KfK 4138
September 1986

Modellierungs- und Simulationsinstrumentarium für fehlertolerante verteilte Systeme angewendet auf verteilte Datenbankverwaltungssysteme

M. Leszak
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 4138

Modellierungs- und Simulationsinstrumentarium für fehlertolerante verteilte Systeme angewendet auf verteilte Datenbankverwaltungssysteme

Marek Leszak

Als Dissertation genehmigt vom Fachbereich 20 (Informatik)
der Technischen Universität Berlin

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

A b s t r a c t

Leszak, Marek:
Modellierungs- und Simulationsinstrumentarium für fehlertolerante verteilte Systeme angewendet auf verteilte Datenbankverwaltungssysteme

Vorliegende Arbeit stellt eine einheitliche Methodik zur Spezifikation und Leistungsvorhersage fehlertoleranter verteilter Systeme vor. Hauptziel dabei ist es, den Entwurf robuster höherer Kommunikationsprotokolle durch begleitende Leistungsuntersuchungen zu unterstützen.

Es wird eine auf höheren stochastischen Petrinetzen basierende Spezifikationssprache vorgeschlagen, die generell zur Beschreibung komplexer Hardware-/Software-Systeme geeignet ist. Die Sprache unterstützt die strukturierte Beschreibung der Systemlast, des funktionalen und stochastischen Verhaltens von Software-Prozessen und -Protokollen, sowie des zeitlichen und fehlerhaften Verhaltens von Hardware-Ressourcen. Spezifikation und Simulation von mit dieser Netzsprache erstellten Modellen werden durch ein Software-Werkzeug unterstützt. Dieses ermittelt die wesentliche Leistungsgrößen und enthält eine komfortable interaktive Ablaufüberwachung mit dem Ziel der Modellüberprüfung. Erweiterungen in Richtung auf analytische Untersuchung prädikativer und quantitativer, stationärer Systemeigenschaften werden vorgeschlagen.

Es wird eine neuartige, effiziente Methode zur integrierten Untersuchung von Leistung und Verfügbarkeit fehlertolerierender Rechensysteme präsentiert, die einen Leistungsvergleich von Entwurfsalternativen mit Einbeziehung von Leistungseinbußen nach Hardware-Teilausfällen in die Gesamtleistung gestattet.

Die entwickelten Methoden und das darauf basierende Werkzeug werden an einem komplexen Anwendungsbeispiel erprobt. Es werden drei Protokolle zur Verarbeitung verteilter Transaktionen auf redundant verteilten Datenbeständen spezifiziert, die Datenkonsistenz sogar nach Netzpartitionierung bewahren und einen robusten Mehrpartner-Kommunikationsdienst benutzen. Die Protokolle werden in unterschiedlichen Anwendungs- und Rechnernetz-Umgebungen durch Simulationsexperimente bewertet. Die Ergebnisse gestatten differenzierte Aussagen über Kosten und Nutzen redundanter verteilter Datenhaltung.

A b s t r a c t

Leszak, Marek:
A Modeling and Simulation Tool for Fault-tolerant Distributed Systems applied to Distributed Database Management Systems

This thesis presents a uniform methodology for specification and performance prediction of fault-tolerant distributed systems. The main goal of our approach is to provide a tool supporting the design and evaluation of robust, higher layer communication protocols.

A specification language based on high-level stochastic Petri Nets is proposed. This language is applicable to the description of complex hardware/software systems in general. It supports a structured system design, containing description and processing of the workload, functional and stochastic behavior of software processes and protocols, and of hardware resources including their erroneous behaviour. Specification of models using this net language, together with model execution by simulation, is supported by a software tool. The tool allows for automatic evaluation of essential performance measures. It features a highly interactive experimentation control facility for model test and inspection. Enhancements towards analytical investigation of predicative properties and steady-state performance are also discussed.

A new efficient method for integrated evaluation of performance and availability of fault-tolerant computer systems is presented. Application of the method yields over-all performance comparisons of design alternatives by considering degraded performance after resource crashes.

The proposed methods and tools are applied to the design of three complete transaction processing protocols for partially replicated databases. The protocols are using a robust multicast interprocess communication service and preserve data consistency even after network partitions. Several performance aspects of the protocols are evaluated in different application and computer network environments by simulation experiments. Various conclusions about cost and benefit of managing redundant distributed data can be inferred from the results presented.

INHALTSVERZEICHNIS

1 EINFÜHRUNG

Verteilte Datenbankverwaltungssysteme (VDBS) stellen ein in den letzten zehn Jahren entwickelte Disziplin dar, die in dem Bestreben entstanden ist, Datenbank- und Rechnernetztechnologie zu integrieren. Eine verteilte Datenbank besteht aus einer Menge logisch zusammenhängender, über mehrere Knoten eines Rechnernetzes verteilter Datenbestände, wobei der Zugriff zu diesen Daten vom VDBS koordiniert und auf mehreren Rechnerknoten durch einheitliche Datenmanipulations- und Datendefinitions-Schnittstellen unterstützt wird. Dabei kann es sich sowohl um ein lokales Rechnernetz (z.B. innerhalb einer Firmenfiliale) oder um ein Rechnernetz mit geographisch verteilter Struktur (z.B. mehrere Filialen in verschiedenen Städten) handeln. Ursprünglich wurden VDBS top-down aus homogenen, auf geographische Verteilung ausgelegten Einzelkomponenten entworfen und realisiert. Heute geht man aber zunehmend dazu über, schon vorhandene zentrale, oft heterogene Datenbanksysteme mittels eines bottom-up konzipierten VDBS unter Ausnutzung lokaler Rechnernetze zu koordinieren.
Als Ergebnis einer sehr intensiven Forschung in verschiedenen VDBS-Teilgebieten gibt es heute etliche experimentelle Prototypen, aber wenige kommerziell verfügbare general-purpose VDBS. Ein wesentlicher Grund hierfür liegt in noch ungelösten Entwurfsproblemen: Es herrscht ein großer Mangel an formalen, ausgereiften Methoden zur Konstruktion, Analyse und Bewertung von VDBS mit dem Ziel, gegebene Anwendungscharakteristika mit bestimmten hohen Leistungs- und Zuverlässigkeits-Anforderungen zu erfüllen. Andererseits sind die potentiellen Vorteile des Einsatzes eines VDBS so vielversprechend, daß Wege zur Lösung dieser Probleme auch zukünftig untersucht werden sollten.

1.1 Verteilte Datenbanksysteme: Motivation, Probleme, Trends

Welches Anwendungsprofil erfordert dezentrale Datenhaltung mit hohen Leistungs- und Verfügbarkeitsanforderungen? Eine typische Charakteristik umfaßt

- komplexe, logisch korrelierte Daten: Erfordernis eines Datenbanksystems

- Mehrbenutzerumgebung mit Dialoganwendungen: Forderung nach guten Antwortzeiten für Datenbankzugriffe (Transaktionen)
- Mehrbenutzerumgebung mit Realzeitanwendungen: Forderung nach sehr guten, vorhersagbaren Antwortzeiten
- Organisation mit mehreren 'gleichwertigen' Filialen: Forderung nach verteilter Datenhaltung
- Produktionsanwendungen oder sicherheitskritische Anwendungen: Forderung nach hoher Fehlertoleranz (Zugriff zu Datenbeständen ist trotz Systemteilausfall zu gewährleisten, Wiederanlaufzeiten sind zu minimieren, Daten müssen konsistent gehalten werden trotz Speicherfehlern) erfordert zuverlässige Datenhaltung in einem fehlertolerant ausgelegten verteilten System. Bei solchen Anwendungen können Beeinträchtigungen der Verfügbarkeit bzw. Zuverlässigkeit Produktionsausfälle mit hohen Folgekosten oder sogar einer Sicherheitsgefährdung zur Folge haben /SCHN84b/.

Beispiele von Anwendungsklassen mit solchen Anforderungen sind z.B.

- Verteilte Realzeit-Datenhaltungssysteme in der Fernwirktechnik zur Energieerzeugung und -verteilung /BDL83/
- Verteilte Lagerhaltungs-Informationssysteme bei Produktionsbetrieben mit mehreren Fertigungsstätten /GODB81/
- Rechnergestützte fehlertolerante Telefonvermittlungssysteme (electronic switching system 5ESS) /NG83/
- Krankenhaus-Informationssysteme

Um verteilte Datenhaltungssysteme mit solch hohen Anforderungen zu realisieren, stellt sich das bisher stark vernachlässigte Problem der **quantitativen Bewertung und Prognose von Leistung und Fehlertoleranz** von VDBS. Der Leistungsbegriff eines komplexen Mehrschichten-Rechensystems umfaßt unterschiedliche Anforderungen an jeweils eine Schicht. Die wichtigste Schicht bietet Dienste für Endbenutzer. Ein VDBS bietet insbesondere einen Dienst zur Realisierung von Datenbankanfragen als **Transaktionen**. Die wichtigsten VDBS-Leistungsmaße aus Benutzersicht sind daher Transaktions-Antwortzeit und -Durchsatz. Diese Maße wurden bisher unter operationalem 'Normalverhalten' eines VDBS untersucht; Fehlerraten wurden nicht einbezogen. Ein sinnvolles Verfügbarkeitsmaß ist die Transaktions-Erfolgsrate, d.h. die Terminierung einer

Transaktion unabhängig von Systemausfällen und das erfolgreiche Einbringen von Datenbasisänderungen. Auch hier ist eine Berücksichtigung von Systemteilausfällen notwendig. Dabei interessieren insbesondere die Einflüsse unterschiedlicher Recovery- (Fehlerbehebungs-), Wiederanlaufs-, und Concurrency-Control- Verfahren auf verteilten redundanten Datenbeständen.
Die erforderliche integrierte Bewertung von Leistung und Verfügbarkeit ist für fehlertolerante Systeme sehr notwendig, wird aber erschwert u.a. durch sich wechselseitig widersprechende Anforderungen.

> **Beispiel:** Minimale Transaktions-Antwortzeit, maximaler -Durchsatz und maximale -Erfolgsquote widersprechen sich wechselseitig und sind in einem System nicht realisierbar: Erhöhung der Erfolgsquote ist erreichbar durch redundante Datenspeicherung mit geeigneten Rekonfigurationsalgorithmen zur Behebung von Systemteilausfällen mit dem Ziel, mindestens eine Datenkopie verfügbar zu erhalten. Um Konsistenz zu sichern, müssen aber Datenänderungen auf allen redundanten Kopien ausgeführt werden, was aber stets Performance-Overhead im Normalfall (höhere Anwortzeiten und geringere Durchsätze) zur Folge hat.
> Auch Antwortzeit und Durchsatz sind nicht gleichzeitig optimierbar, wie in /KOST82/ gezeigt wurde.

Trotz der bestehenden Bewertungsprobleme bringt die Einführung eines VDBS potentiell große Vorteile:

(1) **Leistungssteigerung:** Durch höhere Nebenläufigkeit bei der Bearbeitung von Benutzeranfragen (Transaktionen) an die Datenbank auf mehreren Rechnern, zusammen mit hoher Datenlokalität, lassen sich Antwortzeiten und Durchsätze von Transaktionen steigern. Hohe Lokalität bedeutet eine Zuordnung der Daten zu solchen Rechnern, die von Datenbankbenutzern mit den häufigsten Anfragen zum lokalen Datenzugriff benötigt werden, um Kommunikationskosten klein zu halten. Lokalität ist ein wesentlicher Gesichtspunkt bei der Einführung eines VDBS auch deshalb, weil sie der Philosophie vieler Unternehmen nach autonomer Verantwortlichkeit für lokal gehaltene Datenbestände entspricht. Es wird also eine

Datenverteilung entsprechend der Organisationsstruktur eines Unternehmens angestrebt.

(2) **Erhöhte Fehlertoleranz**: Fehlertoleranz im Datenbankbereich beinhaltet die Verhinderung ungewünschter Datenbasis-Zustände (Bewahrung der logischen Datenintegrität, Zuverlässigkeit), als auch die Bewahrung der Zugriffsmöglichkeiten zu den Datenbeständen (Verfügbarkeit) trotz Auftreten von VDBS-Teilausfällen. Grundlage aller Maßnahmen zur Erzielung von Fehlertoleranz ist die redundante Auslegung von Datenbeständen, VDBS-Funktionen und Hardwarekomponenten wie Rechner oder Massenspeicher, verbunden mit einer Dezentralisierung aller VDBS-Funktionen einschließlich einer Recovery/Restart-Komponente, sodaß ein einzelner Ausfall z.B. eines Rechners nur lokale Auswirkungen auf die Zugänglichkeit der auf ihm gespeicherten Daten hat. Hingegen führt der Rechnerausfall in einem zentralisierten Datenbank-Verwaltungssystem (DBVS) zur totalen Unverfügbarkeit bis zu dessen Wiederanlauf.

(3) **Inkrementelle Erweiterbarkeit**: In einer verteilten Systemumgebung ist die Erweiterung der Datenbasis und der Benutzerumgebung z.B. durch Anschluß weiterer Rechnerknoten einfacher zu erreichen als der Ausbau eines zentralisierten Datenbanksystems. Wesentlich dabei ist, daß das Gesamtsystem nicht längere Zeit abgeschaltet werden muß, um die Erweiterungen zu installieren.

Dem Versuch, diese offensichtlichen Vorzüge verteilter gegenüber zentralisierten DBVS zu realisieren, stehen aber eine Reihe nicht vollständig beherrschter Probleme gegenüber:

(4) **Verteilungs-Komplexität**: Gegenüber zentralen DBVS ergeben sich bedingt durch die Verteilung von Daten und Funktionen neue Probleme. Bei der Verteilung unterscheidet man zwischen Zentralisierung, Partitionierung (disjunkte Teilmengen werden verschiedenen Rechnern zugeordnet), Teil- und Vollreplikation (es gibt Kopien einiger bzw. aller Komponenten). In Abb. 1 sind die sinnvollen und in verschiedenen Systemen auch realisierten Verteilungsalternativen wesentlicher VDBS-Subsysteme wie etwa verteilte Datenbasis, Hardware-Konfiguration, Verteiltes

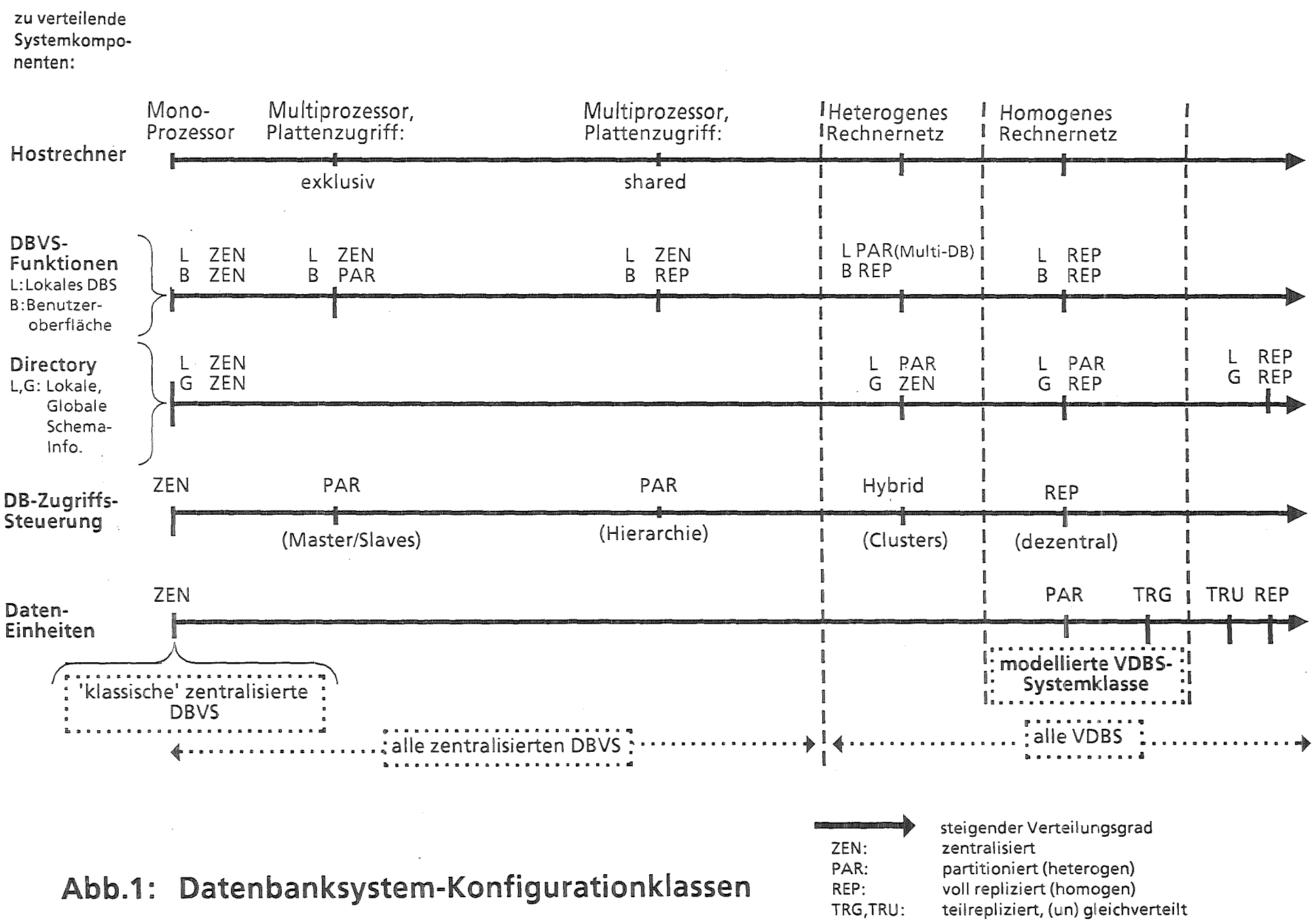


Abb.1: Datenbanksystem-Konfigurationklassen

Directory klassifiziert. Zu beachten ist, daß

- Eine Zentralisierung aller Komponenten eines VDBS-Subsystems nicht sinnvoll ist, da sonst Leistung und Verfügbarkeit des Gesamtsystems wesentlich von Charakteristika dieses kritischen Subsystems (eines 'single failure point') abhängen würden;
- Jede Partitionierung und erst recht Teilreplikation zu exponentiell vielen Verteilungsalternativen pro Subsystem führt;
- Die meisten realisierten VDBS nicht zuletzt aus diesem Grund redundante Datenhaltung nicht unterstützen, d.h. die Verantwortung für Konsistenz und Aktualität physisch mehrfach vorhandener, logisch gleichartiger Dateneinheiten wird dem Endbenutzer aufgebürdet.

Die exponentielle Zahl möglicher Verteilungsalternativen ist die Ursache für die NP-Vollständigkeit /GAJO78/ einiger wesentlicher VDBS-Entwurfsprobleme, z.B. dem Finden der optimalen Datenverteilung oder der optimalen Auswertung von Datenbankanfragen. Diese Probleme werden durch die Komplexität verteilter, kooperierender Programme (**Protokolle**) noch verschärft, sodaß bisher lediglich heuristische Lösungsansätze gefunden werden konnten, die höchstens suboptimale Lösungen liefern.

(5) **Software-Komplexität:** Im Bereich komplexer zentraler Informationssysteme treten bereits große Probleme bei der Erstellung korrekter, wartbarer Software auf, die zu einer allgemeinen Software-Krise geführt haben /SCHN84a/. Im VDBS-Bereich treten noch verschärfende Faktoren hinzu. Es ist wesentlich schwieriger, Protokolle zu entwerfen, intuitiv zu verstehen, und formal zu verifizieren, als sequentielle Programme. Neben der Verteilung trägt die erforderliche Erkennung und Behebung von Störungen des VDBS-Betriebes, z.B. durch Rechnerausfall, wesentlicher zur Erhöhung der Software-Komplexität bei. Hauptursache dafür ist, daß Störungen zu beliebigen Zeitpunkten auftreten können, was die Anzahl der möglichen Zustände konsistenzerhaltender Protokolle stark erhöht.

(6) **Fehlende quantitative Bewertungen:** Trotz hunderter entworfener und z.T. realisierter VDBS-Protokolle zur Kontrolle konkurrierender Datenbankzugriffe (Concurrency Control) und zur Fehlerbehebung (Recovery) besteht weitgehend Unklarheit über die relative Leistungs- und Verfügbarkeitsbewertung dieser Protokolle bei verschiedenen Systemparametern wie z.B. Verteilungsgrad der Datenbestände. Wie in /SEVC83/ konstatiert wird, gehen die meisten quantitativen Untersuchungen in diesem Bereich (s. etwa /CHEN81, DANT80, DECA82, GARC79, GATS84, OZSU83, RIES79/) von unterschiedlichen Annahmen und Anforderungen aus, für die diese Protokolle entworfen wurden. Vergleiche sind daher wenig sinnvoll. Zudem sind die Annahmen so vereinfachend, daß die Relevanz der Ergebnisse für reale Systeme fragwürdig ist:

- Vollredundante Datenverteilung auf alle Rechnerknoten;
- Betrachtung nur von Änderungstransaktionen;
- Beschränkung der Nebenläufigkeiten: keine parallelen Plattenzugriffe, pro Knoten höchstens eine aktive Transaktion;
- Ausklammerung typischer Fehlerfälle wie Knoten- und Leitungsausfall;
- Sehr vergröberte Sicht auf Rechnernetzstruktur und Kommunikationssoftware.

Die genannten Arbeiten verwenden sowohl analytische als auch simulative Untersuchungsmethoden. Die dabei verwendeten Modellannahmen zeigen das beschränkte Einsatzspektrum analytischer Methoden, da dabei sehr restriktive Annahmen erforderlich waren. Heuristische und analytische Methoden lassen für VDBS-Protokolle als Folge der Software-Komplexität und widersprüchlicher Zielanforderungen (trade-offs) nur beschränkte Aussagen zu, sodaß rechnergestützte Simulation als einzig gangbarer Weg erscheint.

Man kann also das Fehlen einer **Konstruktionslehre** feststellen, um ausgehend von einer Anforderungsspezifikation ein "günstiges" oder bezüglich den Anforderungen sogar optimales VDBS zu entwerfen, zu dimensionieren und zu realisieren. Umso wichtiger ist es, **rechnergestützte Werkzeuge** für solche Systeme zu entwickeln. Ein geeignetes Werkzeug muß nicht nur Teillösungen der genannten Probleme

liefern, sondern auch Anforderungen zukünftiger VDBS-Entwicklungen berücksichtigen.
Im wesentlichen lassen sich heute folgende Trends erkennen:

- **Integration heterogener Datenbanken:** Die meisten existierenden VDBS gehen von funktional homogenen Hardware- und Softwarekomponenten aus, da deren Realisierung wesentlich einfacher ist. Als Beispiele seien die Systeme DDM /BERN83a/, ENCOMPASS /BORR81/, JASMIN /WILA84/, POREL /WANE84/, R* /LIND83/, SDD-1 /ROTH80/ und VDN/REFLEX /GMSS81/ genannt. Insbesondere wird das Relationenmodell zur Bildung externer und konzeptioneller Schemata eingesetzt. Zukünftig wird eine zunehmende Integration bereits vorhandener, heterogener Datenbanken in 'Multidatenbanken' erwartet; Prototypen solcher Systeme sind SIRIUS /LITW82/ und MULTIBASE /LARO82/. Einen Überblick dazu vermittelt /GLKE84/. Bei den neuartigen Datenbanksystemen für z.B. Statistik-, Dokumentenretrieval-, Ingenieurentwurfs-, Büro- und Telematik-Anwendungen kann eine Integration mit "klassischen" DBVS erwartet werden, die nicht ohne Auswirkungen auf den VDBS-Bereich bleiben kann.

- **Höhere Endbenutzer-Schnittstellen:** Im Bereich zentraler DBVS ist ein Trend in Richtung höherer Benutzerschnittstellen zu beobachten, charakterisiert durch die Entwicklung graphischer und sogar natürlichsprachlicher Anfragesprachen /SCHN84a/.

- **Innovative Kommunikationstechnologie:** Speziell das Gebiet der lokalen Rechnernetze (LAN) hat in den letzten Jahren eine stürmische Entwicklung erlebt; ihr kommerzielles Einsatzspektrum wächst ständig. Hervorzuheben sind hierbei Breitbandtechnik und Glasfasertechnologie mit wesentlich gesteigertem Leistungsspektrum. Ferner ist durch die Enwicklung kostengünstiger Personal Computer und deren Vernetzung in 'Inhouse'-Bürokommunikationssystemen eine immer stärkere **Dezentralisierung** der Rechnerdienste zu beobachten, die erhöhte Anforderungen an operationale Integrität, Verfügbarkeit und effizientem Zugriff auf verteilte Datenbestände stellt /SCHN84c/. Demgegenüber wurden die meisten VDBS mit dem Ziel entworfen, die

Systeme in geographisch verteilten Netzen einzusetzen. Die Verwendung von LANs kann aber die Komplexität der VDBS-Protokolle wesentlich reduzieren, da bei gewissen Techniken wie Broadcast-LANs deterministische Kommunikationszeiten garantiert werden können. Ein modernes VDBS mit dieser Zielsetzung ist LAMBDA /CHNG84/.

Die Standardisierung der unteren, anwendungsunabhängigen Kommunikationsschichten ist weit fortgeschritten /ISO84a/. Eine entsprechenden Standardisierung höherer Schichten, die z.B. auf VDB-Anwendungen zugeschnitten sind, wird in den Normungsorganisationen ISO und ECMA angestrebt; siehe /ISO84b, ECMA85/. Ziel dabei ist die Bereitstellung von anwendungsnahen Diensten in offenen Kommunikationssystemen, insbesondere von

-- verteilten Transaktionsdiensten
-- Kommunikations- und Datensicherungsdiensten zur Realisierung verteilter Transaktionen /LEBR82, MOPO82/.

- **Fehlertolerierende hochverfügbare Systeme**: Auf dem Gebiet fehlertoleranter DV-Systeme zeichnet sich ein Trend zu hochverfügbaren, kostengünstigen Mehrrechnersystemen für den kommerziellen als auch für den Prozeßrechner-Sektor ab. Einige dieser Systeme sind bereits mit integriertem DBVS ausgestattet; s. /KIM84/.

  Heutige VDBS erkennen und "tolerieren" bereits folgende Störungen des Systembetriebs:
  - Rechner- mit Hauptspeicherausfall;
  - Massenspeicher-Fehler;
  - Fehlerhafte Datenbasisänderungen als Folge von Synchronisationsfehler oder Transaktionsabbruch.

  "Fehlertoleranz" in diesem Kontext bedeutet eine Datenkonsistenzgarantie bei beliebig vielen der genannten Fehler, aber nicht eine Aussage bezüglich der Systemverfügbarkeit. Weitergehende Systeme sind (s.o.)
  - LAMBDA, JASMIN: Daten bleiben trotz beliebigen Rechnernetz-Partitionierungen (nach Leitungsausfall) konsistent;
  - ENCOMPASS: Ausfall einzelner VDBS-Prozesse führt zu einer eingeschränkten Verfügbarkeit, aber nicht zum Totalausfall.

Die Erkennung von HW/SW-Entwurfs- und Realisierungsfehlern und sogar deren Tolerierung (Maskierung der Fehlerauswirkungen vor dem Benutzer, außer Performance-Einbußen) erfordert wesentlich größeren Aufwand (Redundanz, Kosten), sodaß heutige VDBS solche Beeinträchtigungen noch nicht behandeln.

Diese zukünftigen Entwicklungsrichtungen werden die Anforderungen an VDBS-Entwicklungswerkzeuge sicherlich noch steigern. Während im Bereich zentraler Datenbanksysteme bereits eine Reihe integrierter Werkzeuge zur Unterstützung von Datenbankentwurf, Leistungsmessung, -prognose, -tuning sowie Konfigurationsplanung existieren und auch zunehmend kommerziell genutzt werden (s. z.B. /EICK84, MUEL82/ und Realisierungen in den Systemen IMS, ADABAS), wird die Entwicklung entsprechender VDBS-Werkzeuge erst für Systeme der nächsten Generation erwartet /CHAN83/, trotz vielversprechender Ansätze wie etwa in /BIFL79/.

1.2 Modellierungsaspekte von Rechensystemen

Die quantitative Bewertung der Leistung, Zuverlässigkeit und Verfügbarkeit von Rechensystemen spielt eine zunehmend große Rolle. Die Bewertung durch 'Daumenregeln' führt wegen der Komplexität der Systeme zu fragwürdigen Ergebnissen. Mit der Systemgröße stark steigende Softwareentwicklungs- und Wartungskosten erfordern eine Prognose des Systemverhaltens in immer früheren Systementwicklungsphasen wie Anforderungsspezifikations- und Entwurfsphase, um Systemänderungskosten zu minimieren. Leistungsanforderungen eines Rechensystem sind oft gegenseitig widersprüchlich und durch eine einzige Systemkonfiguration nicht erfüllbar. Z.B. kann man ein Datenbanksystem nicht gleichzeitig für gute Antwortzeiten und für optimalen Durchsatz an Transaktionen auslegen; andere sich widersprechende Ziele sind z.B. Zuverlässigkeit - Antwortzeit. Es wird also immer für eine Anwendungscharakteristik optimierte **maßgeschneiderte Systemlösungen** geben, sodaß quantitative Bewertungsmethoden stets wichtig sein werden.

Bewertungsmethoden für Rechensysteme lassen sich grob in implementierungsnahe Methoden und Modellierungsmethoden einordnen. Zur

ersten Gruppe gehören Meß- und Testmethoden am realen System. Diese Methoden sind oft aus Kostengründen nicht praktikabel, z.B. wechselnde Last oder alternative Softwarekomponenten sind nur durch zeitweises Abschalten des Systems untersuchbar. In der Entwurfsphase scheidet diese Vorgehensweise a priori aus.

Daher werden zur quantitativen Untersuchung von Rechensystemen Modelle und Methoden zur Modellauswertung und Gültigkeitsüberprüfung benötigt. Ein **Modell** ist eine abstrakte Beschreibung eines Systems zur Untersuchung seines voraussichtlichen Verhaltens unter Vernachlässigung von (Abstraktion von) weniger relevanten und Betonung der relevantesten Systemcharakteristika. Bei der Bewertung von Rechensystemen unterscheidet man prinzipiell zwischen **Analytischen Modellen** und **Simulationsmodellen**.

Analytische Modellierung besteht in der Aufstellung mathematischer Formeln, mit denen formale Beziehungen zwischen Systemparametern als geschlossener Ausdruck, numerisch oder approximativ hergeleitet werden können. Die bedeutendsten analytischen Bewertungsmethoden für Rechensysteme sind die Warteschlangentheorie und die Theorie der Markovprozesse zur Leistungs- und Zuverlässigkeitsanalyse als Disziplinen der Angewandten Statistik.

Markovmodelle sind spezielle stochastische Zustandsautomaten, die analytisch behandelbar sind. Leistungsaspekte von Rechensystemen können charakterisiert werden durch eine Menge von Hardware- und Software-Betriebsmitteln und eine Menge von Aufträgen und Prozessen, die um Betriebsmittelzugriff konkurrieren. Bei mehreren konkurrierenden Aufträgen/Prozessen und beschränkten Betriebsmittelkapazitäten entstehen Warteschlangen. Daher sind Rechensysteme grob als Warteschlangennetze modellierbar. Analytische Methoden sind oft mit geringem Aufwand einsetzbar; die Ergebnisse vermitteln oft höhere Einsichten in das Systemverhalten bei unterschiedlichen Werten der Eingangsparameter. Andererseits verlangt ihr Einsatz ein großes Expertentum; das Einsatzspektrum ist durch recht restriktive Modellannahmen stark eingeschränkt. Analytische Auswertungsmethoden werden wie folgt klassifiziert:

- Exakte Methoden erfordern geringen Auswertungsaufwand, haben aber relativ kleine Einsatzbreite. Typische Einschränkungen sind: Exponentialverteilte Bedienzeiten, keine nebenläufigen

Betriebsmittelanforderungen, keine Aufspaltung eines Auftrags in Unteraufträge.

- Numerische Methoden besitzen ein erweitertes Einsatzspektrum, die Auswertungskosten wachsen aber meist wesentlich stärker als linear mit der Netzgröße an.
- Approximative Lösungsmethoden besitzen das größte Einsatzspektrum mit vergleichbar geringen Auswertungskosten, sind aber sehr komplex. Die wichtigste Methode in diesem Bereich basiert auf der Dekompositionstheorie, die Aussagen über hierarchische, separat analysierbare Zerlegungen eines Warteschlangennetzes macht.
- Operationale Analyse ist im wesentlichen eine Meßmethode, um aus einem Warteschlangennetz typische Leistungsgrößen durch Beobachtung zu ermitteln, ohne daß ein stochastisches Verhalten der meßbaren Größen wie z.B. Warteschlangenlängen und Auftragsverweilzeiten unterstellt wird.

Typische Anwendungen Analytischer Methoden sind Kapazitätsplanung für Großrechnersysteme und Leistungsvorhersage für E/A-Subsysteme. Hingegen ist die Modellierung von software-spezifischen Systemeigenschaften durch Warteschlangennetze kaum möglich; z.B. kann vom Dateninhalt abhängiger Steuer- oder Nachrichtenfluß nur durch konzeptionelle Erweiterungen solcher Netze modelliert werden.

Simulationsmodelle sind Nachbildungen von statischer Struktur und dynamischem Verhalten eines realen Systems. Erkenntnisse werden durch rechnergestützte Beobachtung der Ausgabegrößen des Modells über die Zeit gewonnen. Modellbeschreibungssprachen basieren z.B. auf algorithmischen Simulationssprachen, erweiterten Petrinetzen oder auch auf Warteschlangennetzen. Die wichtigste Klasse von Simulationsmethoden für Rechensysteme bildet die **zustandsdiskrete stochastische Simulation**, da Rechensysteme meist durch diskret unterschiedene Zustände und Zustandsübergänge beschrieben werden. Die Komplexität heutiger Rechensysteme bedingt eine approximierte Modellbildung gewisser Systemkomponenten durch ein stochastisches Ersatzverhalten, sodaß -im Gegensatz z.B. zur Regelungstheorie- stochastische gegenüber deterministischen Modellen dominieren. Der größte Vorteil von Simulationsmodellen ist ihr unbeschränktes Einsatzspektrum. Systeme können mit beliebigem Detaillierungsgrad modelliert und simuliert werden; sie sind also i.d.R. realistischer

als Analytische Modelle. Während Analytische Modelle stationäre Erwartungswerte für zu untersuchende Zielgrößen liefern, ist mit Simulationsmodellen auch transientes Systemverhalten untersuchbar; als Ergebnisse können statistische Verteilungen der Zielgrößen erhalten werden.
Dem stehen aber eine Reihe von Problemen gegenüber. Die Freiheit in der Wahl des Detaillierungsgrades bringt auch eine Unsicherheit über den geeigneten Grad mit sich, der nur durch exakte Zielvorgaben für die Simulation gemeistert werden kann. Simulation erfordert meist einen großen Rechenaufwand, der oft von der Problemgröße abhängt. Der Aufwand steigt stark mit der Anzahl der Eingabeparameter an, da ihre Wirkung auf die Zielgrößen zunächst nicht bekannt und gerade durch Simulationsexperimente zu ermitteln ist. Die oft vernachlässigte Prüfung der Modellparameter, des Simulationsmodells und der Simulationsergebnisse auf ihre Gültigkeit ist sehr wichtig, aber erfordert sehr gute statistische Kenntnisse für Parametermessungen am realen System, der Abschätzung der tolerierbaren Abweichungen zwischen dem realen System und dem Modell, sowie für die statistisch fundierte Schätzung der erforderlichen Simulationslaufzeit, um gegebene Konfidenzgrade in die Ergebnisse zu erzielen.
Analytische und simulative Methoden sollten nicht exklusiv angewendet, sondern können sinnvoll kombiniert werden.

- Durch ein vergröbertes Analytisches Systemmodell können Simulationsergebnisse validiert und die erforderliche Experimentanzahl reduziert werden.
- Durch den kombinierten Einsatz beider Methoden auf jeweils unterschiedliche Teile eines Modells kann der Untersuchungsaufwand ebenfalls reduziert werden: Nur Submodelle mit gefordertem hohem Detaillierungsgrad werden simuliert; Ergebnisse der analytischen Behandlung anderer Submodelle werden dabei verwendet. Man spricht dann von einer **hybriden Modellierungsmethode** /BEIL81/.

1.3 Ziele der Arbeit

In dieser Arbeit wollen wir einen Beitrag zu einem Instrumentarium zur **quantitativen Leistungs- und Verfügbarkeitsbewertung** verschiedener VDBS-Konfigurationen und -Protokolle leisten. Die vorgeschlagene

Methodik basiert auf zustandsdiskreter, stochastischer Simulation und ist nicht auf die untersuchte Systemklasse beschränkt, sondern auch auf andere stark nebenläufige, fehlertolerante Systeme anwendbar. Stochastische Simulation wird allgemein als einzig überzeugender Weg bei der Leistungs- und Zuverlässigkeitsprognose bzw. -verbesserung von VDBS akzeptiert.
Speziell sollen transaktionsorientierte Protokolle zur Verwaltung dezentralisierter und redundanter Datenbestände untersucht werden, die hohe Leistungs- und Verfügbarkeitsanforderungen gerecht werden könnten. Diese Protokolle beinhalten die Koordination konkurrierender Datenbankzugriffe sowie die Behandlung (Recovery) unterschiedlicher Störungen des Systembetriebs wie Rechnerknoten-, Leitungs- und Transaktionsfehler. Exemplarisch werden dazu drei unterschiedliche Protokolle untersucht, die jeweils in den Systemen VDN/REFLEX und POREL implementiert wurden sowie ein neuartiges Protokoll, das auf einem Mehrheitsentscheid-Prinzip basiert und in Grundzügen bereits in /LEBR82/ und in /BRLE82a,b/ veröffentlicht worden ist.
Bei der quantitativen Untersuchung solcher Protokolle soll die **Fehlertoleranz** eines VDBS bewertet werden, da fehlertolerante System zwar eine immer größere Rolle spielen, aber zur quantitativen Bewertung solcher Systeme bisher nur wenige Forschungsergebnisse vorliegen. Insbesondere interessieren dabei folgende Fragestellungen:

(1) Wie verhalten sich Leistungswerte eines Systems, wenn 'reale' Bedingungen angenommen werden (d.h. unter Berücksichtigung möglicher Fehler), verglichen mit dem fehlerfreien Zustand?

Zu untersuchen wären also die Auswirkung von Fehlerraten auf die Systemleistung. Dazu sollen Leistungs- mit Verfügbarkeits-Kenngrößen kombiniert werden mit dem Ziel, stärker benutzerorientierte Maße zu erhalten. Aus Benutzersicht interessiert z.B. nicht, ob bestimmte Fehler im System aufgetreten sind, sondern wie sich diese Fehler auswirken auf Antwortzeiten, Durchsatz und "Erfolg" seiner Aufträge. Dazu wird eine neue **hierarchische Simulationsmethode** entwickelt.

(2) Welchen Preis -bezogen auf Antwortzeiten und Durchsätze im fehlerfreien Fall- haben Recovery-/Restart-Algorithmen, die die Datenkonsistenz und Systemverfügbarkeit bei möglichst vielen

(Mehrfach-) Fehlern garantieren?

D.h. gefragt ist nach einer **vergleichenden Untersuchung** von Systemen, die jeweils ohne bzw. mit Recovery-/Restart-Algorithmen ausgestattet sind.

Um komplexe Rechensysteme effizient nachbilden zu können, ist eine **höhere, deskriptive Systembeschreibungssprache** erforderlich. In dieser Arbeit wird eine vorhandene, auf zeitattributierten Petrinetzen - den **Funktionsnetzen** - basierende Modellierungsprache /GODB83/ um Konzepte zur effizienten Modellierung stark nebenläufiger, fehlertoleranter Systeme erweitert. Das entsprechend erweiterte Softwarewerkzeug FUN zur stochastischen Simulation von Funktionsnetzmodellen soll für Experimente mit VDBS-Modellen eingesetzt werden. Bei Anwendung komplexer stochastischer Simulationsmodelle ist eine zuverlässige Absicherung der Simulationsergebnisse sehr schwierig. Daher sind Erweiterungen des Werkzeugs auch bezüglich einer statistisch fundierten Schätzung von Leistungs- und Verfügbarkeitskenngrößen notwendig. Teile der erweiterten Modellierungssprache und des darauf basierenden Werkzeugs sind bereits in /LEGO85/ vorgestellt worden.
Bei der Vielzahl möglicher Subsysteme, deren Interaktionen und Leistungsparameter sind Modellbeschränkungen unumgänglich, um die Komplexität beherrschbar zu machen. Es ist auch nicht immer sinnvoll, alle VDBS-Systemkomponenten mit besonders hohem Genauigkeitsgrad zu modellieren, da die Simulationsergebnisse bei Datenbanken stark vom Anwendungsprofil abhängen, während wir das VDBS-Anwendungsprofil selbst stochastisch nachbilden wollen. Folgende **Einschränkungen der VDBS-Systemklasse** und der erforderlichen Simulationsexperimente erscheinen daher sinnvoll:

- Vergröberte Modellierung relevanter VDBS-Subsysteme (lokale Datenbank- und Betriebssystemsoftware, Kommunikationssystem).

- Einschränkung der zu untersuchenden Protokolle zur verteilten Transaktionsverarbeitung auf einige wenige, die hohe Systemverfügbarkeit durch redundante Datenhaltung versprechen, d.h. daß auch bei gewissen mehrfachen Systemfehlern zumindest ein eingeschränkter Systembetrieb aufrechtgehalten werden kann.

Allerdings sollen diese VDBS-Protokolle im Detail nachgebildet werden.

- **Homogenität** des Modells bezüglich jedem Rechnerknoten (VDBS-Prozesse, Transaktionsprofil, Datenreplikation, Ausfallprofil, Rechnernetzstruktur); vergleiche dazu Abb. 1.

- Beschränkung der Experimentauswertung auf wichtige benutzerorientierte Zielgrößen. Als Folge der Knoten-Homogenität reicht es sogar aus, Auswertungen an einem beliebigem Knoten vorzunehmen; die Ergebnisse sind dann auf alle anderen Knoten übertragbar.

- Vermeidung exponentieller Parameterräume bei NP-vollständigen Problemen durch Festlegung plausibler Werte: das betrifft im wesentlichen die Aspekte der Datenreplikation, der Konfigurierung der Netzstruktur, sowie der Auswertungsplan-Generierung.

Trotz dieser Einschränkungen erwarten wir von unserer Untersuchung wesentlich weiterführende Ergebnisse gegenüber anderen oben erwähnten VDBS-Modellierungsstudien, da wir etliche deren Restriktionen nicht machen. Die wichtigsten **Erweiterungen der VDBS-Modellklasse** gegenüber anderen VDBS-Modellen sind

(3) Untersuchung der **Fehlertoleranz** durch Einbeziehung von Knoten- und Leitungsausfällen (insbesondere von **Rechnernetz-Partitionierung**) in die Leistungsanalyse;
(4) Betrachtung realistischer Datenverteilungen, speziell der **Teilreplikation** von Datenfragmenten, unter Berücksichtigung von Rechnernetz-Struktur und Ausfallraten;
(5) Betrachtung detaillierter Anwendungsprofile, insbesondere sowohl query-intensiver als auch update-intensiver Umgebungen;
(6) Detaillierte Modellierung der **Interprozeß-Kommunikation** für kooperierende VDBS-Prozesse.

Aus den Protokoll-Simulationsergebnissen sind Entscheidungskriterien ableitbar, um bei gegebenen Systemparametern und Zielanforderungen das jeweils günstigste Protokoll auszuwählen. Protokollvergleiche und -bewertungen können in fast allen Phasen des VDBS-Lebenszyklus von

Bedeutung sein:

- **Entwurfsphase**: Die Hauptanwendung von Software-Werkzeugen liegt in der Bewertung von Entwurfsalternativen (hier: verschiedenen Protokollen). Ist die Anwendungsumgebung vorgegeben, stellt sich das Problem der Auswahl von Grundsoftware-Moduln, die eine Optimierungsfunktion (z.B. Leistungs- / Kostenverhältnis) für diese Umgebung maximieren oder minimieren bzw. vorgegebene Anspruchsniveaus erreichen /STUE75/. Die hierfür typische Fragestellung ist: Für welche Kombination von Systemparametern ist ein Protokoll voraussichtlich besser geeignet als ein anderes? Man versucht also, bereits in der Entwurfsphase die Systemleistung zu prognostizieren, um unzumutbare Lösungen von vorneherein auszuschließen.

- **Installationsphase**: Die Anwenderumgebung ist hier bereits soweit konkretisiert, daß die Systemkonfiguration dimensioniert werden kann. In dieser Phase können Werkzeuge sowohl zur Identifizierung und Beseitigung von Engpässen eingesetzt werden /BORD81/, als auch zur Prüfung, ob eine geplante Konfiguration die quantitativen Anforderungen erfüllt. Damit können Rückschläge und erhöhte Wartungskosten nach Inbetriebnahme des Systems vermieden werden.

- **Betriebsphase**: Während des Lebenszyklus eines Rechensystems bleibt die Verteilung der Arbeitslast selten konstant. Eine stark veränderte Verteilung hat aber meist eine Verschlechterung der Leistungsgrößen zur Folge. Um diesen unerwünschten Effekten vorzubeugen, werden zukünftige Rechensysteme zunehmend mit automatischen Methoden zum adaptiven, dynamischen Tuning ausgestattet sein. Damit soll eine automatische SW-Einstellung zur Laufzeit bei Laständerungen erfolgen können. Eine Neueinstellung kann in einer SW-Parametervariation oder sogar im Austausch von Moduln bestehen. In beiden Fällen wird i.d.R. ein Erkennen von Änderungen der Stationarität (d.h. von Erwartungswerten des Umgebungsprofils) zur Laufzeit möglich sein, nicht aber die schnelle Auswahl der nach der Änderung optimalen Parameter bzw. Moduln. Es ist also eine Untersuchung der Änderungsalternativen vor der Betriebsphase notwendig.

Im Datenbankbereich wurde eine vergleichende Bewertung von Concurrency-Control-Verfahren bei analoger Zielsetzung vorgeschlagen /ROBI82/. Scheuernstuhl /SCHE81/ plädiert für dynamisch selbstorganisierende DBVS. Leistungsmessungen zur Laufzeit eines Datenbanksystems ergeben Statistiken über das Belastungsprofil, die vom integrierten Data Directory / Dictionary Subsystem (DD/D) verwaltet werden. Falls das aktuelle Profil gegebene Anspruchsniveaus nicht mehr erfüllt, wird die Datenbank reorganisiert.

Damit erscheint eine von uns angestrebte quantitative Untersuchung verschiedener Protokolle mit extensiver Variation der Eingabeparameter bereits in der Entwurfsphase gerechtfertigt, auch wenn die Ergebnisse erst zur Laufzeit eingesetzt würden. Diese Auffassung wird neuerdings auch in der Industrie vertreten (Computerwoche /CW83/, s. auch /MSS83, S.378/).

