlich die Erzeugung des **Prozeßdeskriptors** für p; er wird erzeugt und *deskriptor*(p) wird in die Prozeßliste *PL*, wie in Kapitel 5.1.2.2 erklärt, eingefügt. Zur Beschreibung des Prozesses p gehört wesentlich der Deskriptor des Berechnungszustands *berechnung*(p). Mit *berechnung*(p) erhält p seinen Prozessor durch Erzeugung und Initialisierung von *rp_register* als Teil von *berechnung*(p). *rp_register* ist, wie in Kapitel 5.1.2.2 erklärt, Teil des konzeptionell festgelegten p–privaten Wirkungsbereichs. Zu *rp_register* gehört wenigstens das Befehlsadreßregister; sein Inhalt wird dem Modul $m_0(p)$ entsprechend mit der Adresse des Startbefehls gemäß *code*(p) des Berechnungsobjekts von p festgelegt.
Mit den erklärten Datenobjekten, also mit den Segmenten *cdh_seg*(p) und *k_seg*(p) des Berechnungsobjekts sowie mit den Deskriptoren von p einschließlich *rp_register* als Teil von *berechnung*(p) ist der Prozeß p erzeugt. Der p–private Wirkungsbereich ist mit der Prägung von *VMP*(p) sowie mit *rp_register* definiert. Der Prozeß p existiert im Zustand v.
Den konzeptionellen Festlegungen entsprechend ist der Prozeß p an den Kern gebunden. Diese **Bindung von** p **an** *kern_seg*, mit der sich die Prozeß–Kern–Schnittstelle für p und

die Möglichkeit für p, Kerndienste implizit oder explizit aufzurufen, ergeben, entspricht der Festlegung, daß p das Segment *kern_seg* als Teil von $VM(p)$ mit $VA(p)$ erbt. Die Bindung von p an *kern_seg* erfolgt entsprechend auf der Grundlage des Adreßraums *VAK* und der Mechanismen, die der Prozessor r der HK gemäß Kapitel 5.1.2.2 für Kerndienstaufrufe zur Verfügung stellt.
Mit den erklärten Maßnahmen existiert der Prozeß p im Zustand v. Das Berechnungsobjekt von p und der p–private Wirkungsbereich sind definiert; sie existiert den konzeptionellen Festlegungen entsprechend, und p kann seine Berechnungen abstrakt ausführen. Für reale Ausführungen dieser Berechnungen müssen der abstrakte Speicher $VMP(p)$ realisiert und der p–Prozessor mit *rp_register* zum Kernprozessor mit *rk_register* erweitert werden. Wenn das erfolgt ist, kann der Prozeß p gestartet werden.
Als wesentliche noch zu lösende Aufgabe verbleibt die Realisierung des abstrakten Speichers von p und aller abstrakten Speicher mit den Speichern der HK.

6.2.2 Realisierungen der *VMs*

Für ein System mit der Menge P der koexistierenden Prozesse ergibt sich aus Kapitel 6.2.1, daß die p–privaten Wirkungsbereiche der $p \in P$ i.w. mit den Segmenten $cdh_seg(p)$ und $k_seg(p)$ in $VMP(p)$ und der Kern–Wirkungsbereich mit dem Segment *kern_seg* in *VMK* zur Verfügung stehen. Mit den abstrakten Prozessen ist damit der Parallelitätsgrad des Systems $|P|$. Die Prozesse können ihre Berechnungen real jedoch nur ausführen, wenn ihre Segmente und das Kern–Segment mit den Speichern der HK geeignet realisiert sind. Für den betrachteten Schnappschuß ergibt sich damit die Aufgabe, die Segmentmenge

$$SEG = \bigcup_{p \in P} \{cdh_seg(p), k_seg(p)\} \cup \{kern_seg\}$$

geeignet zu realisieren.
Weil die Prozesse für ihre Berechnungen den realen Prozessor r benötigen und r allein zum AS der 2–stufigen AS–HS–Hierarchie direkt zugreifen kann, kommen primär AS–Realisierungen der $seg \in SEG$ in Betracht. Zusätzlich sind HS–Realisierungen für *SEG*–Elemente dann von Interesse, wenn diese für Berechnungen nicht benötigt werden.
Sei A der **Adreßraum von AS**.
Für AS–Realisierungen der Segmente gibt es mehrere Möglichkeiten. Dazu sei $s \in SEG$. s in einem virtuellen Speicher ist eine Bytefolge. Wenn l_s die Länge dieser Folge ist, ist

$$I(s) = \{0, \ldots, l_s - 1\}$$

die **Byteindexmenge** von s. $I(s)$ entspricht einem Bereich eines virtuellen Adreßraums. Für eine AS–Realisierung von s, sind AS–Zellen für die Speicherung der s–Elemente erforderlich. Wenn diese festgelegt sind, ist die Zuordnung zwischen den $i \in I(s)$ und den entsprechenden $a \in A$ zu beschreiben. Das erfolgt mit einer **Speicherfunktion**

$$\sigma_s : I(s) \to A,$$

die injektiv sein muß.
Das Segment s kann vollständig oder partiell AS–realisiert werden. Wenn s **vollständig AS–realisiert** und σ_s die entsprechende Speicherfunktion ist, dann ist σ_s linkstotal; für alle $i \in I(s)$ enthält $\sigma_s(i)$ das s–Element mit dem Index i. Wenn s **partiell AS–realisiert** ist, ist die Speicherfunktion σ_s partiell und evtl. für alle $i \in I(s)$ undefiniert. Weil s als Datenobjekt definiert ist, müssen, falls s partiell AS–realisiert ist, die nicht AS–realisierten Elemente von s im HS gespeichert sein. s ist dann partiell oder vollständig HS–realisiert.

Für vollständig oder partiell AS–realisiertes s muß die Speicherfunktion σ_s bei jedem Zugriff des Prozessors r zu einem s–Element berechnet werden. Mit der Entscheidung für eine vollständige oder partielle AS–Realisierung von s muß demnach eine **effiziente Speicherfunktion** für s festgelegt werden: σ_s muß effizient beschrieben und berechnet werden können. Die Kombination dieser beiden Kriterien liefert die wesentlichen Verfahren für Realisierungen der Segmente: die segmentbasierten und die seitenbasierten *VM*–Realisierungen.
Für **segmentbasierte *VM*–Realisierungen** gilt folgendes:

- die Segmente werden jeweils vollständig (oder nicht) AS–realisiert;
- jedes AS–realisierte Segment wird mit einem AS–Bereich realisiert.

Wenn das Segment s diesen Festlegungen entsprechend mit dem AS–Bereich mit der Anfangsadresse $a \in A$ realisiert wird, kann für s die **Speicherfunktion**

$$\sigma_s : I(s) \to A \quad \text{(linkstotal und injektiv)}$$
$$\text{mit } \sigma_s(i) = a + i \text{ für alle } i \in I(s)$$

festgelegt werden. σ_s ist einfach beschreibbar (mit a) und einfach berechenbar. σ_s ist eine **1–stufig lineare** Speicherfunktion.
Segmentbasierte *VM*–Realisierungen haben neben dem angegebenen Vorteil i.a. gravierende **Nachteile**. Die angegebenen Festlegungen haben folgende Konsequenzen:

- Die Längen der Segmente werden durch die Kapazität von AS beschränkt.
- Die Festlegung, Segmente mit jeweils einem AS–Bereich zu realisieren, erschwert die AS–Verwaltung. Es werden jeweils AS–Bereiche mit passenden Längen benötigt. Das führt zu **externer Fragmentierung**.
- Die Beschränkung auf vollständige AS–Realisierungen mit jeweils einem Bereich **beschränkt den Parallelitätsgrad** eines Systems, und diese Beschränkung kann gravierend sein.