1.4 Struktur der Arbeit

In den Kapiteln 2 und 3 wird die Ausgangssituation dieser Arbeit beschrieben. Es wir eine Einführung in den Stand der Technik bei Methoden und Werkzeugen zur Spezifikation, Leistungs- und Verfügbarkeitsanalyse gegeben. Aus der Vielzahl existierender Vorschläge werden einige charakteristische herausgegriffen und bezüglich ihrer Eignung zur quantitativen Untersuchung bewertet. Kapitel 3 enthält eine Einführung in Anforderungen, Architektur und Protokolle auf redundanten Datenbeständen verteilter Datenbankverwaltungssysteme (VDBS), um eine uniforme Basis zur Modellierung und vergleichenden Bewertung solcher Systeme zu erzielen. Die wesentlichen Komponenten der VDBS-Umgebung werden im Kapitel 4 modelliert. Das Ergebnis ist das abstrakte, parametrisierte **VDBS-Basisreferenzmodell** (BRM). Das BRM ist ein einheitliches Simulations-"Testbett" für unterschiedliche VDBS-Modelle, deren Abstraktionsebene sich an den oben getroffenen Festlegungen der Systemklasse orientiert. Es besteht aus Anwendungsprofil (Daten- und Transaktionsprofil) und Umgebungsprofil (Hardwarearchitektur-Modell, Fehler-Modell, Kommunikationsmodell). Das stochastische Fehlermodell

berücksichtigt Rechnerknoten-, Leitungs- und Transaktionsfehler. Das Kommunikationsmodell enthält insbesondere eine für Protokolle zur verteilten Transaktionsverarbeitung geeignete robuste Interprozeßkommunikationsschicht, die bereits in /LEBR82/ vorgestellt worden ist. In dieser Schicht werden speziell globale Kommunikationsfehler erkannt, z.T. behoben und an höhere Schichten gemeldet, sodaß VDBS-Protokolle sehr effizient modelliert werden können.

In Kapitel 5 werden simulative Methoden zur Schätzung von Leistung und Verfügbarkeit von VDBS definiert und zu einem **Auswertungsmodell** für VDBS zusammengefaßt. Als mathematische Grundlagen dienen die von Warteschlangennetzen bekannte Theorie der Operationalen Analyse sowie statistische Schätzverfahren für stationäre Zielgrößen. Es werden spezielle Maße zur integrierten Betrachtung von Transaktions-Durchsatz bzw. -Erfolgsrate mit der Verfügbarkeit eines fehlertoleranten Systems definiert und eine **hierarchische Simulationsmethode** zur Schätzung dieser Maße aus Simulationsergebnissen entwickelt. Die Methode gestattet die Einbeziehung der Auswirkung von Systemfehlern in die (VDBS-)Leistung.

In Kapitel 6 wird die auf Funktionsnetzen basierende **Modellierungssprache** konzeptionell erweitert und verbessert, um formale, kompakte VDBS-Simulationsmodelle darstellen zu können. Wesentliche Erweiterungen sind Konstrukte zur Beschreibung von indeterministischem und fehlerhaftem Systemverhalten sowie von zeitverbrauchenden Betriebsmitteln. Die Erweiterungen sind generell zur Modellierung von verteilten bzw. fehlertoleranten Systemen gut geeignet. Die Analysierbarkeit der Netzsprache sowohl bezüglich der Petrinetz- als auch der Warteschlangentheorie wird untersucht. Schließlich werden Architektur und interaktive Benutzerschnittstelle des experimentellen Software-Werkzeugs DAEMON beschrieben, das die Erstellung, Analyse und Simulation von Netzmodellen unterstützt.

In den Kapiteln 3 bis 5 wurden die VDBS-Protokolle, -Komponenten, -Parameter und -Zielgrößen eingeführt. Mit Hilfe der erweiterten Sprache der Funktionsnetze wird im Kapitel 7 das vollständige **VDBS-Simulationsmodell** spezifiziert. Es folgt in Kapitel 8 eine Beschreibung der mit diesen Modellen ausgeführten Simulationsexperimente mit Diskussion der **Simulationsergebnisse**, die

einen Vergleich der VDBS-Protokolle bei unterschiedlichen Mengengerüsten einschließt.

Kapitel 9 faßt die wesentlichen Erkenntnisse der Arbeit zusammen und gibt einem Ausblick auf weitere Anreicherungen eines rechnergestützten Instrumentariums zur quantitativen Untersuchung komplexer Rechensysteme.

Im Anhang werden alle Eingabegrößen des Simulationsmodells spezifiziert. Abschließend ist eine detaillierte Beispielsitzung mit dem DAEMON-System dokumentiert.

2 METHODEN UND WERKZEUGE ZUR SPEZIFIKATION UND BEWERTUNG VON VERTEILTEN SYSTEMEN

Der Schwerpunkt der vorliegenden Arbeit liegt in der Entwicklung eines rechnergestützten Instrumentariums zur simulativen Bewertung von Leistung und Fehlertoleranz (spezieller) verteilter Systeme bis zur Abstraktionsebene der Protokolle. Dieses Teilgebiet läßt sich einordnen in den allgemeinen Bereich der Methoden und Werkzeuge zur Spezifikation und Bewertung von Informationssystemen.
Die Komplexität Verteilter Systeme erfordert eine konstruktive, systematische Vorgehensweise bei der Modellierung. Die dazu notwendigen Schritte werden im nächsten Abschnitt beschrieben. Geeignete Spezifikations- und Bewertungsmethoden müssen eine **konzeptionelle Grundlage** mit einer zugehörigen **Systembeschreibungssprache** aufweisen. Es wird ein Kriterienkatalog für geeignete Methoden aufgestellt und ein repräsentativer Überblick über das große Spektrum existierender Ansätze vermittelt. Abschließend wird die Eignung der Methoden zur Spezifikation und Bewertung von fehlertoleranten verteilten Systemen diskutiert.

2.1 Modellierungsschritte zur Spezifikation und Bewertung von Verteilten Rechensystemen

Der Modellbildungs- und Simulationsprozeß kann vereinfacht in mehrere Phasen zerlegt werden, die in Abb. 2.1 skizziert sind. Eine Analogie zwischen bekannten Phasenmodellen des Software-Entwicklungszyklus und den beschriebenen Modell-Entwicklungsphasen wird dabei deutlich. Neben der Simulation können Modelleigenschaften z.B. durch netztheoretische Analyseverfahren ermittelt werden. Die integrierte Vorgehensweise mit Einbeziehung beider Wege zeigt Abb. 2.2, in die sich die Entwicklungsphasen aus Abb. 2.1 einbetten lassen.

Zur simulativen Untersuchung eines zustandsdiskreten Systems werden folgende Phasen z.T. iterativ durchlaufen:

(1) Systemanalyse: Untersuchung der zu modellierenden Systeme und gegebene Modellierungsziele führen zu (evtl. vereinfachenden) **theoretischen Annahmen** über das System. Das betrachtete System

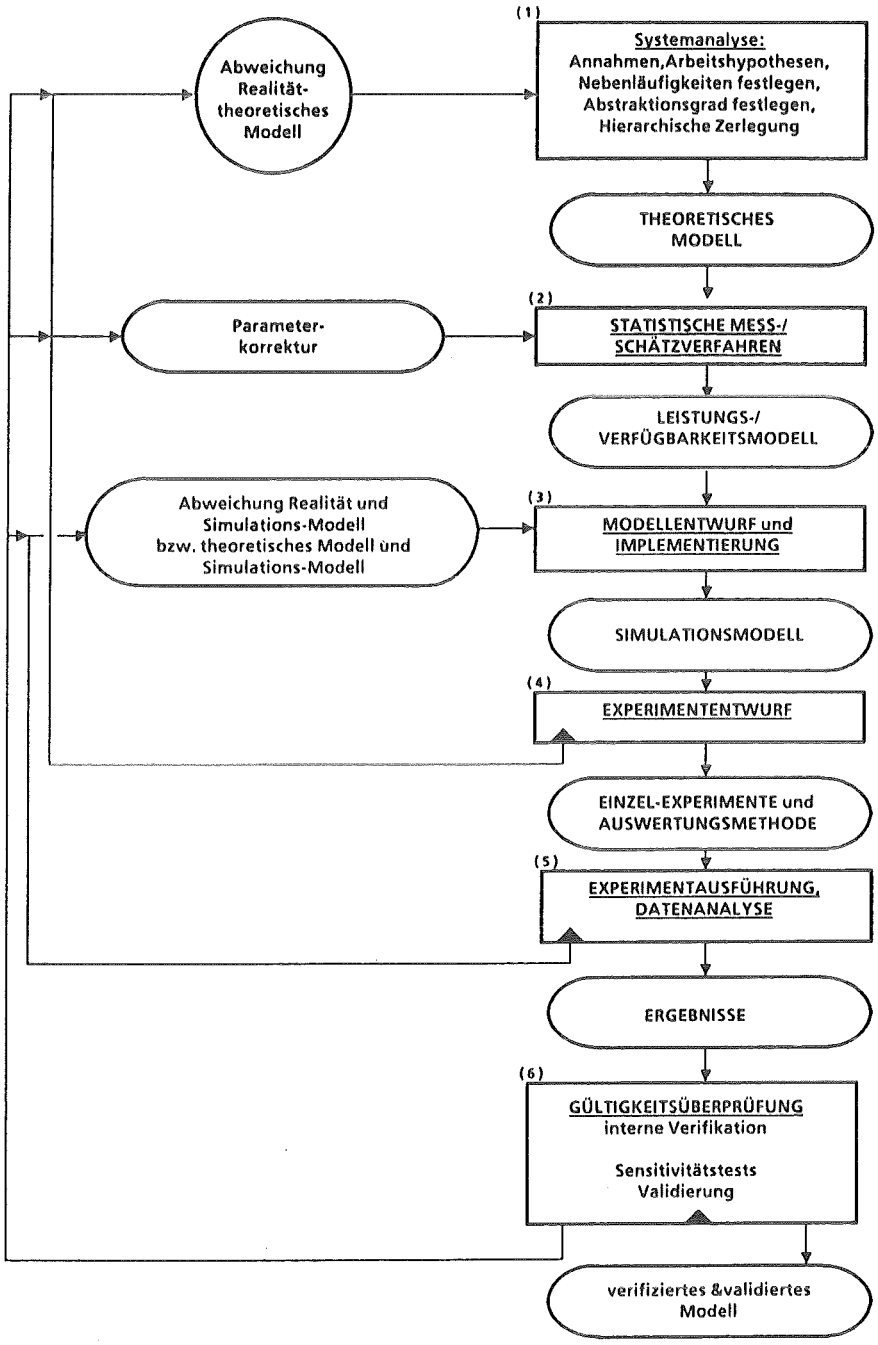


Abb. 2.1: Entwicklungsphasen der (VDBS-)Modellierung und -Simulation

und die Systemumgebung werden grob in Submodelle zerlegt ('top-down Vorgehen').

Interaktionen zwischen den Submodellen definieren die wesentlichen nebenläufigen Modellaktivitäten. Für jedes Submodell ist ein (ggf. unterschiedlicher) Abstraktionsgrad zu wählen. Dieser entspricht der nachgebildeten Hard- und Software; als geeignetes Maß für den Abstraktionsgrad kann die **Zeitgranularität** betrachtet werden, d.h. die mittlere Zeitdauer zwischen zwei

Modellereignissen, die mit der Zeitdauer zwischen zwei Ereignissen im realen System verglichen wird. Bei Rechensystemen kann die Granularität zwischen Nanosekunden und Minuten variieren. Gemäß einer bekannten heuristischen Regel sollen Submodelle, auf die sich die Fragestellung bezieht, möglichst genau modelliert werden. Z.B. ist bei unseren VDBS-Modellen eine Detailmodellierung der verteilten Transaktionsverarbeitung notwendig, da es sich um eine Protokoll-Vergleichsstudie handelt. Hingegen können höhere Endbenutzerschnittstellen, niedere Rechnernetz-Schichten und lokale DBVS-Schichten aggregiert nachgebildet werden. Diese "ausgeblendeten" Subsysteme kann man nun z.B. mit Hilfe einer **hierarchischen Zerlegungsmethode** durch ihr stationäres zeitliches Ersatzverhalten approximieren. Hingegen ist die Detailnachbildung der dem Modellierer am besten bekannten Subsysteme weitverbreitet, aber nicht zu empfehlen.

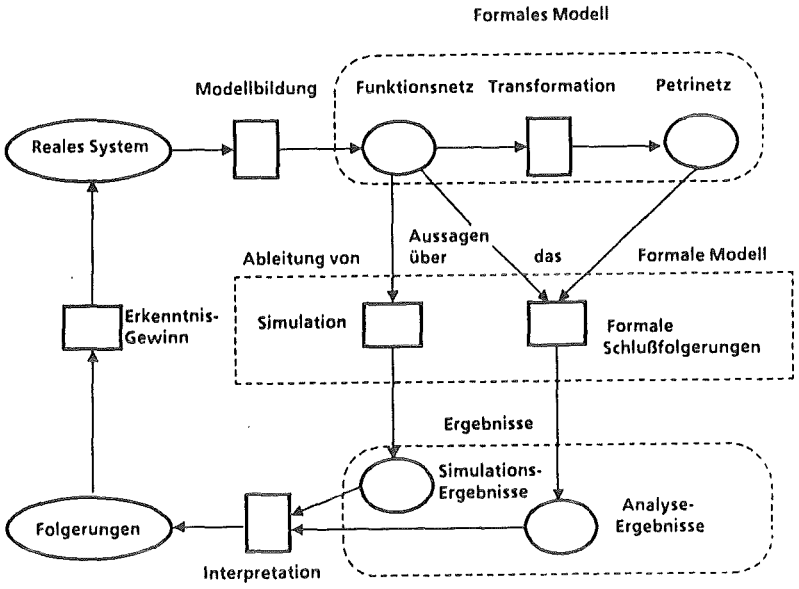


Abb. 2.2: Formales System als Modell eines realen Systems (aus /GODB83/ entnommen)

(2) Festlegung und Kalibrierung von Eingabeparametern; Festlegung der Zielgrößen: Die Wahl der **Eingabegrößen** hängt vom Abstraktionsgrad der einzelnen Submodelle ab. Festlegung der Größen (Mittelwerte, stochastische Verteilungen) pro Submodell erfolgt meist aufbauend auf statistische Messungen am realen System. Da uns gemessene

Werte nicht zur Verfügung stehen, gehen wir von publizierten oder empirisch gewonnenen Werten aus. Das entspricht der Situation in der Systementwurfsphase, in der nur ein konzeptionelles Zielsystem existiert. Es wird daher Expertenwissen zur Parameter-Grobeinstellung verwendet ('empirische Kalibrierung').
Der **Werteraum** von Parametern in VDBS-Modellen ist meist sehr groß, wie bereits in Kap. 1.3 begründet wurde. Unser Simulationsmodell erhebt dagegen keinen Anspruch auf Vollständigkeit, da lediglich eine bestimmte VDBS-Klasse untersucht wird. Daher erscheint es gerechtfertigt, für die praktische Modellierung den Werteraum aller solcher Parameter zu begrenzen, die einen "nichtlinearen" Wertebereich haben, d.h. deren Werteanzahl nichtlinear von bestimmten anderen Parametern abhängt. Beispiele sind die Rechnernetzstruktur oder die Verteilung von Dateneinheiten auf Knoten: In beiden Fällen wächst die Anzahl der Verteilungsalternativen exponentiell mit der Knotenanzahl. In solchen Fällen werden daher beim Parameterentwurf die möglichen Wertebereiche stark reduziert; der Preis ist ein eingeschränkter Gültigkeitsbereich der Simulationsergebnisse.
Aus den globalen Modellierungszielen jedes Submodells wird die Menge der **Zielgrößen** abgeleitet. Zusammen mit statistischen Schätzverfahren zu der Ermittlung stationärer Erwartungswerte der Zielgrößen aus den Ergebnissen erhält man das **Leistungs/Verfügbarkeitsmodell.**

(3) Modellentwurf und -Implementierung: Aufbauend auf (1) und (2), wird das **Simulationsmodell** implementiert. Die **Modellierungssprache** beinflußt dabei den Modellerstellungsaufwand und die Qualitätsmerkmale des Simulationsmodells wesentlich (vergl. Kap 2.1). Jedes Submodell wird dabei strukturell nachgebildet. Dazu gehören die statische Betriebsmittel- Prozeß-, und Protokollstruktur sowie der dynamische Daten- und Kontrollfluß.

(4) Entwurf der Gesamtexperimentserie, bestehend aus Einzelexperimenten. Wir folgen einer Empfehlung von /DIAZ82/: Nach einer top-down Vorgehensweise bei der Modellbildung

verfolgen wir eine bottom-up Strategie bei der Experimentführung, die bei der VDBS-Modellierung zu zwei Gruppen von Experimenten führt. Vor der eigentlichen Simulation der Transaktionsverarbeitung werden Experimente mit den dazu benötigten unteren Diensten (Interprozeß-Kommunikation und lokale Datenverwaltung, s. Abb. 3.2) ausgeführt, um diese bezüglich ihrer Dienstspezifikation funktional zu "verifizieren" (plausibel zu machen).
Eine statistisch fundierte Planung der Experimentserie bei vielen Eingangswerten und mehrdimensionaler Zielfunktion wurde in den meisten Modellstudien stark vernachlässigt: Die Annahme voneinander unabhängiger Eingangsparameter ergibt eine geringere Anzahl von Experimenten, ist aber speziell bei Rechensystemmodellen meist falsch und führt zu inkorrekter Interpretation der Untersuchungsergebnisse. Die Problematik wird in /JOWE84/ aufgezeigt; wir gehen darauf in einem späteren Kapitel noch näher ein.

(5) Ausführung der Einzelexperimente: Für jedes Experiment werden Eingabeparameterwerte entsprechend der Planung in Schritt (4) vorgegeben und die Auswirkungen dieser Werte auf alle Zielgrößen mit statistischen Methoden untersucht. Um einerseits statistisch fundiert vorzugehen, aber andererseits auch den Experimentieraufwand zu begrenzen, sind bei den Simulationsläufen **exakte Start/Stop-Bedingungen** bei erforderlichem Konfidenzgrad für jede Zielgröße einzuhalten.

Gültigkeitsnachweis

Das Ziel der Gültigkeitsüberprüfung von Simulationsmodellen ist nach /STUE75/, /PAGE83/ die Schaffung eines akzeptablen Vertrauensgrades in Korrektheit bzw. Gültigkeit des Modells bezüglich einem realen Zielsystem. Dazu gehört z.B. auch eine statistisch fundierte Experimententwurfsmethode. Prinzipiell kann man nach /MSS83/ für die Gültigkeitsüberprüfung komplexer Simulationsmodelle vier Schritte unterscheiden. Die Schritte (b) bis (d) sind in die Phasen (4) und (5) (Experimententwurf und -ausführung) mit einzubeziehen.

(a) Kalibrierung,

(b) Verifikation,

(c) Sensitivitätsanalyse,

(d) Validierung. Dieser Begriff wird nicht einheitlich verwendet; gemeint ist hier die längerfristige u/o prognostische Gütebewertung des Simulationsmodells an Hand von Outputvergleichen Modell - reales System. Die Validierung ist unter äquivalenten Versuchsbedingungen durchzuführen, d.h. Eingangsparameter von Modell und realem System sind entsprechend gleich zu variieren und die Ergebnisse mit Hilfe statistischer Fehlerabschätzungen zu vergleichen. Die Grenzen der Validierbarkeit werden erreicht, wenn im Rahmen einer **Entwurfsstudie** Entwurfsparameter im Modell variiert werden, während im realen System solche Parameter gar nicht oder nur mit großem Aufwand variiert werden können. Sinnvoller ist es, Simulation zur Entwurfsunterstützung einzusetzen, die Entwurfsparameter entsprechend den Ergebnissen festzulegen, um nach Systemimplementierung eine Validierung mit diesen festliegenden Parametern vorzunehmen.

Wir beschränken uns im folgenden auf die Diskussion der Möglichkeiten, die Korrektheit des Simulationsmodells durch Verifikations- und Sensitivitätsanalysemethoden nachzuweisen. Auf Kalibrierung wurde oben schon eingegangen; zur Validierung stehen uns keine ausreichenden Meßdaten zur Verfügung.

Verifikation

Ein Anspruch auf vollständige Verifikation eines komplexen Modells -wie in /MSS83/ gefordert- kann wegen der bereits erwähnten beschränkten Modellvergleichbarkeit und auch aus Aufwandsgründen nicht erhoben werden. Vielmehr sollte eine Palette von Überprüfungsmethoden angeboten und benutzt werden, die es gestatten, mit vertretbarem Aufwand grobe Unkorrektheiten eines Modells aufdecken zu können.
Ansatzpunkt aller Methoden ist dabei eine hierarchische Zerlegung in Submodelle. Wir schlagen drei Überprüfungsphasen vor, die (im Idealfall) bottom-up auf jedes Submodell und abschließend auf das integrierte Gesamtmodell anzuwenden sind.

(A) **Überprüfung der theoretischen Annahmen** eines (Sub-) Modells, die in Phase (1) festgelegt wurden, z.B. mit Hilfe der allgemeinen Netztheorie (Netzanalyse). Für bestimmte einfache

Klassen von Petrinetzen ist eine Analyse allgemeiner prädikativer Eigenschaften wie Sicherheit, Lebendigkeit und auch der Nachweis spezieller Eigenschaften ('Fakten') mit Hilfe der Erreichbarkeitsanalyse möglich. Diese Analysemethoden sind auf Erweiterte (insbesonders: zeitattributierte) Petrinetze nicht ohne weiteres übertragbar, zumal schon einfache Erweiterungen schnell zur Nichtentscheidbarkeit der Fragestellungen führen /GODB83/. Erfolgreiche Analysen 'höherer' semantischer Modelleigenschaften sind in Einzelfällen gelungen, z.B. /BAGA81/ bzw. /WALT82/ für ein Commit-Protokoll bzw. für ein Modell der globalen Ausfallerkennung.
Netzanalysemethoden sind zwar formal fundiert, aber ohne Rechnerunterstützung sehr schwierig zu handhaben. Daher wird zur Plausibilitätsprüfung von theoretischen Annahmen die Netzauswertungsmethode (s.u.) bevorzugt.

(B) **Überprüfung der statischen und dynamischen Struktur** eines (Sub-) Modells durch
- Netzanalyse (zum Test der statischen Struktur);
- Netzauswertung (Ablaufkontrolle der dynamischen Struktur, z.B. durch Schaltverfolgung). Damit ist ein Test auf **Strukturähnlichkeit** zwischen entworfenem und implementiertem Simulationsmodell möglich.

Z.B. kann geprüft werden, ob Modellentwurf und -implementierung eines Commitprotokolls die gleiche Folge von Zwischenzuständen durchlaufen, und ob die gleichen Endzustände erreicht werden. Allerdings ist diese Methode bei Modellen fehlertolerierender Systeme nur dann sinnvoll, wenn zumindest ein Teil der möglichen Fehlerzustände und -übergänge erfaßt werden, insbesondere die Recoveryverfahren als Reaktion auf einen Fehlerübergang. Eine vollständige Überprüfung aller Übergänge ist sehr aufwendig, kann aber durch eine neuartige Simulationsmethode für fehlertolerante Systeme unterstützt werden (s. Kap. 5).

(C) **Überprüfung des Outputs** durch analytische Vergleichsrechnungen mit entsprechend vergröbertem Modell ('interne Verifikation'). Diese wichtige Methode ist zur Überprüfung von einfacheren

VDBS-Simulationsmodellen bereits eingesetzt worden /GARC79, CHEN81/.

In der Terminologie von /MSS83/ sind die vorgeschlagenen Methoden unter "Verifikation_1" bzw. "Verifikation_2" anzusiedeln.

Sensitivitätsanalyse

Zu dieser Methode vergl. die detaillierte Betrachtung in /STUE75/. Sensitivitätsanalyse sollte gezielt eingesetzt werden, um unsensitive, übersensitive bzw. unstabile Inputparameter des Simulationsmodells zu erkennen, die hinweisen auf

- ein bezüglich den Systemannahmen oder der Modellimplementierung inkorrektes Modell,
- unwesentliche Einflußgrößen,
- kritische Einflußgrößen, um daraufhin das Modell entsprechend zu revidieren.

Allerdings ist eine Interpretation oft problematisch. Z.B. weist eine Unstabilität eines Rechensystemmodells an der Auslastungsgrenze nicht notwendig auf Modellfehler hin, da auch das reale System unter hoher Belastung sich ähnlich verhalten kann.

Eine Modellierungsmethodik für Verteilte Systeme ist nur ansatzweise ausgearbeitet worden, z.B. in /DIAZ82/. Bereits für Einzelrechnersysteme gibt es hierzu nur wenige Untersuchungen. Z.B. ist in /MSS83/ eine Simulationsmethodik mit Schwerpunkt auf Gültigkeitsüberprüfung entwickelt worden; in /ZEIG84/ wird der systematische Entwurf von Rechensystemmodellen beschrieben.

2.2 Anforderungen an geeignete Methoden und Werkzeuge

In diesem Abschnitt wird ein allgemeiner Anforderungskatalog zur Bewertung von Systembeschreibungssprachen für Spezifikation und Bewertung von komplexen Rechensystemen erstellt. Ähnliche Anforderungen sind auch von Godbersen /GODB83/ und Schneider /SCHN81/ entwickelt worden. Anschließend werden die allgemeinen Kriterien um spezifische Anforderungen an eine Beschreibungssprache für die Klasse der fehlertoleranten verteilten Systeme erweitert.

- Höhere Beschreibungssprache: Um den konzeptionellen Abstand zwischen den mentalen Modellen der Werkzeug-Benutzer und der Sprache möglichst klein zu halten, soll die Sprache **deskriptiv** u/o **graphisch** sein. Die Sprache soll auf einem **eindeutigen, formal fundierten Grundkonzept** basieren, um auch die Prüfbarkeit funktionaler Systemeigenschaften zu ermöglichen. Sie soll auf einem **offenen, anwendungsneutralen Ansatz** basieren, sodaß eine möglichst große Klasse von Rechensystemen beschreibbar ist.

- Einheitliche, konsistente Sprache zur Beschreibung
  -- unterschiedlicher Systemkomponenten, insbesondere von Hardware- und Software-Funktionen sowie von komplexen Datenstrukturen (Systemarchitektur, Dienste, Protokolle, Prozesse) mit derselben Methode. Der dynamische Kontroll- und Informationsfluß eines Systems soll ebenfalls einheitlich formulierbar sein.
  -- unterschiedlicher Detaillierungsgrade, insbesondere sollen Modellverfeinerung und -vergröberung mit wohldefinierten Übergängen von der Sprache unterstützt werden. Das Hauptproblem einer Modellierung liegt in der Wahl des geeigneten Abstraktionsgrades. Es muß möglich sein, Subsysteme auf unterschiedlichem Detaillierungsniveau in einem Modell zu repräsentieren. Maße für das Detaillierungsniveau von Rechensystemmodellen sind Hardware- und Softwarekomplexität sowie die minimale Zeitdauer zwischen zwei Modellereignissen. Eine Vergröberung kann auch in der Aggregierung einer Systemkomponente bestehen, die lediglich durch ihr voraussichtliches Zeitverhalten beschrieben wird. Das ist speziell für Systeme mit indeterministischen oder stochastischen Verhalten von Bedeutung. (Anmerkung: Z.B. können Interaktionen zwischen Protokollinstanzen indeterministisch erfolgen, etwa dann, wenn aus der Sicht einer lokalen Instanz mehrere Folgeereignisse möglich sind, ohne daß die Instanz das nächste Ereignis vorhersagen kann.)
  -- unterschiedlicher Systementwicklungsphasen von der Anforderungs- bis zur Implementierungsspezifikation. Die Sprache soll zur Bewertung von Designalternativen, für System- und Anwendungs-Tuning, als auch zur Konfigurationsunter-

stützung geeignet sein.

-- unterschiedlicher Systemsichten und Modellierungsziele: Die Sprache soll insbesondere die Umsetzung eines Spezifikationsmodells in ein Bewertungsmodell unterstützen, d.h. Spezifikation und Untersuchung von Leistungs- und Verfügbarkeitseigenschaften sollen mit einheitlichen Sprachkonstrukten möglich sein.

- Modell-Kompaktheit: Die durch die Sprache formulierbaren Modelle sollen bezüglich ihrer Komplexität möglichst wenig von der Komplexität des zu beschreibenden Systems abhhängen, um unübersichtliche, 'fußballfeldgroße' Modelle zu vermeiden. Die bekannte Zustandsexplosion bei vielen zustandsorientierten Methoden z.B. bei der Modellierung komplexer Datenstrukturen muß verhindert werden.

- Heterogene Lösungstechniken. Die Sprache soll sowohl simulative Lösungstechniken unterstützen, als auch eine analytische Untersuchung und Verifikation des Modells erlauben.

- Rechnerunterstützung: Zur Modellspezifikation und -untersuchung sollen rechnergestützte Werkzeuge verfügbar sein. Der Einsatz von Werkzeugen kann den Aufwand bei der Modellerstellung reduzieren und die Entdeckung von Fehlern in Modellen erleichtern. Viele analytische und simulationsorientierte Modelle können ohne Rechnerunterstützung gar nicht ausgewertet werden. Je größer und komplexer Modelle werden, umso wichtiger werden Werkzeuge, die sowohl eine effiziente, konstruktive Modellerstellung unterstützen, als auch z.B. durch interaktive Simulationslaufkontrolle und flexible Filterung irrelevanter Ausgabeinformation schnell die gewünschten Experimentdaten bereitstellen.

Dieser Katalog erhebt keinen Anspruch auf Vollständigkeit, zeigt jedoch charakteristische Anforderungen an geeignete Spezifikations- und Bewertungsmethoden für Rechensysteme im allgemeinen. Um geeignete Methoden für die uns interessierende Systemklasse zu finden sind zusätzliche Sprachkonstrukte erforderlich zur

- Modellierung von nebenläufigen Prozessen und von komplexen Mehrpartnerprotokollen für **verteilte Systeme**;
- Modellierung sowohl vom operationalen 'Normalverhalten' als auch vom Verhalten bei Teilausfall und Wiederanlauf für **fehlertolerante Systeme**.

Diese Konstrukte sollen einfach formulierbar sein und zu kompakten Modellen führen.

2.3 Klassifikation existierender Methoden und Werkzeuge

Existierende Methoden zur formalen, rechnergestützten Spezifikation und Bewertung komplexer Rechensysteme lassen sich grob in folgende Klassen einteilen:

(1) **Implementierungsnahe Methoden**. Darunter fallen Software- und Hardware-gestützte Meßmethoden zur Untersuchung vorhandener Rechensysteme, die wir aber im Rahmen dieser Arbeit nicht betrachten, da sie nicht zur Entwurfsunterstützung geeignet sind. Eine Ausnahme hiervon bilden integrierte Meß- und Implementierungsmethoden für verteilte Systeme, die **verteilten Testbettsysteme**, die mittlerweise eine gewisse Verbreitung gefunden haben /FRAN82, GGK83, TOMA83/.

(2) **Algebraische Spezifikationsmethoden**. Beispiele solcher Methoden sind Communicating Sequential Processes (CSP) von Hoare /HOAR78/ und Calculus of Communicating Systems (CCS) von Milner /MILN80, MILN82/. Das Ziel dieser Methoden ist eine streng formale Beschreibung der Syntax (durch CSP) und Semantik (durch CCS) von Kommunikationsmechanismen für verteilte Systeme, um rechnergestützte Verifikation dieser Mechanismen zu ermöglichen. CSP-Derivate werden zum Teil auch zur Implementierung verteilter Systeme eingesetzt. Z.B ist in /BAIA84/ eine CSP-basierte Sprache für fehlertolerante Prozeßsysteme beschrieben. Eine graphische Notation ist in diesen Methoden nicht vorgesehen.
In jüngster Zeit wurde die Sprache LOTOS auf der Basis von CCS entworfen und erfolgreich zur Spezifikation komplexer Protokollschichten und -dienste eingesetzt /ISO83/.

CCS-Erweiterungen um die Modellierung des zeitlichen Systemverhaltens erlauben die Anwendung analytischer Methoden zur Leistungsuntersuchung /NOYE84/.

(3) **Indeterministische endliche Transitionssysteme.** Rechensysteme werden meist als Systeme aus endlich vielen Zuständen und evtl. indeterministischen Zustandsübergängen aufgefaßt. Spezielle Transitionssysteme sind

- Erweiterte endliche Automaten. Darauf basieren die weit verbreiteten Protokollspezifikationssprachen ESTELLE /ISO85/ und SDL /DAGO83/, als auch die Methoden in /BAUE84/ und in /NIGA81/. Beide basieren auf einer ESTELLE-ähnlichen Spezifikationssprache, bieten aber darüber hinaus Hilfsmittel zur rechnergestützten Verfifikation und Leistungsuntersuchung an.
- Warteschlangennetze. Darauf basieren die meisten analytischen Untersuchungsmethoden (s. /AKBO81/) und z.B. die Modellierungs- und Simulationswerkzeuge COPE /BEGR82/, ILMAOS /JOBM82/, QSYS /KOST82/ und RESQ /SMK81/.
- Zeitattributierte Petrinetze. Beispiele zugehöriger (Simulations-)Werkzeuge sind Funktionsnetze /GODB83/ und UCLA-Graphs /VSE83/ zur Leistungs- sowie HARP /GTDS83/ zur Zuverlässigkeitsanalyse.

Die Unterscheidung dieser drei Methodenklassen ist nicht besonders scharf, da viele der mit einer Methode formulierten Modelle in Modelle anderer Methoden überführbar sind. Es existiert stets eine graphische Beschreibungssprache, die oft auch Konstrukte aus höheren Programmiersprachen als Beschriftungen der graphischen Sprachelemente enthält, um den bei 'reinen' Transitionssystemen oft auftretenden sehr großen Zustandsraum zu reduzieren. Beschreibungen anderer Ansätze auf der Basis von Transitionssystemen wurden in /GODB83/ beschrieben.

(4) **Hybride Untersuchungsmethoden.** Verschiedene Techniken zur Untersuchung eines Systems werden bei diesen verhältnismäßig neuen Methoden integriert (vergl. /BEIL81, SHSA83/). Der heterogene Ansatz wird meist zusammen mit einer hierarchischen Zerlegung des Modellzustandsraums in Submodelle angewendet, indem

grob modellierte Submodelle analytisch und detailliert modellierte, analytisch nicht oder nur aufwendig behandelbare Submodelle simulativ untersucht werden. Die Ergebnisse werden dann integriert. Gegenüber einer rein simulativen Technik ist der Untersuchungsaufwand oft wesentlich reduziert. Anstatt analytischer Submodell-Lösungen wird manchmal auch ein abgeschätztes stochastisches Ersatzverhalten für ein Submodell eingesetzt.

Hybride Methoden sind oft in Fallbeispielen erfolgreich eingesetzt worden. Ein spezielles Werkzeug, das hybride Methoden zur Untersuchung der Konfigurierung bestimmter Rechnertypen verwendet, ist z.B. SIM8860 /BORD81/; eine Weiterentwicklung ist in /SCHW85/ beschrieben.

Bei dieser Klassifikation wurden nur Methoden bewertet, die ein formales Basiskonzept aufweisen, um auf eine größere Systemklasse anwendbar zu sein. Maßgeschneiderte Modellierungswerkzeuge für dedizierte Anwendungen, z.B. das Werkzeug POSID zur Leistungsbewertung verteilter Datenbanksysteme /BIFL79/, wurden dabei nicht berücksichtigt.

2.4 Beschreibung ausgewählter Methoden und Werkzeuge

2.4.1 Untersuchung mit verteilten Testbettsystemen

Mehrere Verteilte Testbettsysteme (VTBS) wurden in den letzten Jahren entwickelt. Das Ziel dieser Systeme ist, ein Forschungswerkzeug zur Leistungsuntersuchung verschiedener Designalternativen wie z.B. alternativer Concurrency-Control-Verfahren in verteilten Systemen zu realisieren. Dabei wird angenommen, daß reale Systeme selten so modular aufgebaut sind, daß ein einfacher Funktionsaustausch möglich ist, während analytische und simulative Untersuchungen wegen ihren vereinfachenden Annahmen weniger genaue und realistische Ergebnisse liefern können.

Die VTBS-Architektur beinhaltet

- eine verteilte Hardware-Architektur,
- uniforme, generalisierte und erweiterbare Schnittstellen zur

Entwicklung höherer Protokolle, meist als Dienste zur Interprozeß-Kommunikation und für Zugriffe auf lokale Datenbank- oder Dateiverwaltungssysteme;
- Software- und teilweise auch hardware-gestützte Instrumentierung zur Experimentführung, Leistungsmessung und Test von höheren Protokollen;
- Unterstützung bei der Generierung künstlicher Lastprofile.

Die VTBS-Entwicklungskosten sind wesentlich kleiner als bei vergleichbaren realen Systemen, da auf Leistungsoptimierung verzichtet wird: Man nimmt an, daß sich eine schlechtere Leistungscharakteristik in etwa gleich auf zu untersuchende Designalternativen auswirkt. Wenn außerdem wesentliche Komponenten des realen Systems mit genügendem Detaillierungsgrad nachgebildet wurden, wird erwartet, daß vergleichende Leistungsuntersuchungen brauchbare Resultate liefern.

2.4.2 Spezifikation mit algebraischen Methoden

Als Beispiel einer algebraischen Spezifikationsmethode wird Milner's CCS (Calculus of Communicating Systems) besprochen.
CCS ist eine algebraische Theorie zur Beschreibung des Verhaltens kommunizierender Instanzen, z.B. verteilter kooperierender Prozesse oder Protokolle. Ein 'Verhalten' entspricht einer möglichen Ausführungsfolge aller Instanzen eines durch erweiterte reguläre Ausdrücke und Axiome beschriebenen Protokolls. Die Kommunikation zwischen Instanzen in CCS ist direkt (d.h. ohne Verbindungsaufbau/abbau) und synchron entsprechend dem rendez-vous Konzept von CSP. Im erweiterten CCS /MILN82/ ist auch asynchrones Verhalten beschreibbar.
CCS abstrahiert von der Implementierung von Kommunikationsschnittstellen, z.B. von Ports, Mailboxes, gemeinsamen Variablen. Die Einheit der Kommunikation ist ein 'Ereignis'. Ein Ereignis erfolgt entweder durch den Austausch von Variablenwerten durch 'Aktionen' mindestens zweier Instanzen, oder allein durch die Synchronisation der jeweiligen Aktionen. Beteiligt sind stets eine Sende- und eine oder mehrere Empfangsinstanzen. Eine Aktion ist ein Einzelschritt einer Instanz, der nach Ausführung der vorhergehenden Aktion aktivierbar ist. Der Einzelschritt wird genau dann aktiviert,

wenn alle 'zugehörigen' Aktionen der an einem Ereignis beteiligten Instanzen ebenfalls aktivierbar sind. Die Zugehörigkeit ist eindeutig definiert. Die Aktionsausführung -also ein Ereignis- durch Wertaustausch oder durch 'pure' Synchronisation findet atomar statt; Zeitverbrauch ist dabei nicht vorgesehen. Nebenläufige Ereignisse sind möglich, können aber durch Restriktionen beschränkt werden. Indeterministisches Verhalten bezüglich der Ausführungsreihenfolge ist möglich.
Das Grundkonzept von CCS ist die **Beobachtungs-Äquivalenz** verschiedener Verhalten. Intuitiv gesehen, sind zwei Verhalten eines Protokolls zwischen Instanzen äquivalent, wenn sie von einem externen Beobachter des Systems nicht unterschieden werden können. Aus dem regulären Ausdruck aller Instanzen des Systems mit ihren Interaktionen kann durch Anwendung eines Expansionstheorems die Menge aller beobachtungsäquivalenten Verhalten abgeleitet werden (Anmerkung: bei zustandsorientierten Verfahren wird diese Menge durch Erreichbarkeitsanalyse bestimmt). Diese Menge ist eine Kongruenzklasse über der Äquivalenz. Umgekehrt kann die Äuivalenz zweier Verhalten durch ein Reduktionstheorem bestimmt werden.
Asynchrones CCS enthält u. a. Erweiterungen für

- Spezifikation zeitabhängiger Protokolle
- Spezifikation von Protokollen, die auf komplexen Datenstrukturen basieren, indem die Menge der Kommunikationspartner dynamisch verändert werden kann.

CCS ist eine Methode zur Spezifikation und Verifikation von Kommunikationsprotokollen. Das **Einsatzspektrum** ist beschränkt auf das funktionale Verhalten der Kommunikation zwischen sicheren Instanzen; lokale Verarbeitung, fehlerhafte Kommunikation und zeitverbrauchende Instanzen können **nicht formuliert** werden. Allerdings können diese Nachteile durch CCS-Erweiterungen überwunden werden, indem z.B. eine Abbildung von Ereignisse auf Zeitattribute definiert und somit eine Leistungsanalyse ermöglicht wird /NOYE84/.

Die in der ISO-Welt entworfene Sprache LOTOS (Language of Temporal Ordering Specifications, /ISO83/) baut auf CCS-Grundkonzepten auf.

2.4.3 Spezifikation mit erweiterten endlichen Automaten

Auf dem Gebiet der Spezifikation und Verifikation von Protokollen und Diensten für schichtenorientierte verteilte Systeme sind die populärsten Ansätze System Development Language (SDL) in der CCITT-Welt und die Spezifikationssprache ESTELLE in der ISO-Welt. Hauptziel dieser Ansätze ist eine exakte, eindeutige, vollständige und implementierungsunabhängige Protokollspezifikation, die sowohl zur rechnergestützten Protokollimplementierung als auch -validierung wichtiger Protokolleigenschaften wie Sicherheit, Lebendigkeit, Terminierung, Erreichbarkeit verwendet werden kann.
Das ESTELLE-Grundkonzept geht aus von einer abstrakten Protokollmaschine, bestehend aus einer Menge durch 'Kanäle' verbundener 'Moduln'. Moduln repräsentieren Protokollinstanzen oder Komponenten der Protokollumgebung. Interaktion zwischen Moduln findet ausschließlich über Kanäle statt. Ein Kanal beschreibt die abstrakte Schnittstelle zwischen mindestens zwei Moduln. Sowohl asynchrone Kommunikation (Kanäle speichern Interaktionen in einer Warteschlange) als auch synchrone Kommunikation entsprechend dem rendez-vous-Konzept von CSP ist spezifizierbar. Entsprechend dem ISO-7-Schichtenmodell /ISO84a/ bestehen Interaktionen im Nachrichtenaustausch mit Hilfe eines Schichtenprotokolls zur Erbringung eines Dienstes für die nächsthöhere Schicht. Eine Modulspezifikation definiert das Verhalten des Moduls, das an dessen Eingabe- und Ausgabekanälen beobachtet werden kann. Ein Modul, der durch eine Interaktion eine Nachricht empfängt, führt zu einer atomaren Modul-Operation mit anschließendem Zustandsübergang, d.h. Ausgabe von Nachrichten an die Ausgabekanäle des Moduls. Ob eine Operation ausgeführt wird, kann vom Wert einer Vorbedingung abhängig gemacht werden. Einer Operation wird ein evtl. stochastischer Zeitverbrauch zugeordnet, was sowohl die Spezifikation zeitabhängiger Protokolle auch eine Untersuchung von Leistungsaspekten ermöglicht. Operationen können modulinterne Variablen verändern und Ausgabekanäle spezifizieren. Variablen reduzieren den Zustandsraum der Protokollmaschine beträchtlich, da Variablenwerte die möglichen Ausgabeinteraktionen beschränken können. Eine mit ESTELLE spezifizierte Protokollmaschine verhält sich i.allg. indeterministisch, da nebenläufige Zustandsübergänge und

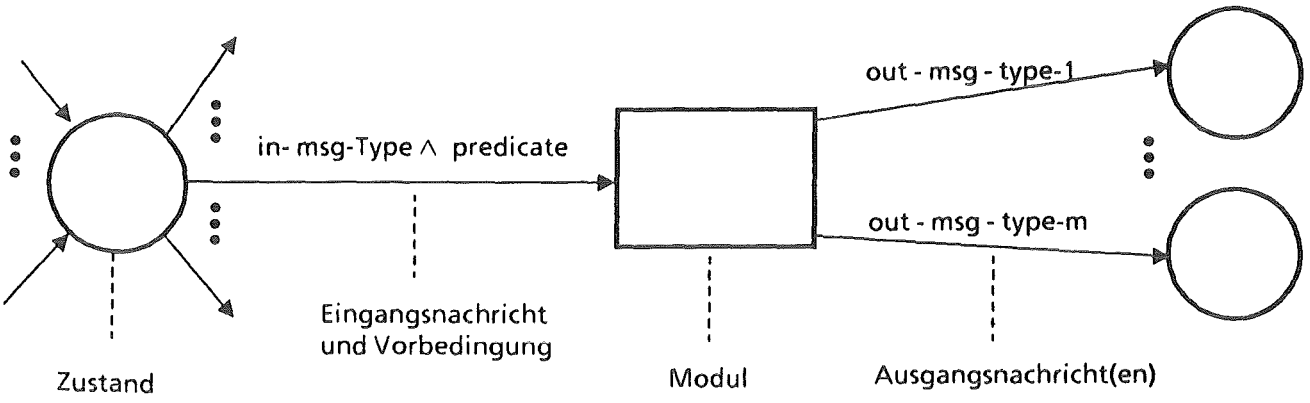


Abb. 2.3: Graphische Grundkonstrukte von ESTELLE

Interaktionen möglich sind.
Eine ESTELLE-Spezifikation kann sowohl durch eine lineare Notation mittels der algorithmischen Sprache Pascal als auch durch eine graphische Notation, erweitert um Deklaration von Variablen und Datenstrukturen, repräsentiert werden. Die graphischen Grundkonstrukte zeigt Abb. 2.3. (Die graphische Notation ist nicht Teil des ISO-Draft Proposals in /ISO85/.)
Werkzeuge zur Leistungsuntersuchung auf ESTELLE-Basis sind dem Autor nicht bekannt. Die verwandte Methode PODEx /BAUE84/ unterstützt eine Spezifikation von Zeitattributen mit anschließender Simulation zur Leistungsanalyse.

2.4.4 Analytische und simulative Untersuchung mit Warteschlangennetzen

Wir beschreiben das Software-Werkzeug COPE (Computer Performance Evaluator /BEGR82/) als ein typisches Beispiel der Anwendung analytischer und simulativer Bewertungsmethoden auf Warteschlangennetzen.
Ein COPE-Leistungsmodell eines Rechensystems wird in einer speziellen Spezifikationssprache formuliert. Falls das resultierende Netz einer speziellen Klasse exakt oder approximativ analysierbarer Warteschlangennetze angehört, werden netzspezifische Erwartungswerte von Leistungsgrößen, wie z.B. Warteschlangenlängen oder Auftragsdurchsatz analytisch ermittelt. Andernfalls wird automatisch ein Simulationsmodell generiert und ausgeführt.

Ein COPE-Modell besteht aus der Spezifikation von Systemarchitektur, Lastprofil sowie einer Ergebnisspezifikation. Die Systemspezifikation beschreibt die Hardware-Architektur als Warteschlangennetz. Ein Netz ist aus folgenden Grundkonstrukten (Netzknoten) aufgebaut:

- SERVER: aktives, zeitverbrauch-orientiertes Betriebsmittel
- RESOURCE: passives, speicherplatz-orientiertes Betriebsmittel
- POOL: Warteschlange auf Bedienung wartender Betriebsmittelanforderungen durch Prozesse, Jobs oder Transaktionen. Durch verschiedene POOL-Typen können unterschiedliche Scheduling-Strategien modelliert werden
- CONNECTOR: Zusammenführung mehrerer Auftragsflüsse bzw. Auswahl des nächsten benötigten Betriebsmittels

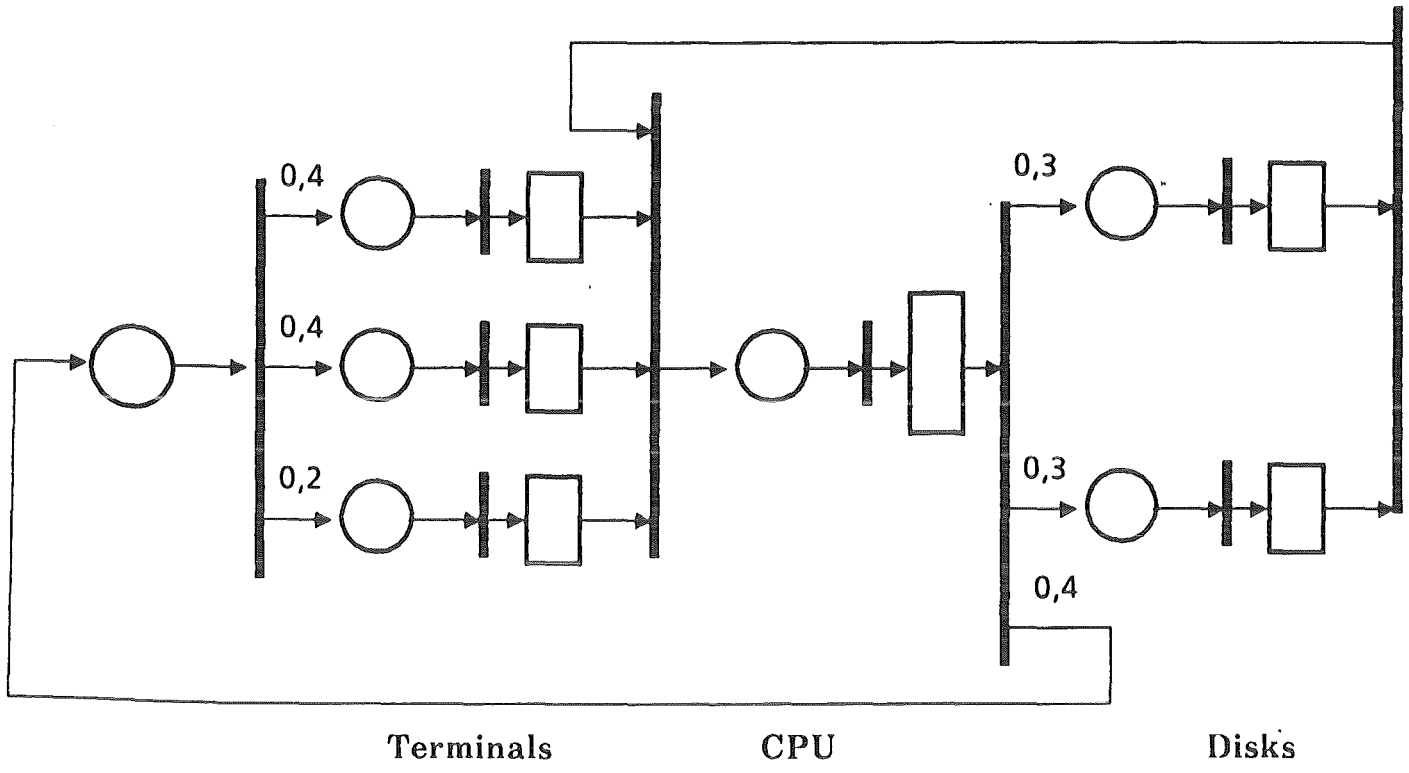
Knoten werden durch unidirektionale Kanten, den LINKs, zu einem Netz verbunden. Dabei ist eine strenge **Zweisortigkeit**, analog allen Methoden auf Petrinetzbasis zu wahren: SERVER- und RESOURCE-Knoten sind mit POOL-Knoten zu verbinden. Allerdings können dazwischen CONNECTOR-Knoten plaziert werden.

Die Lastspezifikation beschreibt den durch die Architektur vorgegebenen Auftragsbearbeitungsfluß genauer. Im einzelnen wird u.a. angegeben

- Deterministische / stochastische Auftrags-Ankunftsraten
- Deterministischer / stochastischer Zeitverbrauch für jede Betriebsmittel-Anforderung
- Transfer-Wahrscheinlichkeiten zur Auswahl der Folge-Anforderung an Konnektoren
- Evtl. nebenläufige Betriebsmittelanforderungen an Konnektoren

Die Ergebnisspezifikation zwingt den Modellierer, die Zielgrößen des Modells genau festzulegen. Zur simulativen Untersuchung können auch Start-/Stop-Bedingungen formuliert werden.

Als Beispiel ist in Abb. 2.4 ein einfaches COPE-Netz gezeigt. Modelliert ist ein geschlossenes, zentrales Rechensystem aus drei Terminals, einer Zentraleinheit und zwei Plattenlaufwerken; es sind ausschließlich diese aktiven Betriebsmittel modelliert. Last- und Ergebnisspezifikation sind separat zu spezifizieren; in der Abbildung sind lediglich die Übergangsraten angegeben. Nebenläufige



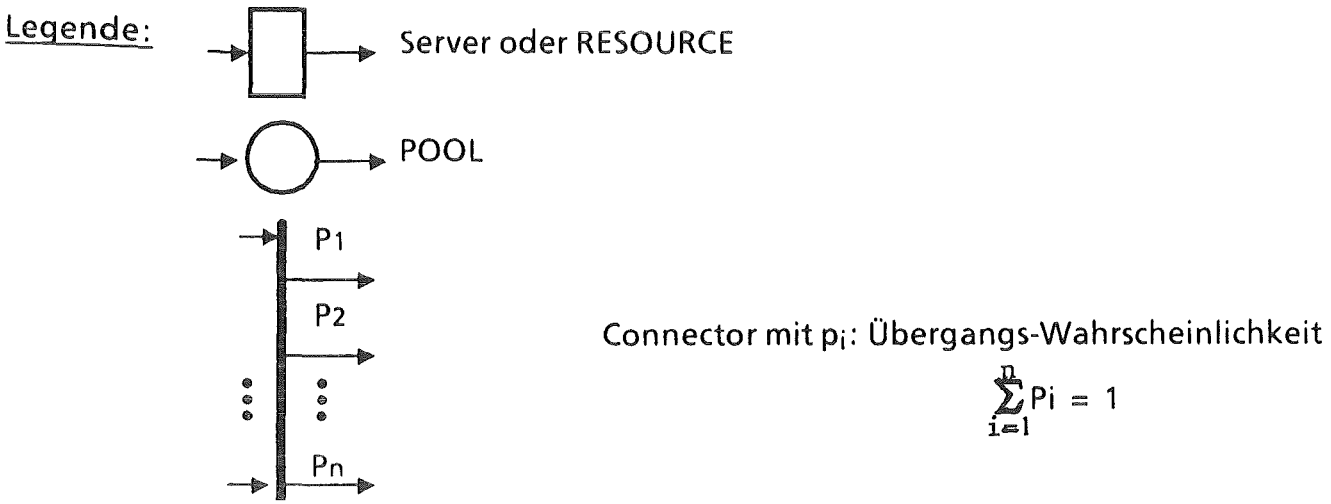


Abb. 2.4: Ein geschlossenes 'Central Server' - Netz in COPE

Anforderungen existieren in diesem Netz nicht. Folgende charakteristischen Beschränkungen ermöglichen eine exakte analytische Behandlung dieses Netzes:

- Exponentialverteilte Bedienraten von Terminals, CPU, Platten
- Feste Transferraten unabhängig von Auftragsprofil und von vorhergehenden Aufträgen
- Keine nebenläufige Bearbeitung durch CPU und Platte innerhalb

desselben Auftrags

Es fällt auf, daß hierarchische Modellierung (Dekomposition / Aggregierung) von dem Instrumentarium nicht unterstützt wird. Simulationsmodelle mit beliebigem Detaillierungsgrad sind nicht formulierbar.

2.4.5 Simulative Untersuchung mit zeitattributierten Petrinetzen

Petrinetze haben wegen ihrer streng formalen Basis in der Allgemeinen Netztheorie und ihrer graphischen Repräsentation eine zunehmende Verbreitung bei der Spezifikation und Gültigkeitsüberprüfung diskreter, insbesondere stark nebenläufiger Systeme - auch außerhalb der Informatik - gefunden. Bei der Anwendung des Konzepts der Stellen-/Transitionsnetze (ST-Netze, /PETE81, BEFE86/) auf komplexe Rechensysteme ergaben sich etliche Mängel des einfachen Konzepts:

- Eine starke Abhängigkeit der Modellkomplexität von der Problemgröße wie auch bei anderen Transitionssystemen (s.o.), die allerdings durch Verallgemeinerung der ST-Netze auf Prädikats-Transitionsnetze weitgehend behoben wurden;
- Fehlende Möglichkeiten zur quantitativen Untersuchung, da Zeitverbrauch und Spezifikation individueller Systemaufträge und Betriebsmittelanforderungen mit Zugriff auf wartende Aufträge nicht explizit vorgesehen sind.

Daher wurden zur Spezifikation, Leistungs- und Zuverlässigkeitsvorhersage komplexer Rechensysteme eine Reihe von verhaltensäquvalenten Interpretationen bzw. Verallgemeinerungen konzipiert, die mächtigere deskriptive Konstrukte beinhalten, z.B.:

- Zeitverbrauch an Transitionen bzw. Stellen
- Individuelle, strukturierte Marken
- Zugriffsstrategien auf strukturierte Marken in Stellen, z.B. FIFO-Strategie
- Transitionen als komplexe, den Markeninhalt verändernde Instanzen
- Transitionen mit selektivem Eingangs- und Ausgangs-Schaltverhalten in Abhängigkeit von booleschen Ausdrücken oder von Markentransfer-Wahrscheinlichkeiten bzw. von Markeninhalten
- Auflösung von Schaltkonflikten durch Priorisierung der nächsten zu schaltenden Transition oder durch Zuordnung von Auflösungsraten

(transfer rates).

Viele dieser Konstrukte (z.B. Priorisierung zusammen mit unbeschränkter Stellenkapazität) bewirken, daß die zugehörige Netzklasse die Modellierungsmächtigkeit von Turingmaschinen erreicht /PETE81/, sodaß die Entscheidbarkeit interessanter Modelleigenschaften wie z.B. Verklemmungsfreiheit nicht mehr gewährleistet ist.

Im folgenden werden einige rechnergestützte Methoden auf der Basis erweiterter, zeitattributierter Petrinetze (SPN) diskutiert, die speziell für quantitative Untersuchungen geeignet sind. Auf einen umfassenden Überblick über andere Netzvarianten verweisen wir auf /GODB83/ und /TPN85/.

In /RAZO84/ wird den Transitionen eines ST-Netzes ein **deterministischer** (d.h. nichtnegativer, konstanter) **Zeitverbrauch** zugeordnet. Diese Netzklasse ermöglicht eine **symbolische Leistungsbewertung**. Aus einem Netz wird ein Erreichbarkeitsgraph konstruiert, aus dem Durchsatz-Formeln für mittlere Leistung in Abhängigkeit von Transitionszeiten und Transferraten hergeleitet werden.

In /MCB84/ haben Transitionen eines ST-Netzes **exponentialverteilte Schaltzeiten**. Zusätzlich ist 'promptes' Schalten (Zeitverbrauch = 0) möglich. Wenn nun das Netz sicher ist, d.h. bei beliebigen Markierungen die Markenanzahl auf jedem Platz beschränkt bleibt, ist der Erreichbarkeitsgraph dieses Netzes endlich. Ein solcher endlicher Graph kann mit Hilfe der Matrizenalgebra automatisch erzeugt werden. Es wird gezeigt, daß dieser Graph isomorph zu einem endlichen, homogenen, zeitkontinuierlichen Markovprozeß ist. Damit ist die klassische Markov-Theorie anwendbar, um 'steady-state' Analysen der Leistungs- u/o Zuverlässigkeitsgrößen vorzunehmen.
Generell ist für alle Erweiterungen der ST-Netzklasse im Hinblick auf Zeitverbrauch anzumerken:

- Allein die **statische Netzstruktur** reicht aus, um die Analyse vorzunehmen
- Die Ergebnisse der Analyse gelten -ausgehend von einer gegebenen Anfangsmarkierung- **für beliebige Ereignisfolgen** (Schaltfolgen), d.h. es wird eine **Verifikation** des Leistungs- u/o

Zuverlässigkeitsverhaltens erzielt

- Bei exponentialverteilten u/o deterministischen Schaltzeiten ist der Erreichbarkeitsgraph identisch mit dem Graph, der sich aus dem entsprechenden 'klassischen' Netz ohne Zeitverhalten ergibt. Damit sind alle Verfahren der Allgemeinen Netztheorie zur Untersuchung prädikativer Netzeigenschaften **unmittelbar anwendbar**.

Andererseits ist einschränkend zu bemerken:

- Markovanalyse ist nur auf endliche Markovprozesse, die aus sicheren Netzen erzeugt werden, anwendbar.
- Bei der Analyse von Markovprozessen kann schnell eine Zustandsexplosion auftreten, wobei einem Zustand in der Kette einer von der Anfangsmarkierung erreichbaren Netzmarkierung entspricht. Methoden zur 'eigenschafts-erhaltenden' Netzreduktion /GODB83, LEFA85/ schaffen hier nur bedingt eine Abhilfe. Zudem konservieren die bekannten Reduktionsmethoden nicht notwendig das Zeitverhalten eines Netzes.
- Bereits die verwendete 'low-level' ST-Netzdarstellung führt schnell zu sehr großen Netzen.
- Es ist nicht ersichtlich, wie symbolische Lösungsmethoden auf Netze mit stochastischem Zeitverbrauch angewendet werden können.
- Analyse liefert stets nur Erwartungswerte relevanter Zielgrößen, während oft deren statistische Verteilung erwünscht ist
- Eine große Klasse von Problemen kann durch ST-Netze nicht mit adäquatem Detaillierungsgrad nachgebildet werden, was sowohl das funktionale als auch das zeitliche Ssytemverhalten betrifft.

Die Beschränkung auf exponentialverteiltem Zeitverbrauch ist in der Netzvariante des Werkzeugs **HARP** (Hybrid Automated Reliability Predictor, /GTDS83/) aufgehoben. Zusätzlich können HARP-Transitionen selektiv schalten, indem Ausgabestellen (konstante) Transfer-Raten zugeordnet werden. HARP-Netze werden durch Simulation untersucht; exakte analytische Behandlung ist bei dieser Netzklasse nicht möglich. Interessant ist die Methodik der Zuverlässigkeitsauswertung in HARP:

(1) Fehlerereignisse werden mittels Markovprozessen modelliert und bezüglich charakteristischer Zuverlässigkeitskenngrößen analytisch ausgewertet, wie in der Zuverlässigkeitstheorie üblich (vergl. /SISW82, Kap.5/).

(2) Die wesentlich komplexeren Modelle der Recovery-, Restart- und Rekonfigurationsalgorithmen zur Fehlererkennung, -diagnose und -behebung werden als HARP-Netz spezifiziert. Fehlerbehebungsraten und -zeiten werden aus dem Netz durch Simulation ermittelt.

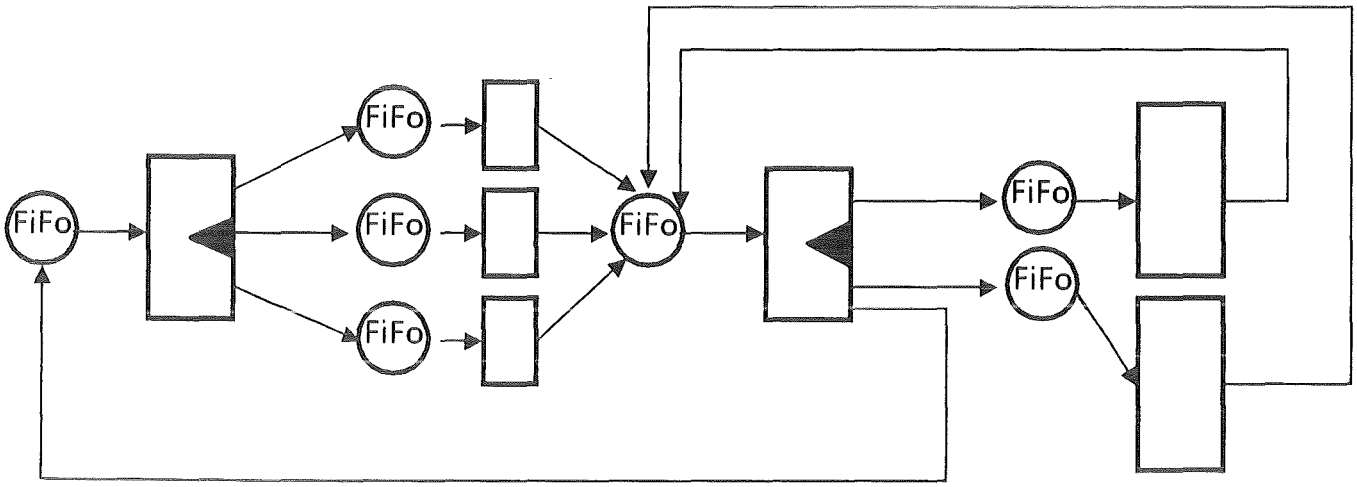
(3) Die Ergebnisse aus (1) und (2) werden in einer aggregierten Kenngröße eingebracht.

Die Gefahr einer Zustandsexplosion wird durch Abkehr von den anonymen Marken der Stellen/Transitionsnetze, verbunden mit algorithmischer Spezifikation der Transformation strukturierter Marken in Transitionen, sowie durch Einführung von Stellenzugriffsverfahren (wie etwa FIFO, LIFO) und markenabhängigen Markenfluß-Entscheidungen stark vermindert. Durch diese sehr nützlichen Konstrukte zeichnen sich z.B. die **Funktionsnetze** (FN) /GODB83/ aus. Abb. 2.5 zeigt ein einfaches graphisches FN-Modell eines Rechensystems, das isomorph zum Warteschlangennetz aus Abb. 2.4 ist. Allerdings können durch Spezifikation komplexer Transitionsalgorithmen -im Gegensatz zu analytisch behandelbaren Warteschlangennetzen- beliebige Auftragsklassen und eine davon abhängige Bearbeitungsfolge auf Recheneinheit und den Speichereinheiten spezifiziert werden (im Bild nicht gezeigt). Das wird ermöglicht durch die Einführung selektiv schaltender Transitionen, die **markenabhängige** Transfer-Entscheidungen treffen können, während durch die Spezifikation fester Transfer-Raten wie in COPE das gewünschte Verhalten nur annähernd beschrieben werden kann. Mit diesen Konstrukten können auch sehr detaillierte Modelle mit Funktionsnetzen formuliert und per Simulation untersucht werden, während klassische Leistungsanalysemethoden nicht a priori anwendbar sind. Wir werden auf dieses Problem in Kapitel 6 zurückkommen.

2.5 Bewertung der Methoden

Das Spektrum der beschriebenen Methoden reicht von algorithmischen Testverfahren über Kommunikationsalgebren bis hin zu stochastischen, graphisch repräsentierten Transitionssystemen, mit steigendem Abstraktionsgrad.

Verteilte Testbettsysteme liefern bei der Bewertung komplexer verteilter Systeme sicherlich die besten Ergebnisse. Ihr Einsatz



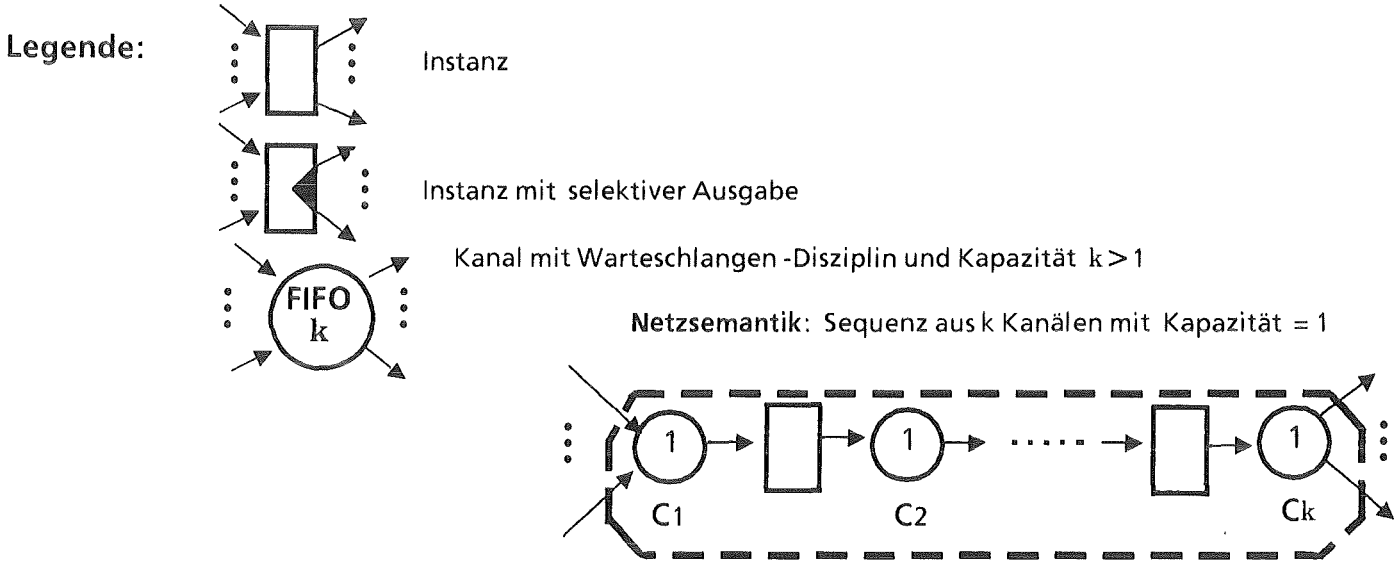


Abb. 2.5: Das Warteschlangenmodell aus Abb. 2.4 als Funktionsnetz

erfordert aber ein funktionsfähiges Rechnernetz; ein die Spezifikation unterstützendes formales Basiskonzept ist nicht vorhanden.

Algebraische Spezifikationssprachen eignen sich hauptsächlich zur Verifikation von Protokolldiensten und -implementierungen. Eine graphische Darstellung für solche Spezifikationen fehlt bisher.

Hierarchische Modellierung ist nicht vorgesehen. Die Beschränkung auf das funktionale Systemverhalten macht den Einsatz für Leistungsvorhersagen fraglich; durch geeignete Erweiterungen ist aber das zeitliche Systemverhalten spezifizierbar. Damit sind algebraische Sprachen prinzipiell zur Leistungsanalyse geeignet, auch wenn dieser Weg bisher kaum beschritten worden ist.

Eine ausgereifte Theorie der **hybriden Modellierung** existiert noch nicht, zumal schon von einer hierarchischen 'top-down' Zerlegung keine allgemeingültige Konstruktionslehre zu erwarten ist /MSS83/, auch wenn für bestimmte schichtenorientierte Modelle verteilter Systeme eine strukturelle Zerlegung entsprechend den Konzepten aus /ISO84a/ naheliegt. Zudem ist auch keine Fehlerabschätzmethode für hybride Techniken bekannt /BEIL81/. Insgesamt kann noch keine schlüssige Aussage bezüglich der Anwendbarkeit dieser Techniken gemacht werden.

Alle **stochastischen Transitionssysteme** basieren auf einem einheitlichen, formalen Grundkonzept. Daher sind viele der darauf basierenden Methoden äquivalent, d.h. Modelle einer Methode können in verhaltensgleiche und oft sogar isomorphe Modelle anderer Methoden transformiert werden. Hierzu kann auf die Zweisortigkeit von Elementen der graphischen Darstellung von Erweiterten Automaten, Warteschlangennetzen und Erweiterten Petrinetzen verwiesen werden, die sich in den Abbildungen 2.3 bis 2.5 zeigt. Stochastische Transitionssysteme erscheinen für unsere Modellierungszwecke noch am geeignetsten zu sein, da viele der darauf basierenden Methoden die oben definierten allgemeinen Anforderungen gut erfüllen, indem sie

- eine graphische Modellierungssprache besitzen
- ein uniformes Beschreibungsmittel sowohl für die Beschreibung von Hardware, Software und Datenstrukturen, als auch von Spezifikationsmodell und Bewertungsmodell bieten
- Hierarchische Modellierung unterstützen
- zu relativ kompakten Modellen führen
- Rechnerunterstützung zur Spezifikation und Bewertung aufweisen

Jedoch erfüllen keine der besprochenen 'höheren' Methoden ESTELLE, Computer Performance Evaluator (COPE) und Funktionsnetze (FN) alle speziellen Anforderungen an Methoden zur Bewertung komplexer

verteilter, fehlertoleranter Rechensysteme:

- Algorithmische Transitions-Spezifikation: in ESTELLE, FN
- Unterscheidbare, strukturierte Marken: in FN
- Flexible Transitions-Schaltregel: Selektives Schalten in FN, Schalt-Vorbedingung in ESTELLE
- Explizite Spezifikation von und unterschiedliche Anforderungsstrategien für Systemresourcen: in COPE
- Rechnerunterstützung zur Modellerstellung und -untersuchung: in COPE, FN
- Modellierung von Betriebsmittel-Ausfall und -Wiederanlauf: in keiner der Methoden vorgesehen

Die **Folgerung** aus dem beschriebenen Stand der Technik ist, daß keine der beschriebenen Methoden alle Anforderungen zur Modellierung verteilter fehlertoleranter Rechensysteme vollständig erfüllt. Als geeignetste Methode wurden **Funktionsnetze** bewertet. Deren Konzeption und Realisierung wird allerdings noch konzeptionell und bezüglich der rechnergestützten Experimentierumgebung zu erweitern sein. Die Erweiterungen sind Inhalt des 6. Kapitels der Arbeit.

3 GRUNDLAGEN VERTEILTER DATENBANKVERWALTUNGSSYSTEME

In diesem Kapitel sollen die Grundlagen zum Verständnis verteilter Datenbank-Verwaltungssysteme (VDBS) gelegt werden. Dazu werden entsprechend dem Stand der Technik Terminologie, Architektur und ausgewählte Protokolle eingeführt, um eine einheitliche Basis zur Modellierung und quantitativen Bewertung unterschiedlicher VDBS zu schaffen. Modellierung und Bewertung folgen in den nächsten Kapiteln.

3.1 Transaktionen in zentralisierten Datenbanksystemen

3.1.1 Transaktions- und Datenbankmodell

Eine **Datenbank** besteht aus einer Menge voneinander unabhängiger **Dateneinheiten**:

$$DB := \{DE_1, \ldots, DE_E\}$$

Diese sind Teil des **Konzeptionellen Schemas** entsprechend dem 3-Schema-Modell /ANSI78/. Basiert dieses Schema auf dem Relationenmodell, dann entspricht einer Dateneinheit eine Subrelation, d.h. dem Teil einer Relation, der nach einer bestimmten horizontalen u/o vertikalen Partitionierung entstanden ist. Man nennt eine solche Subrelation auch **Fragment**. Eine Dateneinheit kann auch als Tupel (Bezeichner, Wert) angesehen werden. Daraus ergibt sich der (aktuelle) **Datenbankzustand** als (aktuelle) Abbildung aller Bezeichner auf Werte. Ein Datenbankzustand ist **korrekt**, wenn er die im konzeptionellen Schema spezifizierten (logischen) **Konsistenzbedingungen** erfüllt.

Benutzer eines Datenbanksystems erwarten, daß alle DB-Operationen **konsistenzerhaltend** sind. Eine **Transaktion** ist eine Folge von DB-Operationen, die als Einheit betrachtet konsistenzerhaltend sind /EGLT76/. D.h. eine Transaktion ist eine Abbildung zwischen konsistenten DB-Zuständen, sodaß entweder alle Operationen der Transaktion auf der Datenbank ausgeführt werden, oder gar keine. Ist der initiale Datenbankzustand konsistent, und werden dann zustandsverändernde Transaktionen ausgeführt, dann folgt aus der Definition:

- Jede Transaktion findet einen konsistenten Zustand vor;

- Jede Transaktion hinterläßt einen konsistenten Zustand.

Technisch gesehen, besteht eine Transaktion aus einer bzw. mehreren Operationen, die in einer Datenmanipulationssprache (DML) formuliert werden. Diese Operationen kann man grob in **Leseoperationen** und **Schreiboperationen** einteilen. Die Menge der dabei zu lesenden bzw. zu verändernden Dateneinheiten wird als **Lesemenge** R(T) bzw. als **Schreibmenge** W(T) der Transaktion T bezeichnet; deren Vereinigung ist die **Basismenge** BS(T). Bei **Lesetransaktionen** besteht die Basismenge nur aus der Lesemenge; andernfalls spricht man von **Schreibtransaktionen**.

Jede Transaktion durchläuft in einem DB-System eine Folge von Transaktions-Zuständen:

(a) Eine Transaktion wird auf der Endbenutzerschnittstelle initiiert entweder explizit als abgeschlossene Folge von DML-Kommandos oder implizit als Teil eines Anwendungsprogramms. Nach Bearbeitungsbeginn der Transaktion durch das System ist diese im Zustand **offen** (interaktive Transaktionen werden nicht betrachtet).

(a1) Zustand = **offen/aktiv**: Die Bearbeitung dauert noch an,

(a2) Zustand = **offen/blockiert**: eine offene Transaktion kann wegen einem Ausfall einer Systemkomponente nicht beendet werden, d.h. die Bearbeitung wird bis zur Reparatur der Komponente verzögert. Dieser Zustand ist höchst unerwünscht, da er die **Verfügbarkeit** des Systems einschränkt.

(b) Zustand = **terminiert**: Eine Transaktion ist beendet, wenn der Benutzer von ihrem Erfolg (Endzustand = COMMIT) bzw. ihrem Mißerfolg (Endzustand = ABORT) durch Mitteilung ihrer Ergebnisse erfährt. Im Falle einer Schreibtransaktion enthält der Datenbankzustand nach deren Beendigung alle von der Transaktion beabsichtigten Änderungen (COMMIT), oder der Zustand wurde nicht verändert (ABORT) gegenüber dem vor Beginn der Bearbeitung dieser Transaktion.

3.1.2 Korrektheit der Transaktionsverarbeitung

Das oben definierte Transaktionsmodell stellt eine idealisierte Sicht dar, da es in realen Systemen unmöglich ist, stets 100-prozentige

Konsistenz zu bewahren, obwohl für die meisten Anwendungen eine inkonsistente Datenbank nicht tolerierbar ist. Das gilt auch für später entdeckte Inkonsistenzen, da dann bereits terminierte Transaktionen inkorrekte Resultate geliefert haben können. Erwünscht ist also **korrekte Transaktionsverarbeitung**. Dieser Begriff kann für zentralisierte Systeme wie folgt definiert werden (vergleiche Abb. 3.1).

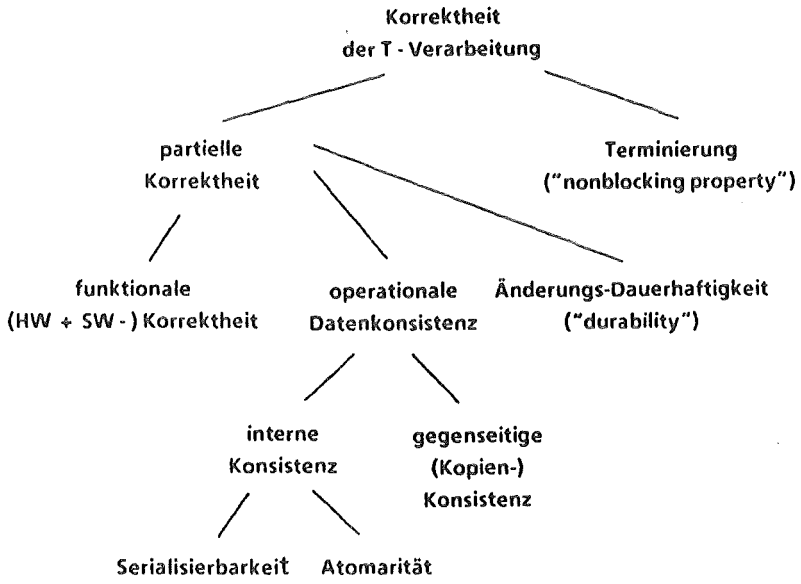


Abb. 3.1: Korrektheit und Terminierung transaktionsverarbeitender Systeme

(1) **Partielle Korrektheit:**

(1.1) System-Hardware und -Software sind entsprechend ihrer Entwurfsspezifikation korrekt realisiert.

(1.2) **Operationale Datenkonsistenz:** im zentralisierten System entspricht das dem Begriff **interne Konsistenz**, bestehend aus

- **Atomarität:** Trotz möglicher Systemfehler werden alle Änderungen einer Schreibtransaktion vollständig oder gar nicht in die Datenbank übernommen.
- **(Konflikt-)Serialisierbarkeit:** Trotz nebenläufig initialisierter u/o bearbeiteter Transaktionen bleibt die

Datenbank in einem log. konsistenten Zustand.

Prinzipiell können Inkonsistenzen nur dann auftreten, wenn

- Offene Schreibtransaktionen nebenläufig mit anderen Transaktionen im System vorhanden sind;
- Operationen verschiedener Transaktionen nebenläufig ausgeführt werden;
- Bei einem Rechner- oder Massenspeicherausfall offene Transaktionen nicht vollständig ausgeführt werden.

Die DBVS-Komponente, die für die Einhaltung korrekter (sprich: konsistenzerhaltender) Ausführungsreihenfolgen zuständig ist, nennt man **Transaktions-Scheduler** /GRAY78/. Ein Scheduler soll in der Lage sein, die Ausführung einzelner Operation ggf. zu verzögern, um die Konsistenz zu erhalten. Das dabei zugrundeliegende Konzept bezeichnet man als **Concurrency Control**. Der Scheduler erhält als Input eine Menge von Transaktionen und erzeugt daraus eine verzahnte Folge von DB-Operationen, die man als **Transaktionsschedule** bezeichnet. Nicht alle Schedules sind korrekt. Inkonsistenzen können bei Auftreten von Konflikten auftreten:

Zwei aufeinanderfolgende Operationen eines Schedules stehen im **Konflikt** <=>

- beide sind Schreiboperationen auf derselben Dateneinheit, oder
- es handelt sich um eine Lese- und eine Schreiboperation auf derselben Dateneinheit.

Ein korrekter Scheduler hat die Aufgabe, im Konflikt stehende Operationen in eine sinnvolle Ausführungsreihenfolge zu bringen, sodaß die daraus resultierende verzahnte Ausführung von Transaktionen die gleiche Folge konsistenter DB-Zustände erzeugt wie **irgendeine** serielle Ausführung. Daher garantiert ein Scheduler mit dieser Eigenschaft Konflikt-Serialisierbarkeit. Aus Benutzersicht bedeutet das, daß die Effekte nebenläufiger Transaktionen diesem nicht sichtbar werden.

(1.3) **Änderungs-Dauerhaftigkeit**, "durability" /GRAY81/:
Die Semantik erfolgreich terminierter Schreibtransaktionen (d.h. mit Endzustand = COMMIT) impliziert, daß ausgeführte Änderungen stabil innerhalb der Datenbank sind, d.h. daß diese

die einzig erwünschten sind. Es soll also niemals notwendig werden, diese Änderungen nochmal einzubringen.

Die Gefahr zur Zerstörung bereits eingebrachter bzw. zum Einbringen unerwünschter Änderungen besteht bei Fehlern des Massenspeichers, auf dem die Datenbasis gespeichert ist. Als Abhilfe werden von der DBVS-Logging-Komponente Daten redundant geführt, mit deren Hilfe ein konsistenter Datenbasiszustand restauriert werden kann. Durch Realisierung eines **transaktionsorientierten Loggingkonzepts** kann sogar stets der aktuellste Zustand restauriert werden, solange keine schwerwiegenden Massenspeicherausfälle auftreten (die wir nicht weiter betrachten).

(2) **Terminierung**: Trotz dem Auftreten von Fehlern terminiert jede Transaktion unabhängig von Reparaturzeiten fehlerhafter Komponenten.

Diese Forderung ist in zentralisierten Systemen nicht durchsetzbar, da ein einziger CPU-Ausfall zur Blockierung aller offenen Transaktionen im System führt.

3.1.3 Beispiele von Konsistenzproblemen

Gegeben sei ein elektronischer Bankschalter, bei dem Kunden von ihren Giro- und Sparkonten durch evtl. nebenläufige Eingabe von Transaktionen Geld abheben, zwischen den Konten transferieren, und Informationen über die Kontostände abrufen können. Die Kontostände eines Girokontos G und eines Sparkontos S seien in der zentralen Datenbasis gespeichert mit dem Anfangswert S=G=1000. Wir betrachten jeweils zwei nebenläufige Transaktionen T_1, T_2 und T_1, T_3 auf dieser Datenbasis.

T1:	T2:	T3:
r(S)	r(S)	r(G)
sub(S, 100)	sub(S, 200)	r(S)
w(S)	w(S)	print(G+S)
r(G)		
add(G, 100)		
w(G)		

Transaktion T_1 liest Konto S, hebt davon DM 100 ab, liest Konto S und addiert DM 100 auf S. Transaktion T_2 liest Konto S und hebt davon DM 200 ab. Transaktion T_3 liest die Konten G und S und gibt deren Summe aus.

Ohne Vorsorgemaßnahmen könnte nun eine verzahnte Abarbeitung dieser Transaktionen zu folgenden Konsistenzverletzungen führen.

- **Verlorene DB-Änderung:** Folgende verzahnte Abarbeitungsfolge der Transaktionen T_1, T_2 ist inkorrekt:

  $r_1(S)$, $sub_1(S, 100)$, $w_1(S)$,
  $r_2(S)$, $sub_2(S, 200)$, $w_2(S)$,
  $w_1(S)$, $r_1(G)$, $add_1(G, 100)$, $w_1(G)$
  (Die Operationen op_i wird von der Transaktion T_i ausgeführt)

  Das korrekte Ergebnis wäre S=700, G=1100, das sich bei irgendeiner sequentiellen Ausführung beider Transaktionen ergibt. Hingegen liefert die obige Folge das Ergebnis S=900, G=1100, d.h. die Änderungsoperation der Transaktion T_2 ist verlorengegangen.

- **Inkonsistentes DB-Lesen:** Die Abarbeitungsfolge für die Transaktionen T_1, T_3

  $r_1(S)$, $sub_1(S, 100)$, $w_1(S)$,
  $r_3(S)$, $r_3(G)$, $print_3(S+G)$,
  $w_1(S)$, $r_1(G)$, $add_1(G, 100)$, $w_1(G)$

  ist inkorrekt, da T_3 die Summe 900+1000 ausgibt, das korrekte Ergebnis aber 900+1100 ist. Ursache dieser Anomalie ist, daß T_3 einen durch T_2 verursachten inkonsistenten Zwischenzustand der Datenbasis liest.

Eine andere Form der Konsistenzverletzung bilden **unvollständige DB-Änderungen**. Fällt z.B. der zentrale Rechner während der T_1-Bearbeitung vor der letzten Operation aus, also vor dem Update des Girokontos, dann ist die Datenbasis ebenfalls inkonsistent, da T_1 unvollständig ausgeführt wurde und jede Transaktion per definitionem nur als ganzes konsistenzerhaltend ist.

3.1.4 Konzepte zur Realisierung korrekter Transaktionsverarbeitung

Die gebräuchlichste Realisierung eines Transaktions-Schedulers basiert auf einem **Sperrprotokoll**: Vor einem Lese- bzw. Schreibzugriff muß die betreffende Dateneinheit durch eine Lese- bzw. Schreibsperre vor inkonsistentem Zugriff geschützt werden. Beim einfachsten Verfahren kann eine Dateneinheit zu einem Zeitpunkt mit mehreren Lese-, aber mit höchstens einer Schreibsperre belegt sein.
Korrekte Schedules erhält man z.B. durch das **2-Phasen-Sperrprotokoll** (2PL) /EGLT76/: Eine Transaktion darf eine Dateneinheit erst dann wieder freigeben (d.h. entsperren), wenn sie alle benötigten Sperren gesetzt hat. Bei der verbreiteten statischen 2PL-Variante werden alle Sperren zu Beginn der Bearbeitung einer Transaktion angefordert. Das setzt voraus, daß die Transaktion als Einheit dem Scheduler übergeben wird (wie oben angenommen). Der Vorteil dieser Variante besteht in der Verhinderung von Deadlocks und damit von aufwendigen Maßnahmen zur Deadlockerkennung.
Eine Vielzahl anderer Concurrency-Control-Verfahren ist in den letzten Jahren entworfen worden; wir beschränken uns aber auf die genannten Sperrverfahren.

3.2 Transaktionen in verteilten Datenbanksystemen

3.2.1 Transaktions- und Datenbankmodell

Die Erweiterung vom zentralisierten auf den verteilten Fall basieren im wesentlichen darauf, daß Fragmente des Konzeptionellen Schemas auf einem oder mehreren Rechnerknoten gespeichert sein können. Wird ein Fragment auf verschiedenen Knoten redundant gespeichert, dann bezeichnen wir ein gespeichertes Fragment als **Kopie**, die Zuordnung Fragmente --> Knoten als **Datenreplikation**. In einer verteilten Datenbank mit E Dateneinheiten auf N Rechnerknoten ($E \gg N$) habe jede Dateneinheit C Kopien, wobei $1 \leq C \leq N$. Jede Kopie eines Fragments wird zum Zweck der Verfügbarkeitssteigerung einem unterschiedlichen Rechnerknoten zugeordnet. Wir setzen voraus, daß Kopien 1:1 auf Dateien abgebildet werden, sodaß eine Kopie gleichzeitig die Einheit zur Datenreplikation, zum -zugriff und zur -synchronisation ist. Dieses einfache Modell war die Grundlage vieler Dissertationen im

Bereich VDBS-Modellierung; es ist im VDBS POREL implementiert worden /WANE84/.

Daraus ergibt sich ein Transaktionsmodell aus Benutzer- und Systemsicht: Aus Benutzersicht sind "logische" Transaktionen weiterhin eine Folge von Operationen auf Fragmenten, während aus Systemsicht der Speicherort der Fragmente relevant ist: "physische" Transaktionen operieren auf Kopien. D.h. aus Benutzersicht soll die Datenreplikation nicht sichtbar sein, so als wäre die Datenbank nicht verteilt. Daraus ergibt sich eine weitere VDB-spezifische Bedingung zur DB-Konsistenzerhaltung; die oben definierten Eigenschaften einer korekten Transaktionsverarbeitung gelten auch im verteilten Fall (vergl. Abb. 3.1):

> **Gegenseitige (Kopien-) Konsistenz** /BERN83b/. Eine Folge von Transaktionen auf eine replizierte Datenbank führt zu derselben Folge von Datenbank-Zuständen wie **irgendeine** Bearbeitungsfolge dieser Transaktionen in der zugehörigen zentralisierten Datenbank.

Diese Definition bedeutet aus Benutzersicht, daß Datenlokation und -replikation dem Benutzer nicht sichtbar sind.

3.2.2 Beispiele von Konsistenzproblemen

Die Beispiele in Kap. 3.1.3 zeigten, wie in zentralisierten DBVS inkorrekte Transaktionschedules bzw. ein Systemausfall zu inkonsistenten Datenbasiszuständen führen können. In verteilten Systemen kommen neue Probleme hinzu. Im folgenden wird demonstriert, daß in einem verteilten System trotz korrekter Schedules lokaler Concurrency-Control-Mechanismen Inkonsistenzen auftreten können. Entsprechend der Problemursache unterscheiden wir

(1) **Indeterministisches Rechnernetzverhalten:** Wir betrachten wieder die Bankanwendung wie in Kapitel 3.1.3. Transaktion T_1, eingegeben auf Knoten A, erhöht das Sparkonto S um die jährlichen Sparzinsen von 5 Prozent (S:=S*1.05) Transaktion T_2, auch auf A initiiert, schreibt dem Konto S DM 100 gut (S:=S+100). Konto S ist in je einer Kopie auf Knoten A, B gespeichert. Der Kontostand

sei anfänglich auf beiden Knoten gleich, d.h. die Kopien sind gegenseitig konsistent. Betrachtet man die sequentiellen Abarbeitungsfolgen

T_1, T_2 auf Knoten A
T_2, T_1 auf Knoten B

dann ist der Kontostand beider Kopien danach nicht mehr identisch, da Multiplikation und Addition nichtkommutative Operationen sind. Diese Folge kann entstehen durch die Transaktionen

T1:	T2:
r(S)	r(S)
S := S•1.05	S := S+100
$w_A(S)$	$w_A(S)$
send(S --> B)	send(S --> B)

wobei die S-Ergebnisse auf Knoten B nicht in Sendereihenfolge ankommen (es handelt sich um einen Überholvorgang).

(2) **VDBS-Teilausfall:** Ausfälle von Knoten oder Leitungen im verteilten System stellen ein besonderes Problem dar, wenn man Kopienkonsistenz auf redundanten Datenbeständen bewahren möchte. Dazu zwei Beispiele:
Wir betrachten die Transaktion T_1 aus Kapitel 3.1.3. Die Konten-Datenbasis sei auf drei Rechnerknoten A,B,C repliziert. Angenommen, A fällt nach dem Abheben von DM 100 vom Sparkonto, aber vor der Gutschrift dieses Betrags auf das Girokonto aus; auf den Knoten B und C wird die Transaktion vollständig ausgeführt. Offensichtlich ist damit die Kopienkonsistenz verletzt.
Abschließend betrachten wir noch Konsistenzverletzung als Folge einer Rechnernetzpartitionierung. Eine auf Knoten A initiierte Transaktion T_4 soll DM 700 vom Sparkonto S abheben. Eine auf Knoten B eingegebene Transaktion T_5 soll DM 500 vom selben Konto abheben. Kopien von S existieren auf beiden Knoten mit dem Anfangswert 1000, d.h. Kopienkonsistenz ist noch gewahrt. Es gelte die semantische Konsistenzbedingung $S \geq 0$, d.h. eine

Kontoüberziehung ist nicht erlaubt.
Angenommen, die Knoten A und B werden durch Leitungsausfall getrennt, d.h. sie liegen in zwei aktiven Partitionen des Rechnernetzes, die nicht miteinander kommunizieren können. Würde T_4 auf A und T_5 auf B ausgeführt, dann wäre das Konto 'global' gesehen um DM 200 überzogen, ohne daß diese Konsistenzverletzung 'lokal' auf A und B erkennbar wäre. Auf dieses Partitionierungsproblem werden wir noch näher eingehen.