Die Beschränkung des Parallelitätsgrads eines Systems durch segmentbasierte *VM*–Realisierungen wird im folgenden genauer erklärt. Für das betrachtete System mit P ist der abstrakte Parallelitätsgrad $|P|$. Von der Segmentmenge *SEG* muß *kern_seg* stets AS–realisiert sein. Von den Prozessen $p \in P$ können Segmente AS–realisiert sein. p ist vollständig AS–realisiert, wenn *cdh_seg*(p) und *k_seg*(p) AS–realisiert sind. Sei P_{in} die Menge der vollständig AS–realisierten Prozesse. Dann können allein die Berechnungen der $p \in P_{in}$ (bzgl. der *VM*–Realisierungen) real fortschreiten. Die Segmente der $p \in P - P_{in}$ sind entweder vollständig oder partiell HS–realisiert. P zerfällt in drei Teilmengen: $P = P_{in} + P_{trans} + P_{out}$. P_{out} ist die Teilmenge der Prozesse, die vollständig HS–realisiert sind; sie sind **ausgelagert** oder aus AS verdrängt; sie belegen keine AS–Bereiche. $P_{trans} = P - P_{in} - P_{out}$; P_{trans} besteht aus den Prozessen, die weder vollständig AS– noch vollständig HS–realisiert sind; es sind die Prozesse, für die AS–HS–Umspeicherungen erfolgen; sie belegen AS–Bereiche. Es ergibt sich, daß der **Parallelitätsgrad** des Systems (bzgl. der *VM*–Realisierungen) $|P_{in}|$ ist; dabei kann $|P_{in}| \ll |P|$ gelten.
Die angegebenen Nachteile segmentbasierter *VM*–Realisierungen gelten für parallele Mehrbenutzersysteme; ihr Gewicht ist natürlich von den jeweiligen Gegebenheiten abhängig. Für Spezialsysteme, insbesondere für Echtzeitsysteme mit festen oder schwach variierenden Prozeßmengen kann bei entsprechenden AS–Kapazitäten $|P| \sim |P_{in}|$ erreicht werden; dann sind segmentbasierte *VM*–Realisierungen gut geeignet.

Für universelle, parallele Mehrbenutzersysteme sind die angegebenen Nachteile gravierend. Für diese sind die seitenbasierten *VM*–Realisierungen, die im folgenden Kapitel behandelt werden, geeignet.

6.2.3 Seitenbasierte *VM*–Realisierungen

Die in Kapitel 6.2.2 angegebenen Nachteile segmentbasierter *VM*–Realisierungen werden mit **seitenbasierten *VM*–Realisierungen** vermieden. Für sie gilt:

- Die Segmente werden in Seiten mit festgelegter Länge aufgeteilt;
- AS wird eine 2–dimensionale Struktur mit Kacheln, die den Seiten entsprechen, aufgeprägt;
- die Segmente werden auf der Grundlage der Seiten und Kacheln partiell AS–realisiert, genauer gemischt AS–HS–realisiert.

Der erste Schritt zu seitenbasierten *VM*–Realisierung erfolgt durch Festlegung der **Seitenlänge** pl. Es ist zweckmäßig, pl als Potenz von 2 zu wählen; übliche Längen sind 2^9 bis 2^{13}. Mit pl sei $D \triangleq \{0, \ldots, pl-1\}$. Mit pl wird dem AS und den Segmenten eine 2–dimensionale Struktur aufgeprägt, die Grundlage für alle Realisierungen ist.
Für AS mit dem Adreßraum A wird $|A| \equiv 0(pl)$ angenommen. Dann seien $|K| \triangleq |A|/pl$ und

$$K = \{0, \ldots, |K|-1\}.$$

K ist der **Kacheladreßraum** von AS. Eine Kachel ist ein AS–Bereich mit Adressen $\{a, a+1, \ldots, a+pl-1\} \subseteq A$ mit $a \equiv 0(pl)$. Der Zusammenhang zwischen der 1–dimensionalen und der 2–dimensionalen Struktur von AS wird mit der Bijektion

$$\beta_{AS} : A \to K \times D$$
$$\text{mit } \beta_{AS}(a) = (k, d) \Leftrightarrow a = k \cdot pl + d \text{ für alle } a \in A$$

hergestellt, so daß beide Strukturen nach Zweckmäßigkeit benutzt werden können.
Einem Segment s mit der Byteindexmenge $I(s) = \{0, \ldots, l_s - 1\}$ wird ebenfalls eine 2–dimensionale Struktur aufgeprägt. Mit $b_s \triangleq \lceil l_s/pl \rceil$ sei

$$B(s) \triangleq \{0, \ldots, b_s - 1\}.$$

$B(s)$ ist die **Seitenindexmenge** von s. Eine Seite von s ist eine Folge von s–Elementen mit den Byteindizes $\{i, i+1, \ldots, i+pl-1\} \subseteq I(s)$ mit $i \equiv 0(pl)$. Der Zusammenhang zwischen der 1– und der 2–dimensionalen Struktur von s wird mit der linkstotalen, injektiven Abbildung

$$\beta_s : I(s) \to B(s) \times D$$
$$\text{mit } \beta_s(i) = (b, d) \Leftrightarrow i = b \cdot pl + d \text{ für alle } i \in I(s)$$

hergestellt, so daß beide Strukturen nach Zweckmäßigkeit benutzt werden können. Weil nicht notwendig $l_s \equiv 0(pl)$ gilt, ist $\beta_s(I(s)) \subset B(s) \times D$ möglich; das ist die schon bekannte **interne Fragmentierung**, die hier auftreten kann.
Für das Segment s ergibt sich eine **seitenbasierte AS–Realisierung** dadurch, daß Seiten von s mit Kacheln von AS realisiert werden. Sie wird durch die injektive Abbildung

$$\sigma_s^{AS} : B(s) \to K,$$

die seitenbasierte **AS–Speicherfunktion** von s, beschrieben. σ_s^{AS} kann linkstotal oder partiell sein; das entspricht einer **vollständigen oder partiellen AS–Realisierung** von s.
Die Seiten von s, die nicht AS–realisiert sind, müssen mit HS realisiert werden. Eine **seitenbasierte HS–Realisierung** wird konstruiert, indem für einen Speicher $h \in H$ Blockbereiche so festgelegt werden, daß sie zum Speichern von Seiten geeignet sind. Für h sei der entsprechende Blockbereichsadreßraum $BA(h)$. Dann ergibt sich eine seitenbasierte HS–Realisierung von s mit h dadurch, daß Seiten von s mit Blockbereichen von h realisiert werden. Sie wird durch eine injektive Abbildung

$$\sigma_s^{HS} : B(s) \to BA(h),$$

die seitenbasierte **HS–Speicherfunktion** von s, beschrieben. σ_s^{HS} kann linkstotal oder partiell sein; das entspricht einer **vollständigen oder partiellen HS–Realisierung** von s.
Die 2–dimensionale Struktur des Segments s mit Seiten und die einheitliche Vorgehensweise für seitenbasierte AS– und HS–Realisierungen ermöglichen es, die Realisierungen von s zu konstruieren, mit denen die für *VM*–Realisierungen erforderliche Flexibilität und Effizienz erreicht wird: die **vollständigen, gemischten AS–HS–Realisierungen**. Für s wird eine gemischte AS–HS–Realisierung mit dem Speicherfunktionenpaar $(\sigma_s^{AS}, \sigma_s^{HS})$ beschrieben. Vollständig trägt dem Rechnung, daß das definierte Datenobjekt stets vollständig gespeichert sein muß. Für Vollständigkeit muß das Paar $(\sigma_s^{AS}, \sigma_s^{HS})$ folgende Bedingungen erfüllen:

- Für alle $b \in B(s)$ ist $\sigma_s^{AS}(b)$ oder $\sigma_s^{HS}(b)$ definiert.
- Wenn für $b \in B(s)$ sowohl $\sigma_s^{AS}(b)$ als auch $\sigma_s^{HS}(b)$ definiert sind, sind der Inhalt von $\sigma_s^{AS}(b)$ und der Inhalt von $\sigma_s^{HS}(b)$ (bis auf Codierungen) gleich.

Im weiteren wird angenommen, daß sämtliche Segmente $seg \in SEG$ vollständig, gemischt AS–HS–realisiert sind.
Wenn das Segment s vollständig, gemischt AS–HS–realisiert ist, sind für Berechnungen, die auf s ausgeführt werden, **Zugriffe des Prozessors** r zu s erforderlich. Diese Zugriffe erfolgen auf der Grundlage der $i \in I(s)$ zur Identifikation der s–Elemente (die virtuellen Adressen), und es sind allein Zugriffe zu den AS–realisierten Seiten von s möglich. Für diese Zugriffe benötigt r eine geeignete **Speicherfunktion**, die als injektive Abbildung

$$\sigma_s : I(s) \to A$$

mit σ_s^{AS} wie folgt definiert ist: für alle $i \in I(s)$ mit $\beta_s(i) = (b, d)$ gilt

$$\sigma_s(i) = \begin{cases} \sigma_s^{AS}(b) \cdot pl + d & , \text{ falls } \sigma_s^{AS}(b) \text{ definiert ist} \\ \perp & , \text{ falls } \sigma_s^{AS}(b) = \perp \end{cases}$$