3.2.3 Protokolle zur Realisierung korrekter Transaktionsverarbeitung

Wie die obigen Beispiele zeigen, ist korrekte Transaktionsverarbeitung in verteilten Systemen viel schwieriger zu erreichen, bedingt durch komplexere Fehlersituationen und indeterministisches Rechnernetzverhalten. Wir diskutieren nun die Korrektheitseigenschaften und geben Realisierungsmöglichkeiten an.

(A) Serialisierbarkeit

Das 2-Phasen-Sperrprotokoll für zentralisierte DBVS führt prinzipiell auch im verteilten Fall zur Serialisierbarkeit, sofern man Ausfälle nicht betrachtet. Bei den verteilten Sperrprotokollen werden im logischen Sinne ganze Dateneinheiten gesperrt. Der Scheduler ist i.d.R. als ein verteiltes Programm realisiert, der Sperren auf Kopien setzt. Die Sperrprotokolle unterscheiden sich im Kopienbehandlungskonzept. Allen diesen Protokollen ist aber folgende Eigenschaft von Transaktions-Schedules gemeinsam, die Serialisierbarkeit auch im verteilten Fall garantiert:

> Zu keinem Zeitpunkt sind zwei Kopien einer Dateneinheit durch Schreibsperren unterschiedlicher Transaktionen belegt.

Die folgenden drei 2PL-Protokolle realisieren diese Eigenschaft. (Eine vollständige, formale Spezifikation dieser Protokolle folgt in Kapitel 7.) Wir betrachten dazu Kopiensperren einer einzelnen Dateneinheit, die gelesen oder verändert werden soll:

- **Basis-2PL** (PROT_ALL):
  Ziel dieses Protokolls ist es, Lesezugriffe auf Kosten der Verfügbarkeit bei Updates zu optimieren.

Zum Lesen wird **irgendeine** Kopie gesperrt und kann dann gelesen werden. Zum Schreiben sind **alle** Kopien zu sperren; die Schreiboperationen sind auf allen auszuführen. Eine Variante dieses Protokolls ist in den Systemen R* bzw. VDN/REFLEX implementiert (/WLS83/ bzw. /MUNZ79/).

- **Primärkopien-2PL** (PROT_PRIM):
  Ziel dieses Protokolls ist es, sowohl Lesezugriffe 'billig' zu machen, als auch die Systemverfügbarkeit trotz Ausfällen für Lese- und Schreibzugriffe zu gewährleisten.
  Für jede Dateneinheit existiert ein bevorzugter Rechnerknoten, auf welchem die Dateneinheit als **Primärkopie** gespeichert wird. Zum Lesen wird diese Primärkopie gesperrt; daraufhin kann eine **beliebige** Kopie gelesen werden (aus Performance-Gründen wird die auf dem Heimatknoten der Transaktion vorhandene Kopie gewählt, wenn sie dort gespeichert ist). Zum Schreiben wird die Primärkopie gesperrt; die Änderungen sind anschließend auf allen weiteren Kopien auszuführen. Darauf basierende Protokolle sind z.B. in den VDBS POREL /WANE84/ und JASMIN /WILA84/ realisiert.

- **Mehrheitsentscheid-2PL** (PROT_MAJ):
  Ziel dieses Protokolls ist es, die Systemverfügbarkeit trotz Ausfällen für Lese- und Schreibzugriffe zu gewährleisten, was auf Kosten der Lesekosten geschieht. Das Einsatzspektrum läge also bei update-intensiven Anwendungsumgebungen mit hohen Verfügbarkeitsanforderungen.
  Zum Lesen ist mindestens eine **Mehrheit** der Kopien zu sperren. Für jede Kopie wird eine **Versionsnummer** geführt, sodaß eine Kopie mit der höchsten Versionsnummer innerhalb der Mehrheit gelesen werden kann. Zum Schreiben wird ebenfalls eine Mehrheit (bezüglich einer Dateneinheit) gesperrt; die Änderungen sind dann mindestens auf der Kopienmehrheit auszuführen, wobei Kopien, deren Versionsnummer kleiner sind als die maximale Versionsnummer innerhalb der Mehrheit, auf den Stand dieser aktuellsten Version "nachzufahren" sind. Eine vollständige Spezifikation dieses Protokolls wurde in /LEBR82/ veröffentlicht. Eine Realisierung eines ähnlichen Protokolls wird in /ELFL83/ beschrieben.

Die Korrektheit dieser Protokolle bezüglich der Serialisierbarkeit ist intuitiv einsichtig und wurde formal nachgewiesen /BERN83b/. Zusätzlich garantieren diese Protokolle Kopienkonsistenz (s.u.).

Wie in zentralisierten DBVS ist auch im verteilten Fall Deadlock-Verhinderung durch Prädeklaration von Sperren möglich. Bei der gängigsten Realisierung werden alle Sperranforderungen an jeweils einen Knoten geschickt und dessen Bestätigung der Auftragsausführung abgewartet, bevor die nächste Anforderung verschickt wird.

(B) **Atomarität**

In verteilten Systemen ist die Gefahr der Atomaritätsverletzung durch Schreibtransaktionen viel größer und erfordert spezielle Terminierungs-Protokolle, wie folgendes Beispiel demonstriert:

> Eine Transaktion T auf dem Heimatknoten HOME(T) soll zwei Kopien C1, C2 zweier Dateneinheiten DE1, DE2 ändern. HOME(T) fällt nach versenden der C1-Änderungsoperation aus, d.h. C2 wurde noch nicht geändert. Zu diesem Zeitpunkt ist die Datenbank inkonsistent. Jeder C2-Zugriff einer anderen Transaktion T' von deren Heimatknoten HOME(T') ≠ HOME(T) wäre inkorrekt.

Hingegen ist dieser Fall in zentralisierten Systemen nicht möglich, da während der HOME(T)-Ausfalldauer keine andere Transaktion auf Daten zugreifen kann.

Atomarität von VDB-Änderungen wird garantiert durch **2-Phasen-Commitprotokolle** (2PC) mit integriertem Restart-Protokoll (vergl. z.B. /MUNZ79, LEBR82, MOLI83, WANE84/). Ein Commitprotokoll ist ein Terminierungs-Protokoll für Schreibtransaktionen, das sowohl die atomare Propagierung von Änderungen als auch das Kopienentsperren beinhaltet. 2PC-Varianten sind abhängig vom gewählten Sperrverfahren. Wir betrachten daher stets **integrierte Transaktionsverarbeitungsprotokolle** (TVP), bestehend aus

- Sperrprotokoll,
- Commitprotokoll,
- Kopienabgleichsprotokoll,
- Restartprotokoll.

(C) **Kopienkonsistenz**

Zunächst ist klar, daß alle drei Protokolle Updates in **höchstens einer**

aktiven Partition des Rechnernetzes zulassen. Wie das Beispiel im letzten Kapitel zeigte, ist dies eine notwendige Bedingung für Kopienkonsistenz, wenn man Leitungsfehler tolerieren will.
Bei den Protokollen PROT_PRIM und PROT_MAJ ist ein Kopienabgleichsprotokoll erforderlich, um Kopienkonsistenz zu garantieren. Probleme ergeben sich z.B. dann, wenn eine "veraltete" Kopie gelesen wird, d.h. ein Lesezugriff stattfindet auf eine Dateneinheit, die von einer im Schedule vorangegangenen Transaktion verändert wurde, diese Änderung aber noch nicht auf allen Kopien vollzogen ist. Beim Protokoll PROT_ALL kann das gar nicht auftreten, da per definitionem **alle** Kopien bis zur Terminierung der Transaktion gesperrt sind.

(D) **Terminierung**

In verteilten Systemen stellt sich das Problem der **Transaktionsblockierung**: Eine blockierte Transaktion wartet auf die Reparatur beteiligter Knoten bzw. der Verbindung zu ihnen. Das ist der Preis der 2PC-Protokolle, da die in der 2. Commitphase befindlichen Transaktionen von ihren DM-Knoten nicht ohne mögliche Atomaritätsverletzung abgebrochen werden dürfen, wenn die Verbindung zum Knoten HOME(T) unterbrochen ist. Dieser Verlust der **Knotenautonomie** führt zu langfristig blockierten Daten und somit zur reduzierten VDBS-Verfügbarkeit. Verbesserungen sind erzielbar durch folgende 2PC-Varianten:

- DM/DM-Verhandlung der an der Ausführung einer Transaktion beteiligten Knoten in 2. Commit-Phase /MUNZ79, LEBR82/;
- 3-Phasen-Commit mit zusätzlichen Backup-Prozessen, die die Koordination der Transaktionsausführung übernehmen sollen, wenn der Heimatknoten einer Transaktion nicht verfügbar wird /BRLE82b/.

Backup-Prozesse werden im folgenden nicht betrachtet; die Protokoll-Komplexität stiege dadurch beträchtlich, ohne daß der Nutzen (Verfügbarkeitssteigerung) offensichtlich wäre.

3.3 Ein operationales Architekturmodell für verteilte Transaktionsverarbeitung

Im folgenden wird ein einfaches Modell der Architektur verteilter Transaktionsverarbeitungssysteme entworfen.

3.3.1 Die VDBS-Systemarchitektur

Die wesentlichen VDBS-Funktionen mit ihren Interaktionen zeigt Abb. 3.2.

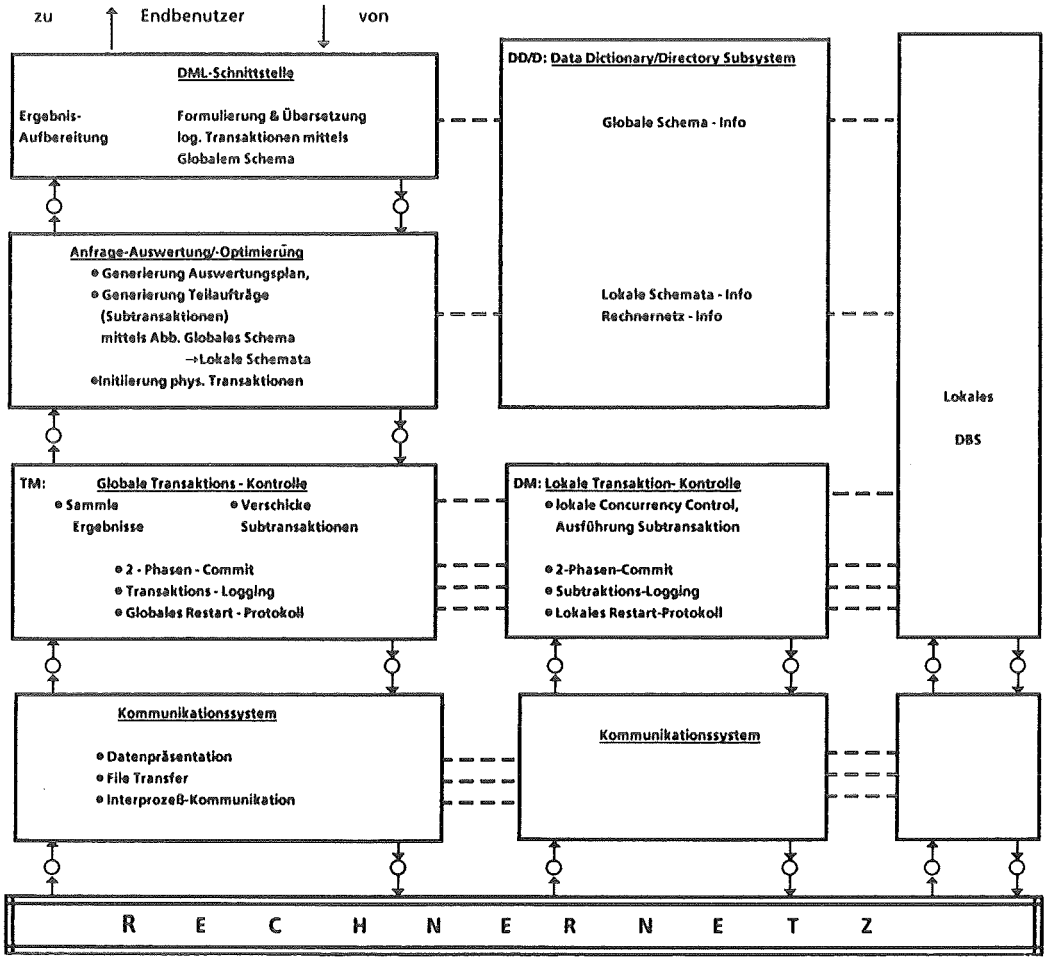
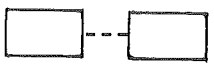


Abb. 3.2: Modell der verteilten Transaktionsverarbeitung


: Schichtenprotokoll

Eine ähnliche Systemarchitektur findet sich in /GLKE84/. Die Funktionsstruktur repräsentiert ein komplexes verteiltes Mehrschichtensystem, wobei eine logische Funktion nicht notwendig

durch einen Betriebssystem-Prozeß realisiert ist. Dem Modell liegen folgende Annahmen zugrunde:

- Die VDBS-Prozeßstruktur ist auf allen Knoten des Rechnernetzes identisch (**Homogenität**).
- Lokale und nichtlokale VDBS-Prozesse können nur über ein Nachrichtentransport-Subsystem miteinander kommunizieren.
- Transaktionen können an jedem Knoten des Rechnernetzes eingegeben werden.
- An jedem Knoten können mehrere Transaktionen nebenläufig aktiv sein.
- Transaktionen werden an der Endbenutzer-Schnittstelle des VDBS eingegeben. Eine direkte Interaktionsmöglichkeit mit dem lokal vorhandenen DBVS besteht nicht.

Abb. 3.2 repräsentiert zwei unterschiedliche Systemsichten: Horizontale Funktionsschnitte ergeben die VDBS-Schichtenstruktur. Vertikale Funktionsschnitte repräsentieren eine 3-Ebenen-Struktur, bestehend aus

A) Transaktionsvorverarbeitung ("User-Frontend"),
B) Transaktionsausführung und -überwachung als Schnittstelle zwischen A) und C),
C) Lokales DBVS ("Database-Backend").

Die einzelnen VDBS-Funktionen werden im folgenden durch die dynamische Bearbeitungsfolge einer Transaktion beschrieben.

3.3.2 Transaktions-Vorverarbeitung

An der Vorverarbeitung einer verteilten Transaktion sind VDBS-Instanzen zur Anfrageübersetzung, -Auswertung sowie -Informierung (Distributed Data Dictionary) beteiligt.

Benutzer-Schnittstelle: Eine Transaktion T wird am Heimatknoten HOME(T) in der Datenmanipulationssprache (DML) des VDBS formuliert. Die DML vermittelt eine uniforme, integrierte Sicht auf das **Globale Datenbankschema**. Zur Übersetzung von T wird das DD/D-Subsystem benötigt, um die Speicherungsorte der benötigten Daten zu ermitteln (s.u.). T wird nur dann bearbeitet, wenn die maximale Anzahl

nebenläufiger Transaktionen pro Knoten nicht überschritten ist.

Anfrageauswertung und -optimierung: Die "logische" Transaktion wird mittels des DD/D-Subsystems auf eine "physische" Transaktion abgebildet: das entspricht der Abbildung Globales Schema --> lokale Schemata, d.h. daß DML-Operationen auf Dateneinheiten in Lese- und Schreiboperationen auf Kopien transformiert werden. Zur Anfrageoptimierung werden zusätzlich Rechnernetz-Leistungsdaten der DD/D-Komponente benutzt.
Aus dem daraus generierten Auswertungsplan werden Teilaufträge an die Transaktions-Partnerknoten COHORT(T) abgeleitet, die die benötigten Kopien enthalten. Ein solcher Teilauftrag wird als **Subtransaktion** bezeichnet. In einer Subtransaktion werden alle DB-Zugriffsanforderungen an das lokale DBVS eines Knotens "gebündelt", um den Aufwand der Interrechner-Kommunikation zu minimieren. Welche Operationen eine Subtransaktion enthält, hängt im wesentlichen vom Kopienhaltungskonzept ab, das den Zeitpunkt des Kopienabgleichs bestimmt.
In die Teilaufträge werden alle Protokollinstanzen eingebaut, die zur lokalen Transaktionskontrolle gehören (s.u.).
Dann kann der Transaktions-ID der "physischen" Transaktion generiert werden. In diesem Modell besteht eine solche Transaktion somit aus folgenden Informationen:

- Transaktions-ID = T
- Heimatknoten = HOME(T)
- Partnerknotenmenge = COHORT(T) = $\{K_1, K_2, \ldots\}$
- Subtransaktionen = $\{SUB_1(T), SUB_2(T), \ldots\}$

Wie werden noch zeigen, daß die Transformation der logischen auf physische Transaktionen, insbesondere die Menge der Partnerknoten und die Struktur der Subtransaktionen, stark vom gewählten TVP abhängt.

Data Dictionary / Directory (DD/D): Dieses Subsystem ist eine Informationsinstanz, die "Metadaten" über das VDBS liefert, die im Rahmen des VDB-Entwurfs festgelegt werden und evtl. im laufenden Betrieb adaptiv anzupassen sind (vergl. den Übersichtsartikel /ALM82/ und die Dissertation /SCHE81/). Dazu gehören

- Kataloginformation über das Globale Schema; dieses entspricht in zentralierten Systemen dem Konzeptionellen Schema.
- Kataloginformation über Lokale Schemata incl. Partitionierung der Datenbasis in logische Fragmente (Dateneinheiten); Zuordnung und Replikation dieser Fragmente auf Knoten des Rechnernetzes; Leistungs- und Zuverlässigkeitskenndaten wie Knotenverfügbarkeit, topologische Rechnernetzstruktur, Leitungskapazitäten.

Diese Daten könnten selbst wieder als verteilte Datenbank aufgefaßt werden; DD/D-Zugriffe müßten dann als spezielle Transaktionen formuliert werden. Zur Modellvereinfachung machen wir aber folgende Annahmen:

- Alle DD/D-Daten sind vollredundant auf allen Rechnerknoten vorhanden,
- Transaktionen, die Schemaänderungen bewirken, kommen nicht vor oder sind so selten, daß sie sich nicht auf das Leistungsverhalten des Systems auswirken.

Aus diesen Annahmen folgt, daß alle Zugriffe auf DD/D-Daten lokal am Heimatknoten einer Transaktion erfolgen können, deren Kosten sich einfacher abschätzen lassen.

3.3.3 Transaktions-Ausführung und -überwachung

Die zentralen Instanzen eines VDBS bestehen in den globalen **(Transaktions-Manager**, TM) und lokalen Transaktionskontrollinstanzen **(Daten-Manager**, DM). Die TM-Instanz überwacht am Heimatknoten einer Transaktion deren korrekte, konsistenzerhaltende Bearbeitung. DM-Instanzen sind an allen solchen Rechnerknoten in der Bearbeitung beteiligt, an denen Subtransaktionen auszuführen sind, d.h. diese Instanzen bilden die Schnittstelle zu den lokalen DBVS.

Zur Bearbeitung gehören folgende Schritte, die wir zusammen als **Transaktionsverarbeitungs-Protokoll** (TVP) bezeichnen:

(1) **Ausführbarkeitstest.** Der TM überprüft die Verfügbarkeit (Zugreifbarkeit) der benötigten Partnerknoten der auszuführenden Transaktion. Sind nicht alle diese Knoten verfügbar, wird die Transaktion abgebrochen (Mißerfolg).

(2) **Sperrphase.** Der TM verschickt Subtransaktionen an die betreffenden Partnerknoten. Ein DM an einem solchen Knoten

versucht, die in der Subtransaktion spezifizierten Sperren zu setzen und quittiert dem TM Erfolg bzw. Mißerfolg. Mißerfolg führt zu Transaktionsabbruch.

(3) **Ausführungsphase**. Die in einer Subtransaktion vermerkten Lese- und Schreiboperationen werden ausgeführt; Schreiboperationen (oder ggf. deren Ergebnisse) werden in einem transaktionsspezifischen Puffer temporär abgelegt.

(4) **Terminierungsphase**. TM und DM führen ein 2-Phasen-Commit aus. Im Erfolgsfall werden im Transaktionspuffer stehende Änderungen permanent in einer atomaren Operation in die lokalen Datenbasen an den beteiligten Partnerknoten geschrieben, die Daten werden entsperrt, und das die vom TM eingesammelten Ergebnisse werden dem Benutzer mitgeteilt.

Ferner existieren folgende Protokolle zur Konsistenzerhaltung:

- TM- und DM-Logging: "crash-relevante" globale und lokale Transaktionszustände werden auf stable storage gespeichert, um bei Wiederanlauf eine konsistente Terminierung offener Transaktionen zu garantieren. Ein TM trägt auch Information über alle Subtransaktionen ins Logbuch ein. DMs tragen auch Information über Vor- u/o Nachzustand von zu ändernden Daten ein.
- TM- und DM-Restart-Protokoll: Folgendes Wiederanlaufkonzept zur Erhaltung der operationalen Datenkonsistenz wird angenommen:
  -- Alle offenen Transaktionen, die sich noch nicht im globalen Zustand "2. Commit-Phase" zum Ausfallzeitpunkt befunden haben, werden abgebrochen.
  -- Auf alle offenen Transaktionen, die sich innerhalb dieser Phase befanden, wird ein TM/DM- bzw. DM/DM-Verhandlungsprotokoll angewendet, um atomare Transaktionsterminierung zu erzielen.
- Kopienabgleichsprotokoll: Zur Erzielung gegenseitiger Konsistenz müssen ggf. Kopien einer zu ändernden Dateneinheit angeglichen werden, um eine einheitliche Sicht auf alle solchen Kopien zu erzwingen. Je nach Kopienhaltungskozept erfolgt der Abgleich vor oder nach der Terminierungsphase: beim Protokoll PROT_PRIM genügt es, Sekundärkopien "im Hintergrund" abzugleichen.

3.3.4 Lokale Datenbankverwaltung

Das an jedem Knoten befindliche lokale Datenbanksystem wird im Rahmen dieser Arbeit nicht detailliert modelliert. Wir beschränken uns auf die Spezifikation der DBVS-Dienstoperationen und die Abschätzung ihrer jeweiligen Kosten (CPU- und Massenspeicher- Zeitverbrauch). Gemäß Abb. 3.2 werden DBVS-Dienste von folgenden VDBS-Prozessen in Anspruch genommen:

- -- TM: Leseoperationen auf Katalog und Leistungskennwerte (DD/D-Zugriff) zur Anfrageauswertung und -Optimierung
- -- DM: Kopienlesen und -ändern; Kopiensperren und -entsperren
- -- TM und DM: Logging-Operation für Transaktionszustände und Kopien auf stabilen Speicher; VDBS-Wiederanlauf

3.3.5 Kommunikations-Subsystem

VDBS-Prozesse setzen spezielle, hohe Anforderungen an den Nachrichtenaustausch. Diese Anforderungen sind bisher nicht standardisiert worden und gehen über die realisierte IPC-Schnittstelle vieler verteilter Betriebssysteme hinaus. Im wesentlichen sind das die Anforderungen

- -- Zuverlässiger Nachrichtenaustausch,
- -- automatische Fehlererkennungsmechanismen,
- -- Mehrpartner-Nachrichtenaustausch (1:n- und n:1-Kommunikation)

Eine solch mächtige IPC-Schnittstelle wird in Kapitel 4 entworfen. Ein Verzicht darauf würde die Komplexität vieler VDBS-Protokolle noch weiter steigern und ihre Verständlichkeit sowie Verifizierbarkeit weiter reduzieren.
In Abb. 3.2 wird neben dem IPC-Dienst auch ein Filetransfer-Dienst und ein Datenpräsentationsdienst gefordert. Der erste Dienst wird z.B. zum Austausch von Zwischenergebnissen (Ausführungsphase) und zum Kopienabgleich verwendet. Der zweite Dienst ist z.B. in heterogenen VDBS erforderlich; wir gehen darauf aber nicht weiter ein.

4 DAS VDBS-BASISREFERENZMODELL (BRM)

Verteilte Transaktionsverarbeitung wird im wesentlichen durch ein Protokoll zwischen den VDBS-Prozessen Transaktions-Manager und Daten-Manager spezifiziert. Um unterschiedliche VDBS-Prozesse und -Protokolle auf einer gemeinsamen, **uniformen Grundlage** modellieren zu können, wird in diesem Kapitel ein formales Modell der zur VDBS-Protokoll-Umgebung gehörenden Systemkomponenten entworfen. Dieses protokollunabhängige, parametrisierte Modell bezeichnen wir als **Basisreferenzmodell** (BRM) für VDBS. Formales Beschreibungsmittel des Modells wird die graphische Sprache der Funktionsnetze sein. BRM-Parameter sind die bei der quantitativen VDBS-Untersuchung zu variierenden Größen, soweit sie nicht zum Transaktions-verarbeitungsprotokoll (TVP) gehören.
Die klare Trennung zwischen VDBS-Protokollen und dem BRM sowie die modulare BRM-Struktur ermöglichen eine vorhergehende Bewertung und Gültigkeitsprüfung des BRM. Das BRM kann als "Simulations-Testbett" für unterschiedliche Protokolle angesehen werden, ähnlich den implementierungsnahen Testbettsystemen (Kap. 2.4.1). In beiden Modellierungskonzeptionen kann sowohl eine Verringerung der Modellerstellungs- und Modellauswertungskosten, als auch ein größeres Vertrauen in die Untersuchungsergebnisse erwartet werden.
Das BRM beinhaltet im einzelnen: Ein Konfigurationsmodell der Rechnernetz-Architektur (Kap. 4.1), ein Modell des Auftretens, der Erkennung und Behandlung datenbankrelevanter Systemstörungen (Kap. 4.2), ein Kommunikationsmodell für verteilte Prozesse bzw. kooperierende Protokollinstanzen (Kap. 4.3), sowie ein Modell der verteilten Daten und Anwendungen (Kap. 4.4).
BRM-Parameterwerte und -Verteilungen sind ausführlich im Anhang A beschrieben.

4.1 Das Rechnernetz-Modell

Wir gehen von einer modernen Architektur eines homogenen Verteilten Systems aus, wie sie Abb. 4.1 zeigt. Die Hardware-Konfiguration eines VDBS bestehe aus N≥2 geographisch verteilten, homogenen **Knoten**, die durch ein **Kommunikationssystem** /SCHN83/ lose gekoppelt sind. Die

Kommunikation zwischen Knoten erfolgt durch Nachrichten auf der Grundlage einer Hierarchie von Protokollen. Jeder Knoten besteht aus einem **Arbeitsrechner** (AR), einem Massenspeichersystem (aus evtl. mehreren physikalischen Speichereinheiten), aus einer Menge von 'intelligenten' **E/A-Stationen** (Terminals), sowie aus einem **Netzzugangsrechner** (net interface unit, NIU).

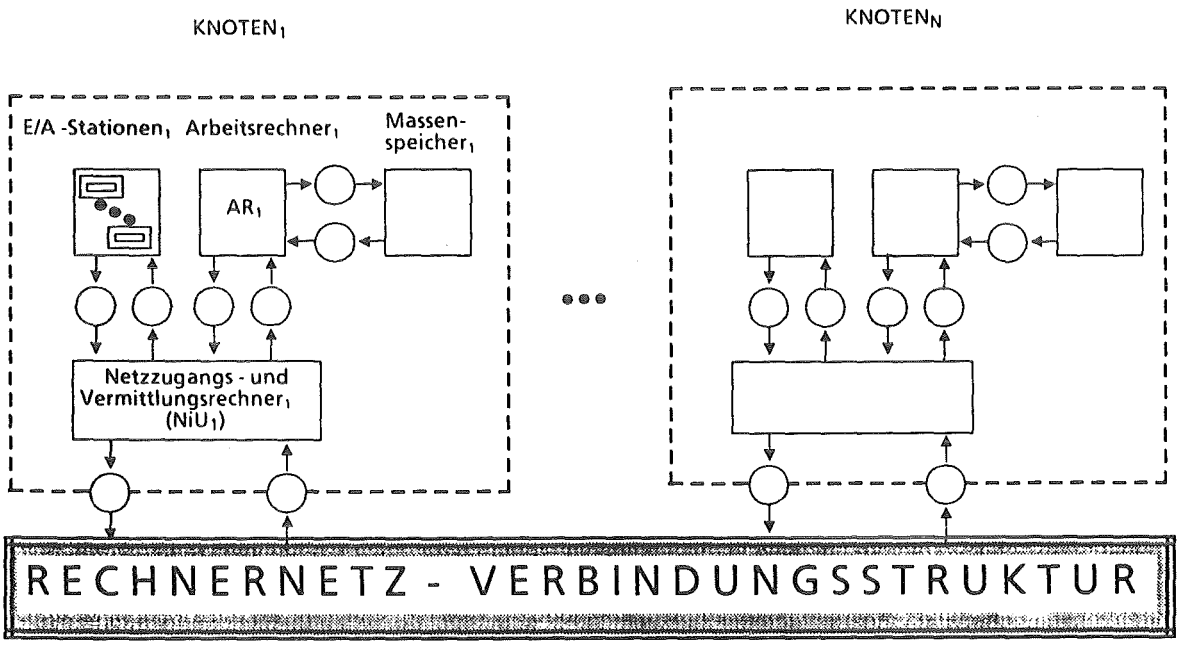


Abb. 4.1: Hardware-Architektur homogener Rechnernetze (vereinfacht)

Eine NIU regelt die lokale Kommunikation zwischen Arbeitsrechner, Hintergrundspeicher und Terminals, hat aber auch die Aufgabe, durch Techniken wie Nachrichten-, Paket- oder Leitungsvermittlung die Nachrichtenübertragung zwischen verschiedenen Knoten zu ermöglichen. Wir gehen von einer intelligenten NIU aus, die entsprechend der ISO-Protokollhierarchie /ISO84a/ Dienste bis einschließlich der **Transportschicht** 'on-the-chip' anbietet, wie bereits von einigen Herstellern angeboten /INTL84/. Zusätzlich wird die Existenz eines **sicheren Transportdienstes** angenommen, der Nachrichtenfehler maskiert. Wir gehen darauf im nächsten Abschnitt noch ein.

Die Festlegung eines NIU pro Knoten schließt bewußt eine zentrale Kontrollinstanz aus Verfügbarkeitsgründen aus (vergl. Abb. 1 !).

Die vorgestellte Architektur ermöglicht verschiedenartige nebenläufige

Aktivitäten:

- Nachrichtenaustausch zwischen Knoten nebenläufig mit lokalen Aktivitäten auf den Knoten;
- nebenläufige lokale Aktivitäten von Terminals, AR, Massenspeicher und NIU innerhalb eines Knotens.

Eine noch höhere Nebenläufigkeit läßt sich erreichen durch Parallelisierung der NIU-Aufgaben auf mehrere Spezialrechner:

- separate Netzanzugangsrechner für alle Terminals auf einem Knoten oder pro Terminal,
- separate Netzzugänge für Zugriffe auf Massenspeicher ('file server')

Um die Komplexität unseres Modells zu begrenzen, betrachten wir diese für schnelle lokale Rechnernetze interessante Verallgemeinerung nicht, zumal dedizierte Server zu heterogenen Rechnernetzen führen.

Die Hardware-Architektur wird entsprechend den ISO-Entwurfsprinzipien einer Protokollhierarchie -wie in Abb. 4.2 dargestellt- auf die Protokollhierarchie eines VDBS abgebildet, die in Abb. 3.2 gezeigt wurde. Als Ergebnis ist in Abb. 4.3 das Modell der logischen VDBS-Architektur, zusammen mit den wesentlichen Einflußgrößen, für einen Knoten des Rechnernetzes dargestellt. Die darin spezifizierten abstrakten Protokollinstanzen werden später bei der Detailmodellierung zu einem System kommunizierender Prozesse verfeinert. Abb. 4.3 zeigt insbesondere folgende Schichtenprotokolle zwischen abstrakten VDBS-Prozessen (Nummerierung wie in der Abbildung):

(1) Transaktions-Übersetzung
(2) Generierung Subtransaktionen
(3) Transaktionsverarbeitungs-Protokoll (TVP)
(4) Sperren /Entsperren von Dateneinheiten, Subtransaktions-DO
(5) DM-Zustandssicherung, Subtransaktions-UNDO, DM-Restart
(6) TM-Zustandssicherung, TM-Restart

Folgende BRM-Einflußgrößen von Arbeitsrechnern und Kommunikationssystem werden als Modellierungparameter eingeführt:

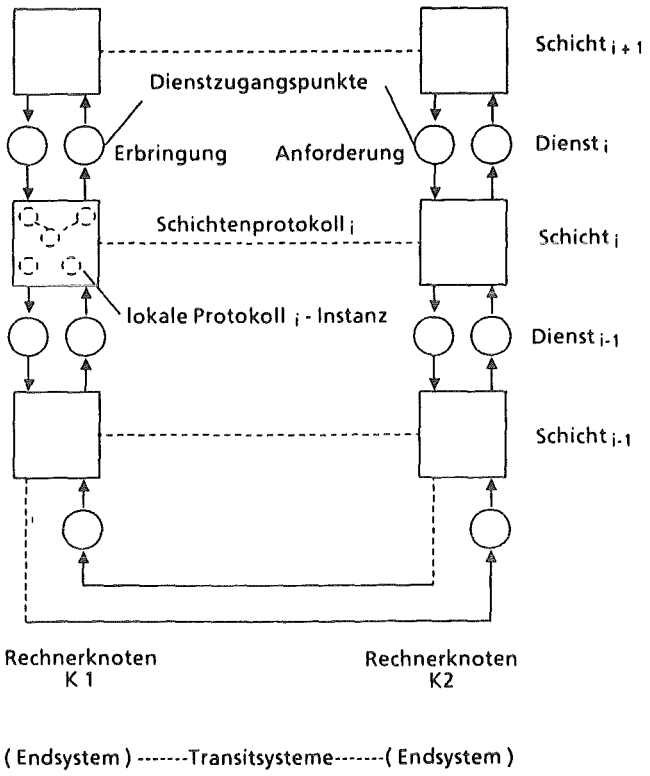


Abb. 4.2: Logische Schichtenarchitektur eines Verteilten Systems nach ISO-OSI

Arbeitsrechner-Profil:

(HW1) **Nebenläufigkeitsmaß:** Als Maß wird die maximale Anzahl aktiver Transaktionen pro Knoten gewählt /SEVC83/, d.h. solcher Transaktionen, die an diesem Knoten eingegeben wurden und noch nicht mit 'COMMIT' bzw. 'ABORT' terminiert haben. In diesem Maß sind implizit enthalten:

- Die Kapazität der lokalen Betriebsmittel Arbeits- und Pufferspeicher, über die das VDBS verfügen kann;
- Die maximale Anzahl von Terminals pro Heimatknoten: Es wird angenommen, daß eine Transaktion an einem Terminal erst dann eingegeben werden kann, wenn die zuletzt bearbeitete abgeschlossen wurde. Zwischen der Terminierung einer Transaktion und der Initiierung einer neuen Transaktion wird eine gewisse Terminal-Denkzeit verbraucht (s.u.). Das liefert ein bezüglich der Systemlast geschlossenes

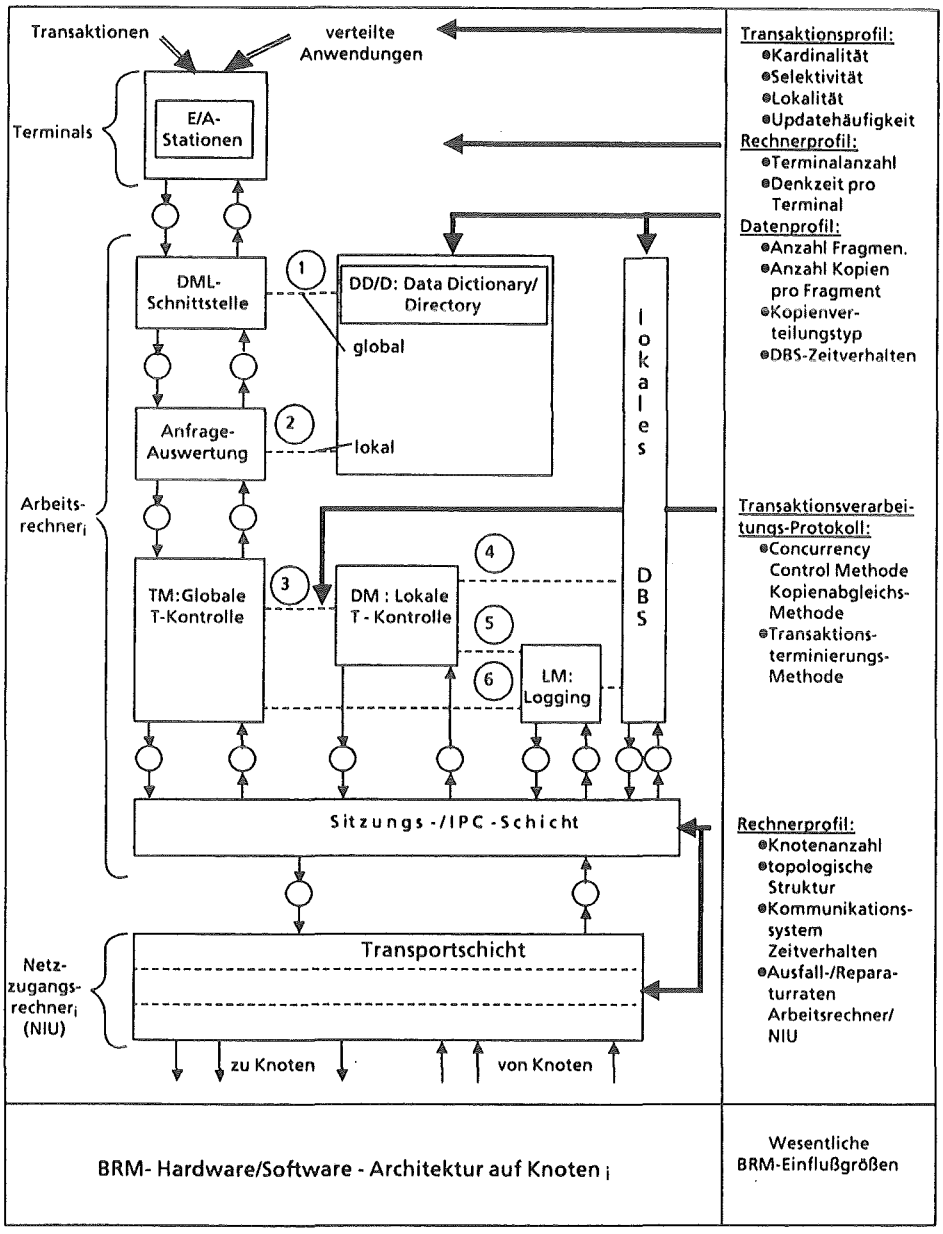
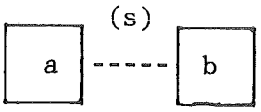


Abb. 4.3: VDBS-Basisreferenzmodell (BRM) auf Knoten$_i$



: Schichtenprotokoll (s) zwischen abstrakten VDBS-Prozessen a und b

: Einfluß des Parameters p auf BRM-Komponente K

Warteschlangenmodell.

Dieses Nebenläufigkeitsmaß beschreibt also eine feste Population interaktiver Benutzer.

Für eine sinnvolle Realisierung der Behebung von Systemfehlern ist dieses Maß besonders wichtig, da bei vielen Verfahren bestimmte Ausfälle eine Blockierung aktiver Transaktionen verursachen, z.B. in der 2.Phase eines Commit-Protokolls. Es ist dann sinnlos, eine beliebige Anzahl neuer Transaktionen zu akzeptieren und zu puffern, wenn sie wegen Sperrung von Daten durch blockierte Transaktionen nicht erfolgreich bearbeitet werden können.

Ein mit diesem Maß verwandter, aber nicht identischer Begriff ist der **Multiprogrammming-Grad** /KOST82/. Wie Kostro ausführt, ist dieser Grad in vielen Systemen höher als die Anzahl aktiver Jobs (z.B. Transaktionen), da diese selbst nebenläufige Anforderungen stellen und Unteraufträge erzeugen können.

(HW2) **Denkzeit pro Terminal:** Zeit zwischen Ergebnisübergabe einer Transaktion an den Terminal-Benutzer und der Initiierung einer neuen Transaktion durch denselben. Denkzeiten werden exponentialverteilt angenommen, da diese 'gedächtnislose' Verteilung der Annahme entspricht, daß aufeinander folgende Transaktionen voneinander unabhängig sind.

Nach /FERR78, S. 211/ kann die Transaktions-Ankunftsrate ARR pro Knoten indirekt aus den genannten Größen näherungsweise bestimmt werden:

$$ARR = (TCO - W) / D \qquad \text{wobei}$$

TCO: Anzahl Terminals pro Knoten
(Transaktions-Nebenläufigkeit)
W: mittl. Anzahl offener Transaktionen pro Knoten, $W \leq TCO$
D: mittl. Denkzeit pro Terminal

W ist eine system- und zustandsabhängige Größe; TCO-W gibt die mittlere Belegung der Warteschlangen vor den Terminals an.
Zur Untersuchung der Auswirkung steigender Transaktionslasten auf das Systemverhalten gibt es u.a. folgende Alternativen:

- TCO konstant; D erniedrigen
- D konstant; TCO erhöhen

In dieser Arbeit wird die zweite Alternative gewählt.

(HW3) Betriebssystem-Charakteristik
(Prozeßscheduling-Strategie, Prozeßumschaltzeiten) :
Da VDBS-Protokolle als kommunizierende Prozesse im Sinne des Betriebssystems modelliert werden, beeinflußt das jeweilige lokale Betriebssystem eines Arbeitsrechners auch das Protokoll-Zeitverhalten. Die relevantesten Größen sind dabei Scheduling-Strategie und der Systemoverhead bei Prozeßwechsel. Der Overhead wird im Rahmen unseres Modell-Abstraktionsgrades als vernachlässigbar angesehen. Auf Scheduling-Strategien wird in Kap. 6 noch näher eingegangen.

Hingegen wird die Verarbeitungsgeschwindigkeit nicht als Parameter modelliert. Wegen der Homogenität aller Knoten sind die mittleren Geschwindigkeiten (Anzahl Instruktionen pro Zeiteinheit) aller Rechner gleich; sie werden indirekt über die Verarbeitungszeiten der auf dem Arbeitsrechner ablaufenden Prozesse beschrieben.

Profil des Kommunikationssystems:

Im Gegensatz zu /WILD79/, der sehr detailliert verteilte Systeme klassifiziert und ihr Leistungsverhalten analysiert, erscheint für VDBS-Leistungsmodelle eine stark vergröbernde Modellierung des Zeit- und Verfügbarkeitsverhaltens ausreichend. Das Verhalten homogener verteilter Systeme auf der für unsere VDBS-Modelle adäquaten Abstraktionsebene kann durch lediglich vier Modellparameter hinreichend charakterisiert werden:

(HW4) **Knotenanzahl**
(HW5) **Verbindungsstruktur**
(HW6) **Knoten-Knoten-Übertragungszeitverteilung** (End-To-End Delay)
(HW7) **Nebenläufigkeitsmass**

Es wird ein **Ring** als physikalische Verbindungsstruktur zwischen den N Knoten (genauer: NIUs) angenommen. Ein Ring bietet gegenüber anderen Strukturen wie Stern, Bus und Baum folgende Vorteile:

- In lokalen Netzen haben sich die "einfachen" Strukturen durchgesetzt, da sie potentiell höhere Zuverlässigkeit und Verfügbarkeit als unregelmäßige Strukturen aufweisen. Da der Schwerpunkt dieser Arbeit auf lokal verteilten Datenbanken liegen soll, bevorzugen wir einfache Strukturen.
- In günstig organisierten Ringsystemen (Rechnernetzsteuerung dezentral; bidirektionale Leitungen oder Doppelringsystem) führt ein Ausfall von einer Leitung bzw. einem Knoten höchstens zu Beeinträchtigungen der anderen N-1 Rechnerknoten, nicht aber zu wechselseitiger Rechnernetz-Partitionierung zwischen jedem Knotenpaar. Ein Ring ist eine Struktur mit minimaler Verbindungsanzahl bezüglich dieser Eigenschaft.
- Ringstrukturen sind anderen regelmäßigen Strukturen wie etwa Bussystemen im Hochlastbereich überlegen /SPAN82/; es kann sogar eine Durchsatzgarantie gegeben werden, da die angewendeten Nachrichtenscheduling-Verfahren fair bezüglich allen Rechnerknoten sind. Eine solche hohe Kapazitätsauslastung ist für viele VDBS-Anwendung wahrscheinlich.

Alternative Modelle der Verteilung von **Übertragungszeiten** zur Übermittlung einer Nachricht zwischen Instanzen der Anwendungsschicht eines verteilten Systems sind, mit steigendem Detaillierungsgrad:

(a) konstante (last-, zeit- und zielunabhängige) Verzögerung V
(b) konstante (last-, zeitunabhängige), aber zielabhängige Verzögerung V(i,j) für ein Knotenpaar i,j ε [1,N]
(c) Statistisch verteilte Zeit mit konstantem, zielunabhängigen Erwartungswert V
(d) Statistisch verteilte Zeit mit zielabhängigem Erwartungswert V(i,j)
(e) Die Verteilung der Verzögerungszeiten ergibt sich durch das komplexe Zusammenspiel detaillierter Einzelfunktionen, z.B. durch Algorithmen für Wegewahl und Flußkontrolle unterer Rechnernetzschichten

Wir nehmen eine Verteilung der Übertragungszeiten entsprechend Modell (c) an. Dabei werden mittlere Übertragungszeiten unabhängig von Quell-/Zielknoten und damit von der gewählten Rechnernetz-Struktur

festgelegt.
Konstante Zeiten bei (a),(b) erscheinen zu unrealistisch; Modell (d) wäre zu stark von einer bestimmten Übertragungstechnik und der Netzstruktur abhängig; bei Modell (e) wäre die gesamte Kommunikationssoftware im Detail nachzubilden.
Charakteristische Übertragungszeiten ("end-to-end Verzögerung") sind 0.02 bzw. 0.5 sec pro Nachricht für lokale bzw. long-haul Rechnernetze bei einer mittleren Nachrichtenlänge von 150 bytes (nach /WILD79/). Die Wahl von z.B. normalverteilten Übertragungszeiten gestattet die flexible Spezifikation unterschiedlicher Rechnernetz-Klassen durch die Variierung der Varianz:

- Kleine Varianz repräsentiert ein Netz mit hoher Bandbreite, geringer Auslastung und kleinen Nachrichtenfehlerraten
- Große Varianz repräsentiert ein Netz mit niedriger Bandbreite, hoher Auslastung und größeren Fehlerraten.