Das ist bei Verwendung einer Tabelle für σ_s^{AS} eine **2–stufig lineare** Speicherfunktion.
Wenn bei einem Versuch von r, zum Element von s mit dem Index i zuzugreifen, das Bild $\sigma_s(i)$ definiert ist, kann der Zugriff erfolgen und erfolgt. Wenn $\sigma_s(i) = \perp$ gilt, tritt ein sog. **Seitenfehler** auf. Er bewirkt eine Programmausführungs–Unterbrechung für r; der Zugriffsversuch zu i von s muß wiederholt werden; er ist erst dann erfolgreich, wenn die Seite von s, zu der das Element mit i gehört, AS–realisiert ist.
Das Speicherfunktionenpaar $(\sigma_s^{AS}, \sigma_s^{HS})$ einer vollständigen, gemischten AS–HS–Realisierung von s muß beschrieben und gespeichert werden; das erfolgt mit entsprechenden Tabellen. Die **Seiten–Kachel–Tabelle** $SKT(s)$ dient dazu, $\sigma_s^{AS}(B(s))$ zu speichern. Die Elemente von $SKT(s)$ sind Seitendeskriptoren, die für $b \in B(s)$ neben $\sigma_s^{AS}(b)$ weitere Verwaltungsinformation enthalten. Die **Seiten–Blockbereichs–Tabelle** $SBT(s)$ dient dazu, $\sigma_s^{HS}(B(s))$ zu speichern. Die Elemente von $SBT(s)$ sind Seitendeskriptoren, die für $b \in B(s)$ neben $\sigma_s^{HS}(b)$

weitere Verwaltungsinformation enthalten. Die $SKT(\cdot)$ und $SBT(\cdot)$ der Segmente $seg \in SEG$ sind als Datenobjekte des Kerns Datenobjekte in *kern_seg*.
Der Prozessor r muß für alle Zugriffe zu Segmenten gemäß SEG die oben für das Segment s erklärte 2–stufig lineare Speicherfunktion σ_s berechnen. Dazu sind Zugriffe zu den $SKT(\cdot)$ erforderlich. Zur Verkürzung der Berechnungszeiten für diese Speicherfunktionen ist r mit einem speziellen Assoziativspeicher für einen Teil der $SKT(\cdot)$, einem *TLB* für **Translation Look–aside Buffer**, und einer *MMU* für **Memory–Management Unit** ausgestattet.
Das bisher für seitenbasierte *VM*–Realisierungen Erklärte liefert die Rahmenbedingungen für die noch zu lösende Aufgabe: Die Segmente, die als Datenobjekte in den *VMs* existieren, müssen mit den Speichern der HK so realisiert werden, daß die Berechnungen des Systems effizient ausgeführt werden können. Mit der Menge P der koexistierenden Prozesse und der in Kapitel 6.2.2 eingeführten Segmentmenge SEG gilt dabei:

- Die Segmente $s \in SEG$ werden jeweils vollständig und gemischt AS–HS–realisiert.

Dabei sind für die einzelnen Segmente der Prozesse auch die Sonderfälle vollständiger AS– und vollständiger HS–Realisierungen zugelassen. Für die Prozesse wird zudem festgelegt, daß sie mit vollständig HS–realisierten privaten Segmenten ($cdh_seg(p)$ und $k_seg(p)$ für $p \in P$) erzeugt werden. Das genannte Ziel der Realisierungen der Segmente mit den Speichern der HK, effiziente Ausführungen der Berechnungen des Systems, wird im folgenden mit dem Parallelitätsgrad des Systems bzgl. der *VM*–Realisierungen präzisiert.
Mit den erklärten Realisierungsmöglichkeiten der Segmente $s \in SEG$ ergibt sich zunächst, daß die Prozeßmenge in drei Teilmengen zerfällt: $P = P_{in} + P_{mix} + P_{out}$. Dabei ist P_{in} (P_{out}) die Teilmenge der Prozesse, für die beide privaten Segmente vollständig AS–realisiert (vollständig HS–realisiert) sind; P_{mix} ist die Teilmenge der Prozesse, deren private Segmente echt gemischt AS–HS–realisiert sind. Für die Prozesse $p \in P_{in}$ ($p \in P_{out}$) ist klar, daß deren Berechnungen bzgl. der *VM*–Realisierungen fortschreiten (nicht fortschreiten) können. Die Prozeßmenge P_{mix} zerfällt ihrerseits in zwei Teilmengen: $P_{mix} = P_{a-mix} + P_{w-mix}$. P_{a-mix} besteht aus den Prozessen, deren Berechnungen bzgl. der *VM*–Realisierungen fortschreiten können; die Seiten, die sie dazu benötigen, sind AS–realisiert. P_{w-mix} besteht aus den Prozessen, die blockiert sind, weil die Seiten, die sie benötigen, nicht AS–realisiert sind; sie bleiben blockiert, bis die benötigten Seiten AS–realisiert zur Verfügung stehen. Damit ergibt sich $|P_{in}| + |P_{a-mix}|$ als **Parallelitätsgrad** des Systems bzgl. der *VM*–Realisierungen. Die angegebene Zerlegung von P ist die eines Schnappschusses. Mit geeigneten Verfahren zur Realisierung der Segmente mit den Speichern der HK muß erreicht werden, daß der erklärte Parallelitätsgrad des Systems im zeitlichen Mittel hoch ist.

6.3 Strategien seitenbasierter Speicherverwaltung

Mit den in Kapitel 6.2.3 erklärten Rahmenbedingungen für seitenbasierte *VM*–Realisierungen bleibt die Aufgabe zu lösen, die mit der HK zur Verfügung stehenden Speicher zu verwalten und für vollständige, gemischte AS–HS–Realisierungen der Segmente so einzusetzen, daß die Berechnungen des Systems effizient ausgeführt werden können.
Zur **Verwaltung** der Speicher der HK gehört, daß die Belegungszustände *frei/belegt* der Speicherkomponenten, also der Kacheln von AS und der Blockbereiche der $h \in H$, registriert und fortgeschrieben werden. Zur Verwaltung gehört vor allem, daß die für die Segment–Realisierungen benötigten Kacheln und Blockbereiche den jeweiligen Anforderungen entsprechend zur Verfügung gestellt werden. Gefordert ist, daß die Berechnungen des Systems, also die Kern– und insbesondere die Prozeß–Berechnungen, effizient fortschreiten können.

Den Gegebenheiten für dieses Fortschreiten entsprechend ist damit gefordert, daß die jeweils für die Berechnungen benötigten Seiten AS–realisiert zur Verfügung gestellt werden. Seitenbasierte Speicherverwaltung erfordert deshalb primär leistungsfähige **Strategien zur kachelbasierten AS–Verwaltung**: Für die Seiten der Segmente werden AS–Realisierungen benötigt; HS–Realisierungen dienen dazu, trotz beschränkter AS–Kapazität effiziente AS–Realisierungen zu erreichen.
Für Lösungen der gestellten Verwaltungsaufgabe sind geeignete Verhaltensbeschreibungen des Systems und geeignete Leistungskriterien erforderlich. Für Verhaltensbeschreibungen des Systems ist zu beachten, daß Berechnungen dann fortschreiten, wenn für sie ein Prozessor eingesetzt ist. Mit dem R–Prozessor r der HK ergibt sich, daß das Verhalten geeignet mit Seitenreferenzspuren beschrieben werden kann. Eine **Seitenreferenzspur** R ist eine Folge

$$R = r_1 r_2 r_3 \ldots$$

mit Bezeichnern r_i für Segment–Seiten–Paare, welche die Seiten in der Reihenfolge der Zugriffe bei Ausführung der sequentiellen Berechnungsschritte des Prozessors r identifizieren. Die lineare Ordnung der Spur ist einerseits von den Berechnungsschritten des Prozessors r bestimmt; sie entspricht also den Prozessor–Programmkonzepten. Die Spur enthält andererseits Bezeichner für Segment–Seiten–Paare, so daß die Zuordnungen der Zugriffe zu den Prozeß–Segmenten und zum Kern–Segment erfaßt werden. Damit wird erreicht, daß modifizierte Seitenreferenzspuren als Verhaltensbeschreibungen für Lösungen der gestellten Verwaltungsaufgaben benutzt werden können.
Mit $n \in \mathbb{N}$ sei $R_n = r_1 \ldots r_n$ ein Präfix der Referenzspur R. Eine Referenz $r_i, 1 \leq i \leq n$, kann in Abhängigkeit davon, ob die mit r_i identifizierte Seite AS– oder HS–realisiert ist, entweder zu einem erfolgreichen Zugriff zur Seite, einem Treffer, oder zu einem nicht erfolgreichen Zugriffsversuch, einem sog. Seitenfehler, führen. Diese beiden Alternativen, die unterscheidbar sind, weil r_i ein Segment–Seiten–Paar–Bezeichner ist, führen zu Leistungskriterien für Strategien kachelbasierter AS–Verwaltung. Wenn f_n die Anzahl der bei Abarbeitung von R_n auftretenden Seitenfehler ist, ist $\varphi_n = f_n/n$ die **Seitenfehlerrate**; $1 - \varphi_n$ ist die entsprechende **Trefferrate**. Ein Optimierungsziel für Strategien kachelbasierter AS–Verwaltung ist demnach, bei gegebenen Referenzspuren möglichst kleine Seitenfehlerraten zu erreichen. Was dabei klein bedeutet, ergibt sich mit der Zugriffszeit t_{AS} zu AS und der Zugriffszeit t_{HS} zu einem Speicher $h \in H$ wie folgt: Mit der Seitenfehlerrate φ ist die (mittlere) Zugriffszeit zu gemischt AS–HS–realisierten Segmenten