Ferner werden folgende Annahmen getroffen:

(1) Der Mittelwert der Verzögerungszeit gelte für alle Nachrichten zwischen beliebigen Knoten i, j.

Nachrichtenlängen werden damit als konstant vorausgesetzt. Diese vereinfachende Annahme ist im VDBS-Bereich üblich, sofern keine größeren Datenmengen zu transferieren sind (s. etwa /OZSU83, SEVC83/). Dafür würde sich eher ein Dateitransferdienst eignen. Ein solcher Dienst kann auf der Basis eines Dienstes zur Interprozeß-Kommunikation realisiert werden, sodaß die Annahme keine Einschränkung bedeutet.

(2) Der Mittelwert der Verzögerungszeit gelte auch für Multicast-Nachrichten, d.h. daß $n \geq 1$ Nachrichten nebenläufig übermittelt werden können.

Reale Rechnernetze werden durch einen konstanten Mittelwert nur unterhalb des Mittel- und Hochlastbereichs genügend genau beschrieben. Realistischere Übertragungszeiten ergeben sich bei Berücksichtigung der zulässigen Nebenläufigkeiten im Rechnernetz.

Das **Nebenläufigkeits-Mass** des Kommunikationsprofils beschreibt den Dezentralisierungsgrad der Nachrichtenvermittlungsinstanzen im

Rechnernetz. Zur Verdeutlichung dienen folgende Extremfälle.

- Vollständig **zentrale Kontrolle**: Es gibt genau einen Kommunikationsrechner, z.B. einen Bus Controller in bestimmten lokalen Rechnernetzen. Auch ein "token ring" /SPAN82/ verhält sich wie ein Netz mit zentraler Kontrollinstanz: Zu einem Zeitpunkt kann sich höchstens eine Nachricht (pro Token) auf dem Ring befinden.
- Vollständig **dezentrale Kontrolle**: Es gibt keine dedizierten Kommunikationsrechner; jede der N NIUs kann Nachrichten bzw. Pakete vermitteln. Zu einem Zeitpunkt können Nachrichten zwischen beliebigen, disjunkten Knotenpaaren nebenläufig übertragen werden.

Wir nehmen maximale Nebenläufigkeit im Rechnernetz an, die aber im Hochlastbereich durch die Übertragungskapazitäten und Auslastungen der NIUs begrenzt werden.

In diesem Kapitel wurden die hardware-orientierten Modellparameter des Rechnernetzes definiert, soweit sie dessen operationales, störungsfreies Verhalten betreffen. Die Erweiterung auf die Charakterisierung des **Fehlverhaltens** wird im nächsten Abschnitt besprochen.

4.2 Ausfallauftretens- und Ausfallbehandlungsmodell

Im folgenden werden Systemfehler und Ausfälle berücksichtigt, soweit sie eine ordnungsgemäße Funktion des Systemdienstes "Transaktionsverarbeitung" beeinträchtigen können. Sogar falls eine Fehlerbehandlung (Restart und Recovery) für alle möglichen Fehlerfälle operationale Datenkonsistenz sichern würde, können Beeinträchtigungen ("performance degradation") der folgenden Art auftreten:

- Eine Transaktion kann nicht gestartet werden und wird abgewiesen,
- Eine Transaktion wird während ihrer Bearbeitung vom System abgebrochen aus Gründen, die nicht der Benutzer zu vertreten hat,
- Eine Transaktion ist während der gesamten Dauer einer bzw. mehrerer Fehler blockiert, d.h. deren Terminierung hängt von einer erfolgreichen Reparatur bestimmter Fehler ab. (Wie in /SKEE81/ bewiesen wurde, kann es keine nichtblockierende Transaktionsterminierung bei beliebigen Knotenausfällen geben.)

Solche Beeinträchtigungen u/o Dateninkonsistenzen können durch eines oder mehrere der folgenden Ereignisse ausgelöst werden: **Systemteilausfall** (Ausfall von Arbeitsrechnern, NIU, Leitungen), **Massenspeicherteilausfall** und **Transaktions(teil)abbruch**. Andere mögliche Ausfälle sind so unwahrscheinlich, daß ihre Betrachtung für die Praxis irrelevant ist, oder es sind Fehler, die durch untere Schichten eines VDBS "maskiert" werden. Diese Fälle werden am Kapitelende diskutiert.

Im folgenden wird idealisierend bzgl. allen betrachteten Fehlern **Fehlererkennbarkeit** vorausgesetzt. Das schließt insbesondere **transiente Fehler** aus, die sporadisch auftreten und vor einer Erkennung wieder verschwunden sind (im Gegensatz zu permanenten Fehlern).

4.2.1 Grundlegende Definitionen

Im Gebiet "fehlertolerierende (Mehr-)Rechnersysteme" ist zu beobachten, daß eine einheitliche und konsistente Begriffsbildung keineswegs abgeschlossen ist, vergl. /KOPE82, DILGE83, NTG82/. Im folgenden wird eine konsistente Anwendung der Begriffe für den VDBS-Bereich versucht. Dabei wird eine **Systemdienst-Sicht** eingenommen. Aus Hardwaresicht sind z.T. andere Definitionen sinnvoller /SISW82/.

(1) Ein **Fehler** ist ein Zustand eines Systems, der sich im Widerspruch zur Systemspezifikation befindet und durch die Ereignisse **Ausfall** bzw. **Wiederinbetriebnahme** ausgelöst bzw. beendet wird (nach /KOPE82/). Nach einem HW-Ausfall eines Rechners spricht man von **Wiederanlauf** (Restart) anstatt von Wiederinbetriebnahme.

(2) **Fehlertoleranz** ist die Fähigkeit eines Systems, auch mit einer begrenzten Zahl fehlerhafter Subsysteme seine spezifizierte Funktion zu erfüllen (nach /DILGE83/).
Ersetzt man "begrenzte Zahl" durch "n", dann erhält man den konkreteren Begriff **Ausfallsicherheit(n)** oder "n-failure-resiliency" /LEBR82/. Zwei Interpretationen dieses Begriffs sind möglich, wobei von Zeitabhängigkeiten abstrahiert wird:
- **Zuverlässigkeit(n)**, abgekürzt REL(n), bedeutet daß mehr als n

gleichzeitig aufgetretene Fehler zu einem entsprechend der Spezifikation inkorrektem Systemverhalten führen.

- **Verfügbarkeit(n)**, abgekürzt AVA(n), bedeutet daß mehr als n gleichzeitig aufgetretene Fehler eine Ausführung der Systemfunktion verhindern.

Verfügbarkeit wird oft als Teil der Zuverlässigkeit betrachtet, da eine rechtzeitige Funktionsausführung oft Bestandteil der Systemspezifikation ist, wie z.B. in Echtzeitsystemen.
Um Fehlertoleranz zu erzielen, ist stets **Redundanz** erforderlich. Redundante Komponenten sind (Hardware-, Software-, Daten-) Ersatzkomponenten zur Steigerung der Fehlertoleranz.

Beispiel: Ein Dateiverwaltungssystem benötigt mindestens n+1 Speichermedien mit voneinander unabhängigem Fehlermodus, um REL(n) gegenüber Fehlern des Speichermediums zu erreichen. Zusätzlich sind mindestens n+1 unabhängige physische Zugriffswege zu jedem Speichermedium erforderlich, um AVA(n) zu garantieren. Prinzipiell sind also Verfügbarkeitsanforderungen schwerer zu erfüllen.

(3) **Recovery** ist eine (SW- oder HW-) Maßnahme, um aus einem Fehlerzustand in einen fehlerfreien "Normalzustand" zu gelangen, in dem alle Fehlereffekte eliminiert sind, nachdem der (die) Fehler erkannt und lokalisiert wurden.
Der Übergang zurück zum fehlerfreien Zustand wird oft als Wiederanlauf bzw. Restart bezeichnet, was inkonsistent zur obigen Definition ist, da nicht alle Fehler Hardwareursachen haben. Wir bevorzugen daher den gebräuchlicheren Begriff "Recovery" und wollen darin auch Wiederanlaufmaßnahmen enthalten wissen.
Man unterscheidet insbesondere

- **Vorwärts-Recovery**: Übergang in aktuellen fehlerfreien Zustand, der ohne Auftreten von Fehlern erreicht worden wäre; z.B. REDO einer Transaktion oder bei unzugänglichen Kopien die Beschaffung aktueller Versionen bereits aktualisierter Kopien,
- **Rückwärts-Recovery**: Übergang in früheren fehlerfreien Zustand; z.B. UNDO einer Transaktion oder Ersetzung fehlerhafter Daten durch eine ältere, aber konsistente Version dieser Daten.

Bei der Definition von Fehlermodellen komplexer (insbesondere: verteilter) Systeme hat es sich bewährt, von einem **Schichtenmodell** des Systems auszugehen. Damit können sich die definierten Begriffe Fehler, Ausfall, Recovery, Restart, REL(n), AVA(n) sowohl auf die interne Realisierung einer Schicht(i) als auch auf den von ihr erbrachten Dienst für die nächsthöhere Schicht(i+1) beziehen.
In geschichteten Systemen ist es wichtig, innerhalb einer Schicht entdeckte Fehler soweit zu tolerieren, daß sie nicht bis zur nächsthöheren Schicht gelangen. Dazu definieren wir den Begriff

(4) **Fehlermaskierung(FK,i)** einer Schicht(i) von Fehlern der Klasse FK bezüglich Schicht(i+1) bedeutet, daß beliebig viele Fehler aus FK, die auf einer Schicht(i') mit $i' \leq i$ auftreten, auf der Schicht(i+1) nicht sichtbar sind, da die Fehler spätestens auf Schicht(i) erkannt und ihre Folgen behoben worden sind.

Diese idealisierte Annahme der Tolerierung beliebig vieler Fehler erlaubt es, bei der Betrachtung von Schicht(j), $j>i$, diese Fehler zu ignorieren, da sie garantiert von unteren Schichten behandelt werden. Beispiel: Fehlerhafter Nachrichtenaustausch kann in Protokollen oberhalb der Sitzungsschicht oft ignoriert werden.

4.2.2 Arbeitsrechner- und NIU-Ausfall

Im folgenden wird die Ausfallcharakteristik der wichtigsten Hardwareeinheiten, Arbeitsrechner und NIUs, betrachtet. Andere Einheiten wie z.B. NIU-NIU-Verbindungen werden als so robust angenommen, daß sie nie ausfallen.
Die Ausfallcharakteristik soll nicht nur Hardware-Ausfälle umfassen, sondern alle anderen Fehlerarten wie z.B. Operateurfehler oder fehlerhafte Systemsoftware, die zu einer Nichtverfügbarkeit der jeweiligen Komponente aus **Benutzersicht** führen können.

Stochastische Annahmen

Es wird von folgendem einfachen stochastischen Modell der Ausfall- / Reparaturzeiten von Arbeitsrechnern und NIUs eines Rechnernetzes ausgegangen. Wir betrachten dazu die Ausfall-Charakteristik von homogenen Komponenten K_1, K_2, ... eines Subsystems, z.B. aller Arbeitsrechner.

(F0) Jeder Ausfall ist in endlicher Zeit **erkennbar** und **reparierbar**.

(F1) Jede Komponente K_i verhält sich **fail-stop**: Entweder sie ist **intakt** ('UP') d.h. funktioniert korrekt gemäß der Hardware- und Systemsoftware-Spezifikation, oder sie ist total **ausgefallen** ('DOWN').

(F2) Mittlere Ausfall- und Reparaturzeiten aller Komponenten sind zeitunabhängig ("stationär"). Insbesondere ist ein Komponentenausfall stochastisch unabhängig externen Bedingungen, insbesondere von
- Ausfällen / Reparaturen anderer Komponenten,
- Alterung der Komponente,
- Systembelastung, speziell vom Einfluß der Auftragsankunfts-Verteilung.

(F3) Mittlere Reparaturzeiten $MTTR_i$ und Ausfallzeiten $MTTF_i$ sind für alle homogenen Komponenten K_i eines Subsystems gleich.

(F4) Fehlererkennungszeiten werden vernachlässigt, da sie i.d.R. klein im Vergleich zu Reparaturzeiten sind.

Nach diesen allgemeinen werden nun spezielle Annahmen über Ausfälle und Reparaturen von Arbeitsrechnern und NIUs definiert.

Annahmen zum Arbeitsrechner-Ausfall:

Ein AR_i-Ausfall ist aus dessen Sicht charakterisiert durch

(AR1) Inhalte von Hauptspeicher und Puffer gehen verloren (speziell: alle Inhalte von Kanälen, sofern nicht auf Platte o.ä. gerettet). Alle aktiven Prozesse auf dem Arbeitsrechner werden abgebrochen, ebenso die Bearbeitung von (Sub-) Transaktionen, Aufträgen und Nachrichten.

(AR2) Daten auf Hintergrundspeicher sind nicht betroffen (d.h. Unabhängigkeit der Fehlermodi Knoten - Platte, s. (F2)).

Ein AR_i-Ausfall aus Sicht anderer Knoten ist charakterisiert durch

(AR3) Es ist nicht mehr möglich, mit AR_i zu kommunizieren. Das wird von jedem Rechner AR_j, der mit AR_i kommunizieren möchte, in endlicher Zeit erkannt (Forderung an Dienst zur Interprozeß-Kommunikation !).

(AR4) AR_i-Ausfall und Ausfall aller physischen Verbindungen zu AR_i (d.h. Rechnernetzpartitionierung zwischen AR_i und alle anderen Knoten) sind ununterscheidbar, solange der Ausfall nicht repariert ist (diese Annahme ist realistisch, s. z.B. /PESI83, S.472/).

(AR5) Ein Knoten AR_j, der die Partitionierung von AR_i erkannt hat, kann nicht entscheiden, ob Nachrichten von AR_j an AR_i, deren Erhalt an AR_j nicht quittiert worden ist, von AR_i bereits erhalten wurden u/o ob AR_i diese quittieren konnte.

Oft wird bei Knotenausfall unterschieden zwischen

- 'weicher' Ausfall: wegen Operateur- oder Anwendungs-SW-Fehler begibt sich das Betriebssystem in seinen Grundzustand. D.h. MTTR wäre hier annähernd gleich null; Restart und Recovery könnten sofort einsetzen.
- 'harter' Ausfall: HW-Ausfall mit MTTR >> 0. Restart und Recovery erst nach erfolgter Reparatur.

Diese Unterscheidung kann ggf. durch eine entsprechende MTTR-Verteilungsfunktion eines Knotens modelliert werden.

Die Restart-Prozedur des Arbeitsrechners ist z.T. anwendungsabhängig und wird weiter unten modelliert.

Annahmen zu NIU-Ausfall und Rechnernetz-Partitionierung

(N1) **Adaptive Wegeermittlung** /SCHN83, S.594/: Ein Ausfall einer phys. Verbindung (Leitungen, NIUs) zwischen AR_i, AR_j wird stets vor der IPC-Schicht und darüberliegenden Schichten maskiert, solange es andere intakte Wege zwischen diesen Knoten gibt.

Eine Abtrennung **aller** Wege zwischen AR_i, AR_j (wegen Leitungs-, NIU- u/o Arbeitsrechnerausfällen) wird als **Rechnernetzpartitionierung** (RNP) bezeichnet.

Schreibweise: RNP(<Arbeitsrechnermenge1> , <Arbeitsrechnermenge2>).

(N2) Ein NIU_i-Ausfall bewirkt
- Falls AR_i intakt ist, nimmt dieser Arbeitrechner eine RNP zu allen anderen Arbeitsrechnern an. Ein intakter AR_j ($i \neq j$) nimmt eine RNP zu AR_i an.
- Falls es andere Arbeitsrechner AR_{j1}, AR_{j2} gibt, die beide intakt sind und für die der einzige Kommunikationsweg über NIU_i verlief, nehmen beide eine RNP zum jeweils anderen Arbeitsrechner an.

(N3) Eine NIU_i-Reparatur bewirkt
Alle intakten Arbeitrechner AR_i, AR_j, die durch diese Reparatur wieder verbunden sind, erkennen die Beendigung der gegenseitigen RNP. Diese Reintegration der vorher isolierten Teilsysteme bezeichnet man als 'partition merge'.

(N4) Aus (N1)-(N3) folgt **Kommutativität** für beliebige disjunkte AR-Mengen AM, AM':

$$\neg RNP(AM, AM') \Leftrightarrow \neg RNP(AM', AM)$$

In realen Systemen wird zwischen einem Ausfall und der gegenseitigen RNP-Erkennung eine gewisse Zeit verstreichen. Bei der Erkennung kann wegen dem Fehlen einer globalen Uhr im Verteilten System keine 'Gleichzeitigkeit' definiert werden. Die Kommutativität folgt aber bereits aus unserer Modellannahme (F4): Danach werden Partitionsänderungen 'sofort' -d.h. ohne Zeitverzögerung- erkannt.

(N5) Aus (N1)-(N3) folgt **Transitivität** für beliebige disjunkte Mengen von Arbeitsrechnern AM, AM', AM":

$$\neg RNP(AM , AM') \wedge \neg RNP(AM' , AM") \Rightarrow \neg RNP(AM , AM")$$

Andernfalls wäre ein AM"-Rechner über den Umweg eines AM'-Rechner nicht erreichbar; das ist ein Widerspruch zum Konzept der adaptiven Wegeermittlung (N1).

Das geschilderte Ausfall/Reparaturmodell ist in Abb. 4.4 als "Markovmodell" in an Petrinetzen orientierter Notation veranschaulicht. Die Abbildung stellt repräsentativ für das Gesamtmodell die möglichen Systemzustände und -übergänge zweier

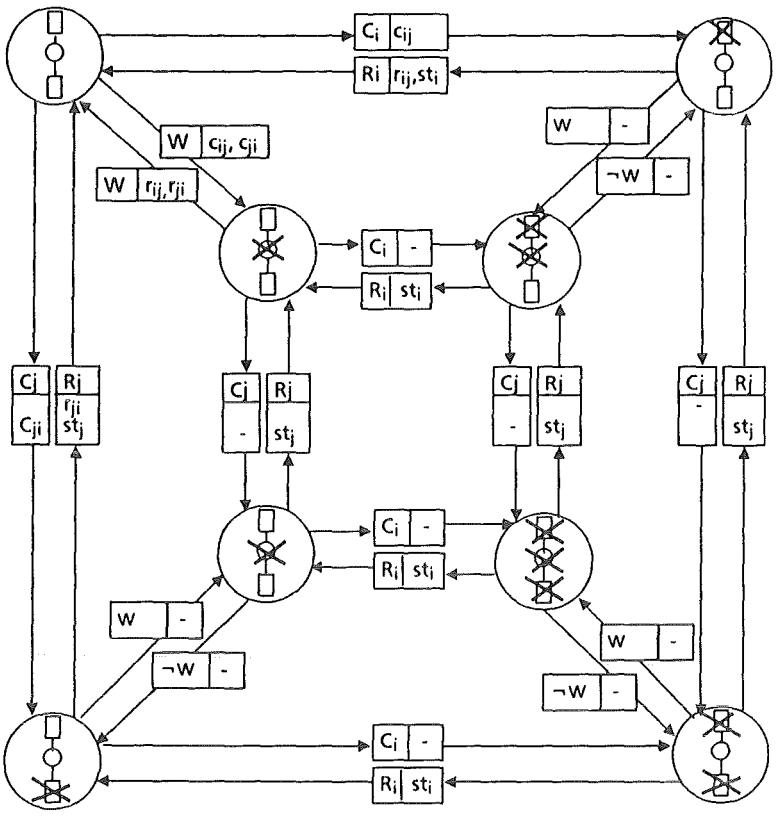
beliebiger Arbeitsrechner und deren Verbindungsstruktur dar. Insbesondere wird die wechselseitige Erkennung des Systemzustands aus Sicht eines Arbeitsrechners mit den entsprechenden (Un-)Verfügbarkeitsmeldungen veranschaulicht, die genau dem oben definierten Verhalten von Arbeitsrechnern und NIUs entspricht.

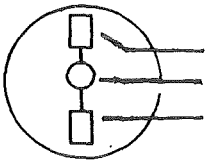
Die Wahrscheinlichkeit eines RNP-Auftretens ist wesentlich von der **Verbindungsstruktur** im Rechnernetz abhängig. Die Wahrscheinlichkeit kann reduziert werden, indem z.B. zwischen je zwei NIUs mindestens zwei alternative Wege geschaffen werden. Trotz einer solchen Verbindungs-Redundanz erzeugt aber bereits ein einziger NIU-Ausfall eine RNP.
Viele TVP sehen keine Maßnahmen im Fall eines RNP-Auftretens vor, da ein solches Ereignis angeblich sehr unwahrscheinlich sei (so bei SDD-1 nach /ROTH80/, VDN nach /MUNZ80/). Dazu folgende Gegenbeispiele:

- Ein RNP-ähnliches Verhalten kann leicht entstehen, wenn überlange Kommunikationsverzögerungszeiten bei stark ausgelastetem Rechnernetz auftreten (congestion).

- In jüngster Zeit werden verstärkt (lokale) Rechnernetze durch dedizierte **Kommunikationsrechner** (Gateways) verbunden; damit steigt die RNP-Wahrscheinlichkeit gegenüber einzelnen Rechnernetzen, da ein Gateway die einzige Verbindung zwischen je zwei Rechnernetzen darstellt.

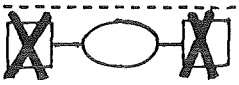
- Laut /JOHN77/ wurden in den Jahren 1972-76 im ARPANET Ausfallraten gemessen, die folgende Wahrscheinlichkeiten ergaben:
  p1 := p(IMP ausgefallen) = 0.0031
  p2 := p(Leitungsausfall) = 0.01
  Ein IMP im ARPANET ist ein spezieller NIU; Arbeitrechner werden 'Hosts' genannt. Abb. 4.5 zeigt das einfachste nichttriviale Rechnernetz vom Typ ARPANET.
  Es wird dabei von der beim ARPANET vorhandenen Zuverlässigkeitsanforderung ausgegangen, daß mindestens zwei physische IMP-IMP-Wege existieren. Unter Vernachlässigung von Hostknoten- und Konzentratorausfällen unter unter der Annahme unabhängiger Leitungs- und IMP-Ausfallraten ergibt sich:





Verfügbarkeitsstatus zwischen beliebigen Arbeitsrechnern:
Arbeitsrechner AR(i)
WEG(i,j): phys. Verbindung NIU(i) <--> ... <--> NIU(j)
Arbeitsrechner AR(j)

Tritt das (Crash/Restart-) Ereignis E auf, erfolgt Zustandsänderung Z --> Z', und eine Nachricht N wird als lokale Sicht des globalen Ausfallzustands erzeugt

Ereignis E	Nachzustand (Z')	Bedeutung
C_i, C_j		Crash von AR_i, AR_j
R_i, R_j		Restart von AR_i, AR_j
W		WEG(i,j) ausgefallen (keine phys. Verbindung zw. AR(i) und AR(j))

Nachricht N : c_{ij} AR_j erfährt "AR_i nicht verfügbar"
r_{ij} AR_j erfährt "AR_i wieder verfügbar"
st_j AR_j erfährt "Verfügbarkeitsstatus aller AR_i"

Abb. 4.4: Zustandsdiagramm des Ausfall-/Reparaturverhaltens aus Sicht zweier Arbeitsrechner AR_i, AR_j

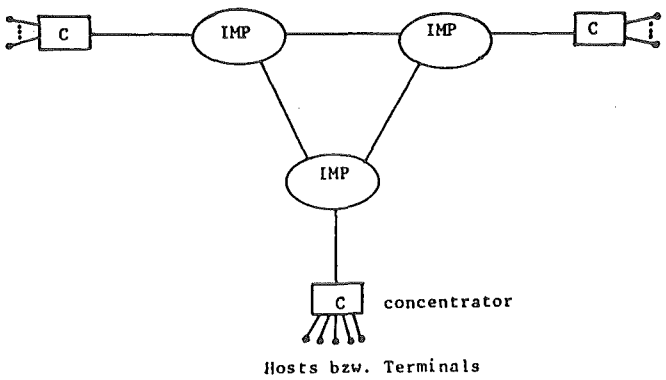


Abb. 4.5: Rechnernetzstruktur vom Typ ARPANET (Beispiel)

p(irgendein RNP zwischen beliebigen Hosts)
= 1 - p(alle Hosts sind physikalisch verbunden)
= 1 - [p(alle 3 IMPs 'up')
•p(alle 3 IMP-Konzentrator Leitungen 'up')
•p(≥2 IMP-IMP Leitungen 'up')]
= 1 - [$(1-p1)^3$
•$(1-p2)^3$
•$((1-p2)^3 + 3•(1-p2)^2•p2)$]
= 0.0119 für p1=0.0031, p2=0.01

Daraus läßt sich die MTTF einfach bestimmen:

p(irgendein RNP zwischen beliebigen Hosts)
= MTTR / (MTTF+MTTR)

Bei einer MTTR von 1 Stunde ergibt sich MTTF = 83 Stunden.
Damit ist bei diesem Beispiel eine RNP in etwa gleichwahrscheinlich wie ein beliebiger Hostknotenausfall (bei p(Knotenausfall) = 0.01)!

4.2.3 Transaktions(teil)abbruch

Darunter verstehen wir einen Abbruch einer Subtransaktion z.B. wegen Verletzung von Datenzugriffsrechten, log. Integritätsbedingungen oder Plattenzugriffsstörungen. Als Folge kann die gesamte Transaktion abgebrochen werden, muß aber nicht (protokollabhängig). Hingegen wird ein globaler Abbruch einer Transaktion (i.d.R. wegen deadlock) nicht betrachtet: Die vergleichende Bewertung beschränkt sich auf deadlockfreie Verfahren.

4.2.4 Nicht betrachtete Ausfallklassen

Nicht berücksichtigt werden

- Fehlerhafter Nachrichtenaustausch
- Ausfälle von Massenspeicher, Terminals, Leitungen
- Inkorrekte entworfene bzw. implementierte Software

Nachrichtenaustausch-Fehler

Verschiedene Rechnernetz-spezifische Fehlerquellen können den korrekten Nachrichtenaustausch beeinträchtigen. Mögliche Quellen sind

- Synchronisationsverlust zwischen verteilten kooperierenden Prozessen
- Inkorrekter Entwurf oder Implementierung von Kommunikationsprotokollen
- Nachrichten- (Paket-) Verlust z.B. wegen Netzüberlastung
- nicht behandelte Verklemmung bei der Packetvermittlung
- Verfälschung von Nachrichteninhalten durch 'Kanalrauschen'

Die Behebung solcher Fehler kann in offenen Kommunikationssystemen gemäß dem Standard einer sicheren Transportschicht (Klasse 4) realisiert werden /ISO84a/. Aufbauend darauf wird im nächsten Kapitel ein sicherer IPC-Dienst modelliert, der solche Fehler vor oberen Schichten, insbesondere Funktionen der Transaktionsverarbeitung, maskiert. Im einzelnen wird in Anlehnung an den ISO-Vorschlag folgendes garantiert:

(1) Keine Nachrichten-Verfälschungen (erkannt und beseitigt durch z.B. cyclic-redundancy-checks);
(2) Keine Nachrichten-Fehladressierungen (ebenfalls durch solche Checks);
(3) Nachrichten zwischen zwei Kommunikationspartnern kommen in derselben Reihenfolge wie abgesendet an (d.h. Erzwingen der FIFO-Kanaleigenschaft, i.d.R. durch Nachrichten-Sequenznummern);
(4) Keine Nachrichten-Duplikate (erkannt und beseitigt durch Sequenznummern);
(5) Nachrichten zwischen nicht ausgefallenen und durch mind. einen physischen Weg verbundenen Knoten gehen nicht verloren (erzwungen durch adaptive Wegeermittlung und durch "Retransmit

upon timeout"-Mechanismus).

Eine realitätsnahe Modellierung der IPC-Schicht kann den durch diese Fehlerarten bewirkten höheren Kommunikationszeiten z.B. durch höhere Varianz dieser Zeiten berücksichtigen.

Ausfälle von Massenspeicher, Terminals, Leitungen

Ausfälle von Hintergrundspeichern werden nicht betrachtet, da sie durch geeignete Maßnahmen praktisch ausgeschlossen werden können. Dazu kann ein "stable storage" Konzept verwendet werden: Jede Dateneinheit (oder eine feinere Granularität wie etwa ein Satz) wird redundant auf zwei bezüglich ihrer Ausfallcharakteristik unabhängigen Speichermedien gehalten. Datenänderungen werden "vorsichtig" gemacht: Zuerst wird das Datum auf dem ersten Medium geändert und die Korrektheit geprüft, dann erst wird es auch auf das zweite Medium geschrieben.
Üblich ist, als Speichermedium Platten zu verwenden, man spricht dann von Spiegelplatten. Damit wird Ausfallsicherheit bezüglich einzelnen Plattenhardware-Fehlern garantiert (d.h. es gilt REL(1)). Falls zusätzlich unabhängige phys. Zugriffswege auf beide Platten realisiert werden, läßt sich sogar AVA(1) garantieren. Unter Verwendung heutiger Technologie erreicht man damit eine MTTF von ca. 800 Jahren /GRAY81/.
Stable Storage erleichtert speziell die Modellierung von Wiederanlaufverfahren für Transaktionsverarbeitung: Hierzu wird bzgl. der Restaurierbarkeit aktueller Verarbeitungszustände von Transaktionen angenommen "the log never fails", d.h. stable storage wird als plattenfehlermaskierend angenommen.

Ausfälle anderer Harewarekomponenten werden ebenfalls als unwahrscheinlich betrachtet, sodaß eine Modellierung nicht erfolgt.

Softwareentwurfs- und Implementierungsfehler

Sowohl fehlerhafte System-SW (Betriebssystem, VDBS, Kommunikationssystem) als auch fehlerhafte Anwendungs-SW können einen temporären Ausfall von Betriebssystem u/o DBVS zur Folge haben kann.
Softwarefehler sind bereits in der Ausfallrate der Arbeitsrechner berücksichtigt; interne Auswirkungen auf das DBVS werden nicht modelliert. Es wird trotz möglicher Fehler angenommen, daß Transaktionsprogramme logisch korrekt sind, d.h. keinen inkonsistenten Datenbasiszustand hinterlassen.

4.3 Operationales Modell der verteilten Prozess-Kommunikation (IPC)

Ein verteiltes Prozeßsystem, bestehend aus Transaktions-Managern (TM), Daten-Managern (DM) und den lokalen Datenverwaltungssystemen, kooperiert mit Hilfe von Nachrichtenaustausch entsprechend den Regeln eines Protokolls zur Abarbeitung von verteilten Transaktionen. Die Kooperation dieser lokalen und nichtlokalen VDBS-Prozesse muß durch einen Dienst zur Interprozeß-Kommunikation, dem **IPC-Dienst**, unterstützt werden. Damit wird eine einheitliche Modellierungsumgebung für eine große Klasse existierender VDBS-Protokolle festgelegt.
Heutige Kommunikationssysteme bieten einen für **transaktionsorientierte** verteilte Anwendungen geeigneten IPC-Dienst meist nicht an, da die gängigsten verteilten Anwendungen **sitzungsorientiert** sind (z.B. remote job entry) und insbesondere keine kooperierenden 'Anwendungs'-Prozesse kennen. Daher verwenden die meisten realisierten VDBS **maßgeschneiderte IPC-Dienste**. Eine Standardisierung der erforderlichen Dienste ist bisher nicht erfolgt.
Einigkeit besteht aber dahingehend, daß der IPC-Dienst mindestens auf der Kommunikationssteuerungs-Schicht (OSI-Schicht 5, vergl. /SCHN83/) anzusiedeln ist. Abb. 4.6 enthält einen Vorschlag der Einbettung der Interprozeß-Kommunikation in die ISO-OSI-Architektur, der im folgenden im Detail spezifiziert wird.

4.3.1 Kommunikationsanforderungen von VDBS-Prozessen

Bei der Spezifikation des IPC-Dienstes für VDBS-Prozesse gehen wir von folgenden Anforderungen aus:

(1) Einheitlichkeit: Die IPC-Dienstprimitive sollen zur Kommunikation sowohl zwischen verteilten als auch zwischen lokalen Prozessen geeignet sein.

(2) Vollständigkeit und Minimalität: Gesucht ist eine minimale Anzahl von Dienstprimitiven, die in ihrer Funktionalität alle Kommunikationsanforderungen von VDBS-Protokollen erfüllen.

(3) Unabhängigkeit vom Nachrichtentransportdienst: Der IPC-Dienst soll die VDBS-Prozesse unabhängig vom der Charakteristik der

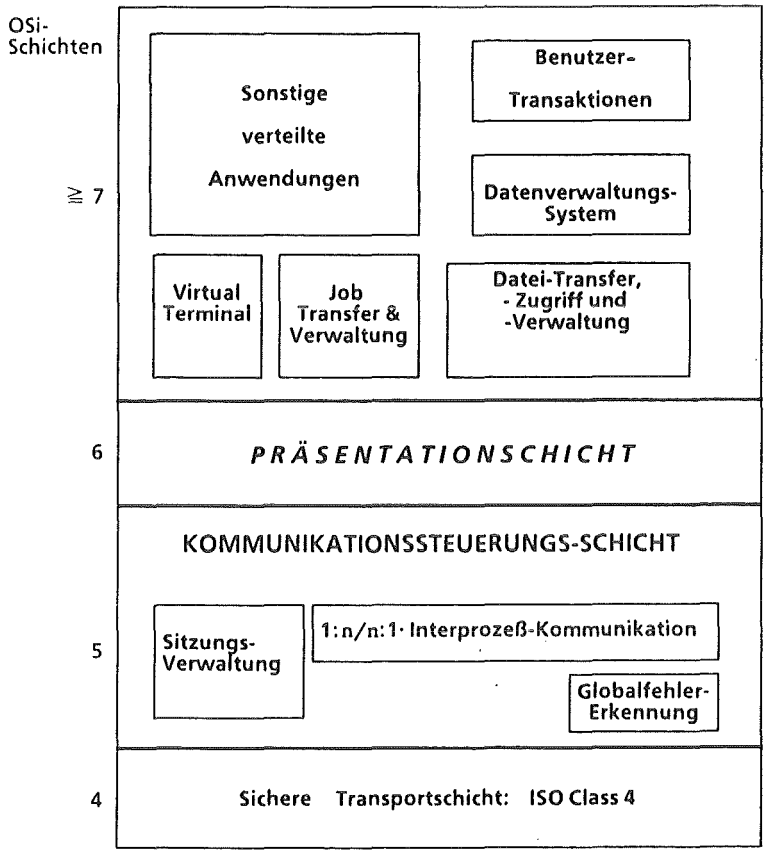


Abb. 4.6: Einbettung der Interprozeßkommunikation in die Verteilte Systemarchitektur

darunterliegenden Transportschicht machen. Es wird angenommen, daß der IPC-Dienst einen **sicheren Nachrichtenaustausch** (Kap. 4.2.4) zwischen Arbeitsrechnern garantiert. Insbesondere soll die Realisierung des IPC-Dienstes sowohl auf einem verbindungslosen Datagramm-Dienst der Transportschicht /DAVS79/, als auch auf einer verbindungsorientierten Transportschicht /ISO84a/ möglich sein. im letzteren Fall kann der sichere Nachrichtenaustausch von der Transportschicht realisiert werden, andernfalls von der darüberliegenden Schicht.

Eine effiziente Realisierung von Kommunikation zwischen verteilten Prozessen auf der Transportschicht, wie z.B. des nachrichtenintensiven TVPs, ist nicht möglich, wenn bei einem verbindungsorientierten Transport für jeden Nachrichtenaustausch

Verbindungen auf- und abzubauen sind. Daher ist ein Verbindungsaufbau und -abbau zwischen VDBS-Prozessen nur bei Systeminitialisierung und nach Wiederanlauf einzelner Rechner sinnvoll. Bei unserer Modellierung des IPC-Dienstes wird ein Verbindungsaufbau und -abbau nicht berücksichtigt; es handelt sich damit um einen 'direkten' IPC-Dienst /PESI83/.

(4) Unabhängigkeit vom lokalen Betriebssystem: Der IPC-Dienst soll offenlassen, ob für jede neue Transaktion dynamisch neue TM- und DM-Prozesse generiert werden, oder ob diese VDBS-Prozesse als mehrfach aktivierbare **Dienstprozesse** realisiert werden. Ein Dienstprozeß kann zu einem Zeitpunkt mehrere Transaktionen quasi-parallel bearbeiten; diese können verschieden weit fortgeschritten sein. Unser IPC-Modell sieht keine Primitive zur Kreierung und Zerstörung von VDBS-Prozessen vor.

(5) Mächtigkeit: Die Komplexität von VDBS-Protokollen erfordert einen 'höheren' IPC-Dienst, als ihn die meisten (zentralen und verteilten) Betriebssysteme heute anbieten. Konkret wird gefordert:

Remote Procedure Call

Bei der Kommunikation zwischen VDBS-Prozessen besteht die Semantik einer empfangenen Nachricht meist in der Aufforderung, eine bestimmte Aktion auszuführen und die erfolgreiche Ausführung dem Absender zu quittieren bzw. als Quittung die Ergebnisse der Ausführung zu übermitteln. Diese Kommunikationsart ist mit üblichen Sendeoperationen speziell dann kaum realisierbar, wenn der Sendeprozeß ein Ergebnis auf jeden Fall erwartet, auch wenn der Zielrechner während der Ausführung ausfällt. D.h. erforderlich ist, daß die IPC-Schnittstelle in endlicher Zeit entweder im Normalfall das Ergebnis der Anforderung liefert, oder im Fehlerfall den Sendeprozeß von der **Nichtverfügbarkeit** des Zielrechners informiert. Diese Informierung soll auch dann erfolgen, wenn der Zielrechner erst nach Erhalt der Anforderung unverfügbar werden sollte, ohne daß der sendende Benutzerprozeß etwa zyklisch den Zielrechnerstatus abzufragen hätte. Dieser Mechanismus wird als 'remote procedure call' oder als 'request/reply'-Mechanismus bezeichnet. Voraussetzung zu deren Realisierung ist eine integrierte

Ausfall/Wiederanlauferkennungs-Instanz:

Erkennung und Meldung des globalen Systemzustands

(Verfügbarkeit / Unverfügbarkeit entfernter Arbeitsrechner).
Die IPC-Schicht (oder eine darunterliegende Schicht) soll eine entsprechende Instanz realisieren, die auf der Basis ihrer lokalen Sicht den Verfügbarkeitsstatus aller entfernter Arbeitsrechner verwaltet. Schnittstelle zu den 'Anwendungs'-Prozessen sind Dienstprimitive, die die (Nicht-)Verfügbarkeit eines Arbeitsrechners melden. Eine Integration in den request/reply Mechanismus ist ebenfalls vorzusehen. Für IPC-Dienstbenutzer soll von Realisierungsaspekten der Instanz abstrahiert werden; insbesondere ist die Spezifikation von Zeitüberwachungsschranken (time-outs) für Verfügbarkeitstests innerhalb des IPC-Protokolls vorzunehmen.

1:n/n:1-Kommunikation

Zur Transaktionsverarbeitung kooperieren ein Transaktionsmanager mit $n \geq 1$ Datenmanagern auf Arbeitsrechnern, die die benötigten Dateneinheiten besitzen. In gewissen Ausfallsituationen kommunizieren auch die Datenmanager untereinander. Die Menge der Kooperationspartner variiert bei verschiedenen Transaktionen, sodaß die Anwendung einer 1:1-Sendeoperation zu wesentlich komplexeren VDBS-Protokollen führen würde. Daher soll der IPC-Dienst 'Multicast'-Primitive anbieten, deren Realisierung möglichst durch eine geeignete Transportschicht zu unterstützen ist. Diese Schicht wird durch eine NIU realisiert. Eine NIU soll ein effizientes 1:n-Senden ermöglichen, um die Kommunikationskosten für verteilte Transaktionen zu begrenzen. Die Forderung lautet, die Übertragungszeit für n Nachrichten in der gleichen Größenordnung wie für eine einzelne Nachricht zu halten. Ein IPC-Protokoll, das diese Leistung annähernd erreicht, ist z.B. im VDBS LAMBDA realisiert worden /CHNG84/.
Eine mächtige Erweiterung des zuverlässigen 1:1-request/reply erhält man durch 1:n-request / n:1-reply Primitive.

Selektiver Nachrichtenempfang und verteilte Adresshierarchie

Zur 'direkten', datagramm-orientierten Kommunikation zwischen transaktionsverarbeitenden Prozessen reicht die Addressierung eines netzweit eindeutigen Zielprozesses nicht aus. Ein VDBS-Prozeß wie z.B.

ein Transaktionsmanager kann zu einem Zeitpunkt mehrere Transaktionen in Bearbeitung haben. Benötigt wird also ein netzweit eindeutiger Bezeichner für jede Transaktion. Dieser ist als 'message tag' jeder Nachricht des Protokolls anzuhängen. Der Zielprozeß der Nachricht wartet **selektiv** auf das Eintreffen einer Nachricht mit genau diesem Bezeichner.

Weiterhin kann die Bearbeitung einer Transaktion durch einen VDBS-Prozeß verschieden weit fortgeschritten sein. Für Nachrichten zwischen VDBS-Prozessen wird also genau eine **Mailbox** für jeden Empfangszustand benötigt, in die eine Nachricht einer Transaktion in einem wohldefinierten Bearbeitungsstatus innerhalb eines VDBS-Prozesses geschickt wird. Da eine Transaktion sich in höchstens einem Bearbeitungszustand innerhalb eines Prozesses befinden kann, gibt es für jeden Bearbeitungszustand höchstens eine Mailbox, in die Nachrichten an diese Transaktion innerhalb eines Prozesses zu schicken sind. Da mehrere Transaktionen sich in demselben Zustand befinden können, kann eine Mailbox gleichzeitig mehrere Nachrichten enthalten, wobei jede Nachricht jeweils für eine unterschiedliche Ziel-Transaktion bestimmt ist. Kommunizierende Prozesse sind also aus VDBS-Sicht Kooperationspartner einer bestimmten Transaktion.

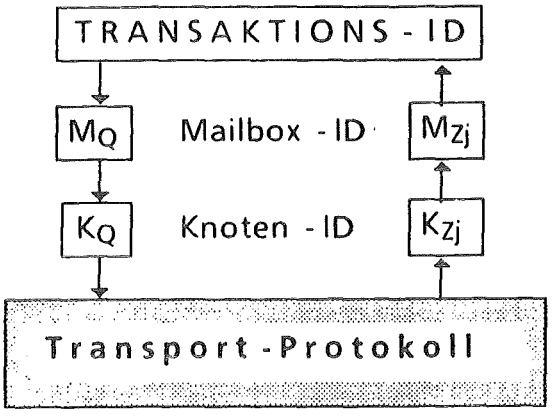


Abb. 4.7: Die VDBS-Adreßhierarchie

Die Grundlage der verteilten Transaktionsverarbeitung ist eine **verteilte Adresshierarchie**, wie in Abb. 4.7 gezeigt wird. Ein Transaktions-ID wird netzweit eindeutig durch ein 2-Tupel, bestehend aus seinem Heimatknoten HOME und einer monoton steigenden Sequenznummer (Zeitstempel) realisiert. M_Q ist die Mailbox der vom Sendeprozeß bearbeiteten Transaktion, auf den die Transaktion auf ein

'reply' (Antwort vom Zielprozeß) wartet. M_Z ist die Zielmailbox derselben Transaktion, die vom Zielprozeß bearbeitet wird oder werden soll. Mailboxes sind innerhalb eines Knotens eindeutig definiert; jede Mailbox gehört dabei genau einem Prozeß. Damit entfällt die explizite Adressierung von Quell- bzw. Zielprozeß. K_Z und K_Q sind netzweit eindeutige Ziel- und Quellknoten-IDs, wobei K_Q nicht notwendig identisch mit HOME ist. Ein reservierter Bezeichner OWN_NODE dient zur Identifizierung lokaler Nachrichten.

Die geforderte IPC-Dienstschnittstelle ist auf verteilte transaktionsorientierte Systeme zugeschnitten, aber allgemein genug auch für viele andere verteilte Anwendungen. Ein zuverlässiger IPC-Dienst für VDBS-Anwendungen als Teil der Schicht zur Kommunikationssteuerung wurde auch in /BRLE82/, /MOPO82/ gefordert. Einen Vorschlag zur Implementierung der Erkennung der Nichtverfügbarkeit entfernter Knoten enthalten /TOMA83/, /WALT82/; letzterer ist effizient im VDBS "DDM" /BERN83a/ implementiert. Realisierungen 'höherer' IPC-Primitive zum Mehrfach-Senden / -Empfangen sowie selektiven Senden / Empfangen (z.B. mittels Transaktions-IDs als 'message tag') sind beschrieben in /BEGI84, LIND83, NIGA81, TOMA83/.

4.3.2 Spezifikation der IPC-Dienstprimitive

Verteilte Algorithmen behandeln Nachrichten als höhere parametrisierte Objekte oder Operationen. Nachrichten werden in eine begrenzte Anzahl von Nachrichtentypen eingeteilt, die im wesentlichen durch die Primitive zur Interprozeß-Kommunikation bestimmt sind.
Der IPC-Dienst besteht aus folgenden parametrisierten Primitiven:

SEND_MSG (T; Z_Q; M_Z, AM, Nutzdaten)

REQU_MSG (T; Z_Q; M_Z, AM, Nutzdaten)
returns ((AR_1,$reply_1$), ... , (AR_n,$reply_n$)) **at** M_Q

REQU_REPLY(T; Z_Q; M_Z, AM, Nutzdaten)
returns ((AR_1,$reply_1$), ... , (AR_n,$reply_n$)) **at** M_Q

SEND_REPLY(T; Z_Q; M_Z, AM, Nutzdaten)

WATCH_AV (T; Z_Q; AM) **returns** (AR_i, 'AV') **at** M_Q

STOP_WATCH(T; M_Q)

GET_STATUS(T; Z_Q; AM) **returns** ($STATUS_1$, ... ,$STATUS_n$) **at** M_Q

mit AM: Zielknotenmenge {AR_1, ... ,AR_n}
T: Transaktions-ID
AV: Verfügbarkeitsmeldung für einen Arbeitsrechner

Die Nutzdaten einer Nachricht enthalten die zwischen IPC-Dienstbenutzern auszutauschende Information. Konventionen zur Syntax und Semantik der Nutzdaten werden deshalb allein von kommunizierenden VDBS-Prozessen festgelegt und sind daher für IPC-Dienst und -Schicht nicht relevant.

Mit SEND_MSG werden n≥1 Nachrichten abgesendet. Es handelt sich um ein nichtblockierendes, 'unzuverlässiges' Senden. Der Sendeprozeß erwartet keine Antworten. Diese Operation wird verwendet, wenn ein Warten auf Antwort von der Ziel-Transaktion nicht notwendig ist. Ist der Zielrechner nicht verfügbar, geht die betreffende Nachricht verloren, ohne daß der Sendeprozeß davon erfährt.