$$t_\varphi = (1 - \varphi) t_{AS} + \varphi t_{HS}.$$

Dabei ist der Quotient t_{HS}/t_{AS} von der Größenordnung 10^5.
Mit dem bisher Erklärten ergibt sich für die zu lösende Verwaltungsaufgabe, daß von Verhaltensbeschreibungen eines Systems mit Referenzspuren ausgehend Strategien kachelbasierter AS–Verwaltung mit möglichst kleinen Seitenfehlerraten zu konstruieren sind. Die Referenzspuren beschreiben die Anforderungen, die gestellt werden; sie sollen für die Berechnungen, die auszuführen sind, die Gesetzmäßigkeiten der Zugriffe zu Seiten erfassen. Die oben erklärte Referenzspur R erfaßt die Referenzen aus der Sicht des Prozessors r; sie enthält entsprechend durchmischt Referenzen für die Segmente der Prozesse $p \in P$ und für das Kern–Segment. Für die zu lösende Verwaltungsaufgabe sind diese Segmente unterschiedlich zu behandeln; es ist deshalb geboten, die Referenzspuren separat zu erfassen. Das kann dadurch erfolgen, daß die Referenzspur R durch Projektionen aufgeteilt wird; alternativ dazu kann angenommen werden, daß Referenzspuren bzgl. des Kern–Segments und bzgl. der privaten Segmente der Prozesse $p \in P$ vorliegen.

Die AS–Realisierungen der Seiten des Kern–Segments erfordern eine Sonderbehandlung, weil dieses Segment von allen Prozessen benötigt wird. Im folgenden wird *kern_seg* zunächst nicht weiter berücksichtigt; es wird als AS–realisiert angenommen. Für die Prozesse $p \in P$ wird angenommen, daß ihr Verhalten mit Referenzspuren $R(p)$ beschrieben wird; dabei sind in $R(p)$ die Referenzen für die beiden privaten Segmente *cdh_seg*(p) und *k_seg*(p) zusammengefaßt.
Von diesen Gegebenheiten ausgehend gibt es für die gestellte Verwaltungsaufgabe zahlreiche Lösungsansätze; sie führen zu Strategieklassen und Strategien, die mehr oder weniger geeignete Lösungen liefern. Die Verfahren können danach klassifiziert werden, wann die Realisierungsentscheidungen getroffen werden; das führt zu vorausplanenden Strategien und zu Anforderungsstrategien. Mit **vorausplanenden Strategien** (prepaging) wird versucht, den Prozessen vorausplanend, also bevor die entsprechenden Zugriffe erfolgen, die jeweils benötigten Seiten AS–realisiert zur Verfügung zu stellen. Mit **Anforderungsstrategien** (demand paging) werden die Realisierungsentscheidungen bei Anforderungen, also bei Zugriffsversuchen, die zu Seitenfehlern führen, getroffen. Für beide Strategieklassen gibt es Varianten in Abhängigkeit davon, für welche Seitenmengen (ein– oder mehrelementige) die Realisierungsentscheidungen getroffen werden. Die Verfahren können weiter danach klassifiziert werden, ob die Realisierungsentscheidungen für die einzelnen Prozesse separat oder unter Berücksichtigung jeweils aller koexistierenden Prozesse getroffen werden. Das führt zur Klasse der **prozeßlokalen Strategien** und zur Klasse der **prozeßglobalen Strategien** für kachelbasierte AS–Verwaltung.

6.3.1 Prozeßlokale Einzelseiten–Anforderungsstrategien

Die in mehrfacher Hinsicht einfachsten Strategien zur kachelbasierten AS–Verwaltung sind die **prozeßlokalen Einzelseiten–Anforderungsstrategien**. Für sie gilt folgendes:

- Die einzelnen koexistierenden Prozesse gemäß P werden separat behandelt. Einem Prozeß $p \in P$ wird für die Ausführung seiner Berechnungen eine feste Kachelmenge K_p zur Verfügung gestellt.

- Für einen Prozeß p, dem die Kachelmenge K_p zugeordnet ist, werden die Realisierungsentscheidungen auf der Grundlage einer Referenzspur R_p bzgl. der Segmente *cdh_seg*(p) und *k_seg*(p) getroffen. Die jeweils benötigten Seiten werden mit K_p AS–realisiert, und die Realisierungsentscheidungen werden jeweils für einzelne Seiten auf Anforderung, also beim Auftreten eines entsprechenden Seitenfehlers, getroffen.

Wenn P die Menge der koexistierenden Prozesse eines Systems und K' die Menge der für die Prozesse verfügbaren Kacheln (nach Abzug der für *kern_seg* benötigten) von AS sind, wird für die hier betrachteten Strategien wie folgt verfahren:

- Die Kacheln gemäß K' werden in Mengen zerlegt, und diese Kachelmengen werden Prozessen von P zugeordnet; damit ergibt sich eine Aufteilung der Prozeßmenge: $P = P_a + P_b$.

- Den Prozessen $p \in P_a$ sind Kachelmengen $K_p \neq \emptyset$ zugeordnet; sie können ihre Berechnungen ausführen; dabei sind die Seiten, die p benötigt, mit den Kacheln gemäß K_p zu realisieren.

- Den Prozessen $p \in P_b$ sind keine Kacheln zugeordnet; sie können ihre Berechnungen nicht ausführen und bleiben bis zu einer Revision der Verteilung der Kacheln gemäß K' blockiert.

- Eine Revision der Verteilung der Kacheln gemäß K' erfolgt, wenn einer der Prozesse $p \in P_a$ die Ausführung seiner Berechnungen abgeschlossen hat. Dann werden die Kacheln gemäß K_p frei; sie werden einem Prozeß oder mehreren Prozessen aus P_b zugeteilt.

Wenn sich mit diesem Vorgehen die Aufteilung der Prozeßmenge $P = P_a + P_b$ ergibt, ist $|P_a|$ obere Schranke für den Parallelitätsgrad des Systems bzgl. der *VM*–Realisierungen. Sie gilt für das lange Zeitintervall bis zur nächsten Revision der K'–Verteilung. In diesem Zeitintervall werden die prozeßlokalen Realisierungsentscheidungen für die einzelnen $p \in P_a$ getroffen, aus denen sich der Parallelitätsgrad des Systems bzgl. der *VM*–Realisierungen ergibt.
Die **charakteristischen Eigenschaften** der Einzelseiten–Anforderungsstrategien werden im folgenden für einen Prozeß p mit der Referenzspur R_p erklärt. Dabei wird angenommen, daß p eine Kachelmenge mit $m \in I\!N$ Elementen zugeordnet ist.
Die Maßnahmen und Wirkungen einer Strategie für p werden mit einer **Zustandsspur** $Z_0 Z_1 \ldots$ beschrieben. Ein Zustand Z_i ist jeweils die Menge der Seiten von p, die mit den zugeordneten Kacheln AS–realisiert sind; es gilt also $0 \leq |Z_i| \leq m$. Der Anfangszustand ist $Z_0 \triangleq \emptyset$.
Mit einer festgelegten Strategie ergibt sich von Z_0 ausgehend mit den Schritten der Referenzspur R_p eine Zustandsspur: die $r_1 r_2 \ldots$ führen zu Seitenfehlern; die entsprechenden Seiten werden AS–realisiert und damit ergibt sich $Z_0 Z_1 Z_2 \ldots$. Diese Schritte erfordern keine wesentlichen Entscheidungen, bis für ein n erstmals ein Zustand Z_n mit $|Z_n| = m$ erreicht wird. Für den Folgeschritt mit r_{n+1} sind die beiden Fälle $r_{n+1} \notin Z_n$ und $r_{n+1} \in Z_n$ möglich. Für $r_{n+1} \notin Z_n$ ist eine relevante Realisierungsentscheidung erforderlich:

- Eine Seite $r \in Z_n$ muß ausgewählt und **aus AS verdrängt** werden: Für r ist anstelle der vorliegenden AS–Realisierung eine HS–Realisierung erforderlich; sie wird erreicht, indem r von AS in HS umgespeichert wird.