Mit REQU_MSG, REQU_REPLY und SEND_REPLY werden 'remote procedure calls' mit der Erweiterung auf 1:n/n:1-Kommunikation und integrierter Ausfallerkennung realisiert:

- Mit REQU_MSG wird die Sende-Transaktion blockiert, bis 'replys' von **allen** n Zielprozessen eingetroffen sind, d.h. die Transaktion erwartet an der Quellmailbox n 'Replys'. Diese Zielprozesse werden bezüglich derselben Transaktion erst bei Empfang des Requests aktiviert. Zu beachten ist, daß entsprechend unserem Modell der verteilten Transaktionsverarbeitung die n Zielprozesse stets auf **n unterschiedlichen Rechnern** liegen.

- Mit REQU_REPLY wird eine Art der 'Ping-Pong' Kommunikation realisiert: Der Quellprozeß verschickt 'Requests' an n

Zielprozesse. Die Quell-Transaktion wird blockiert, bis alle n 'Replys' eingetroffen sind. Umgekehrt erwartet jeder Zielprozeß diesen 'Request' als 'Reply' auf einen vom ihm vorher abgesetzte REQU_REPLY. Die Verarbeitung der Transaktion im Zielprozeß ist blockiert, bis der jeweilige 'Request' an der Zielmailbox eingetroffen ist.
Die VDBS-Prozesse Transaktionsmanager und n Datenmanager kommunizieren im fehlerfreien Fall meist durch wechselseitige REQU_REPLY-Operationen, nachdem der 'Ping-Pong'-Mechanismus anfangs vom Transaktionsmanager durch die REQU_MSG-Operation an alle Partner-Datenmanager initiiert wurde.

- Mit SEND_REPLY wird eine Folge von wechselseitigen REQU_REPLY-Operationen beendet: Im Gegensatz zu REQU_REPLY erwartet die Quell-Transaktion keine 'Replys'. Damit kann ein Kommunikationspartner von seiner Seite aus die Transaktion terminieren. Ist der Zielknoten nicht verfügbar, dann geht die betreffende Nachricht verloren, ohne daß die Quell-Transaktion davon erfährt.

Abb. 4.8 zeigt das Prinzip der Mailbox-Kommunikation mit den genannten IPC-Primitiven an einem einfachen Protokoll zwischen zwei zyklischen Partnerprozessen P1, P2. Ausfallfreie Kommunikation wird hier vorausgesetzt. Prozeß P1 aktiviert P2 durch die Operation REQU_MSG. Beide kommunizieren anschließend über n≥0 REQU_REPLYs. Das Protokoll wird von P2 durch SEND_REPLY beendet. Ein erneuter Protokollstart kann durch die Wartebedingung W ausgelöst werden. Die Anzahl initialer Marken auf Kanal K1 (K2) gibt die maximale Anzahl nebenläufiger P1- (P2-) Aktivierungen vor.

Bei den Operationen REQU_MSG, REQU_REPLY und SEND_REPLY wird eine Blockierung der Transaktion auf der Sender- u/o Empfängerseite in **endlicher Zeit** aufgehoben: Der IPC-Dienst wird so realisiert, daß eine wartende Transaktion in endlicher Zeit garantiert n Antwortnachrichten erhält. Wird ein Zielknoten AR_i unverfügbar wegen Arbeitsrechner- oder NIU-Ausfällen, dann erzeugt das IPC-Protokoll anstatt der erwarteten Antwort vom Zielprozeß die Meldung

$CP(AR_i)$ (CP = Crashed / Partitioned)

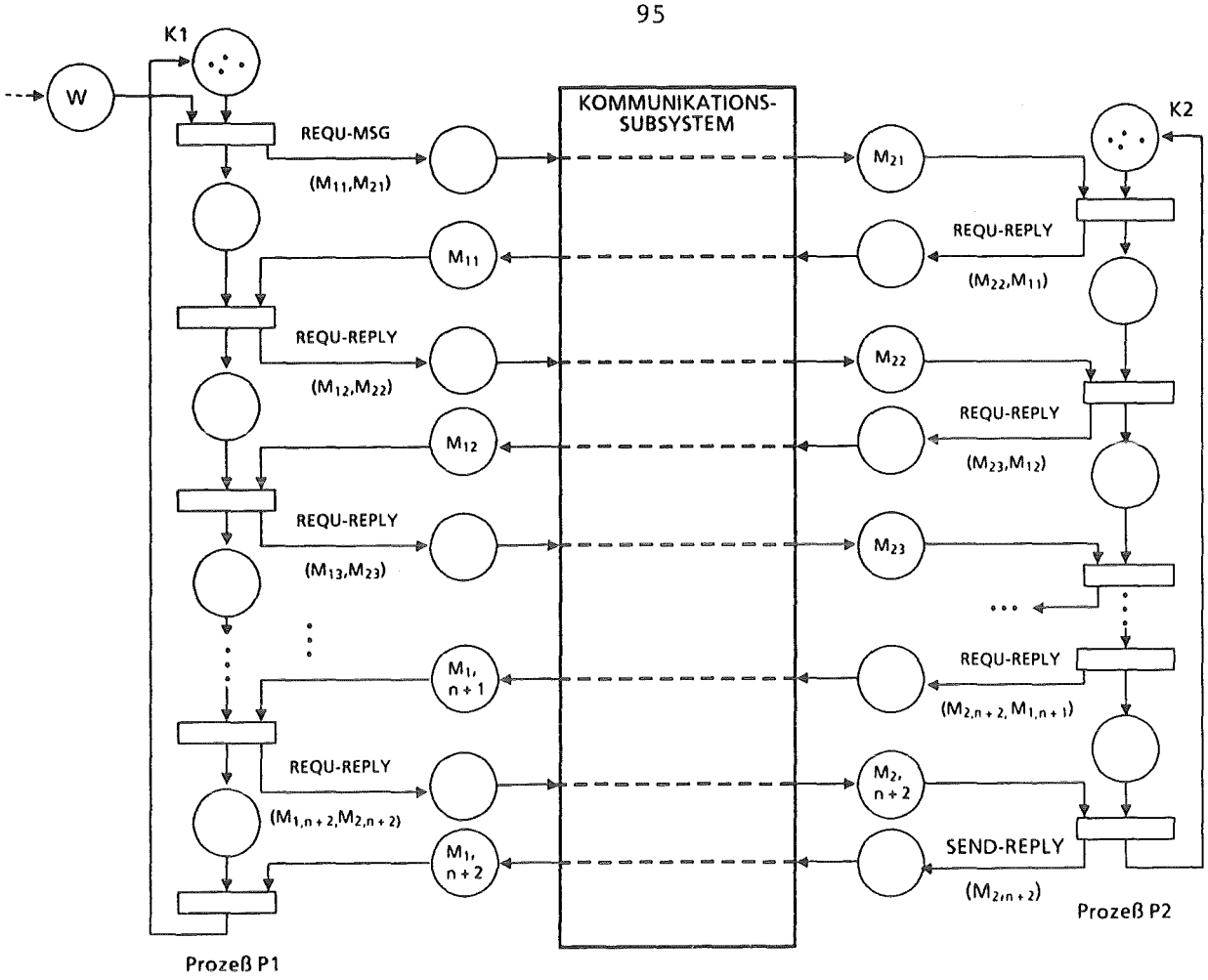


Abb. 4.8: Mailbox-Kommunikation zweier sequentieller, zyklischer Prozesse P1, P2 (Beispiel)

M_{ij} : j-te Mailbox von Prozeß P_i

op(MQ, MZ) : Aufruf IPC-Operation "op" mit
MZ: Ziel-Mailbox MQ: REPLY-Mailbox

und schickt diese an die anfordernde Mailbox.

Hat eine Transaktion diese Nichtverfügbarkeitsmeldung erhalten, hat sie die Option, auf die Wieder-Verfügbarkeit des betreffenden Arbeitsrechners durch Aufruf der Operation WATCH_AV zu warten. Bei einer Menge AM mehrerer unverfügbarer Arbeitsrechner AM = $\{AR_1,\ldots,AR_n\}$ (n>1) kann durch einen WATCH_AV-Aufruf auf die Wiederverfügbarkeit aller dieser Rechner gewartet werden. Die Quell-Transaktion wird von der (Wieder-)Verfügbarkeit jedes Arbeitsrechners $AR_i \in AM$ durch die Meldung $AV(AR_i)$ **asynchron** benachrichtigt, d.h. jede Wiederverfügbarkeit erzeugt eine Meldung an die Transaktion. Eine n:1-Sammelmeldung, d.h. das Warten auf Verfügbarkeit **aller** angesprochenen Zielrechner ist nicht immer

sinnvoll, da dann die Wartezeit abhängig von Intakt- und Reparaturzeiten im Rechnernetz wäre. Eine darauf wartende Transaktion würde unnötig lange blockiert, obwohl sie evtl. bereits mit einer Teilmenge der angeforderten Rechnerzustände fortgesetzt werden kann. Hingegen wird die Wartezeit auf alle Antworten eines mehrfachen remote procedure call der wartenden Transaktion zugemutet.
Mit STOP_WATCH wird das Warten auf jegliche Verfügbarkeitsmeldungen für eine vom Prozeß p_Q bearbeitete Quell-Transaktion beendet, d.h. eine vorher aufgerufene WATCH_AV-Operation wird so vorzeitig abgebrochen.
Mit GET_STATUS wird der aus Sicht des Quellknotens aktuelle Verfügbarkeits-Zustand aller angeforderter entfernter Arbeitsrechner sofort gemeldet.

Ein Beispiel des dynamischen Ablaufs der Interprozeß-Kommunikation zwischen zwei auf unterschiedlichen Rechnern liegenden Prozessen ist in Abb. 4.9 als vereinfachtes Zeitdiagramm gegeben. Der Prozeß auf Rechner AR_1 (z.B. ein TM) kommuniziert dabei mit n anderen Prozessen. Einer dieser Prozesse (z.B. ein DM) befindet sich auf Rechner AR_2.

4.3.3 Modellierung der IPC-Schicht

Die IPC-Schicht wird in Abb. 4.10 als Teil einer ISO-OSI-ähnlichen Schichtenarchitektur gezeigt (vergl. Abb. 4.2 als Vergröberung von Abb. 4.10). Der IPC-Dienst wird dabei an den FIFO-Schnittstellenports ρ(PROZESSE,IPC) und ρ(IPC,PROZESSE) erbracht; diese Ports bilden im Sinne von ISO-OSI **Dienstzugangspunkte**. Diese bestimmen die Interaktionen zwischen Kommunikationsinstanzen benachbarter Schichten. Die Modellierung mit Petrinetzen erfolgt so, daß jede Kommunikationsinstanz durch ein **stellenberandetes Teilnetz** beschrieben wird, wobei die Randstellen die Schnittstellen der Instanz zur über- und untergeordneten Schicht definieren. Die Menge der Randstellen zur übergeordneten Schicht ist genau die Menge der Dienstzugangspunkte dieser Schicht /BUEC82/.
Die IPC-Schicht auf einem Knoten besteht aus Sende- und Empfangsinstanzen. Instanzen auf verschiedenen Knoten kooperieren gemäß den formalen Regeln des **IPC-Protokolls**, um den verlangten IPC-Dienst für die den Dienst nutzenden VDBS-Prozesse zu erbringen.

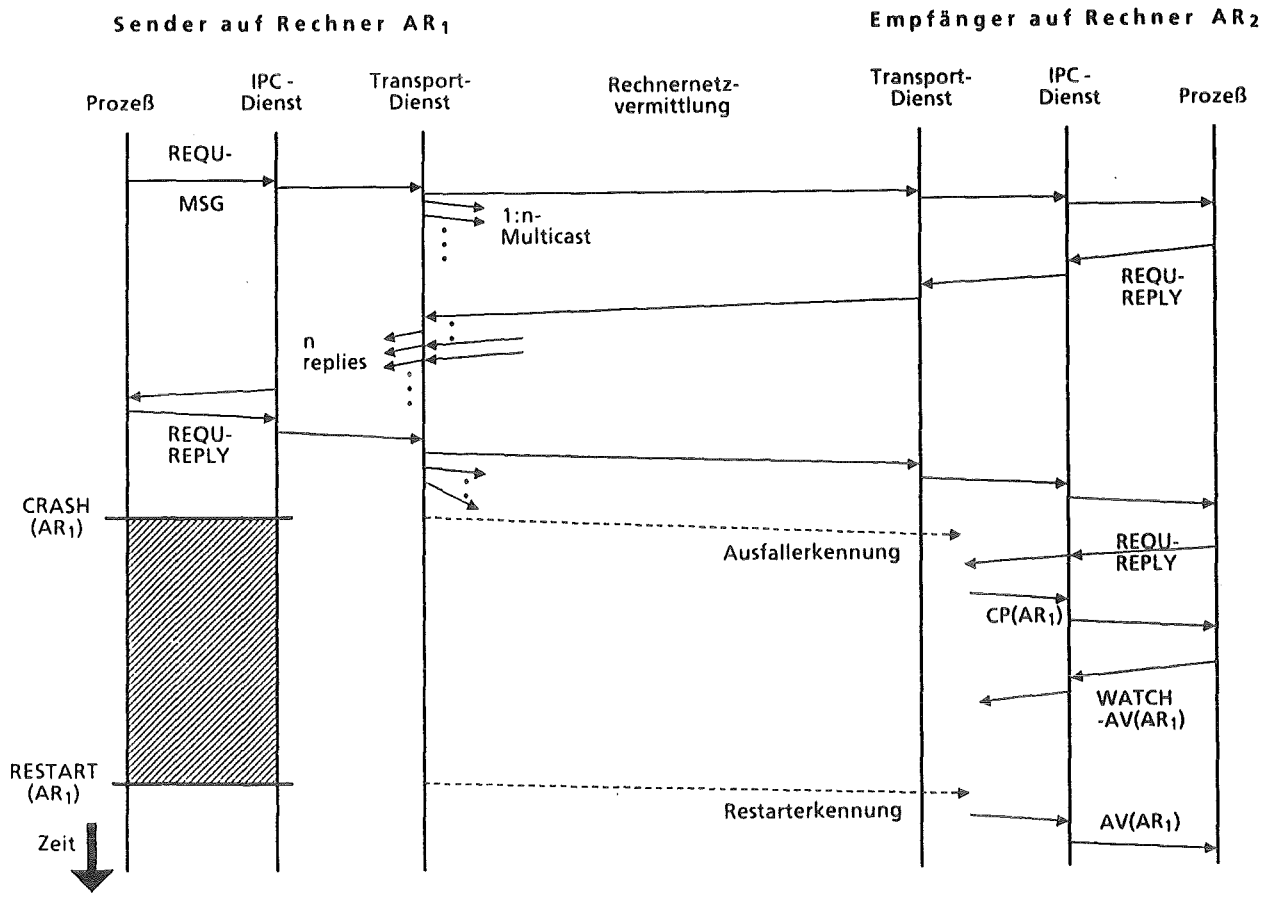


Abb. 4.9: Scenario zweier kommunizierender Prozesse

Auf der lokalen Network Interface Unit (NIU) sind die Transportinstanz und alle unteren Instanzen realisiert. Die Transportinstanz leitet lokale Nachrichten an die IPC-Empfangsinstanz weiter; Nachrichten an entfernte Knoten werden durch eine 'Multicast'-Operation unterstützt. Die integrierte Instanz zur Erkennung und Meldung von Verfügbarkeitsänderungen entfernter Knoten stellt die entsprechenden Meldungen der IPC-Schicht zur Verfügung.

Ein wichtiges Problem bei der transaktionsorientierter Kommunikation mit Hilfe von Funktionsnetzen bildet die Modellierung des **stochastischen Zeitverhaltens** des VDBS und seiner Umgebung: Die Abarbeitung von Transaktionen bzw. Subtransaktionen entsprechend einer FIFO-Warteschlangendisziplin ist nicht möglich, da Transaktionen, die durch den gleichen Dienstprozeß bearbeitet werden, sich gegenseitig "überholen" können. Beispiel hierfür ist die **selektive Empfangsoperation**, bei der die zeitliche Ankunftsreihenfolge von zwei n:1-Sammelmeldungen für unterschiedliche Transaktionen nicht

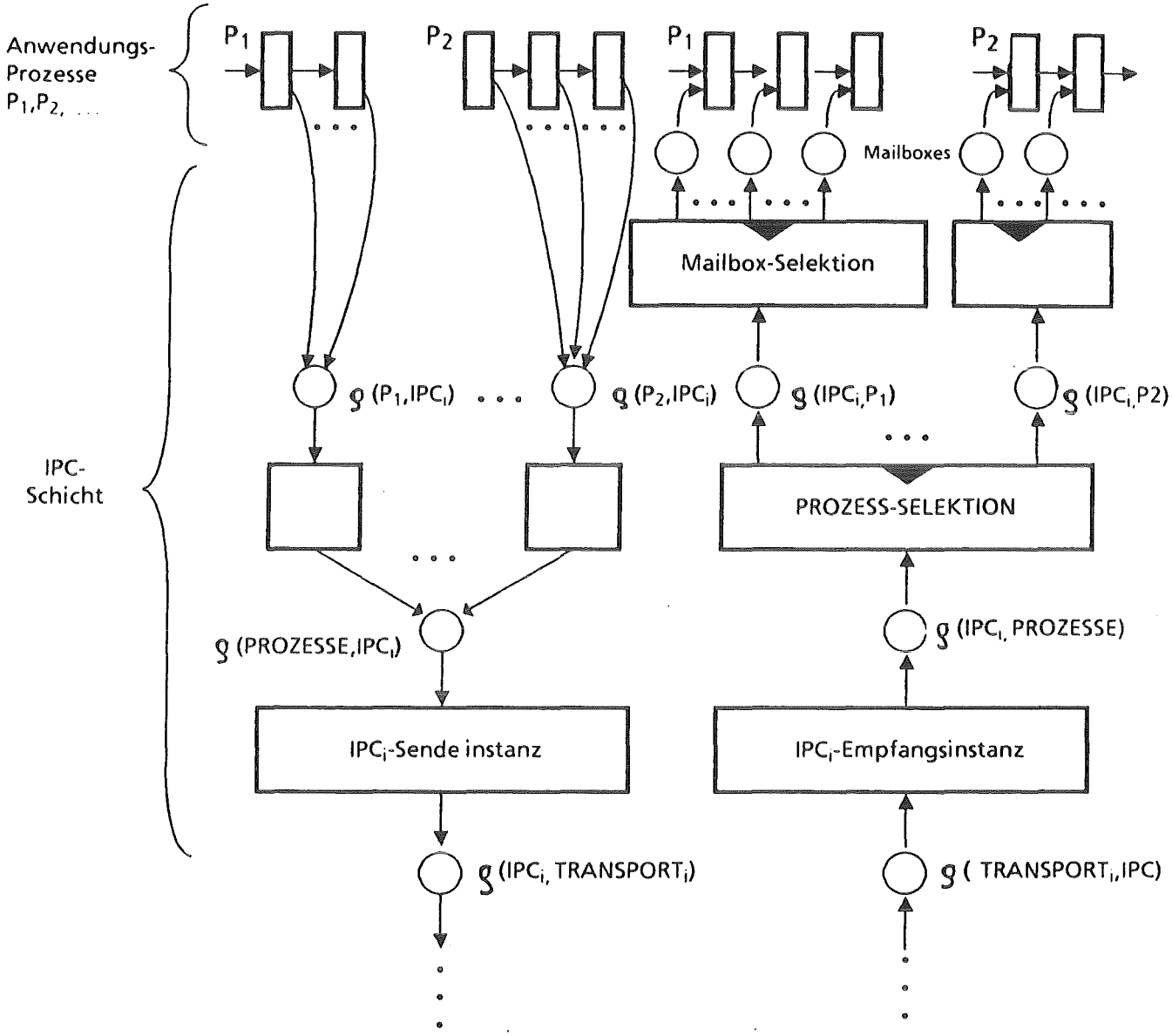


Abb. 4.10: Modell der Kommunikation zwischen VDBS-Prozessen
(Beispiel mit sequentiellen Prozessen P_1, P_2)
$\rho(S_i,S_j)$: FIFO-Sendeport von Schicht$_i$ zu Schicht$_j$

vorhersagbar ist. Wie auch in der Realisierung ähnlicher IPC-Protokolle in /TOMA83/, wird selektives Warten durch einen 'wahlfreien' Mengenzugriff auf strukturierte Marken mit Hilfe eines **Auswahlprädikats** realisiert, das gleichzeitig als Schalt-Vorbedingung der wartenden Instanz aufzufassen ist. Der in den strukturierten Marken gespeicherte Transaktions-ID "TID" dient dabei als Marken-Schlüssel. Die Modellierung einer Empfangsoperation, zusammen mit den zum selektiven Markenzugriff erforderlichen Erweiterungen des Funktionsnetz-Konzepts, sind in Abb. 4.11 graphisch veranschaulicht.

Ein analoges Problem ergibt sich beim Zugriff auf die Wartemengen

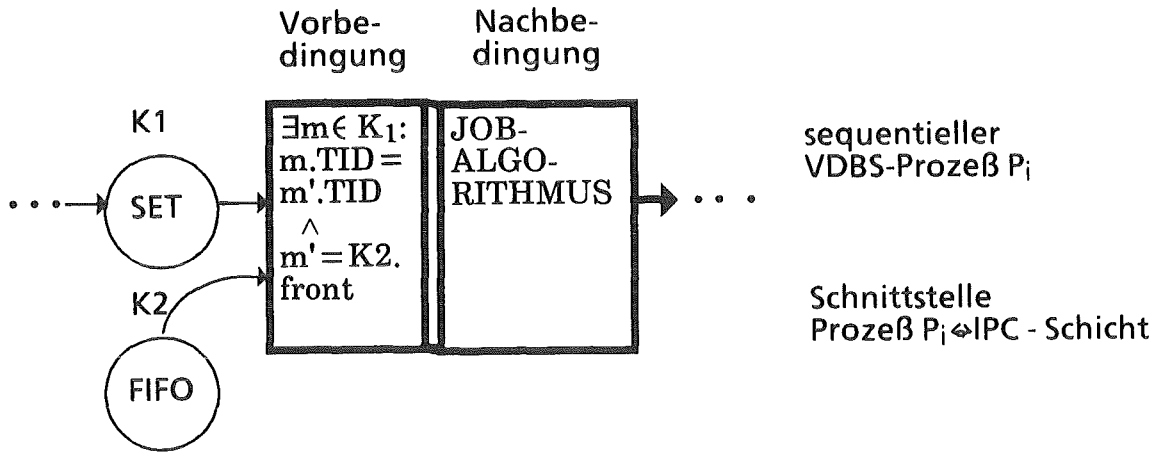


Abb. 4.11: Selektiver Nachrichtenempfang mittels Transaktions-ID

innerhalb des IPC-Protokolls. In Abb. 4.12 ist der lokale Nachrichten- und Kontrollfluß des IPC-Protokolls und des Transportprotokolls zur Erbringung des IPC-Dienstes als erweitertes Funktionsnetz dargestellt, welches nun erläutert wird.

- Eine Nachricht 'MSG' von einem lokalen VDBS-Prozeß kommt am Port ρ(PROZESSE, IPC) an. Die IPC-Sendeinstanz leitet MSG an die Transportinstanz weiter ((1) in der Abbildung), falls MSG nicht vom Typ STOPWATCH ist.
- Die Transportinstanz transferiert die an einen lokalen Prozeß bestimmte Teilnachricht der Mehrfach-Nachricht MSG an den lokalen Port ρ(TRANSPORT,IPC), s. (2). Bei einer Nachricht vom Typ GET_STATUS wird eine Kopie des Verfügbarkeits-Statusvektors (s.u.) beschafft (3); dieser wird ebenfalls als Ergebnis an die lokale IPC-Empfangsinstanz transferiert.
  Eine Teilnachricht an einen Prozeß auf einem entfernten Rechner AR_i wird entsprechend obiger Spezifikation wie folgt behandelt. Zunächst wird der Verfügbarkeitsstatus von AR_i beschafft (3), der 'AV' (bei Verfügbarkeit) oder 'CP' (sonst) liefert.
  -- Falls MSG-Typ REQU_MSG oder REQU_REPLY und '$CP(AR_i)$', dann wird diese Nichtverfügbarkeit als Reply-Nachricht lokal weitergegeben (2), da ein Sendeversuch an einen unerrreichbaren Rechner sinnlos wäre.
  -- Falls MSG-Typ WATCH_AV und '$AV(AR_i)$', dann wird diese Verfügbarkeitsinformation als Reply lokal weitergegeben (2).

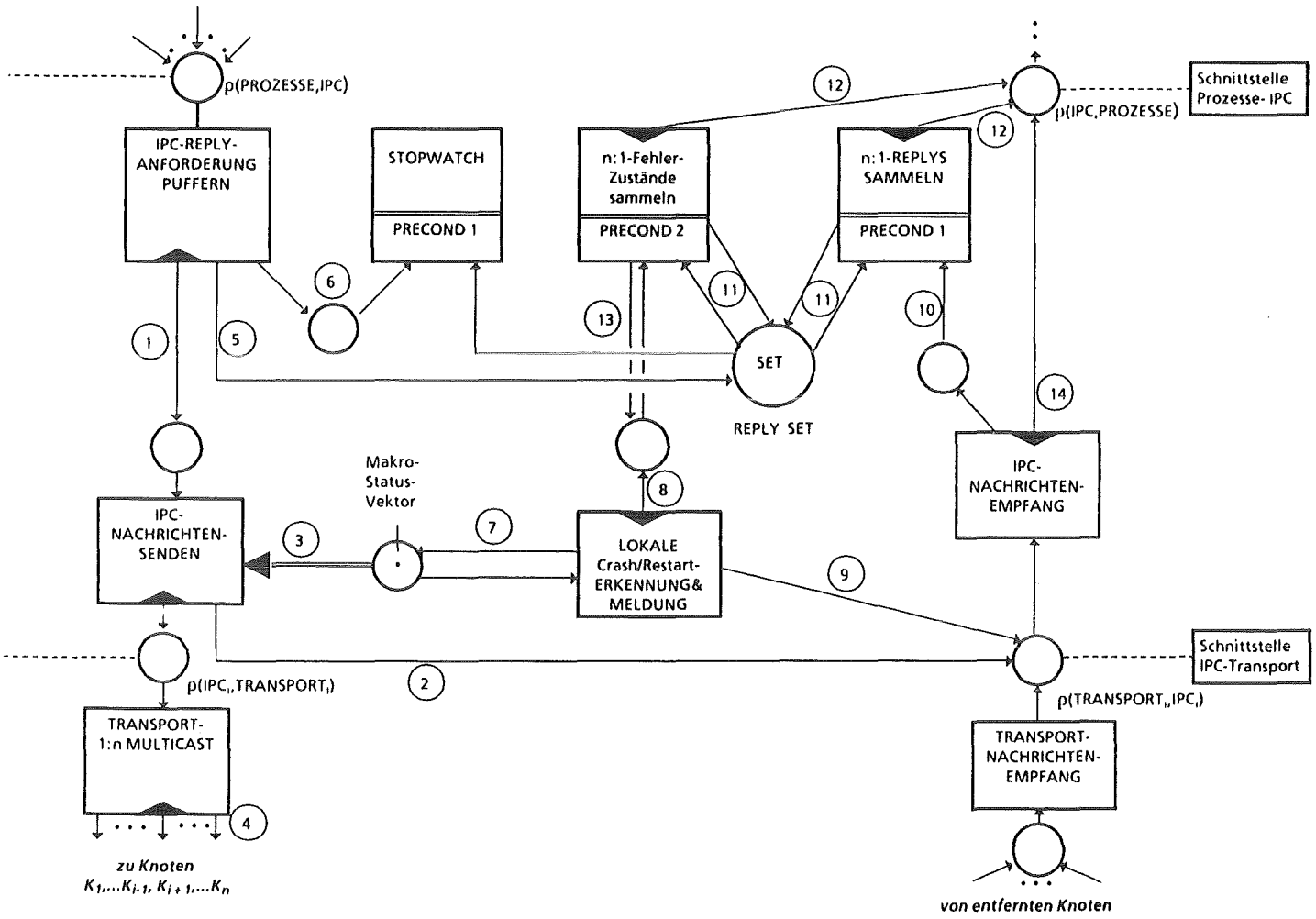


Abb. 4.12: Modell des Kommunikations-Subsystems
(Verfeinerung der IPC-Sende- und IPC-Empfangsinstanz in Abb. 4.10)

⟹ kopierendes Lesen aus Inputkanal

Selektionsbedingungen:

PRECOND1: **select** m∈REPLYSET: m.TID=m'.TID ∧ m.KQ=m'.KZ

PRECOND2: **select** m∈REPLYSET: ∃i∈[1,NKZ] :
(m'.STATUS(m.KZ(i)) = 'down' ∧ m.MSG_TYP≠WATCH_UP)
∨ (m'.STATUS(m.KZ(i)) = 'up' ∧ m.MSG_TYP=WATCH_UP)

mit NKZ: Anzahl Zielknoten, von denen REPLYs erwartet werden
KZ(1), ... , KZ(NKZ) : entsprechende Zielknotenmenge

-- Alle anderen Nachrichten werden an entfernte Rechner mittels einer **Multicast-Operation** übermittelt (4). Dabei ist ein Nachrichtenverlust nicht auszuschließen, falls die lokale NIU

während der Übermittlung oder z.B. die Ziel-NIU vor dem Empfang oder während des Empfangs der Teilnachricht ausfällt.

- Bei den MSG-Typen REQU_MSG, REQU_REPLY und WATCH_AV werden n≥1 **Antworten** (Replys) erwartet. Dazu speichert die IPC-Sendeinstanz die für eine an einer Mailbox MQ wartende Transaktion T bestimmte Wartemenge REPLYSET(T, MQ) in einen lokalen Puffer (5). Wartemengen dienen zum Sammeln von n Einzelnachrichten in eine aggregierte Sammelnachricht, die als Reply an (T,MQ) übermittelt wird (s.u.).
- Bei MSG-Typ STOP_WATCH wird ein bestimmter REPLYSET(T, MQ) vernichtet (6); die Transaktion T erwartet dann an der Mailbox MQ keine Replys mehr.
- Entsprechend dem lokalen Kenntnisstand der lokalen NIU wird ein **Verfügbarkeits-Statusvektor** entfernter Arbeitsrechner verwaltet und bei jeder entdeckten Statusänderung aktualisiert (7). Die Beziehung zwischen Ausfall und Wiederanlauf von Arbeitsrechnern und NIUs und zwischen den lokalen Kenntisständen und lokalen Meldungen wird durch die formale Spezifikation in Kap. 4.2.2 eindeutig festgelegt. Bei Wiederanlauf des eigenen Arbeitsrechners wird eine entsprechende RESTART-Meldung an den lokalen Logging-Prozeß geschickt (9). Jede erkannte AR_i-Statusänderung eines entfernten Arbeitsrechners wird als (IPC-schichtinterne) Statusmeldung $CP(AR_i)$ bzw. $AV(AR_i)$, $i \varepsilon [1,N]$ an die IPC-Instanz zum Eintragen von Statusänderungen in auf Replys wartende Ergebnismengen geleitet (8). Diese behandelt jede solche Meldung ähnlich wie einen Reply von einem entfernten Prozeß (10). Genauer gilt für alle lokal gespeicherten Wartemengen $REPLYSET(T,M_Q,\ldots,AR_i,\ldots)$, die auf eine Reply von AR_i warten:
  - Im Falle eines MSG-Typs SEND_REPLY ODER REQU_REPLY wird eine Statusänderung $CP(AR_i)$ als Reply in die Wartemenge gespeichert (s. (11)). Dabei ist sicherzustellen, daß die wartende Transaktion für jeden der n requests genau einen Reply erhält: Daher wird eine Statusänderung ignoriert, wenn der 'echte' Reply vorher schon eingetroffen ist und in der Wartemenge gespeichert wurde. Umgekehrt wird ein 'echtes' Reply vernichtet, wenn es erst nach Eintreffen und Speichern der Stautsänderung ankommt.

Die Wartemenge wird als Sammelantwort an (T,MQ) übergeben, falls alle n angeforderten 'echten' bzw. 'CP-' Antworten eingetroffen sind (12). Wesentlich dabei ist, daß entsprechend obiger Spezifikation des Request/Reply die n Antworten von **n unterschiedlichen Rechnern** erwartet werden.

-- Im Falle eines MSG-Typs WATCH_AV wird eine Statusänderung $AV(AR_i)$ sofort erzeugt und als Reply an (T,MQ) weitergegeben (s. (12)).

- Falls eine Statusänderung mehrere wartende Transaktionen betrifft, ist eine mehrmalige Aktivierung der Instanz erforderlich, die CP-/AV-Änderungen in REPLYSET einträgt (s. (13)).
- Alle ankommenden Nachrichten, auf die keine Transaktion T an einer Mailbox MQ wartet, also solche vom Typ RESTART, SEND_MSG und REQU_MSG, werden direkt zu der entsprechenden Zielmailbox m.MZ geleitet (14).

4.4 Modell der verteilten Daten und Anwendungen

Die Ergebnisse einer Leistungs- oder Verfügbarkeitsprognose eines Datenbanksystems werden maßgeblich von der **Arbeitslast** des Systems beeinflußt. Die Arbeitslast besteht aus dem **Datenprofil** (Charakterisierung der logischen und physischen Datenbankstruktur) und dem **Transaktionsprofil** (Charakterisierung des Verhaltens von Datenbankbenutzern).

Im nächsten Abschnitt werden zunächst die Begriffe Zentralisierung, Partitionierung, Replikation und Homogenität von Komponenten (Daten, Funktionen) eines Verteilten Systems formal definiert. Darauf aufbauend, werden anschließend Zuordnungsstrategien von Daten zu Knoten diskutiert und eine geeignete Strategie als Teil des Datenprofils des VDBS-Simulationsmodells entwickelt. Schließlich folgt die Modellierung des Transaktionsprofils.

4.4.1 Formale Beschreibung der Verteilung von Systemkomponenten

Gegeben sei

- Ein verteiltes System, bestehend aus $N \geq 2$ Knoten

  $VS := \{S_1, \ldots, S_N\}$

- Ein Subsystem SUB von VS, bestehend aus k funktional unterscheidbaren Komponenten

$$SUB := \{K_1, \ldots, K_k\} \quad (k \in \mathbb{N})$$

Beispiele für Subsysteme:

-- Menge aller Fragmente der Datenbasis

-- Menge aller VDBS-Funktionen

Auf SUB sei die **Komponentengleichheit** definiert:

Zwei Komponenten K_i, K_j sind genau dann **homogen**, wenn sie bezüglich ihrem funktionellen Verhalten nicht unterscheidbar sind. Bei gleicher Realisierung der Komponenten werden diese als **Kopien** derselben Komponente bezeichnet.

Im folgenden treffen wir keine Unterscheidung zwischen Kopien und homogenen Komponenten.

- Eine Funktion

$$C: SUB \rightarrow \mathbb{N}$$

die jeder SUB-Komponente eine Anzahl Kopien zuordnet. Schreibweise:

$$C_i := C(K_i)$$

$$\mu := \sum_{i=1}^{k} C_i / k \quad \text{(mittl. Kopienanzahl pro Komponente)}$$

- Eine **Komponenten-Rechner-Zuordnungsfunktion**

$$f: SUB \rightarrow 2^{VS}$$

die jede SUB-Komponente K_i auf ein oder mehrere Knoten verteilt. Schreibweise:

$|f(K_i)|$:= Anzahl Knoten, die eine K_i-Kopie besitzen

Es gelte:

$$\forall\, i \in [1,k]: |f(K_i)| = C_i$$

d.h. jede K_i-Kopie wird **genau einem** Knoten zugeordnet. Es gilt also bei N Knoten

$$\forall\, i \in [1,k]: C_i \in [1,N]$$

- Eine **Lokalisierungs-Funktion**

$$g: VS \rightarrow 2^{SUB}$$

(2^{SUB} stellt die Potenzmenge über den SUB-Komponenten dar)
die zu jedem Knoten in VS die darauf befindlichen SUB-Komponenten angibt. Die Funktion g ist durch die Zuordnungsfunktion f eindeutig bestimmt. Schreibweisen:

$|g(S_j)|$:= Anzahl der auf S_j liegenden SUB-Komponenten ($\varepsilon[0,k]$)

m := Anzahl VS-Knoten, die irgendeine SUB-Komponente besitzen ($\varepsilon[1,N]$)

Die folgenden Definitionen charakterisieren Komponenten-Rechner-Zuordnung und Kopienanzahl genauer. Eine Anwendung dieser Charakterisierung auf wesentliche VDBS-Subsysteme zeigte die Abb. 1.

Das Subsystem SUB ist

- **zentralisiert** (ZEN):
  $\exists\, j\varepsilon[1,N]\ \forall\, i\varepsilon[1,k] : f(K_i) = S_j$
  d.h. alle SUB-Komponenten liegen auf genau einem Knoten

- **partitioniert** (PAR):
  $\forall\, j1,j2\ \varepsilon\ [1,N]: j1 \neq j2 \Rightarrow g(S_{j1}) \cap g(S_{j2}) \neq \phi$
  d.h. disjunkte SUB-Teilmengen werden auf unterschiedliche Knoten gelegt. Daraus folgt insbesondere **Redundanzfreiheit**, d.h.
  $\forall\, i\varepsilon[1,k] : C_i = 1$

- **teilrepliziert** (TR):
  $\exists\, i\varepsilon[1,k] : 1 < C_i < N$
  d.h. bestimmte SUB-Komponenten sind auf eine echte Teilmenge der Knoten des verteilten Systems verteilt.

Ein wichtiger Spezialfall der Teilreplikation ist die Eigenschaft

- **teilrepliziert-gleichverteilt** (TRG):
  - $\exists c\ \forall i\varepsilon[1,k] : C_i = c \ \wedge\ 1<c<N$
  - $\forall j\varepsilon[1,N] : E(\ |g(S_j)|\ ) = k \cdot c/N$

  d.h. jede Komponente kommt in genau c Kopien vor, und der Erwartungs-

wert der Anzahl SUB-Komponenten ist für alle Knoten gleich.

- **vollrepliziert** (REP):
  $\forall j \varepsilon [1,N] : g(S_j) = SUB$
  d.h. jeder VS-Knoten enthält alle SUB-Komponenten. Insbesondere gilt
  $\forall i \varepsilon [1,k] : C_i = N$
  Man bezeichnet diese Eigenschaft von SUB auch als **Vollredundanz**.

Um diese Verteilungs-Eigenschaften besser einschätzen zu können, definieren wir auf diesen die Maße
- V-ANZ : Anzahl möglicher f-Zuordnungen für SUB
- V-GRAD : **Verteilungsgrad** von SUB

$$V\text{-}GRAD := (\mu - 1/m) / (N - 1/N)$$

Der Verteilungsgrad hängt also vom 'Redundanzfaktor' μ, der Anzahl m Knoten, denen Komponenten zugeordnet sind, sowie der Gesamtknotenanzahl N ab. Es ergeben sich folgende Beziehungen zwischen diesen Maßen und den Verteilungstypen:

VERTEILUNGSTYP	V-GRAD	V-ANZ
ZEN	0	1
PAR	$\varepsilon[0,1]$	N^k
TR	$\varepsilon[0,1]$	$\binom{N}{C_1}\binom{N}{C_2}\cdots\binom{N}{C_k}$
REP	1	1

Der so definierte Verteilungsgrad entspricht der intuitiven Vorstellung: Er ist minimal bei der Zentralisierung (V-GRAD = 0), maximal bei der Vollreplikation (V-Grad = 1), und er liegt bei allen anderen Verteilungsalternativen zwischen 0 und 1. V-GRAD wächst bei steigendem m und bei steigendem μ. V-GRAD fällt mit steigendem N, was ebenfalls sinnvoll ist, da dann die gleiche Anzahl Komponentenkopien auf ein größeres Rechnernetz verteilt werden.
Die kombinatorische Formel für beliebige Zuordnungstypen liefert der Typ TR; die Formeln für die anderen Typen sind Spezialfälle hiervon.
In der VDBS-Forschung ist fast ausschließlich der Typ REP (Vollreplikation) bei Fragment-Rechner-Zuordnungen in verteilten

Systemen untersucht worden. Ursache dafür sind die exponentiell vielen Möglichkeiten der allgemeinen Teilreplikation, obwohl gerade diese bei realen VDBS vorherrscht.
Der oben definierte Verteilungstyp der gleichverteilten Teilreplikation schränkt die Anzahl der Verteilungsalternativen stark ein. Damit erreichen wir einen Kompromiß zwischen realitätsnaher Modellierung und Anzahl zu untersuchender Fälle. Wir gehen darauf unten bei der Spezifikation des Datenprofils noch ein.

4.4.2 Datenpartitionierungs- und Datenreplikations-Strategien

Ein sehr schwieriges Problem des physischen VDB-Entwurfs ist der Entwurf der einzelnen lokalen Schemata, d.h. die Zuordnung von Fragmenten auf Rechnerknoten. In der VDB-Entwurfsphase (vergl. /TEFR82, S.423 ff/) muß der Datenbank-Administrator entsprechend einer vorgegebenen **Zuordnungsstrategie** lokale Schemata generieren und in die DD/D-Komponenten aller Knoten einbringen. Die einzelnen Phasen des verteilten Datenbankentwurfs mit den wichtigsten Einflußgrößen zeigt Abb. 4.13.

Kosten/Nutzen von Datenzuordnung und insbesondere von Datenreplikation in verteilten Systemen stellt ein nicht vollständig verstandenes Problem dar, was eine Folge sowohl der Vielzahl der Einflußparameter, als auch der Komplexität der Verteilungsalternativen ist. Bei der Wahl einer Datenzuordnungsstrategie müssen folgende sich widersprechende Anforderungen berücksichtigt werden:

- Meist sollen Fragmente repliziert werden, um die Lokalität von Lesezugriffen zu erhöhen. Anderseits führt jede zusätzliche Kopie zu erhöhtem Aufwand bei Schreibzugriffen, wenn man korrekte DB-Zugriffe (hier: Kopien-Konsistenz) garantieren will, sowie zu komplexeren TVP.
- Auch die Umkehrung gilt: Werden Fragmente zur Optimierung von Schreibzugriffen repliziert, geht das auf Kosten der Lesezugriffs-Effizienz.
- Repliziert man Fragmente, um die VDB-Verfügbarkeit zu optimieren, kann sich das negativ auf die Effizienz von Lese- u/o von Schreibzugriffen auswirken, da Verfügbarkeit und Kopien-Konsistenz

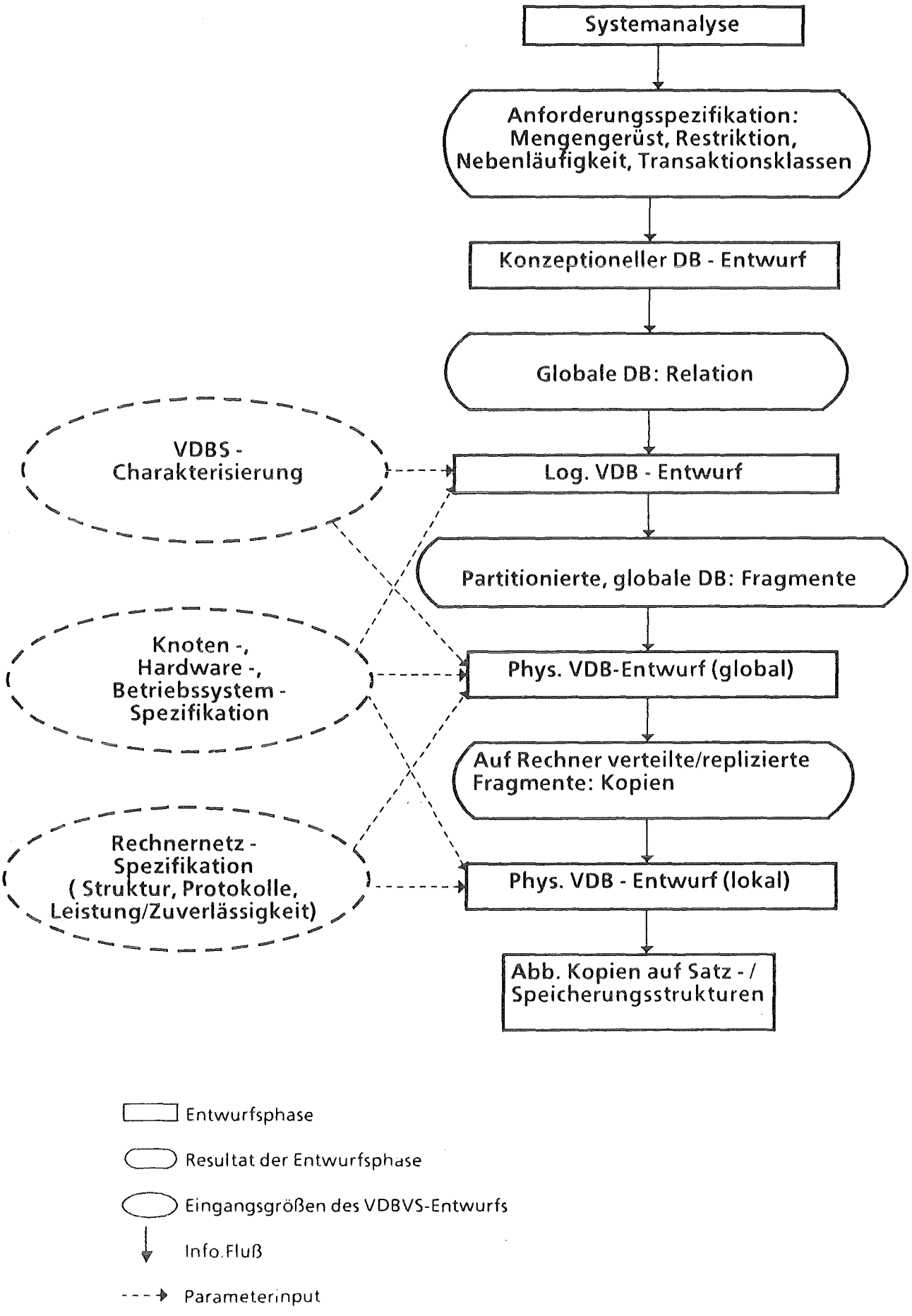


Abb. 4.13: Entwurfsmethodik Verteilter Datenhaltungssysteme

sich widersprechende Forderungen sind.

Zuordnungsstrategien lassen sich in vier Gruppen einteilen:

(1) Zuordnung durch Prädikate, die aus Benutzeranforderungen abgeleitet werden /SCHW77/;

(2) Zuordnung nach Anwendung eines linearen Optimierungsverfahrens /CMP82/;

(3) Zuordnung durch statistische Annahmen über das voraussichtliche VDB-Zugriffsverhalten;

(4) Dynamisch einstellbare Zuordnung durch adaptive Methoden während der Betriebsphase /SCHE81/.

Die ersten beiden Strategien setzen a-priori-Wissen über das voraussichtliche VDB-Anforderungsprofil voraus.
Bei Strategie (1) werden Prädikate aus diesem Profil abgeleitet, die sowohl Fragmente als auch (redundante) Speicherungsorte der Fragmente festlegen. Bei dieser Methode soll das **Lokalitätsverhalten** optimiert werden: Die Idee dabei ist, daß in verteilten Systemen Kommunikationskosten dominieren, sodaß Transaktionen mit überwiegend lokalen Zugriffen optimales Antwortzeitverhalten ergeben.
Eine ähnliche Strategie zur Minimierung der Kommunikationskosten wird in /WONG83/ verfolgt. Es wird gezeigt, daß im allgemeinen Replikation von Fragmenten notwendig ist, um den aufwendigen Austausch von Zwischenergebnissen während der Bearbeitung von Subtransaktionen zu vermeiden: Bei Verwendung der hier eingeführten **lokal-genügenden Zuordnungsmethode** reicht es aus, die Ergebnisse aller Subtransaktionen nach deren Beendigung zu sammeln.
Strategie (2) versucht, aus einem gegebenen VDB-Gesamtprofil nach einer oder mehrerer Zielfunktionen zu optimieren. Gegeben sind z.B.

- Leistungs-, Zuverlässigkeitsanforderungen;
- Transaktions- und Datenprofil (Globale Schema-Information);
- Rechnernetzkapazität, -struktur

Gesucht wird eine Datenzuordnung, die bezüglich einer der folgenden (sich i.a. gegenseitig widersprechenden) Zielgrößen optimal ist:

- Antwortzeit von Lese- bzw. Update-Transaktionen
- Durchsatz von Lese- bzw. Update-Transaktionen
- Verfügbarkeit von Lese- bzw. Update-Transaktionen

(Transaktions-Erfolgsrate)

Die auf den ersten Blick sehr effektive Strategie (2) hat aber zwei wesentliche Schwächen:

-- Es ist bekannt, daß dieses Optimierungsproblem bereits bei einer Zielgröße NP-vollständig ist. Die Vielzahl von Input- und Output-parametern führt daher in der Praxis schnell zur Nichtauswertbarkeit.

-- Ein wesentlicher Einflußfaktor auf alle genannten Zielgrößen ist nicht berücksichtigt: **Protokolle zur Transaktionsverarbeitung** beeinflussen diese Größen wesentlich (s.u.), ohne daß sie durch analytische Methoden genügend detailliert modellierbar wären.

In der Strategie (3) wird auf der Basis stochastischer Annahmen ein Datenprofil als Teil des Simulationsmodells erzeugt. Dieser Weg wird in der vorliegenden Arbeit verfolgt.

Das folgende Beispiel soll den Nachweis erbringen, daß VDBS-Zielgrößen wesentlich von der Protokollauswahl beeinflußt werden. Zielgröße sei die Terminierungswahrscheinlichkeit für Lese- und Schreib-Transaktionen. Betrachtet werden folgende zur Auswahl stehenden Entwurfs-Alternativen:

- Zentralisierte Datenbank;
- Verteilte Datenbank mit Vollzustimmungsprotokoll (PROT_ALL);
- Verteilte Datenbank mit Mehrheitsentscheidprotokoll (PROT_MAJ).

Es wird von folgendem einfachen Systemmodell ausgegangen:

-- Rechnernetz mit N Knoten (Im zentralisierten Fall gilt N=1)

-- Vollreplizierte DB, d.h. die Basismenge, auf die jede Transaktion zugreift, ist auf allen N Knoten identisch vorhanden.

-- Fehlermodell: Nur Knotenausfälle werden betrachtet mit folgenden Wahrscheinlichkeiten:

- p_K: Wahrscheinlichkeit, daß ein Knoten intakt ist, wobei diese unabhängig von anderen Knoten sein soll
- p_H: (bedingte) Wahrscheinlichkeit, daß der Heimatknoten einer Transaktion während deren Bearbeitung intakt bleibt, wenn er bei deren Initiierung intakt war
- p_C: (bedingte) Wahrscheinlichkeit, daß ein Partnerknoten einer

Transaktion während deren Bearbeitung intakt bleibt, wenn er bei deren Initiierung intakt war

Sei

- n_r, n_w : Anzahl der Partnerknoten von Lese-, Schreibtransaktionen;
- $TERM_r$, $TERM_w$: Terminierungswahrscheinlichkeit für Lese-, Schreibtransaktionen.

Dann ergibt sich

Zentralisierte Datenbank:

$n_r = 1$ $\quad TERM_r = p_K \cdot p_H$

$n_w = 1$ $\quad TERM_w = TERM_r$

PROT_ALL:

$n_r = 1$ $\quad TERM_r$ = Wahrscheinlichkeit, daß mind. ein Knoten intakt ist und daß der Heimatknoten intakt bleibt

$= [1-(1-p_K)^N] \cdot p_H$

$n_w = N$ $\quad TERM_w$ = Wahrscheinlichkeit, daß alle Knoten intakt sind und daß Heimatknoten und alle Partnerknoten intakt bleiben

$= p_K^N \cdot p_H \cdot p_C^{N-1}$

PROT_MAJ:

$n_r = n_w = \lfloor (N+1)/2 \rfloor$

$TERM_r = TERM_w$ = Wahrscheinlichkeit, daß mind. eine Mehrheit aller Knoten intakt ist und bleibt (Binomialverteilung)

$$= \sum_{i=n_w}^{N} \binom{N}{i} p_K^i \cdot (1-p_K)^{N-i} \cdot p_H \cdot p_C^{i-1}$$

Ein Vergleich dieser Aussagen ergibt: Bei Protokoll PROT_ALL ist die Terminierungswahrscheinlichkeit von Lesetransaktionen höher, die von Schreibtransaktionen aber niedriger als bei einer zentralisierten Datenbank. Bei Protokoll PROT_MAJ hängt das Vergleichsergebnis von den einzelnen Wahrscheinlichkeiten und von der Knotenanzahl ab.
Ferner wird das Vergleichsergebnis von der **Datenreplikationsmethode** wesentlich beeinflußt. Geht man etwa von einer partiellen Replikation mit C Kopien pro Fragment aus ($C \ll N$), dann sind für n_w bei PROT_ALL

bzw. PROT_MAJ im günstigsten Fall nur noch C bzw. $\lfloor (C+1)/2 \rfloor$ Partnerknoten nötig. $TERM_W$ verbessert sich bei beiden Protokollen; die untere Grenze ergibt sich bei Ersetzung von N durch C in obige Formeln.

Auf der Basis dieser Problematik wird im nächsten Abschnitt eine geeignete Methode zur Datenreplikation entworfen und in das modellierte Datenprofil eingebracht.

4.4.3 Spezifikation des Datenprofils

Zur Ermittlung geeigneter VDBS-Lastprofile sind prinzipiell zwei Wege gangbar:

- Daten- und Transaktionsprofile werden nach einer für eine bestimmte Anwendungsklasse spezifischen Anforderungsanalyse festgelegt, wie z.B. in /GODB81/. Man erhält auf diese Weise einige wenige, überwiegend statisch vorgegebene Transaktionsklassen, die auf eine relativ kleine Menge verteilter Dateneinheiten zugreifen.
- Daten- und Transaktionsprofile werden durch statistische Verteilungen auf der Basis empirisch ermittelter bzw. gemessener Werte modelliert.

Während beim ersten Weg der Wert der Ergebnisse über die untersuchte Anwendungsklasse hinaus fragwürdig ist, basieren Ergebnisse beim zweiten Weg auf über viele Anwendungen gemittelte Profile unter Annahme theoretischer Verteilungsfunktionen. Damit sind die daraus abgeleiteten Aussagen allgemeiner, haben jedoch geringere Aussagekraft bezüglich einer gegebenen Anwendung. Im folgenden verfolgen wir den zweiten Weg.
Die wesentliche Parameter des Datenprofils sind:

(DP1) Anzahl log. Dateneiheiten (Fragmente) in Datenbasis
(DP2) Fragment-Verteilungstyp: Zuordnungsstrategie Daten --> Rechner
(DP3) Fragment-Verteilungsgrad: Anzahl Kopien pro Fragment

Ein Fragment wird in unserem Datenbankmodell als die Dateneinheit mit einheitlicher Granularität zum Verteilen, Lesen, Update und Sperren betrachtet (Kap. 3.2). Die Fragmentanzahl der Datenbasis sei zeitlich invariant, d.h. das Datenbankschemema ändernde Transaktionen werden

nicht betrachtet. Wir machen keine Annahmen über die Kardinalität (z.B. Recordanzahl) von Fragmenten. Diese läßt sich aber indirekt beschreiben: Größere Fragmente sind durch höheren Zeitverbrauch bei der Fragmentbearbeitung, zusammen mit kleinerer Fragment-Gesamtanzahl, charakterisiert.

Alternative Datenreplikationskonzepte wurden im letzten Abschnitt diskutiert. Die Annahme der Vollreplikation ist in den bisherigen Studien zur VDBS-Leistungsvorhersage bevorzugt worden, obwohl Leistungseinbußen als Folge der Vollreplikation nicht zu vermeiden sind. Die folgende Methode der **gleichverteilten Teilreplikation** von Fragmenten verspricht für die zu modellierenden Protokolle -unter noch zu definierenden Annahmen- besseres Leistungs- und Verfügbarkeitsverhalten.
Bei der Konstruktion einer Methode muß man folgende Aspekte berücksichtigen:

- Eine für beliebige Systeme und für alle Anforderungen optimale Methode existiert nicht. Bestenfalls kann man bei bekanntem System-Gesamtprofil und sich nicht widersprechenden Anforderungen eine günstige Replikation konstruieren.
- Da es exponentiell viel Kombinationen gibt, Fragmente zu Rechnern zuzuordnen (vergl. Kap. 4.4.1), erscheint es wenig sinnvoll, alle Kombinationen per Simulation zu studieren. Offensichtlich resultieren aber je nach Zuordnung sehr unterschiedliche Ergebnisse bzgl. Transaktions-Antwortzeiten und Ausfallsicherheit, wie das obige Beispiel zeigte.
- Wie soll man Austausch von Transaktions-Zwischenergebnissen bei der Anfrageausführung modellieren, wenn nicht alle benötigten Daten lokal vorhanden sind? Unser aggregiertes Modell ist hierfür wenig geeignet, da die bekannten Anfrageauswertungsstrategien beim Datenprofil von detaillierteren DB-Parametern ausgehen (z.B. Fragmentgröße, Record-Größe, Selektivitäten, Schlüsselverteilungen), während beim Transaktionsprofil meist von relationalen Operationen wie PROJECT, SELECT, JOIN anstatt unseren Dateioperationen READ, WRITE ausgegangen wird.
  Wir nehmen aber an, daß 'fast' alle benötigten Daten lokal (am Eingabenknoten einer Transaktion) gespeichert wurden, was z.B.

durch Anwendung der lokal-genügenden Zuordnungsmethode (s.o.) möglich ist. Daher kann eine einfache Funktion der lokalen Anfragekosten definiert werden. Der Modellierungsfehler ist dabei umso kleiner, je höher die Interrechner-Kommunikationskosten gegenüber lokalen Verarbeitungskosten sind.

Diese Probleme werden durch folgende **stochastische Datenreplikations-Methode** vereinfacht.

4.4.4 Eine stochastische Datenreplikations-Methode

Die Methode basiert auf folgenden Annahmen:

(i) Transaktionslasten sind auf allen Rechnerknoten gleichverteilt. Transaktionen haben im Mittel ein hohes Lokalitätsverhalten.

(ii) Das zugrundegelegte Rechnernetz hat eine Ringstruktur: Es besteht aus N Knoten, die von 1 bis N durchnummeriert sind; ein Knoten K(i) ist mit seinem linken und rechten Nachbarn

K(**mod**(i+N-2,N)+1) und K(**mod**(i,N)+1)

physisch verbunden (N≥2).

Die Methode ordnet Fragmentkopien auf Rechner in zwei Schritten zu:

(a) Jedes der E Fragment wird auf genau eine **Primärlokation** (d.h. einen bestimmten Rechnerknoten) gespeichert. Dabei sei die Auswahl jedes Knotens K(1), ... ,K(N) gleichwahrscheinlich, d.h. im Mittel bekommt ein Knoten $\lceil E/N \rceil$ Primärkopien zugewiesen.

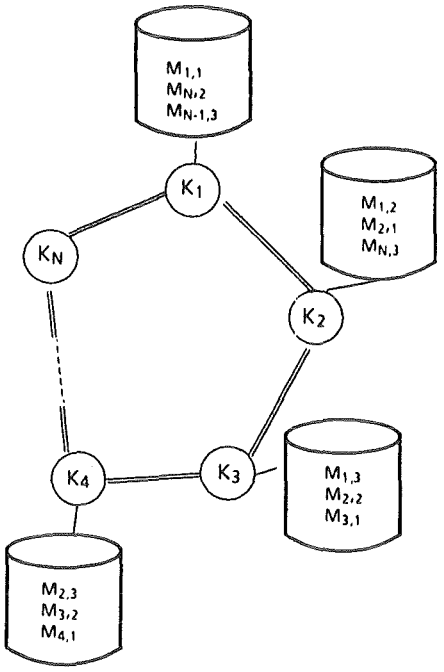
(b) Für jede Primärkopie auf K(i), 1≤i≤N, werden C-1 Sekundärkopien desselben Fragments auf die C-1 K(i)-Nachfolgerknoten verteilt:

K(**mod**(i+j-1,N)+1) für j∈[1,C-1]

Dieses Verfahren wird in Abb. 4.14 an einem Beispiel erläutert.

Was bewirkt diese Methode unter den gegebenen Randbedingungen (i) und (ii) voraussichtlich?

(1) Die Forderung nach hoher Lokalität wird durch die Methode nicht eingeschränkt, da die Wahrscheinlichkeit, eine bestimmte Primärkopie lokal vorzufinden, für allen Knoten gleich ist. Die Methode ist also **fair** bezüglich dem Lokalitätsverhalten. Nichtlokale Anfrage-Auswertungskosten werden daher



Legende:	$K_1,\ldots,K_N$	Rechnerknoten
	═══	phys. Knotenverbindung(hier Ringtopologie)
	$M_{i,j}$	Menge aller Kopien, deren Primärkopie sich auf Knoten i befindet j ∈ (1, 3) ist Kopien-Nr.
	$M_{i,1}$	Menge aller Primärkopien auf Knoten$_i$
	$M_{i,2}$ $M_{i,3}$	Menge aller Sekundärkopien auf Knoten$_i$

Abb. 4.14 Konzept der stochastischen Datenreplikation (Beispiel)

vernachlässigt.

(2) Die Anzahl der Partnerknoten einer Transaktion wird minimiert, da folgendes gilt:

Für die Menge aller an einem Knoten vorhandener Primärkopien existiert eine (Partner-)Knotenmenge mit genau C-1 Knoten, auf der sich **alle** Sekundärkopien derselben Fragmente befinden.

Bei vollständiger Lokalität liegen also alle Kopien von Fragmenten der Basismenge einer Transaktion auf genau C Knoten, was offensichtlich bezüglich Kommunikationskosten und

Datenverfügbarkeit optimal ist.

(3) Die Methode verspricht ein faires Datenverfügbarkeitsverhalten: Wegen Annahme (ii) und den Homogenitätsannahmen des Fehlermodells ist die Wahrscheinlichkeit einer Knoten-Nichtverfügbarkeit für jeden Knoten gleich. Da die Replikationsmethode bewirkt, daß auf allen Knoten gleich viele Kopien liegen, wird bei jeder auftretenden Nichtverfügbarkeit dieselbe Anzahl Kopien für andere Knoten unzugänglich.

Bei unserer homogenen Replikationsmethode kann insbesondere der Zusammenhang zwischen Lokalität und Ausfallsicherheit untersucht werden, ohne daß der Einfluß einer ungünstigen Datenverteilung die Ergebnisse entwerten würde.

Da Kopien insbesondere zur Erhöhung der Ausfallsicherheit gehalten werden, wird eine konstante Kopienanzahl für alle Fragmente angenommen (statisch, fest). Dabei werden jeweils eine und drei Kopien pro Fragment untersucht:

- Neben der Teilreplikation wird zum Vergleich auch eine partitionierte (nichtredundante) verteilte Datenbank untersucht, d.h. der Fall genau einer Kopie pro Fragment.
- Eine gerade Kopienanzahl wird nicht untersucht, da die Performance von Update-Transaktionen schlechter ist als bei einer Kopie weniger; zudem wird die Ausfallsicherheit bei Anwendung bekannter Sperrverfahren auf redundante Daten durch diese zusätzliche Kopie nicht verbessert.
- Eine Kopienanzahl ≥ 5 wird nicht untersucht, da das voraussichtlich sehr schlechte Antwortzeitverhalten von Update-Transaktionen die erhöhte Ausfallsicherheit nicht rechtfertigen würde.

4.4.5 Spezifikation des Transaktionsprofils

Das Transaktionsprofil wird durch folgende Eingabeparameter modelliert, wobei die Charakteristik des Datenprofils berücksichtigt ist:

(TP1) Transaktions-Komplexität:

Verteilungsfunktion der Fragmentanzahl pro Transaktion

(TP2) Transaktions-Lokalitätsfaktor

(TP3) Schreib/Lesefaktor (Update-Häufigkeit)

(TP4) Fragment-Zugriffshäufigkeiten

Die Transaktions-Ankunftsrate pro Knoten ist kein Eingabeparameter, sondern ergibt sich in unserem geschlossenen Netzmodell indirekt aus der Terminalanzahl pro Knoten, der mittleren Denkzeit pro Terminal sowie der mittleren Antwortzeit pro Transaktion (vergl. Kap. 4.1).

Parameter und Annahmen des Transaktionsprofils legen die **Basismenge** BS(T) einer Transaktion T fest, also derjenigen Fragmente, auf die durch Lese- und Schreiboperationen innerhalb der Transaktion zugegriffen wird (vergl. Kap. 3.2.1). Aus der Basismenge können -abhängig von Datenverteilung und Verarbeitungsprotokoll- die Subtransaktionen einer Transaktion erzeugt werden.
Ein Maß für die **Transaktionskomplexität** ist die Anzahl und Verteilungsfunktion der Fragmente in BS(T). Damit wird die Anzahl von Verknüpfungsoperationen in T erfaßt. (Anmerkung: der Zusammenhang zwischen den Operationen und der Fragmentanzahl ist komplex, da z.B. eine relationale SELECT- oder PROJECT-Operation mehrere Fragmente erfordert, wenn eine Relation in mehrere Fragmente partitioniert wurde.) Die Kardinalität |BS(T)| der Basismenge legt eine grobe Einteilung in **Transaktionsklassen** fest. Eine empirische Verteilung der Basismengen (relative Kardinalität-Häufigkeiten) legt den 'Transaktionsmix' fest: Bei 'einfachen' Anwendungen gehen wir von kleinen Basismengen wegen vielen ungeübten, gelegentlichen Benutzern aus. Bei komplexen Anwendungen werden häufiger mehrere Dateneinheiten durch komplexe Suchkriterien verknüpft; die Basismengen sind hier im Mittel größer.

Kardinalität und Inhalt der Basismenge BS(T) einer Transaktion T werden durch folgende Schritte ermittelt:

(s1) Die Kardinalität |BS(T)| wird aus einer empirischen Verteilung generiert

(s2) Aufspaltung in Anzahl lokaler und nichtlokaler Fragmente. Der **Lokalitätsfaktor** legt die Anzahl lokal -also an T's Heimatknoten

HOME(T)- gespeicherter Primärkopien pro Transaktion T fest (Durch die Datenreplikationsmethode wurde vorher jedem Fragment ein primärer Speicherort zugewiesen). Dieser Faktor ist ein Maß für die **Güte der Datenreplikationsmethode**: Je höher die Lokalität der verteilten Dateneinheiten, umso geringer ist die Anzahl Unteraufträge, die Kommunikations- und Verarbeitungskosten an lokal nicht vorhandenen Daten verursachen. Es wird daher ein Lokalitätsfaktor >> 0.5 gefordert /GODB81, MUNZ79/, der Idealwert 1.0 ist aber unrealistisch.
Der Anteil lokal vorhandener Primärkopien bestimmt sich zu

$|LOC(T)| := \lceil |BS(T)| \cdot LOC \rceil$ LOC: Lokalitätsfaktor

(s3) Aufspaltung in lokale und nichtlokale Lese- und Schreibmenge. Die **Update-Häufigkeit** WR definiert den Anteil der Basismenge, der geändert werden soll (evtl. nach vorhergehendem Lesezugriff). Eine 'query-intensive' Anwendung wird durch WR<0.5 charakterisiert. Die Anzahl lokal/nichtlokal vorhandener Lese/Schreib-Primärkopien ergibt sich jeweils aus:
$|WL(T)| := \lceil |LOC(T)| \cdot WR \rceil$ Anzahl lokaler Schreib-Primärkopien
$|RL(T)| := |LOC(T)| - |WL(T)|$ Anzahl lokaler Lese-Primärkopien
Die nichtlokalen Anteile werden entsprechend berechnet.

(s4) Bestimmung des Inhalts der 'logischen' Basismenge durch Zuordnung konkreter Primärkopien. Dazu wird die Fragment-Zugriffshäufigkeit benötigt. Wir nehmen hier **Gleichverteilung** an. Um die Zuordnung auszuführen, ist ein **erwartungstreues Auswahlverfahren** erforderlich: Aus einer Menge mit M Elementen sind m≤M Elemente 'ohne Zurücklegen' so zu ziehen, daß das Ziehen eines beliebigen Elements stets mit Wahrscheinlichkeit 1/m erfolgt, und jede mögliche Folge von m Ziehungen mit der gleichen Wahrscheinlichkeit $1/\binom{M}{m}$ auftreten kann. Effiziente Auswahlverfahren, deren Zeitkomplexität von m (aber nicht von M) abhängt, werden in /VITT84/ beschrieben.
Damit können Primärkopien für jeden der vier Anteile der Basismenge entsprechend der geforderten Zugriffshäufigkeit erzeugt werden.

(s5) Bestimmung des Inhalts der 'physischen' Basismenge durch

Zuordnung konkreter Sekundärkopien. Durch das im letzten Abschnitt entwickelte Kopienverteilungsverfahren ist diese Zuordnung eindeutig bestimmt.

Aus der in (s1) bis (s5) erhaltenen Basismenge werden alle Subtransaktionen in Abhängigkeit vom gewählten TVP generiert, wie unten noch erläutert wird.

5 MODELL DER LEISTUNGS-/VERFÜGBARKEITSSCHÄTZUNG FEHLERTOLERANTER VERTEILTER SYSTEME

Bei der Simulation von VDBS-Protokollen sollen hauptsächlich **benutzerorientierte Leistungsgrößen** wie Antwortzeit und Durchsatz von Transaktionen ermittelt werden. Diese Zielgrößen werden als relevanter angesehen als z.B. HW-Auslastung von Betriebsmitteln und SW-Auslastung von VDBS-Prozessen, da diese systemorientierten Größen als Einflußfaktoren von Antwortzeiten und Durchsätzen anzusehen sind, die mehr für das Systemmanagement (Engpaßentdeckung, Kapazitätsplanung) von Interesse sind.

Bei der Bestimmung dieser Größen ist die traditionelle Methode, jegliche Fehler a priori auszuschließen. Hingegen soll in dieser Arbeit das Ausfallverhalten eines Rechensystems durch **Integration von Leistungs- und Verfügbarkeitsgrößen** in die Leistungsanalyse einbezogen. Hierfür werden relevante Maße definiert und eine neue Simulationsmethodik entwickelt.

Prinzipiell können quantitative Protokolluntersuchungen verwendet werden, um

(a) Ein einzelnes Protokoll im Sinne einer Kostenfunktion zu optimieren,

(b) Bei einem einzelnen Protokoll Leistung unter realen mit Leistung unter idealen Bedingungen zu vergleichen,

(c) Verschiedene Protokolle bei gleichen Eingabeparametern zu vergleichen.

Eine Protokolloptimierung nach (a) erscheint im Rahmen dieser Arbeit nicht sinnvoll. Mangels Meßwerten von existierenden VDBS wäre eine Optimierung von geringem praktischen Wert.
Hingegen sind vergleichende Untersuchungen nach (b) bzw. (c) unter den vorgegebenen Randbedingungen überzeugender, da sie realistischere Annahmen über das Systemverhalten machen. Zudem kann erwartet werden, daß Ergebnisse einer Vergleichsstudie auch dann gültig sind, wenn die Absolutwerte der erhaltenen Zielgrößen stark von den in realen VDBS gemessenen Werten abweichen. **Protokollvergleiche** sind allerdings mit Sorgfalt vorzunehmen, um signifikante Aussagen zu erhalten. Ein

Vergleich kann nur dann sinnvolle Ergebnisse liefern, wenn die zu **vergleichenden Protokolle unter gleichen Umgebungsannahmen** analysiert werden. (Bei dem VDBS-Modell sind diese Annahmen Teil des Transaktions-, Daten-, Rechnernetz- und Ausfallprofils.)

Die wesentliche **Zielsetzung** bei Vergleichs-Untersuchungen von Designalternativen eines fehlertoleranten Rechensystems ist:

> Wie hoch ist der voraussichtliche **Recoveryaufwand** des Gesamtsystems -bewertet durch Antwortzeiten und Durchsätze- unter der Bedingung, daß Ausfälle eintreten können, verglichen mit dem Aufwand ohne Ausfallmöglichkeit? Genauer:

	Ausfälle berück-sichtigt	Recovery realisiert	Erläuterung der Modellklasse
Klasse 1	nein	ja	fehlertol. System ohne Ausfall
2	ja	ja	fehlertol. System mit Ausfall
3	ja	nein	fehlerintol. System mit Ausfall
4	nein	nein	fehlerintol. System ohne Ausfall

Dabei repräsentiert jede Klasse eine Menge von VDBS-Modellen, in denen jeweils Ausfälle und Recovery-Verfahren modelliert sind oder nicht. Der "reale" Recoveryaufwand (Ergebnis von Untersuchungen der Klasse 2) ist zu normieren mit dem "normalen" Aufwand (Ergebnis von Untersuchungen der Klasse 1). Das Ergebnis beschreibt den Grad der erzielten **Fehlertoleranz** des Systems und dessen Kosten bezüglich dem Leistungsverhalten des Systems. Eine solche Normierung ist speziell für Systemvergleiche von Nutzen. Die Integration von Leistungs- mit Verfügbarkeitsmaßen würde den Vergleich eines Protokolls gestatten, das die Leistung im fehlerfreien Fall optimieren soll, mit einem Protokoll, das günstige Leistung nach Ausfällen verspricht, durch ein **einheitliches Zielkriterium.**

Klasse 3 und 4 werden nicht weiter betrachtet. Klasse 3 beschreibt ein inkorrektes System, das Ausfälle nicht erkennt und/oder nicht behandelt. Klasse 4 bietet sich zur Normierung ebenfalls an, ist aber schwierig zu realisieren, da man im Simulationsmodell alle zur Ausfallbehandlung gehörigen Funktionen (z.B. 2-Phasen-Commit, Kopienabgleich) "unterdrücken" müßte. Hingegen ist Klasse 1 aus 2 einfach ableitbar, indem lediglich der ausfallfreie Anfangszustand des

Rechnernetzes betrachtet wird.

Im folgenden wird zunächst eine Methode zur Ermittlung von Zielgrößen unter der Annahme statistisch idealer Bedingungen (Stationarität) eingeführt (Kap. 5.1). Statistische Schätzmethoden unter realistischeren Bedingungen werden in Kapitel 5.2 besprochen. besprochen. In Kapitel 5.3 wird eine Methode zur integrierten Leistungs-Verfügbarkeitsuntersuchung für fehlertolerante Systeme ausgearbeitet. In Kapitel 5.4 werden diese Methoden zum Entwurf des **VDBS-Auswertungsmodells** angewendet.

5.1 Operationale Analyse von Warteschlangennetzen

Bevor wir die Zielgrößen bei der VDBS-Simulation im einzelnen festlegen, müssen die statistischen Grundlagen zur Ermittlung bzw. zuverlässigen Schätzung dieser Größen definiert werden. Hierzu bietet sich im Rahmen der Theorie stochastischer Prozesse die Theorie der **Warteschlangennetze** an. Diese Theorie geht aus von einem Systemmodell, bestehend aus Aufträgen, auftragsbearbeitenden Diensten und Auftrags-Warteschlangen vor Diensten. Offensichtlich besteht eine Analogie zwischen Aufträgen, Diensten und Warteschlangen in dieser Theorie und Marken, Instanzen und Kanälen in (Petri-)Netzen. Allerdings ist die Klasse der durch z.B. Funktionsnetze modellierbaren Systeme größer als die Klasse der durch Warteschlangennetze modellierbaren. Wir werden daher die Netzklasse zum Zwecke der Auswertung einschränken, um die Definition operationaler Zielgrößen aus der Warteschlangentheorie übernehmen zu können.

Aus dem Bereich der Warteschlangentheorie interessiert im Rahmen unserer Arbeit insbesondere, welche Größen in einem gegebenen Beobachtungszeitraum gemessen und wie daraus sinnvolle Zielgrößen abgeleitet werden können. Bei allen abgeleiteten Größen handelt es sich um **statistische Schätzungen**, die unter der (approximierenden) Annahme eines stationären Gleichgewichts des betrachteten Systems definiert werden. Wir treffen dabei keine Aussagen über den Konfidenzgrad solcher Schätzungen, bzw. über den statistischen Fehler, den wir bei den Schätzungen machen. Die Warteschlangentheorie versucht, Zielgrößen analytisch zu bestimmen, während wir lediglich

deren Schätzungen heranziehen, um unsere Simulationsexperimente sinnvoll auszuwerten.

Die folgenden Definitionen basieren auf der **Operationalen Analyse** von Warteschlangensystemen /DEBU78/, die einfache Schätzungen von Leistungsmaßen in solchen Systemen erlaubt. Zunächst werden Leistungsmaße eines Einzeldienst-Modells definiert. Es folgt die Verallgemeinerung auf ein System mit K Diensten. Schließlich definieren wir die für uns relevantesten Maße Antwortzeit und Durchsatz einzelner Dienste, eines Subsystems und des Gesamtsystems, sowie einige Verfügbarkeitsmaße.

Einzeldienst-Modell

Grundlage aller folgenden Definitionen sei ein **Beobachtungszeitraum** $[0,\tau]$, $\tau \in (0,\infty)$. Alle Größen sind von τ abhängige, stochastische Variablen. Innerhalb des Zeitraums τ werden folgende Größen beobachtet (vergl. Abb. 5.1-a):

ANK Anzahl Auftrags-Ankünfte vor dem Dienst
TERM Anzahl Auftrags-Terminierungen (vom Dienst bearbeitet)
TAK Summe der Bearbeitungszeiten des Dienstes (TAK≤τ)

In einem hinreichend großen Zeitraum $[0,\tau]$ werden $n(\tau)$ Leistungswerte $L_n := \{A(1), \ldots, A(n)\}$ beobachtet. Sei

$$E_n(L) := \sum_{i=1}^{n} A(i)/n$$

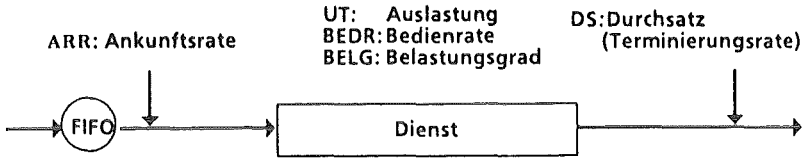
eine Schätzung des Erwartungswerts von L_n. Der durch diese Schätzung induzierte Fehler gegenüber dem 'wahren' Erwartungswert

$$E(L) := \lim_{n\to\infty} E_n(L)$$

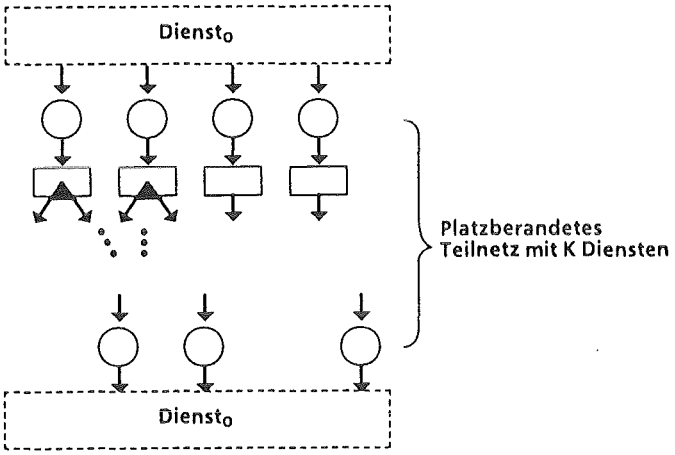
ist vernachlässigbar, falls dieser Erwartungswert existiert und endlich ist (**Stationarität**).

Damit erhalten wir folgende geschätzte Leistungsmaße:

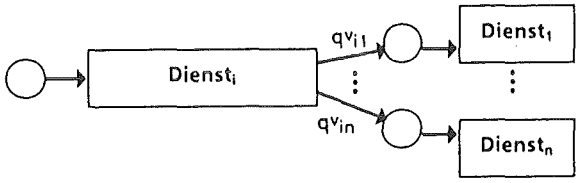
ARR Ankunftsrate, ARR := ANK/τ (arrival rate)
DS Terminierungsrate, DS := TERM/τ (Durchsatz, throughput)
UT Dienst-Auslastung, UT := TAK/τ (utilization)
BEDR Bedienrate, BEDR := TERM/TAK (service rate)



a) Einzeldienst-Auswertungsmodell



b) K - Dienst - Auswertungsmodell



c) Verzweigungswahrscheinlichkeiten $qv_{i1}, \ldots, qv_{in}$ eines Dienstes$_i$

Abb. 5.1: Dienstauswertungsmodell von Warteschlangennetzen

BELG Belastungsgrad, BELG := ARR/BEDR (traffic intensity)
Dieses systemorientierte Maß ist wichtig zur Dimensionierung von Kanälen und bei der Engpaßanalyse. Ein Dienst mit Belastungsgrad ≥1 ist überlastet, seine Eingangswarteschlange kann in ihrer Kapazität nicht begrenzt werden. In einem geschlossenen System ist der Belastungsgrad stets <1.

K-Dienst-Modell

Bei der Verallgemeinerung auf K Dienste wird ein Modell betrachtet, wie es Abb. 5.1-b zeigt. Das Modell besteht aus einem warteschlangen-berandeten Netz, das die Bearbeitung von Aufträgen

durch $K \geq 1$ Dienste beschreibt. Dienste können zwar nebenläufig aktiv sein, aber ein einzelner Auftrag wird zu einem Zeitpunkt von höchstens einem Dienst bearbeitet; speziell werden damit Unteraufträge nicht explizit modelliert.

Aus Notationsgründen wird die Umgebung des Netzes als ein Dienst D_0 aufgefaßt. Nicht festgelegt ist, ob dieses Netz geschlossen im Sinne der Warteschlangentheorie ist, d.h. ob alle das Netz verlassenden Aufträge sofort wieder neue Aufträge produzieren. Um ein solches Netz auf ein Funktionsnetz abzubilden, sind folgende Einschränkungen der Netzklasse notwendig (vergl. Abb. 5.1-c):

- Alle Kanäle haben FIFO-Zugriffsmodus.
- Jeder Dienst wird abstrahierend durch sein 'black-box' Verhalten mit seiner Umgebung beschrieben. Ein Dienst wird auf genau eine Instanz im Netz abgebildet, oder auf ein Teilnetz, dessen internes Verhalten für die Leistungsanalyse nicht interessiert.
- Bei mehreren Outputkanälen einer Instanz kann ein Auftrag von höchstens einer Folgeinstanz bearbeitet werden (logische XOR-Schaltregel entsprechend dem partiellen SOME-OUT Schalten von Instanzen in Funktionsnetzen /GODB83/). Damit wird eine auftragsinterne Nebenläufigkeit verhindert.

Auf einer höheren Abstraktionsebene sind diese Einschränkungen für die Modellierung eines Auswertungsmodells unwesentlich, wie das unten entworfene VDBS-Auswertungsmodell zeigen wird.

Im Zeitraum $[0,\tau]$ werden an jedem $Dienst_i$, $i \varepsilon [0,K]$ folgende Größen beobachtet:

ANK_i Anzahl Ankünfte vor $Dienst_i$

TAK_i Summe der Aktivzeiten von $Dienst_i$

$TERM_{ij}$ Anzahl Aufträge, die von $Dienst_i$ und von einem nachfolgenden $Dienst_j$ bedient werden
($TERM_{ij}=0$, falls von $Dienst_i$ zu $Dienst_j$ kein Weg existiert.
$TERM_{ij}>0$, falls $Dienst_i$-Outputkanal = $Dienst_j$-Inputkanal)

ANK_{0j} Anzahl der Aufträge, die aus der Systemumgebung zuerst in der $Dienst_j$-Warteschlange ankommen

$TERM_{j0}$ Anzahl der Aufträge, die zuletzt vom $Dienst_j$ bedient werden und anschließend das System verlassen

Daraus leiten sich folgende Leistungsmaße für das K-Dienst-Modell ab:

$ANK_0 := \sum_{j=1}^{k} ANK_{0j}$ Anzahl Ankünfte im Gesamtsystem

$TERM_0 := \sum_{i=1}^{k} TERM_{i0}$ Anzahl beendeter Aufträge im Gesamtsystem

$ARR_i := ANK_i/\tau$ Ankunftsrate vor $Dienst_i$

$TERM_i := \sum_{j=0}^{k} TERM_{ij}$ Anzahl vom $Dienst_i$ ausgeführter Aufträge

$UT_i := TAK_i/\tau$ Auslastung von $Dienst_i$

$UT := \sum_{i=1}^{k} TAK_i/(\tau \cdot k)$ Auslastung im Gesamtsystem

$BEDR_i := TERM_i/TAK_i$ Bedienrate von $Dienst_i$

$BEDR := k \cdot TERM_0 / \sum_{i=1}^{k} TAK_i$ Bedienrate im Gesamtsystem

$BELG_i := ARR_i/BEDR_i$ Belastungsgrad von $Dienst_i$

$$qv_{ij} := \begin{cases} TERM_{ij}/TERM_i, & \text{falls } i \varepsilon [1,K] \text{ und } TERM_i \neq 0 \\ ANK_{0j}/ANK_0, & \text{falls } i=0 \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

$Verzweigungswahrscheinlichkeit_j$ von $Dienst_i$ ("routing frequency"), d.h. Anteil der Aufträge, die nach Bearbeitung durch $Dienst_i$ von einem nachfolgenden $Dienst_j$ bearbeitet werden (vergl. Abb. 5.1-c). Damit gilt für jedes i:

$$\sum_{j=1}^{k} qv_{ij} = 1$$

Antwortzeit und Durchsatz im K-Dienst-Modell

Um diese Maße sinnvoll definieren zu können, gelte folgende **Auftrags-Gleichgewichtsannahme:**

$ANK_i = TERM_i$ für alle $i \varepsilon [0, K]$

Im Zeitraum τ sollen also nur terminierte Aufträge betrachtet werden. Diese Annahme ist auch bei auftragsorientierten Simulationsmodellen

üblich, da die Betrachtung offener Aufträge die Ergebnisse verfälschen würden.

Daraus ergibt sich folgende Schätzung des **mittleren Durchsatzes** im Zeitraum $[0,\tau]$ (Anzahl Aufträge pro Zeiteinheit, die sowohl von $Dienst_i$ als auch von einem nachfolgenden $Dienst_j$ bearbeitet werden):

$$DS_{ij} := TERM_{ij}/\tau$$

Im Gleichgewicht ist der Durchsatz DS_i von $Dienst_i$ gleich der Ankunftsrate ARR_i von Aufträgen an $Dienst_i$.

Zur Schätzung der $Antwortzeit_{ij}$, d.h. der mittleren Verweilzeit eines Auftrags zwischen der Ankunft in der $Dienst_i$-Warteschlange und der beendeten Bearbeitung durch einen nachfolgenden $Dienst_j$ benötigen wir eine weitere Meßgröße:

$ASUM_{ij}$ Summe der Verweilzeiten aller Aufträge, die vor $Dienst_i$ ankommen und von einem nachfolgenden $Dienst_j$ bearbeitet wurden, im Zeitraum τ

Die **mittlere Antwortzeit** wird dann wie folgt geschätzt:

$$RESP_{ij} := ASUM_{ij}/TERM_j, \text{ falls } TERM_j \neq 0$$

Wichtig sind folgende Spezialfälle:

-- $RESP_{ij} = 0$, falls kein Weg von $Dienst_i$ zu $Dienst_j$ existiert;
-- $RESP_{0j}$: Verweilzeit aller Aufträge zwischen Ankunft im System und Bearbeitung durch $Dienst_j$;
-- $RESP_{i0}$: Verweilzeit aller Aufträge zwischen Ankunft in $Dienst_i$-Warteschlange und Verlassen des Systems.
-- $RESP_{00}$: mittl. Antwortzeit aller Aufträge im Gesamtsystem, $RESP := RESP_{00}$.

N-Knoten/K-Dienst-Modell

Das K-Dienst Modell kann wie folgt verallgemeinert werden. Gegeben sei ein System aus $N \geq 1$ Knoten. Jeder Knoten $NODE_\theta$, $\theta \in [1,N]$, habe ein homogene Dienststruktur, d.h. er besteht aus den gleichartigen Diensten $D_0(\theta)$, ... , $D_K(\theta)$. Die Knoten sind wechselseitig autonom, d.h. Aufträge können nebenläufig an jedem Dienst $D_0(\theta)$ ankommen und

werden dann ausschließlich von Diensten desselben (Heimat-)Knotens $NODE_\theta$ bearbeitet.

In diesem Modell der zentralisierten Auftragsbeobachtung werden Unteraufträge an Dienste auf lokale und/oder entfernte Knoten nicht explizit beschrieben; solche Dienste sind kein Teil der K-Dienst-Struktur. Da aber angenommen wird, daß auch Aufträge mit Unteraufträgen schließlich an ihrem jeweiligen Heimatknoten terminieren, geht das Leistungsverhalten der Unteraufträge implizit in die Leistungsgrößen der Aufträge mit ein.

Damit sind alle oben für das K-Dienst-Modell definierten Größen für jeden der N Knoten gültig. Jede Größe G des K-Dienstmodells wird im N-Knoten-Modell mit

$$G(\theta)$$

bezeichnet, wobei der Index $\theta \varepsilon [1,N]$ angibt, auf welchem Knoten die Größe beobachtet wird.

Daraus kann jeweils eine Leistungsgröße für das N-Knoten-Gesamtsystem definiert werden; diese bezeichnen wir mit

$$G_{NET}\ .$$

Insbesondere seien Gesamt-Antwortzeit und Gesamt-Verzweigungswahrscheinlichkeit als arithmetischer Mittelwert der Antwortzeiten und Wahrscheinlichkeiten auf allen Knoten definiert:

$$RESP_{ij,\ NET} := \sum_{\theta=1}^{N} RESP_{ij}(\theta) \ /\ N$$

$$qv_{ij,\ NET} := \sum_{\theta=1}^{N} qv_{ij}(\theta) \ /\ N$$

Der Gesamtdurchsatz ist die Summe der Durchsätze pro Knoten:

$$DS_{ij,\ NET} := \sum_{\theta=1}^{N} DS_{ij}(\theta)$$

Das Leistungsverhalten von Transaktionen, die durch ein VDBS mit homogener Hardware-/Software-Architektur bearbeitet werden (s. Kap. 4), kann auf einfache Weise durch dieses N-Knoten/K-Dienst-Modell spezifiziert werden. Wir kommen darauf in Kürze zurück.

5.2 Statistische Schätzung stationärer Zielgrössen

Simulationsmodelle von Rechensystemen werden durch stochastisch generierte Lastprofile angetrieben und erzeugen Zeitreihen für die beobachteten Werte jeder Zielgröße. Bei jedem Experiment, das unter gleichen Eingangsparametern, aber unterschiedlichen Lasten (z.B. wegen unterschiedlichen Zufallszahlenfolgen) ausgeführt wird, erhält man z.T. stark differierende Ausgabe-Zeitreihen. Daher sind fundierte statistische Methoden zur Lösung folgender Problem erforderlich:

- Eliminierung der initialen "Füllphase", in der kein stabiles (erwartungstreues) Verhalten der Zielgrößen erwartet werden kann, d.h. daß meist $E_n(L) \neq E(L)$ sein wird, wenn die Leistungswerte $\lambda(1),\ldots,\lambda(n)$ im Zeitraum $[0,\tau]$ beobachtet werden. Gesucht ist ein Zeitpunkt $t'>0$, sodaß im Zeitraum $[t',\tau]$ die Werte $l(n'),\ldots,l(n)$ die Schätzung $E_{n-n'+1}(L)$ liefern, sodaß diese Schätzung eine geringere Abweichung vom realen Erwartungswert $E(L)$ hat als $E_n(L)$. Der Zeitpunkt t' ist so zu wählen, daß sich für $t > t'$ die Zielgröße L annähernd stationär verhält.
- Bestimmung der minimalen Simulationslaufzeit, um bei der Schätzung des Erwartungswerts jeder Zielgröße keinen zu großen Fehler zu machen. Für transaktionsorientierte Beobachtungen ist die Mindestanzahl terminierter Transaktionen ein Maß für die Laufzeit.
- Schätzung stationärer Erwartungswerte und Varianzen aller Zielgrößen. Bestimmung von Konfidenzintervallen für alle geschätzten Größen bei gegebenem Konfidenzgrad.

Für die Eliminierung transienter Beobachtungen gibt es bis heute keine allgemein anerkannte Methode /WIPR78/. In unseren VDBS-Modell-experimenten wurde ein bestimmter Anteil initialer Transaktionen bei der Auswertung eliminiert in der Erwartung, daß alle transaktionsorientierten Zielgrößen ab dem Ende der eliminierten Phase stationär sind, und daß daher die Anwendung Operationaler Analysemethoden auf Beobachtungen in der restlichen (Simulations-)Zeitdauer erwartungstreue Schätzungen der Zielgrößen liefern.

In stochastischen Warteschlangenmodellen kann eine z.T. starke

Autokorrelation der Zielgrößenwerte beobachtet werden, die auch noch nach der transienten Phase eine Rolle spielt. Transaktionsorientierte Beobachtungen sind autokorreliert, wenn z.B. eine Folge von Antwortzeiten von sich zeitlich überlappenden Transaktionen stochastisch nicht unabhängig ist. Intuitiv gesehen, hängt das mit der Charakteristik der Warteschlangen im Rechensystem zusammen: Ist die Wartezeit vor einer Bedieneinheit (Instanz) für einen Auftrag hoch, dann ist sie oft auch für den Folgeauftrag hoch. Die bekannten Methoden zur Ermittlung von Varianzen und Konfidenzintervallen setzen aber zwingend unabhängige Beobachtungen voraus, da nur dann für große Anzahl von beobachteten Werten diese annähernd normalverteilt sind. Bekanntlich führt eine Ignorierung dieses Sachverhalts zu einer Unterschätzung der Varianzen und Konfidenzintervalle. Im wesentlichen wurden zwei Verfahren zur "Verhinderung" autokorrelierter Beobachtungen vorgeschlagen, die zur Simulation von Rechensystemmodellen geeignet sind (vergl. /WLCH83/). Beide basieren auf folgendem **Konzept**.