- Nach Verdrängung von r kann die Seite r_{n+1} mit der bisher für r belegten Kachel AS–realisiert werden; das wird erreicht, indem die Seite r_{n+1}, die HS–realisiert vorliegt, von HS **in AS geladen** wird.

Für die beiden Fälle mit r_{n+1} ergeben sich also:

- falls $r_{n+1} \in Z_n$: der Zugriff zu r_{n+1} ist ohne weitere Maßnahmen möglich und es gilt $Z_{n+1} \triangleq Z_n$.

- falls $r_{n+1} \notin Z_n$: der Zugriffsversuch zu r_{n+1} führt zu einem Seitenfehler; nach Auswahl eines $r \in Z_n$, Verdrängung von r und Laden von r_{n+1} gilt $Z_{n+1} \triangleq (Z_n - \{r\}) + \{r_{n+1}\}$; damit ist r_{n+1} AS–realisiert und der Zugriff zu r_{n+1} kann erfolgen.

Der für den Schritt mit r_{n+1} erklärte Zustandsübergang

$$Z_{n+1} \triangleq \begin{cases} Z_n & \text{, falls } r_{n+1} \in Z_n \\ (Z_n - \{r\}) + \{r_{n+1}\} \text{ mit } r \in Z_n & \text{, falls } r_{n+1} \notin Z_n \end{cases}$$

gilt entsprechend für alle weiteren Schritte der Spur R_p. Die strategische Entscheidung, die zu treffen ist, ist die Auswahl von r, also eine Verdrängungs–Entscheidung. Für diese Entscheidungen sind geeignete Kriterien erforderlich; Einzelseiten–Anforderungsstrategien sind also i.w. Einzelseiten–**Verdrängungsstrategien**. Wenn die Verdrängungs–Entscheidung getroffen ist, sind die weiteren erforderlichen Maßnahmen festgelegt.

Für die notwendigen Verdrängungs–Entscheidungen gibt es zahlreiche Kriterien; sie führen zu Verdrängungsstrategien mit entsprechenden Bezeichnungen. Einige dieser Kriterien und Strategien sind im folgenden angegeben:

- RANDOM: die zu verdrängende Seite wird zufällig, d.h. mit der Wahrscheinlichkeit $1/m$ für alle $r \in Z$, ausgewählt.
- FIFO: die Seiten werden in der Reihenfolge verdrängt, in der sie geladen werden.
- LIFO: die Seiten werden antiton zur Ladereihenfolge verdrängt.
- BO — Belady's Optimal: die Seiten mit dem jeweils größten Vorwärtsabstand (Abstand in der Zukunft gemäß R_p) werden verdrängt. Diese Strategie ist bzgl. der Seitenfehler optimal; sie ist jedoch unrealistisch, weil R_p im voraus bekannt sein muß, was i.a nicht der Fall ist.
- LRU — Least Recently Used: die Seiten mit dem jeweils größten Rückwärtsabstand (Abstand in der Vergangenheit gemäß R_p) werden verdrängt. Diese Strategie beobachtet die Vergangenheit und schließt aus ihr auf die Zukunft; sie berücksichtigt die Lokalität des Zugriffsverhaltens gemessen in Schrittfolgelängen.
- LFU — Least Frequently Used: die Seiten mit der jeweils kleinsten Rückwärtshäufigkeit (Häufigkeit in der Vergangenheit gemäß R_p) werden verdrängt. Diese Strategie beobachtet die Vergangenheit und schließt aus ihr auf die Zukunft; sie berücksichtigt die Lokalität des Zugriffsverhaltens gemessen in Zugriffshäufigkeiten.

Abbildung 6.5 zeigt für eine gegebene Referenzspur R_p die Zustandsspuren, die sich mit den Strategien FIFO und LRU bei 3 bzw. 4 zugeordneten Kacheln ergeben. Zustandsübergänge mit Seitenfehlern sind mit $*$ markiert; die Anzahl der Seitenfehler ist jeweils als f–Wert angegeben.

	R_p :	0	1	2	3	0	1	4	0	1	0	1	2	3	4	
	0	0^*	0	0	3^*	3	3	4^*	4	4	4	4	4	4	4	
FIFO	1		1^*	1	1	0^*	0	0	0	0	0	0	2^*	2	2	
	2			2^*	2	2	1^*	1	1	1	1	1	1	3^*	3	$f = 9$
	0	0^*	0	0	3^*	3	3	4^*	4	4	4	4	2^*	2	2	
LRU	1		1^*	1	1	0^*	0	0	0	0	0	0	0	3^*	3	
	2			2^*	2	2	1^*	1	1	1	1	1	1	1	4^*	$f = 10$
	0	0^*	0	0	0	0	0	4^*	4	4	4	4	4	3^*	3	
FIFO	1		1^*	1	1	1	1	1	0^*	0	0	0	0	0	4^*	
	2			2^*	2	2	2	2	2	1^*	1	1	1	1	1	
	3				3^*	3	3	3	3	3	3	3	2^*	2	2	$f = 10$
	0	0^*	0	0	0	0	0	0	0	0	0	0	0	0	4^*	
LRU	1		1^*	1	1	1	1	1	1	1	1	1	1	1	1	
	2			2^*	2	2	2	4^*	4	4	4	4	4	3^*	3	
	3				3^*	3	3	3	3	3	3	3	2^*	2	2	$f = 8$

Abbildung 6.5: Beispiele für Einzelseiten–Anforderungsstrategien

Die hier angegebenen Strategien lehren Typisches für die hier betrachtete Strategienklasse:

- Von den 6 Strategien geht RANDOM davon aus, daß keine Kenntnisse über das Referenzverhalten des Prozesses p vorliegen; die übrigen Strategien gehen von entsprechenden Kenntnissen aus und werten sie aus. RANDOM kann, weil auf Beobachtungen verzichtet wird, mit geringem Aufwand implementiert werden; für die Verdrängungsentscheidung ist lediglich ein Zufallszahlengenerator erforderlich. Eine allgemeine Bewertung der Leistungsfähigkeit von RANDOM im Vergleich mit den beobachtenden Strategien ist nicht möglich (von BO abgesehen), weil der Wert der Beobachtungsergebnisse nicht allgemein bekannt ist.

- Die Strategie BO ist optimal; sie setzt voraus, daß die Referenzspur R_p bekannt ist. Für Berechnungen, die häufig ausgeführt werden, können Kenntnisse über das Referenzverhalten gesammelt und dann ausgewertet werden.

- Im allgemeinen ist das Referenzverhalten von Berechnungen im voraus unbekannt. Dann bleibt, wenn nicht wie bei RANDOM zufällig entschieden werden soll, das Beobachten des Referenzverhaltens und die Erwartung, aus dem Verhalten in der Vergangenheit auf das zukünftige Verhalten schließen zu können. Verhaltensbeobachtungen erfordern zusätzlichen Aufwand, und die Auswertung dieser Beobachtungsergebnisse für Verdrängungs–Entscheidungen erfordern ebenfalls zusätzlichen Aufwand. Dieser Aufwand kann für die einzelnen Strategien wesentlich verschieden sein; LRU und LFU sind Beispiele hierfür. Dieser Aufwand ist gerechtfertigt, wenn sich mit den entsprechenden Strategien kleine Seitenfehlerraten erreichen lassen. Analysen haben gezeigt, daß Lokalität des Zugriffsverhaltens von Berechnungen häufig auftritt und erfolgreich genutzt werden kann.

- Die beiden Strategien LRU und LFU, die wesentlich die Lokalität des Zugriffsverhaltens von Berechnungen nutzen, zeigen typisch die Wechselbeziehungen, die zwischen dem Aufwand für Beobachtungen und den Kenntnissen, die mit Beobachtungen gewonnen werden, bestehen. Für den Wert von Beobachtungsergebnissen kommt hier jedoch noch die Unsicherheit des Schließens von der Vergangenheit auf die Zukunft hinzu.

- Alle gewonnenen Beobachtungsergebnisse müssen für ihre Auswertungen bei Verdrängungs–Entscheidungen gespeichert werden; die naheliegende Möglichkeit hierfür sind die Seitendeskriptoren der entsprechenden Tabellen, die für Berechnungen der Speicherfunktionen benötigt werden.

An den Beispielen für Referenz– und Zustandsspuren, die in Abbildung 6.5 gezeigt sind, ist natürlich die Anzahl zugeordneter Kacheln unrealistisch; in Wirklichkeit werden wesentlich mächtigere Kachelmengen zugeordnet. Die Beispiele zeigen jedoch ein für die hier betrachtete Strategienklasse wichtiges Phänomen: Die Vergrößerung der Menge zugeordneter Kacheln hat nicht notwendig eine Reduktion der Anzahl der Seitenfehler zur Folge. Im Beispiel tritt dies als sog. **FIFO–Anomalie** auf.
Diese Beobachtung führt zu einer Minimalforderung an geeignete Einzelseiten–Anforderungsstrategien: Eine Verdrängungsstrategie soll die Eigenschaft haben, daß eine Vergrößerung der Anzahl zugeordneter Kacheln kein Anwachsen der Anzahl der Seitenfehler zur Folge hat. Die Klasse der Verdrängungsstrategien, die diese Forderung erfüllt, ist die Klasse der **Kellerstrategien**. Für sie gilt

$$Z_n(R_p, m) \subseteq Z_n(R_p, m+1)$$
für alle Referenzspuren $R_p, n \in \mathbb{N}_0$ und $m \in \mathbb{N}$.

Dabei ist $Z_n(R_p, x)$ der n–te Zustand bei Abarbeitung von R_p mit x zugeordneten Kacheln. Die Bezeichnung der Strategie–Klasse ergibt sich daraus, daß mit dem Verdrängungskriterium einer entsprechenden Strategie für alle $n \in \mathbb{N}$ die Seiten des Präfix $r_1 \ldots r_n$ einer Referenzspur in einem Keller der Tiefe $|\{r_1, \ldots, r_n\}|$ so geordnet werden, daß sie bei Verdrängung wegen der Anzahl m zugeordneter Kacheln in der festgelegten Reihenfolge vom Kellerkopf zum Kellerfuß verdrängt werden. LRU ist ein Beispiel für eine Kellerstrategie. Abbildung 6.6 zeigt LRU als Kellerstrategie für die Referenzspur und die Anzahl zugeordneter Kacheln, die schon von Abbildung 6.5 bekannt sind.

	4						2	2	2	2	2	3	4	0	m=4
LRU–	3				0	1	2	3	3	3	3	3	4	0	1	m=3
Keller	2			0	1	2	3	0	1	4	4	4	0	1	2	
	1		0	1	2	3	0	1	4	0	1	0	1	2	3	
	0	0	1	2	3	0	1	4	0	1	0	1	2	3	4	
	R_p	0	1	2	3	0	1	4	0	1	0	1	2	3	4	

Abbildung 6.6: LRU als Beispiel für eine Keller–Strategie

Auch mit Kellerstrategien können hohe Seitenfehlerraten nicht ausgeschlossen werden. Hohe Seitenfehlerraten können zum sog. **Seitenflattern** (Trashing) führen, was bedeutet, daß die Berechnungen eines Prozesses mehrerer oder aller Prozesse wegen der notwendigen AS–HS–Umspeicherungen von Seiten nicht mehr fortschreiten. Die wesentliche Ursache hierfür ist die für lange Zeitintervalle geltende, starre Zuteilung von Kacheln an Prozesse, die für die hier betrachtete Klasse von Strategien charakteristisch ist. Zur Verbesserung der Effizienz eines Systems und insbesondere zur Verhinderung von Seitenflattern sind Strategien mit flexiblen, anpassungsfähigen Kachelzuteilungen erforderlich.

6.3.2 Prozeßglobale, anpassungsfähige Strategien

Zur Erklärung der prozeßglobalen, anpassungsfähigen Strategien für kachelbasierte AS–Verwaltung, mit denen die gravierenden Mängel der Einzelseiten–Anforderungsstrategien überwunden werden können, wird analog zu Kapitel 6.3.1 ein System mit der Prozeßmenge P und der Kachelmenge K' betrachtet. Mit den Strategien wird wie folgt verfahren:

- Die Kacheln gemäß K' werden in Mengen zerlegt, und diese werden Prozessen von P zugeordnet; damit ergibt sich eine Aufteilung der Prozeßmenge: $P = P_a + P_b$.
- Den Prozessen $p \in P_a$ sind Kachelmengen $K_p \neq \emptyset$ zugeordnet; sie können ihre Berechnungen ausführen; dabei sind die Seiten, die p benötigt, mit den Kacheln gemäß K_p zu realisieren.
- Den Prozessen $p \in P_b$ sind keine Kacheln zugeordnet; sie können ihre Berechnungen nicht ausführen und bleiben bis zu einer Revision der Verteilung der Kacheln gemäß K' blockiert.
- Dieser Teil der strategischen Maßnahmen ist denen der in Kapitel 6.3.1 für die dort erklärten Strategien ähnlich. Sie unterscheiden sich von diesen wie folgt:
  - Die Menge K' wird so zerlegt, daß sich für die Prozesse $p \in P_a$ Kachelmenge K_p ergeben, mit denen die Prozesse ihre Berechnungen mit kleinen Seitenfehlerraten

ausführen können; die Seiten, die p benötigt, werden initial mit den Kacheln gemäß K_p realisiert.

– Eine Revision der Verteilung der Kacheln gemäß K' erfolgt immer dann, wenn Beobachtungen des Verhaltens eines $p \in P_a$ ergeben, daß die Kachelmenge K_p ungeeignet festgelegt ist.

Wenn sich mit diesem Vorgehen die Aufteilung der Prozeßmenge $P = P_a + P_b$ ergibt, gilt zunächst für P_a das in Kapitel 6.3.1 Gesagte entsprechend; für die $p \in P_a$ sind jedoch die K_p geeignet gewählt, und die Seiten, die p für seine Berechnungen benötigt, werden initial mit Kacheln von K_p realisiert. Der weitere Unterschied besteht darin, daß die Verteilung der Kacheln häufiger und bei Bedarf, der durch Beobachtungen der $p \in P_a$ ermittelt wird, revidiert wird.
Zu den Maßnahmen der hier zu erklärenden Strategien gehört wesentlich, daß das Verhalten der Prozesse, denen Kacheln zugeordnet sind, stets beobachtet wird; aus den Beobachtungsergebnissen wird ermittelt, welche Kachelmengen und Seiten die Prozesse benötigen; zudem wird ermittelt, ob die jeweils zugeordnete Kachelmenge geeignet oder ungeeignet festgelegt ist. Ausschlaggebend für diese Strategien ist also die Wahl der Kenngrößen für das Verhalten der Prozesse, die beobachtet und ausgewertet werden sollen. Die hierfür geeigneten Kenngrößen sind die Arbeitsmengen der Prozesse.
Die **Arbeitsmengen** (Working Sets) der Prozesse sind die Mengen der Seiten, zu denen die Prozesse in Zeitintervallen mit festgelegten Längen in der Vergangenheit bei Ausführungen ihrer Berechnungen zugegriffen haben. Die Arbeitsmengen ergeben sich also aus Beobachtungen des Referenzverhaltens in der Vergangenheit; wie für die entsprechenden Strategien in Kapitel 6.3.1 wird erwartet, daß daraus auf das Verhalten in der Zukunft geschlossen werden kann.
Die Arbeitsmengen werden im folgenden für einen Prozeß p erklärt. Dazu sei $R_p = r_1 r_2 \ldots$ eine Referenzspur von p. Mit $m, n \in I\!N, n \geq m$, ist

$$W(R_p, m, n) \triangleq \{r_{n-m+1}, r_{n-m+2}, \ldots, r_n\}$$

die Arbeitsmenge von p zur Zeit n mit der Fenstergröße m bzgl. der Referenzspur R_p. Die Zeit n entspricht dem aktuellen Schritt der Berechnung von p. Die Fenstergröße m ist ein Parameter der Dimension Zeit, mit dem festgelegt wird, wie weit zurück das Referenzverhalten jeweils beobachtet werden soll. $W(R_p, m, n)$ ist unabhängig davon, ob die erfaßten Seiten AS– oder HS–realisiert sind; die Arbeitsmenge ist also unabhängig von der jeweils eingesetzten Strategie zur kachelbasierten AS–Verwaltung.
Die Arbeitsmengen der Prozesse liefern also Aussagen über das Referenzverhalten der Prozesse; sie liefern keine Strategien zur AS–Verwaltung; sie sind jedoch eine geeignete Basis zur Festlegung von Strategien. **Arbeitsmengen–Strategien** zur kachelbasierten AS–Verwaltung sind Strategien, für die gilt:

- die Arbeitsmengen der Prozesse werden stets beobachtet;
- die Realisierungsentscheidungen werden auf der Grundlage der jeweiligen Arbeitsmengen der Prozesse getroffen.

In diesem Rahmen sind die weiteren Entscheidungsregeln für Arbeitsmengen–Strategien festzulegen; dabei gilt folgende **Grundregel**:

- Den Prozessen, für die entschieden wird, daß sie ihre Berechnungen ausführen sollen, sollen die Seiten ihrer jeweiligen Arbeitsmengen AS–realisiert zur Verfügung stehen.

Für Arbeitsmengen–Strategien werden Regeln für Realisierungs–Entscheidungen auf zwei Ebenen festgelegt: Mit Entscheidungen im Großen wird die Prozeßmenge in Prozesse, die ihre Berechnungen ausführen sollen, und in Prozesse, die blockiert werden und warten sollen, aufgeteilt. Mit Entscheidungen im Kleinen wird für die Prozesse, die ihre Berechnungen ausführen sollen, der Einsatz der zugeordneten Kacheln bestimmt. Die entsprechenden Regeln werden im folgenden für ein System mit der Prozeßmenge P und der Kachelmenge K' erklärt.
Mit **Entscheidungen im Großen** wird die Prozeßmenge jeweils in $P = P_a + P_b$ aufgeteilt: Die $p \in P_a$ sollen ihre Berechnungen ausführen; die $p \in P_b$ werden blockiert und sollen auf die Weiterführung ihrer Berechnungen warten.
Die Prozesse $p \in P_a$ werden der Belegung von K' entsprechend nach ihren Arbeitsmengen ausgewählt und für die Ausführung ihrer Berechnungen vorbereitet:

- p wird eine Kachelmenge K_p gemäß der jeweiligen Arbeitsmenge W_p (bei gegebenen R_p, m und n) zugeordnet, und
- die Seiten gemäß W_p werden (durch Umspeichern) mit Kacheln aus K_p AS–realisiert.

Wenn W_p AS–realisiert ist, wird p zur Ausführung seiner Berechnungen entblockiert. Entsprechend wird mit allen Prozessen von P_a verfahren.
Die Prozesse $p \in P_b$, die blockiert sind und auf die Weiterführung ihrer Berechnungen warten sollen, werden aus AS verdrängt.
Bei festgelegter Aufteilung $P = P_a + P_b$ ergibt sich, daß die $p \in P_a$ nach den entsprechenden Vorbereitungen ihre Berechnungen mit den ihnen zugeordneten Kacheln ausführen können.
Die Aufteilung $P = P_a + P_b$ gilt bis zu ihrer Revision; sie wird revidiert, wenn sich mit dem Fortschreiten der Berechnungen der $p \in P_a$ wesentlich veränderte Rahmenbedingungen ergeben. Veränderte Rahmenbedingungen, die eine Revision der Aufteilung von P erfordern, sind die folgenden:

- Ein Prozeß $p \in P_a$, dem k_p Kacheln zugeordnet sind, benötigt einen wesentlichen Teil dieser Kacheln nicht weiter, und mit diesen von p nicht weiter benötigten Kacheln kann die Arbeitsmenge eines $q \in P_b$ AS–realisiert werden. In diesem Fall wird die Aufteilung von P in $P = P'_a + P'_b$ mit $P'_a = P_a + \{q\}$ und $P'_b = P_b - \{q\}$ verändert.
- Ein Prozeß $p \in P_a$, dem k_p Kacheln zugeordnet sind, benötigt zur AS–Realisierung einer Arbeitsmenge wesentlich mehr Kacheln, und die gemäß K' zur Verfügung stehenden Kacheln sind den Prozessen von P_a zugeteilt. In diesem Fall wird ein Prozeß $q \in P_a$ blockiert und aus AS verdrängt; die Aufteilung von P wird in $P = P'_a + P'_b$ mit $P'_a = P_a - \{q\}$ und $P'_b = P_b + \{q\}$ verändert.

In den jetzt erklärten Realisierungs–Entscheidungen, die im Großen getroffen werden, treten Kriterien auf, die noch zu präzisieren sind. Diese Kriterien betreffen das Verhalten der Prozesse, die ihre Berechnungen ausführen, und die Realisierungs–Entscheidungen für einzelne Seiten, die noch zu erklären sind.
Mit **Entscheidungen im Kleinen** wird für die Prozesse, die ihre Berechnungen ausführen sollen, festgelegt, welche Seiten mit den Kacheln, die den Prozessen zugeordnet sind, AS–realisiert werden. Diese Entscheidungen ergänzen die, welche im Großen getroffen werden. Sie werden im folgenden für einen Prozeß p erklärt, für den im Großen entschieden ist, daß er seine Berechnungen ausführen soll.
Nach dem bereits Erklärten gilt für einen Prozeß p folgendes: p ist die Kachelmenge K_p zugeordnet; mit dieser ist die Arbeitsmenge W_p AS–realisiert. In diesem Zustand wird p zur Ausführung seiner Berechnungen entblockiert.

Diese Gegebenheiten für die Weiterführung der Berechnungen des Prozesses p sind den in Kapitel 6.3.1 für die Anforderungsstrategien betrachteten ähnlich; das weitere Vorgehen wird entsprechend erklärt. Von der Referenzspur R_p sei das Präfix $r_1 \ldots r_n$ bereits abgearbeitet. Die Arbeitsmenge W_p ist also (bei festgelegter Fenstergröße) die zur Zeit n : $W_p(n)$. Dem Prozeß p sind k_0 Kacheln mit $|K_p| \leq k_0$ zugeordnet; mit ihnen ist der Anfangszustand für die Weiterführung der Berechnungen $Z_0 \triangleq W_p(n)$. Für die weiteren Berechnungsschritte von p mit der Referenzfolge $r_{n+1} \ldots$ gilt folgendes:

- Die Arbeitsmengen von p werden beobachtet; sie seien $W_p(n+l)$ für $l = 1, 2, \ldots$.
- Mit den Referenzen der Berechnungsschritte werden die von p benötigten Seiten AS–realisiert; damit ergibt sich die Zustandsspur $Z_0 Z_1 \ldots Z_l$ für $l = 1, 2, \ldots$.

Für die Entscheidungen, die im Kleinen getroffen werden, ist wesentlich, daß die Folge $W_p(n+l)$ der Arbeitsmengen das Referenzverhalten von p strategieunabhängig beschreibt. Die Zustände $Z_1 \ldots Z_l$ sind dagegen die Ergebnisse der Realisierungs–Entscheidungen, die bei der Abarbeitung der Referenzfolge getroffen werden. Dabei können analog zu Kapitel 6.3.1 Seitenfehler auftreten; dem entsprechend kann es notwendig sein, Seiten aus AS zu verdrängen und Seiten in AS zu laden. Für den Anfangszustand gilt voraussetzungsgemäß $Z_0 = W_p(n)$; für die Folgezustände gilt dagegen i.a. $Z_i \neq W_p(n+i)$. Vergleiche der jeweiligen Zustände mit den Arbeitsmengen liefern entsprechend die Kriterien für die Entscheidungen, die im Kleinen zu treffen sind; sie liefern auch die Präzisierungen der Kriterien für die Entscheidungen im Großen, die noch nachzutragen sind.
Für die Entscheidungen, die im Kleinen zu treffen sind, sind modifizierte Einzelseiten–**Anforderungsstrategien** geeignet; das wird im folgenden für den Zustand Z_l, die Arbeitsmenge $W_p(n+l)$ und die Referenz r_{n+l+1} des Prozesses p unter der Annahme, daß diese Referenz zu einem Seitenfehler führt, erklärt. Die Anzahl der Kacheln, die p zugeordnet sind, sei k_0 und es gelte $|Z_l| = k_0$.
Wegen $r_{n+l+1} \notin Z_l$ muß die Seite in AS geladen werden; dafür wird eine freie Kachel benötigt. Für die Auswahl dieser Kachel gibt es mehrere Möglichkeiten; dabei kommt die oben angegebene Grundregel zur Anwendung. Für die erforderliche Realisierungs–Entscheidung werden im folgenden drei Fälle betrachtet.
Der **erste Fall** liegt vor, wenn $W_p(n+l) \subset Z_l$ gilt. Dann wird eine Seite $r \in Z_l - W_p(n+l)$ ausgewählt; diese Seite wird aus AS verdrängt und die Seite r_{n+l+1} wird in die entsprechende Kachel geladen. Die Seite r_{n+l+1} ist damit AS–realisiert und die p–Berechnung kann fortgesetzt werden.
Für diesen ersten Fall ist charakteristisch, daß die Zustandsüberführung $Z_l \rightarrow Z_{l+1}$, die mit dem auftretenden Seitenfehler notwendig wird, mit der Grundregel verträglich erfolgen kann. Für die weiteren Fälle gilt $Z_l \subseteq W_p(n+l)$. Nach der Grundregel sollen Seiten der Arbeitsmengen nicht aus AS verdrängt werden; daraus ergeben sich die beiden folgenden Verfahrensfälle.
Der **zweite Fall** liegt vor, wenn $Z_l \subseteq W_p(n+l)$ gilt und die Anzahl der p zugeordneten Kacheln von k_0 auf k_1, $k_0 < k_1$, vergrößert werden kann. Diese Anforderung einer zusätzlichen Kachel (oder einiger zusätzlicher Kacheln) für p ist eine Modifikation der in Kapitel 6.3.1 erklärten Anforderungsstrategien. Wenn sie erfüllt werden kann und erfüllt wird, ist die notwendige Zustandsüberführung $Z_l \rightarrow Z_{l+1}$ mit der Grundregel verträglich durchführbar.
Der **dritte Fall** liegt vor, wenn $Z_l \subseteq W_p(n+l)$ gilt und die Anzahl der p zugeordneten Kacheln nicht vergrößert werden kann. Dann muß gegen die Grundregel eine Seite $r \in Z_l$ aus AS verdrängt werden, und die Seite r_{n+l+1} wird mit der entsprechenden Kachel AS–realisiert. Die Seite, die verdrängt wird, wird nach einem der in Kapitel 6.3.1 angegebenen Kriterien (z.B. LRU) ausgewählt.

Die jetzt erklärten Entscheidungen im Kleinen werden im Rahmen einer festliegenden Aufteilung der Prozeßmenge $P = P_a + P_b$ getroffen; mit ihnen soll erreicht werden, daß die Berechnungen der Prozesse $p \in P_a$ mit kleiner Seitenfehlerrate effizient ausgeführt werden. Mit den Verhaltensbeobachtungen, die für die Entscheidungen im Kleinen erfolgen, kann auch beurteilt werden, ob die Aufteilung $P = P_a + P_b$ zweckmäßig ist oder verändert werden soll.
Für die Entscheidungen, die im Kleinen getroffen werden, werden jeweils die Arbeitsmengen und die AS–Belegungszustände der Prozesse $p \in P_a$, die ihre Berechnungen ausführen sollen, verglichen. Dem entsprechend ergeben sich für diese Prozesse Aussagen über die Eignung der zugeordneten Kachelmengen für die AS–Realisierungen der Arbeitsmengen, die mit der Grundregel gefordert sind. Diese Vergleiche liefern die Kriterien für die Realisierungs–Entscheidungen im Kleinen; sie liefern auch die Kriterien für die Entscheidungen, die im Großen zu treffen sind. Nach den früheren Erklärungen wird die Aufteilung der Prozeßmenge P dann verändert, wenn wesentliche Abweichungen beobachtet werden; dabei wird wesentlich quantifiziert, indem entsprechende **Schwellenwerte** festgelegt werden. Die Vorgehensweise wird im folgenden für einen Prozeß $p \in P_a$, dem k Kacheln zugeordnet sind, unter Verwendung der oben eingeführten Bezeichnungen für Arbeitsmengen und Zustände erklärt.
Veränderungen der Aufteilung der Prozeßmenge P können dann erforderlich sein, wenn der Prozeß $p \in P_a$ die ihm zugeordneten Kacheln nicht benötigt. Das ist die Situation im oben angegebenen ersten Fall, wenn ein Schwellenwert überschritten wird. Diese **Freigabeschwelle** sei für den Prozeß p mit $u \in \mathbb{N}$ festgelegt. Damit ergibt sich folgendes:

- Wenn für p der Fall $W_p(n+l) \subset Z_l$ mit $|Z_l| = k$ vorliegt und $k - |W_p(n+l)| \geq u$ gilt, dann werden Kacheln, die p zugeordnet sind, freigegeben.

Veränderungen der Aufteilung der Prozeßmenge P können weiter dann erforderlich sein, wenn der Prozeß $p \in P_a$ zusätzliche zugeordnete Kacheln benötigt. Das ist die Situation im oben angegebenen zweiten und dritten Fall, wenn ein Schwellenwert überschritten wird. Diese **Anforderungsschwelle** sei für den Prozeß p mit $o \in \mathbb{N}$ festgelegt. Damit ergibt sich folgendes:

- Wenn für p der Fall $Z_l \subset W_p(n+l)$ mit $|Z_l| = k$ vorliegt und $|W_p(n+l)| - k \geq o$ gilt, dann werden für p zusätzliche Kacheln angefordert.

Die beiden Regeln legen Kriterien für das Zusammenwirken zwischen den Entscheidungen im Kleinen und den Entscheidungen im Großen bei Arbeitsmengen–Strategien zur kachelbasierten AS–Verwaltung fest. Mit ihnen kann erreicht werden, daß die Realisierungs–Entscheidungen prozeßglobal, also unter Berücksichtigung aller jeweils koexistierenden Prozesse, und anpassungsfähig, also unter Berücksichtigung der jeweiligen Anforderungen der Prozesse, getroffen werden. Abbildung 6.7 veranschaulicht das Zusammenwirken der zu treffenden Entscheidungen für anpassungsfähige Arbeitsmengen–Strategien.
Die jetzt erklärten Arbeitsmengen–Strategien ermöglichen effiziente, kachelbasierte AS–Verwaltung durch Anpassung an die jeweiligen Anforderungen der Prozesse. Die Anpassungen setzen voraus, daß das Referenzverhalten der Prozesse stets beobachtet und ausgewertet wird. Für einen Prozeß p mit der Referenzspur R_p werden für die Realisierungs–Entscheidungen einerseits die jeweilige Arbeitsmenge $W(R_p, m, n)$ zur Zeit n bei der Fenstergröße m und andererseits für den jeweiligen Belegungszustand Z_n zur Zeit n Aussagen über die zeitliche Verteilung der Zugriffe zu den Seiten aus Z_n, z.B. die Rückwärtsabstände, benötigt. Diese Beobachtungen des Zugriffsverhaltens der Prozesse über der Zeit sind aufwendig. Mit

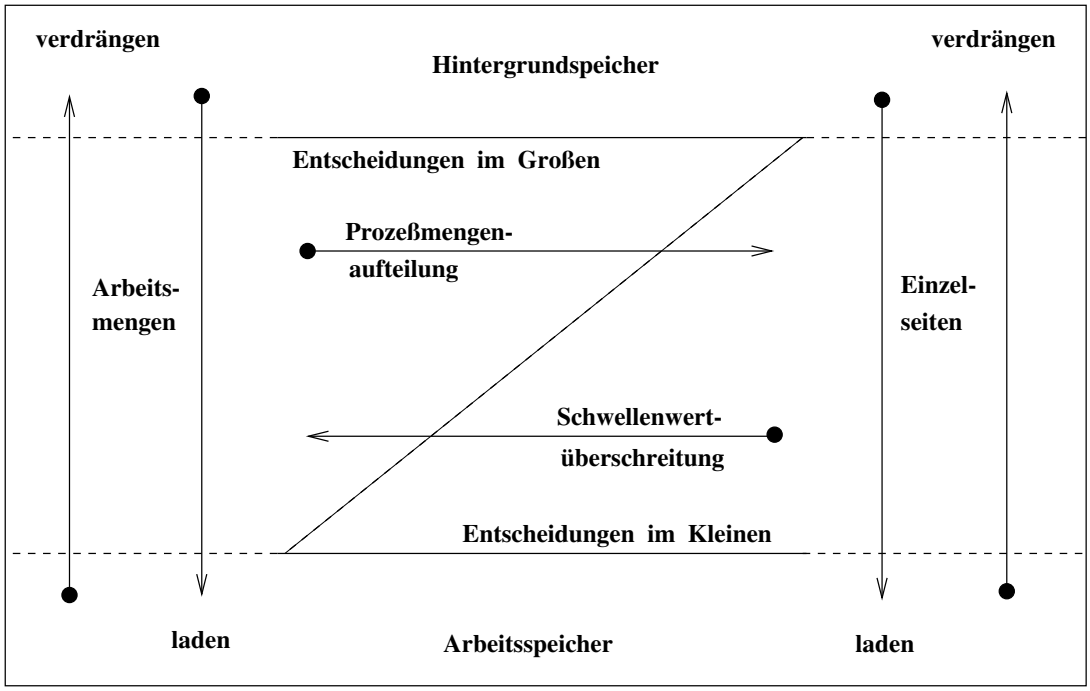


Abbildung 6.7: Anpassungsfähige Arbeitsmengen–Strategien

geringerem Aufwand, aber auch mit geringerem Informationsgewinn können die Seitenfehlerraten der Prozesse beobachtet werden. Sie sind, im Gegensatz zu den Arbeitsmengen, nicht strategieunabhängig. Man kann mit ihnen jedoch **Seitenfehlerraten–Strategien** (PFF–Strategien mit PFF für Page Fault Frequency) konstruieren, die ebenfalls mit Entscheidungen im Großen und im Kleinen sowie mit Schwellenwerten anpassungsfähige, kachelbasierte AS–Verwaltung ermöglichen.