> Legt man genügend viele Einzelbeobachtungen in 'Gruppen' oder 'Blöcke' und faßt diese selbst als Beobachtungen auf, dann hat diese vergröberte Sicht u.U. eine wesentlich geringere Autokorrelation als die ursprünglichen Werte:

- Methode der **unabhängigen Replikationen**. Es werden mehrere Experimente mit jeweils unterschiedlichen, möglichst unabhängigen Zufallszahlenfolgen für alle stochastischen Eingabeparameter ausgeführt. Diese klassische Methode erscheint uns für komplexe Simulationsmodelle als zu aufwendig.
- Methode der **unabhängigen Blöcke** (Batch Means). Dabei werden die P beobachteten Werte eines einzigen Experiments, ggf. nach Eliminierung der transienten Beobachtungen, in Q Blöcke (Batches) der Länge P/Q aufgeteilt. Die Blockgröße soll so gewählt werden, daß die j-te Beobachtung im $Block_i$ möglichst unabhängig von der j-ten Beobachtung im $Block_{i+1}$ ist. (Der Unabhängigkeitsgrad kann durch Korrelationskoeffizienten abgeschätzt werden.) Dann enthält die Folge der Q Mittelwerte annähernd stochastisch unabhängige Werte, sodaß man Normalverteilung annimmt. Für den kumulierten Mittelwert und die Varianz werden Konfidenzintervalle bestimmt.
  Das Problem dieser Methode liegt in der Wahl der Blockanzahl Q.

Die in /WLCH83/ definierte Heuristik zur Wahl von Q in Abhängigkeit von der Autokorrelation der Beobachtungen verspricht aber befriedigende Ergebnisse.
Daher wurden unsere VDBS-Untersuchungen mit Hilfe der Batch-Means-Methode zur Analyse aller transaktionsorientierten Beobachtungen realisiert.

5.3 Methode zur integrierten Leistungs-Verfügbarkeitsuntersuchung

Es wurde eine Simulationsmethode für fehlertolerierende Rechensysteme entwickelt, die es gestattet, Leistung und Verfügbarkeit solcher Systeme mit praktikablem Aufwand zu bestimmen. Die statistischen Randbedingungen der Methode werden angegeben. Anschließend wird die Methode in das VDBS-Auswertungsmodell integriert, um sie experimentell anwenden zu können.

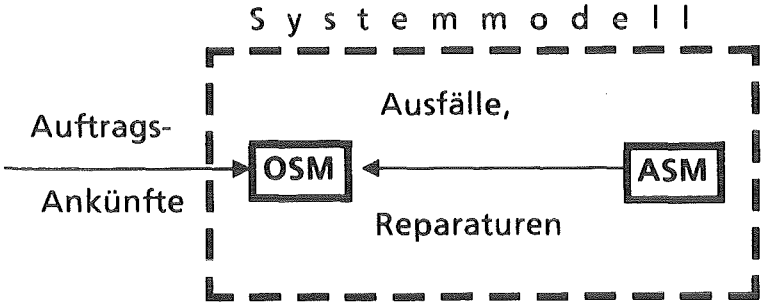
5.3.1 Problemdefinition

Ein verteiltes fehlertolerantes Rechensystem hat folgende Charakteristik:

(1) Es besteht aus zwei stochastischen Prozeß-Typen:
- Ein auftragsorientierter **Mikroprozeß** umfaßt alle HW-/SW-Komponenten, die für die Auftragsabwicklung von der Ankunft bis zur Terminierung von Aufträgen verantwortlich sind.
- Ein ausfallorientierter **Makroprozeß** umfaßt Ausfälle und Reparaturen bestimmter HW-Komponenten (vergl. Kap. 4.2.2).
- Mikroereignisse treten im Mittel wesentlich häufiger auf als Makroereignisse; die jeweiligen dynamischen Prozesse laufen mit stark unterschiedlicher **Zeitgranularität** ab, die ein Maß für die minimale **Zwischenereigniszeit** im jeweiligen Prozeß darstellt.
  In unserem Modell des transaktionsverarbeitenden Mikroprozesses liegt die Zeit zwischen relevanten Systemereignissen in der Größenordnung von einer Millisekunde, während beim Makroprozeß (Arbeitsrechner und NIU-Ausfall / -Wiederanlauf) die Größenordnung 10^9 Millisekunden beträgt

(vergl. Simulationsparameter im Anhang A).

Mikroprozeß und Makroprozeß zerlegen also ein System in eine **Zeithierarchie**, im Gegensatz zur bekannten Zerlegung entsprechend der funktionalen Systemstruktur. Ein Modell eines solchen Systems besteht aus einem **Operations-Submodell** (OSM), in dem Mikroereignisse stattfinden, und einem **Ausfall-Submodell** (ASM). Das 'treibende Element' /MSS83/ dieses Systems ist der (externe) stochastische Prozeß der Auftragsankünfte:



(2) Prinzip des **sanften Leistungsabfalls** ("graceful degradation"):
Ein Ausfall einer einzelnen Komponente führt nicht notwendig zum Gesamtausfall, d.h. das Gesamtsystem reagiert nach jedem weiteren Ausfall mit verminderter Leistung, sofern eine bestimmte Fehleranzahl nicht überschritten ist.
In jedem Zustand verminderter Leistung bestehen unterschiedliche Grade der Erfüllung von Anforderungen, z.B. bezüglich der Systemverfügbarkeit aus Benutzersicht.

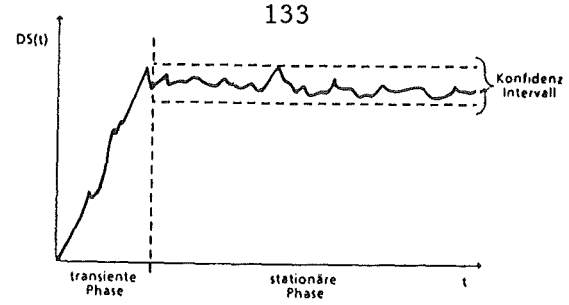
Traditionell werden fehlertolerante Rechensysteme mit folgenden Methoden untersucht:

- Die Systemverfügbarkeit wird durch Anwendung der **Zuverlässigkeitstheorie** (Teil der Theorie stochastischer Prozesse) analytisch untersucht. Mikroereignisse werden dabei nicht betrachtet.
- Die Systemleistung wird (ab einer gewissen Modellkomplexität) durch Simulation ermittelt. Dabei wird **Fehlerfreiheit** angenommen, d.h. Makroereignisse werden nicht betrachtet.

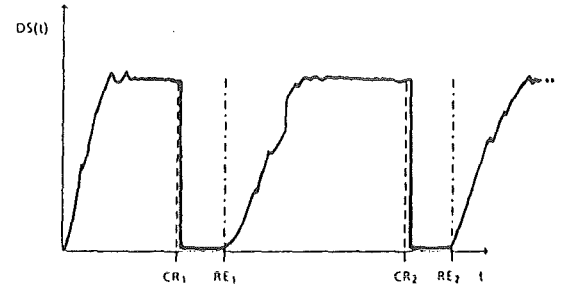
Die Anwendung nur einer dieser Methoden liefert für Systeme mit sanften Leistungsabfall keine befriedigenden Bewertungsmaße für das reale Systemverhalten. Verfügbarkeitsmaße vernachlässigen, daß die

mittlere Leistung in jedem Systemzustand einen anderen Wert haben kann. Leistungsmaße für den fehlerfreien Zustand negieren sowohl verminderte, aber noch existierende Leistung in degradierten Zuständen, als auch deren Ursache, die durch den Makroprozeß beschrieben wird. Hingegen würde die Integration beider Methoden mächtigere Aussagen über das Systemverhalten gestatten, als es jede einzelne Methode -separat angewendet- erlaubt.
Der direkte Ansatz, die Makroereignisse einfach in der Simulation zu berücksichtigen, scheitert in der Praxis am Simulationsaufwand als Folge der zeitlichen Häufigkeitsunterschiede zwischen Mikro- und Makroereignissen.

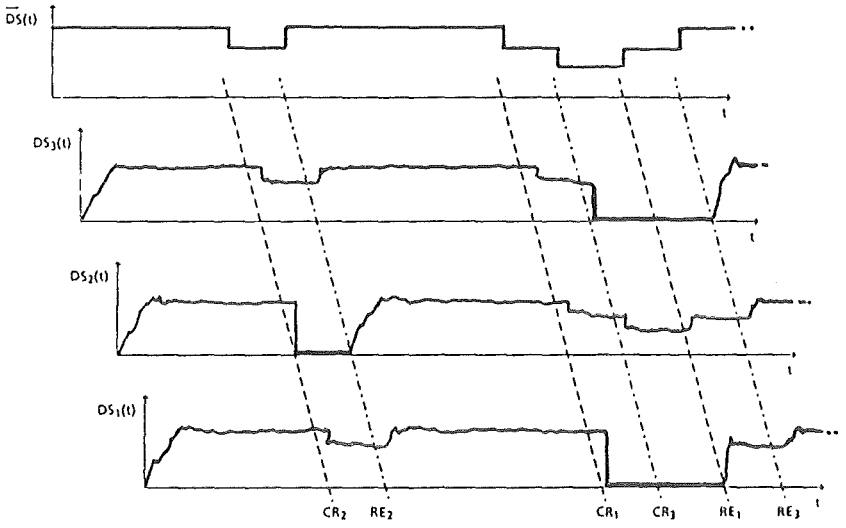
Der Mikroprozeß eines fehlertoleranten Systems soll nun genauer charakterisiert werden. Für den Durchsatz eines fehlerfreien Systems, der bei einer stochastischen Simulation ermittelt wird, ergibt sich ein zeitlicher Verlauf etwa wie in Abb. 5.2-a dargestellt. Nach einer Einschwingphase sollte der Mikroprozeß stationär werden, d.h. mit gegebenem Konfidenzgrad kann der Erwartungswert des Durchsatzes geschätzt werden.
Betrachtet man ein 1-Komponenten-System mit Ausfällen und Restarts, ergibt sich ein zeitlicher Verlauf des Durchsatzes wie in Abb. 5.2-b. Während der Ausfallzeit Δt ist der Durchsatz $DS(\Delta t) = 0$, nach Wiederanlauf sollte sich das System nach einer Einschwingphase stabilisieren.
Nun betrachten wir ein fehlertolerantes 3-Komponenten-System, dessen sanftes Leistungsabfallverhalten in Abb. 5.2-c demonstriert wird. Zu erkennen ist, daß ein Ausfall einer Komponente K_i im Zeitintervall Δt_i zu Durchsatz $DS_i(\Delta t) = 0$ bei allen Aufträgen führt, die diese Komponente bearbeitet. Dieser Ausfall führt bei anderen Komponenten zu Durchsatzeinbußen, sofern die dort bearbeiteten Aufträge Betriebsmittel benötigen, die die ausgefallene Komponente besitzt. Solche Aufträge warten z.B. auf die Reparatur der ausgefallenen Komponente. Die Makroprozeß-Abhängigkeit wird besonders bei der kumulierten Durchsatz-Verteilung DS(t) aller Aufträge im System deutlich.



(a) Durchsatz DS eines fehlerfreien Systems



(b) Durchsatz DS eines Systems mit Reparatur
CR, RE: Crash-, Restart-Ereignis



(c) Durchsatz eines fehlertoleranten Systems aus N = 3 Komponenten

$DS_i(t)$: Durchsatz der an Komponente$_i$ ankommenden Aufträge

$\overline{DS}(t)$: mittlerer Systemdurchsatz

CR_i, RE_i: Crash-, Restart-Ereignis der Komponente$_i$

Abb. 5.2: Integrierte Durchsatz-Verfügbarkeit (Prinzip)

5.3.2 Charakterisierung des Ausfall-Submodells

Annahme 5.1: Makroprozess (vergl. Kap. 4.2.2)

Gegeben sei ein System aus M **gleichartigen**, voneinander **unabhängigen** Komponenten mit fail-stop Ausfallverhalten und konstanten stationären Ausfall- und Reparaturwahrscheinlichkeiten.

Das Ausfall- / Reparaturverhalten aller relevanten (Hardware-)Komponenten eines Systems bezeichnen wir als (stochastischen) **Makroprozess**. Dieser Makroprozeß wird durch einen Zustandsraum Z charakterisiert:

$$Z = \{z_i \mid z_i \in \{UP,DOWN\}^M \wedge i \in [1,q]\}\ ,\ q := 2^M \text{ mit}$$

z_i: Statusvektor,

$z_i = (ST_{i1}, \ldots, ST_{iM})$, $ST_{ij} \in \{UP,DOWN\}$, $i \in [1,q]$, $j \in [1,M]$

Es ist $z_1 = (UP,\ldots,UP)$ der ausfallfreie Zustand,

$z_q = (DOWN,\ldots,DOWN)$ der Totalausfall-Zustand

Im Anfangszustand des Systems seien alle Komponenten intakt, d.h. Anfangszustand ist z_1.

Die Zeit zwischen zwei Ausfällen sei eine exponentialverteilte Zufallsvariable. Die Folge dieser Zeiten bildet einen **homogenen Poisson-Prozess** mit konstantem Erwartungswert ('Mean Time Between Failures', MTBF):

MTBF := MTTF + MTTR

(mittl. Intaktzeit + mittl. Reparaturzeit)

Die Ausfallwahrscheinlichkeit einer Komponente K ist dann proportional zur Ausfallzeit (s. /SISW82/):

p (K zu beliebigem Zeitpunkt ausgefallen)

= mittl. Nichtverfügbarkeit von K,
normiert auf Zeitintervall [0,1]

= MTTR / (MTTF + MTTR)

Aus den Annahmen folgt, daß der Makroprozeß ein **homogener zeitkontinuierlicher Markov-Prozess** mit endlichem Zustandsraum ist (s.

/SISW82, Kap. 5/). Die Homogenität aller betrachteten Komponenten bedeutet zeitlich invariante Ausfall- und Reparaturraten. Bei zeitkontinuierlichem Verhalten können Ausfall- und Reparaturereignisse zu beliebigen Zeitpunkten stattfinden.
Ohne Beweis seien die für uns bedeutenden Eigenschaften eines solchen stochastischen Prozesses genannt:

Satz 5.1:

(a) Die Aufenthaltsdauern in einem Zustand $z_i \in Z$ sind exponentialverteilt.

(b) Die Übergangswahrscheinlichkeiten von Zustand z_i nach z_j hängen nur vom Zustand z_i ab.

Sei $p_i(t)$ die Wahrscheinlichkeit, daß zum Zeitpunkt t das System im Zustand z_i ist. Dann gilt:

(c) Es existiert der Grenzwert

$$p_i := \lim_{t\to\infty} p_i(t) \qquad \text{mit} \sum_{i=1}^{q} p_i = 1$$

(d) Die mittlere Zeit, die in einem Zustand z_i verbracht wird, ist proportional zur Wahrscheinlichkeit p_i.

Die stationären Zustandswahrscheinlichkeiten $p_1,\ldots,p_q$ aller Makrozustände können wie folgt aus den MTTF- und MTTR-Werten berechnet werden, die nach Annahme für alle M betrachteten Systemkomponenten identisch sind. (Eine Verallgemeinerung auf unterschiedliche Ausfallraten findet sich in /SCWS80/.) Dabei wird der Zustandsraum Z entsprechend der Anzahl von Ausfällen partitioniert. Für Beweise der folgenden einfachen Sätze sei ebenfalls auf /SCWS80/ verwiesen.

Satz 5.2: Zustandswahrscheinlichkeiten

Es gelte die Annahme 5.1.
Sei π_j die stationäre Wahrscheinlichkeit jedes Makrozustandes mit genau j Ausfällen, $j\in[0,M]$. Die π_j sind binomialverteilt:

$$\pi_j = a^j(1-a)^{M-j}, \text{ wobei}$$

a: stationäre Nichtverfügbarkeit einer Komponente,

a = MTTR / (MTTR+MTTF)

Sei C_j die Anzahl der Zustände mit genau j Ausfällen:

$$C_j = \binom{M}{j}$$

Sei ANZ_Y die Anzahl der Zustände mit ≤Y Ausfällen:

$$ANZ_Y = \sum_{i=0}^{Y} C_i$$

Sei P_Y die stationäre Wahrscheinlichkeit, daß ≤Y Ausfälle eintreten, Yε[0,M]. Es gilt

$$P_Y = \sum_{k=0}^{Y} C_k \pi_k \text{ mit}$$

$$P_M = 1$$

$$P_0 = (1-a)^M \text{ (stationäre Systemverfügbarkeit)}$$

Schließlich besteht zwischen den π_j (jε[0,M]), und den p_i (iε[1,q]), folgender Zusammenhang:

$$\forall\, k\varepsilon[0,M]:\ \pi_k = p_v = p_{v+1} = \ldots = p_{v+w-1} \text{ wobei } v = ANZ_{k-1},\ w = C_k$$

Die Zustandswahrscheinlichkeiten p_i sind wegen der identischen Ausfallrate aller M Komponenten für eine bestimmte Anzahl k von Ausfällen gleich.

Satz 5.3: Halbordnung der Zustandswahrscheinlichkeiten

Es gelte die Annahme 5.1 und zusätzlich MTTF ≥ MTTR. Dann folgt:

(a) $a \leq \frac{1}{2}$

(b) $\forall\, i\varepsilon[1,q]:\ p_1 \geq p_i \geq p_q$

Wenn also die mittlere Komponentenausfallzeit größer ist als die mittlere Reparaturzeit, ist die Zustandswahrscheinlichkeit im ausfallfreien Fall maximal und bei Totalausfall minimal.

Definition 5.1: Zustandsinklusion

Es gelte die Annahme 5.1.

$\forall\, i,j\ \varepsilon\ [1,q]$:

$z_i \subset z_j$ ("z_i ist in z_j enthalten")

$\Leftrightarrow \forall\, m\varepsilon[1,M]:\ ST_{im} = UP \vee (ST_{im} = DOWN \Rightarrow ST_{jm} = DOWN)$ wobei

$z_i = (ST_{i1},\ldots,ST_{iM})$, $z_j = (ST_{j1},\ldots,ST_{jM})$

Die Zustandsinklusion definiert eine Halbordnung auf dem Zustandsraum. Es ist ein Makrozustand in einem anderen enthalten, wenn im letzteren

mindestens die schon im ersteren ausgefallenen Komponenten ebenfalls ausgefallen sind. Daraus folgen folgende einfachen Beziehungen:

Lemma 5.1: Zustandsinklusion und Zustandswahrscheinlichkeiten

Annahme 5.1 sei vorausgesetzt.
Dann folgt $\forall$ i,j ε [1,q] :

(a) $z_1 \subset z_i \subset z_q$

(b) falls zusätzlich MTTF $\geq$ MTTR, folgt:

$$z_i \subset z_j \Rightarrow p_i \geq p_j$$

Die Zustandsinklusion wird in Kürze zur Charakterisierung des Mikroprozesses benötigt.

5.3.3 Charakterisierung des Operations-Submodells

Ausgangspunkt der Betrachtung sei eine Leistungsgröße eines fehlertolerierenden Systems mit Makro-Zustandsraum Z. Diese Leistungsgröße wird im folgenden mit Λ bezeichnet.

Annahme 5.2: Mikroprozess

Es gelte die Annahme 5.1.
Für jeden Makrozustand z_i, iε[1,q], gelte:

(a) Es existiert ein stationärer, reeller Erwartungswert Λ_i

(b) Der Erwartungswert Λ_i hängt nicht vom z_i-Vorzustand ab (MARKOV-Eigenschaft)

(c) $\Lambda_i \geq 0$

(d) $\Lambda_1 > 0$ (Im Normalfall ist eine nichtleere Leistung vorhanden)

Damit wird vom Systemverhalten speziell bei Zustandsübergängen abstrahiert: Das "Füllen" des Systems mit Aufträgen im ausfallfreien Anfangszustand und nach Reparaturen wird nicht betrachtet; insbesondere werden Fehlererkennungs- und Fehlerbehebungskosten (Restart- und Recoveryaufwand) entsprechend den Modellannahmen in Kap. 4.2.2 vernachlässigt.

Annahme 5.3: Leistungs-Halbordnung

Es gelte die Annahme 5.2. Dann folgt $\forall$ i,j ε [1,q] :

$$z_i \subset z_j \Rightarrow \Lambda_i \geq \Lambda_j$$

Diese Annahme besagt also, daß ausgehend von einem Makrozustand, bei Ausfall weiterer Komponenten (und sonst gleichbleibenden Bedingungen) die jeweilige Leistung höchstens gleichbleiben, sich aber nie verbessern kann. Die Annahme gilt sicherlich nicht für alle denkbaren Leistungsgrößen; die Betrachtung sei aber im folgenden auf genau diejenigen eingeschränkt, für die die Annahme erfüllt ist. In der kumulierten Durchsatz-Verfügbarkeitskurve von Abb. 5.2(c) ist dieses Verhalten angedeutet: Der Gesamtdurchsatz sinkt nach dem 1. Ausfall; er sinkt weiter nach dem 2. Ausfall usw.
Damit haben wir ein fehlertolerantes System hinreichend genau charakterisiert. Im folgenden benötigen wir aber lediglich eine schwächere Aussage:

Satz 5.4: Leistungs-Extremwerte

Es gelte die Annahme 5.3.
Dann folgt $\forall\, i \in [1,q]$:

$$\Lambda_1 \geq \Lambda_i \geq \Lambda_q$$

Der Satz folgt unmittelbar aus Lemma 5.1 (a).

Aus dieser Charakterisierung des Verhaltens eines fehlertoleranten Systems ergibt sich nun folgende **Problemstellung**:

> Gesucht ist eine Methode zur Führung von Simulationsexperimenten, die Zielgrößen des Operations-Submodells unter Einbeziehung von Systemzuständen des Ausfall-Submodells mit vertretbaren Simulationsaufwand und befriedigendem Konfidenzgrad erzeugt.

5.3.4 Die allgemeine Methode

Wir treffen folgende zwei Voraussetzungen:

- Der Zustandsraum Z des Makroprozesses beschreibt das Fehlverhalten der N Arbeitsrechner und N NIUs eines Rechnernetzes, sodaß Annahme 5.1 erfüllt ist:

  $$Z = \{z_i \mid i \in [1,q]\}\ ,\ q := 2^M,\ M := N+N$$

- Gegeben ist eine Leistungsgröße Λ mit stationären Erwartungswerten Λ_i für jeden Makrozustand z_i, entsprechend Annahme 5.2. Auf diesen

Werten sei die in Annahme 5.3 definierte Halbordnung gültig.

Der Methode liegt folgende **Lösungsidee** zugrunde:

- Bestimme durch Simulationsexperimente die stationären Erwartungswerte Λ_i der Leistungsgröße Λ für jedes $i\varepsilon[1,q]$, unter der Bedingung, daß während jedes Experiments keine Zustandsänderung auftritt.
- Wichte jedes Λ_i mit der mittleren Zeitdauer, in dem das System sich im Zustand z_i befindet.
- Die kumulierte Leistung des Systems ergibt sich dann aus der Summe der gewichteten Einzelleistungen.

Intuitiv gesehen, ist dieses gewichtete Maß nur dann sinnvoll, wenn die Leistung Λ_1 im ausfallfreien Zustand z_1 maximal und im Zustand z_q (alle Rechner und NIUs ausgefallen) minimal ist, was genau der Aussage von Satz 5.4 entspricht. Leistungsmaße mit dieser Eigenschaft sind etwa Durchsatz bzw. Erfolgsrate von Transaktionen. (Letzteres Maß wird im nächsten Abschnitt definiert.)
Nach Satz 5.1-d sind für einen Markovschen Makroprozeß die mittleren Aufenthaltszeiten in einem Zustand z_i proportional zur Wahrscheinlichkeit p_i; letztere kann entsprechend unseren Annahmen nach Satz 5.2 einfach berechnet werden. Das führt zur

Definition 5.2: Absolutes Leistungs-Verfügbarkeits-Mass

$$\text{ILVM} := \sum_{i=1}^{q} \Lambda_i \cdot p_i$$

ILVM-Werte liegen stets im Intervall $[\Lambda_1 p_1, \Lambda_1]$:

$$\sum_{i=1}^{q} \Lambda_i p_i \geq \Lambda_1 p_1 \text{ wegen } \Lambda_i \geq 0 \text{ nach Ann. 5.2(c)}$$

$$\sum_{i=1}^{q} \Lambda_i p_i \leq \sum_{i=1}^{q} \Lambda_1 p_i = \Lambda_1 \text{ wegen } \Lambda_i \leq \Lambda_1 \text{ nach Satz 5.4 und } \sum_{i=1}^{q} p_i = 1$$

ILVM ist ein absolutes Maß, mit dem die Leistung verschiedener Protokolle unter realen Bedingungen verglichen werden kann.
Wenn die mittlere Leistung L_A eines Protokolls A in jedem Makrozustand besser als die entsprechende Leistung L_B eines Protokolls B (unter gleichen Umgebungsbedingungen), ist die Aussage "A ist besser als B bezüglich L" stets erfüllt:

$\forall i \varepsilon [1,q]: \Lambda_{i,A}$ 'besser-als' $\Lambda_{i,B}$
$\Rightarrow \quad L_A$ 'besser-als' L_B, mit

$$L_A = (\Lambda_{1,A}, \ldots, \Lambda_{q,A}) , \qquad L_B = (\Lambda_{1,B}, \ldots, \Lambda_{q,B})$$

Wenn hingegen L_A im Normalfall besser, sonst aber schlechter ist als L_B (oder umgekehrt), bietet unser integriertes Maß eine Vergleichsmöglichkeit. Das Ergebnis des Vergleichs hängt dann sowohl von der jeweiligen Leistungsdifferenz L_A-L_B in jedem Makrozustand als auch von der Wichtung dieser Differenzen durch die Zustandswahrscheinlichkeiten ab.

Definition 5.3: Relatives Leistungs-Verfügbarkeits-Mass

$$\boxed{ILVM_{rel} := ILVM \,/\, \Lambda_1} \qquad \text{(nach Ann. 5.2(d) gilt } \Lambda_1 > 0)$$

Das relative Maß gestattet die Bewertung der erzielten Fehlertoleranz eine gegebenen Protokolls, bezogen auf dessen Leistung im ausfallfreien Fall. Dieses Maß kann als **Degradierungsfaktor** eines Systems bezüglich des Leistungsmasses Λ interpretiert werden.

Spezialfälle

Den Spezialfall ILVM = Λ_1 erhält man, wenn man ein ausfallfreies System betrachtet (p_1=1, $p_2 = \ldots = p_q = 0$), bzw. ein ideales System, dessen Leistung trotz Ausfällen nicht abfällt ($\Lambda_2 = \ldots = \Lambda_q = \Lambda_1$). Ein fehler-intolerantes System liefert ILVM = $\Lambda_1 p_1$ wegen $\Lambda_2 = \ldots = \Lambda_q = 0$.

In realen fehlertolerierenden Systemen ist $\Lambda_1 p_1 <$ ILVM $< \Lambda_1$. Je größer ILVM, umso höher ist die Fehlertoleranz des untersuchten Systems.

Damit leisten wir einen Beitrag zur **hierarchischen und hybriden Simulationsmethodik**, indem Simulationsergebnisse der "schnellen" Mikrozeitebene eingesetzt werden in analytische Ergebnisse der "langsamen" Makrozeitebene, ohne das Mikro-Simulationsmodell so weit zu vergröbern, daß es auf der Makroebene analytisch behandelbar würde. Damit geht das **Ersatzverhalten** der schnellen Prozesse durch die Erwartungswerte der Zielgrößen in die Methode ein, was den Begriff "hierarchische Simulation" rechtfertigt.

Unsere Methode zur integrierten Leistungs-Verfügbarkeitsbewertung basiert auf den Arbeiten von J.F. Meyer /MEYE80, MEYE85/. Dort wird das integrierte Maß als **performability** eingeführt. Allerdings ist dieses Maß als Verfeinerung des oben definierten aufzufassen, da anstatt den Erwartungswerten in allen Makrozuständen deren **Verteilungsfunktion** bestimmt wird. Anwendungen des von uns definierten Maßes finden sich z.B. in /LISI84/ zur Bewertung von Kommunikationssystemen bzw. in /TRYU83/ zur Bewertung lokaler Rechnernetze. Die bisherigen Anwendungen setzten die Analysierbarkeit des Operations-Submodells mittels Warteschlangentheorie voraus; daher wurden nur sehr einfache Modelle untersucht. Die vorliegende Arbeit leistet erstmalig einen Beitrag zur Ermittlung der 'performability' für die Systemklasse der verteilten fehlertoleranten Datenbanksysteme.

Folgende Aspekte sind bei Anwendung unserer Methode zu berücksichtigen:

(1) Durch die Stationaritäts-Annahme 5.2(a) des Mikroprozesses in jedem Makro-Zustand machen wir einen durch das stochastische Systemverhalten bedingten Fehler. Insbesondere werden die durch Wiederanlauf einer Komponente verursachten Kosten vernachlässigt. Auch wenn man die Einschwingphasen bei jedem Makro-Zustand z_i berücksichtigen würde, ist nicht bekannt, ob und wieweit A_i vom Vorzustand von z_i abhängt, d.h. ob die Annahme 5.2(b) realistisch ist.

Die genauere, wenn auch sehr viel aufwendigere Methode zur Ermittlung einer Leistung A_i im Makrozustand z_i wäre die folgende:

-- Bestimme die Übergangs-Wahrscheinlichkeiten p_{ji} von beliebigen Zuständen z_j nach z_i analytisch ($j \in [1,q]$, $j \neq i$)

-- Führe q-1 Simulationsexperimente, indem für jeden möglichen Übergang z_j --> z_i der stationäre Erwartungswert $A_j(i)$ geschätzt wird.

-- Wichte die Ergebnisse:

$$A_i := \sum_{i \neq j}^{q} A_j(i) * p_{ji}$$

Ob dieser Aufwand gerechfertigt ist, hängt von den Eigenschaften des jeweiligen Modells ab. Die Betrachtung aller Zustandsübergänge ist

sehr aufwendig, bietet andererseits aber die Möglichkeit, die im Operations-Submodell realisierten Restart-Verfahren vollständig auf ihre funktionale Korrektheit zu testen.

(2) Der offensichtliche Nachteil der Methode ist der mit der Komponentenanzahl des Makromodells exponentiell wachsende Zustandsraum mit entsprechend wachsender Anzahl Simulationsexperimente. Konkret ist die Anzahl für einen gegebenen Eingabeparameter-Vektor:
   - q bei der allgemeinen Methode,
   - q*(q-1) bei der verbesserten Methode mit Berücksichtigung der Übergänge.

Das in /MEYE85/ entwickelte alternative Verfahren zur Bestimmung der Verteilungsfunktion einer Leistungsgröße in jedem Makrozustand ist noch wesentlich aufwendiger.
Der Untersuchungsaufwand macht beide Methoden schon für kleinere Komponentenanzahl M (M≥5) unpraktikabel. Gesucht wird also eine effizientere Methode, die den exponentiellen Zustandsraum durch Vernachlässigung der **weniger wahrscheinlichen** Fehlerfälle reduziert, ohne den Fehler dabei zu groß werden zu lassen.

5.3.5 Die effiziente Näherungsmethode

Der Zustandsraum Z wird nun soweit reduziert, daß ein hoher, gegebener Anteil wahrscheinlichster Zustände erfaßt und der restliche Zustandsraum vernachlässigt wird. Es wird gezeigt, daß der durch diese Approximation induzierte Fehler tolerierbar ist, auch wenn nur ein Bruchteil des gesamten Zustandsraums betrachtet wird.

Es gelten die gleichen Annahmen wie bei der allgemeinen Methode; zusätzlich sei MTTF ≥ MTTR, sodaß nach Satz 5.3 p_1, die Zustandswahrscheinlichkeit im Normalfall, am größten ist.

Gesucht wird die minimale Anzahl von x Zuständen, $x \leq q$ ($q=2^M$), sodaß gilt

$$p_1 \geq p_2 \geq \ldots \geq p_x$$

$$\text{mit } P_\alpha := \sum_{i=1}^{x} p_i \text{ sodaß}$$

$$1 \geq P_\alpha \geq 1-\alpha$$

Dabei ist α ein gegebener Anteil nicht erfaßter Zustände. $1-\alpha$ heißt der **Überdeckungsfaktor** des Zustandsraums Z. Nach Satz 5.1-d entspricht α auch dem nicht erfaßten Zeitanteil aller M Ausfall-/Wiederanlaufprozesse im Rechnernetz, wenn man die Beobachtungszeit als unbeschränkt betrachtet.
Gemäß Satz 5.2 kann bei Unabhängigkeit der Makroeignisse und einer für alle M Komponenten gleicher Ausfallrate a die minimale Anzahl x zu betrachtender Makrozustände wie folgt bestimmt werden:

- Gegeben seien M, a und $1-\alpha$

- Finde die kleinste Ausfallanzahl Y, sodaß $P_Y \geq 1-\alpha$ ($Y \varepsilon [0,M]$)
  (P_Y ist die in Satz 5.2 definierte stationäre Wahrscheinlichkeit für $\leq Y$ Ausfälle)

- Setze $x := ANZ_Y$
  (P_Y ist die Anzahl Makrozustände mit $\leq Y$ Ausfällen)

In Abhängigkeit vom Überdeckungsfaktor sei nun

Definition 5.4: Approximierte Leistungs-Verfügbarkeit

$$ILVM_\alpha := \sum_{i=1}^{x} (\Lambda_i \cdot p_i) \; / \sum_{i=1}^{x} p_i$$

Der Spezialfall x=q ergibt das allgemeine Maß ILVM, bei x=1 reduziert sich die Approximation auf die Leistung Λ_1 im Normalfall.
Es kann gezeigt werden, daß unter Annahme 5.3 der Leistungs-Halbordnung für den Approximationsfehler F,

$$F = |ILVM - ILVM_\alpha| \; / \; ILVM$$

stets gilt

$$F \leq \alpha / p_1$$

Beweis

Sei $S_{u,o} := \sum_{i=u}^{o} \Lambda_i p_i$, $P_\alpha = \sum_{i=1}^{x} p_i$
Da keine Aussage über das Vorzeichen von $ILVM - ILVM_\alpha$ möglich ist, machen wir eine Fallunterscheidung:

Fall 1: $ILVM - ILVM_\alpha \geq 0$

$$
\begin{aligned}
F &= (ILVM - ILVM_\alpha) / ILVM \\
&= (P_\alpha S_{1,q} - S_{1,x}) / (P_\alpha S_{1,q}) \\
&= (P_\alpha (S_{1,x} + S_{x+1,q}) - S_{1,x}) / (P_\alpha S_{1,q}) \\
&= (S_{x+1,q} P_\alpha - S_{1,x}(1-P_\alpha)) / (P_\alpha S_{1,q}) \\
&\leq (\Lambda_1 (1-P_\alpha) P_\alpha - S_{1,x}(1-P_\alpha)) / (P_\alpha S_{1,q}) \\
&\qquad \text{wegen } S_{x+1,q} \leq \Lambda_1 (1-P_\alpha) \text{ (Ann. 5.3)} \\
&\leq (\Lambda_1 (1-P_\alpha) P_\alpha - S_{1,x}(1-P_\alpha)) / (\Lambda_1 p_1 P_\alpha) \\
&\qquad \text{wegen } S_{1,q} \geq \Lambda_1 p_1 \qquad \text{(Ann. 5.2)} \\
&= (\Lambda_1 P_\alpha - S_{1,x})(1 - P_\alpha) / (\Lambda_1 p_1 P_\alpha) \\
&\leq \Lambda_1 P_\alpha (1 - P_\alpha) / (\Lambda_1 p_1 P_\alpha) \\
&= (1 - P_\alpha) / p_1 \\
&\leq \alpha / p_1 \qquad \text{da } P_\alpha \geq 1-\alpha \quad \text{(nach Voraussetzung)}
\end{aligned}
$$

Fall 2: $ILVM - ILVM_\alpha < 0$

$$
\begin{aligned}
F &= (ILVM_\alpha - ILVM) / ILVM \\
&= (S_{1,x} - P_\alpha S_{1,q}) / (S_{1,q} P_\alpha) \\
&= (S_{1,x} - P_\alpha (S_{1,x} + S_{x+1,q})) / (S_{1,q} P_\alpha) \\
&= (S_{1,x}(1-P_\alpha) - S_{x+1,q} P_\alpha) / (S_{1,q} P_\alpha) \\
&\leq (S_{1,x}(1-P_\alpha) / (S_{1,q} P_\alpha) \\
&\leq (\Lambda_1 P_\alpha (1-P_\alpha) / (S_{1,q} P_\alpha) \quad \text{wegen } S_{1,x} \leq \Lambda_1 P_\alpha \text{ (Ann. 5.3)} \\
&\leq (\Lambda_1 P_\alpha (1-P_\alpha) / (\Lambda_1 p_1 P_\alpha) \quad \text{wegen } S_{1,q} \geq \Lambda_1 p_1 \text{ (Ann. 5.2)} \\
&= (1 - P_\alpha) / p_1 \\
&\leq \alpha / p_1 \qquad \text{da } P_\alpha \geq 1-\alpha \quad \text{(nach Voraussetzung)} \quad \square
\end{aligned}
$$

Je kleiner also der Anteil α nicht erfaßter Makrozustände gegenüber der Systemverfügbarkeit p_1 ist, umso geringer ist der Approximationsfehler. Insbesondere ist der Fehler unabhängig von den Leistungswerten, was ja ein recht erstaunliches Ergebnis darstellt.

Damit ist eine ausreichende Rechtfertigung der Aproximationsmethode

gegeben. Abschließend soll demonstriert werden, daß bei realistischen Ausfallwahrscheinlichkeiten bereits durch sehr wenige Zustände eine hohe Überdeckung des Zustandsraums erreicht wird, d.h. daß α und damit der Approximationsfehler klein gehalten werden kann. Die diesem Beispiel zugrundeliegenden Annahmen entsprechen den in Anhang A definierten Parametern unseres VDBS-Simulationsmodells.

Gegeben sei

- Rechnernetz mit 7 Arbeitsrechnern und 7 NIUs
- Identische stationäre Ausfall- und Reparaturzeiten
  MTTF = 100h, MTTR = 2.5h für alle Arbeitsrechner und NIUs
- Überdeckungsfaktor $1-\alpha = 95\ \%$

Es folgt $P_1 = 0.955 > 0.95$ und $x = ANZ_1 = 15$
Berücksichtigt man also den fehlerfreien Zustand sowie alle 14 Einzelausfall-Zustände, dann erzielt man eine Überdeckung von 95% des gesamten, aus 2^{14} Zuständen bestehenden Zustandsraums. Der relative Fehler F bei der Bestimmung der Leistung-Verfügbarkeit beträgt höchstens 7.1% , da

$$F \leq \alpha/p_1 = 0.05/0.707727 = 0.0706$$

5.3.6 Randbedingungen zur Anwendung der Methode

Die wesentlichen Randbedingungen zur Anwendung der allgemeinen Methode sind:

- Unabhängigkeit der Ausfallereignisse. Diese folgt bereits aus dem oben angenommenen einfachen Markovschen Modell. Komplexere Fehlermodelle sind aber nur dann sinnvoll einsetzbar, wenn über Ausfall- und Reparaturraten gesicherte Erkenntnisse vorliegen.

- Stationarität der Mikroprozesse in jedem Makrozustand. Auch diese Abstraktion ist akzeptabel, da transiente Effekte in den Mikroprozessen vernachlässigbar sind:
  -- Nach Modellannahme gilt $MTTF_K \gg MTTR_K \gg$ Wiederanlaufzeiten der durch K-Ausfall betroffenen Mikroprozesse für alle Komponenten K, die ausfallen können. Daher können diese Zeiten vernachlässigt werden.
  -- Nach Modellannahme sind 'operationsorientierte' Ereignisse im Mittel wesentlich häufiger als 'ausfallorientierte'

Ereignisse, sodaß erwartet werden kann, daß Mikroprozesse in ausfallfreien Phasen tatsächlich stationär werden können.

- Wichtung der Leistungen entsprechend den mittleren Zustandsdauern. Diese Wichtung in der ILVM-Definition sind zwar intuitiv gesehen sinnvoll, entsprechen aber nicht notwendig den Benutzeranforderungen: Z.B. sind in bestimmten Anwendungen Systemteilausfälle so gravierend, daß sie in die integrierte Leistungs/Verfügbarkeitsbewertung stärker eingehen sollten als nur mit der mittleren Ausfalldauer. Daher kann die ILVM-Definition verallgemeinert werden, indem die stationären Verfügbarkeiten p_i ersetzt werden durch entsprechend den Anforderungen **intervallskalierten Werten**.

- Statistisches Genauigkeitsproblem. Eine Anwendung der Methode erscheint wenig sinnvoll, wenn die mittlere System-Unverfügbarkeit nicht wesentlich größer ist als der geschätzte Fehler bei der Ermittlung der stationären Erwartungswerte, der durch die Konfidenzintervalle dargestellt wird. Andernfalls würde die aufwendige Ermittlung dieser Werte in den Ausfallzuständen nur zu einer Korrektur der Werte führen, die sich innerhalb der Genauigkeit der Erwartungswerte selbst bewegt.
  Bezogen auf das Beispiel im letzten Abschnitt bedeutet das: Die Nichtverfügbarkeit in unserem VDBS-Modell beträgt bei 7 Knoten 29.2% . Die Konfidenzintervalle der Erwartungswerte sollten also weit unter diesem Wert liegen.

Insgesamt erscheinen die Randbedingungen der allgemeinen Methode akzeptabel. Die wesentliche Randbedingung zur Anwendung der Näherungsmethode ist die Wahl eines genügend großen Überdeckungsfaktors, sodaß der ausgesparte Rest-Zustandsraum wesentlich kleiner als die stationäre Systemverfügbarkeit ist. Das obige Beispiel hat gezeigt, daß bei gängigen Zuverlässigkeits-Kenngrößen bereits die Betrachtung eines Bruchteils des gesamten Zustandsraums dieses Kriterium erfüllt.

5.4 Das VDBS-Auswertungsmodell

Die wesentlichen **relevanten Zielgrößen eines VDBS** aus Benutzersicht sind

- **Transaktions-Antwortzeit, -Durchsatz** als Leistungsmaße;
- **Transaktions-Erfolgsrate, -Blockierungsrate** als Verfügbarkeitsmaße.

Dazu wird im folgenden ein **Auswertungsmodell** für die von uns betrachtete Klasse von VDBS entwickelt. Ausgangspunkt ist das in Kap. 3.3 entworfene VDBS-Architekturmodell, wie es Abb. 3.2 zeigte. Um daraus ein Auswertungsmodell zu konstruieren, ist folgendes zu beachten:

- Das Auswertungsmodell soll abstrakt sein, d.h. es soll die Vorschriften zur Auswertung aller Zielgrößen enthalten; die Vorschriften sollen unabhängig von einem konkreten VDBS-Protokoll definiert sein.

- Das Auswertungsmodell soll lediglich solche Subsysteme enthalten, die zur Ermittlung der obigen Größen notwendig sind, d.h. dieses Modell ist als Vergröberung des vollständigen VDBS-Simulationsmodells aufzufassen.

- Es genügt, lediglich die Transaktions-Koordinationsprozesse $TM_1,\ldots,TM_N$ auf den N Knoten des Rechnernetzes zu betrachten, um die obigen Zielgrößen zu ermitteln, da genau diese die Ankunft und Terminierung aller Transaktionen überwachen. Damit gelten die Voraussetzungen des N-Knoten-Modells von Kap. 5.1. Ein Prozeß TM_i wir dabei entweder 1:1 auf ein 1-Dienstmodell abgebildet, oder durch entsprechende Verfeinerung auf K>1 Dienste. Die Verfeinerung kann für verschiedene Zielgrößen unterschiedlich erfolgen.

- Das Netz der benötigten Auswertungs-Instanzen soll den in Kap. 5.1 festgelegten Einschränkungen der Netzklasse genügen, um die dort definierten Leistungsmaße anwendbar zu machen.

Ein mit diesen Einschränkungen konformes Auswertungsnetz ist in Abb. 5.3 gezeigt. Gegenüber dem Modell in Abb. 3.2 sind alle Verarbeitungsfunktionen außer der Transaktions-Terminierung in einer

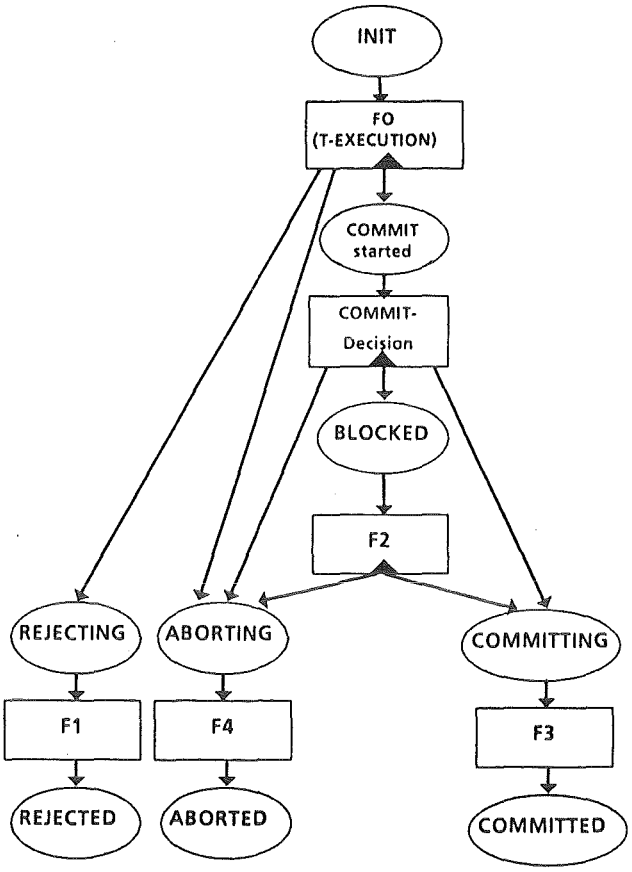


Abb. 5.3: VDBS-Auswertungsinstanzen auf Transaktionsmanager-Prozeß

Instanz F0 vergröbert. Vor ihrer Terminierung wird jede Transaktion von einer der Instanzen F1, F3 oder F4 bearbeitet. Eine Transaktion kann nur dann blockiert werden (Instanz F2), wenn bestimmte Rechnerausfälle während der 1. Commit-Phase auftreten (vergl. Transaktionsausführung, Kap. 3.3.4).

Im VDBS-Auswertungsmodell werden folgende Methoden zur Ermittlung transaktionsorientierter Zielgrößen verwendet:

- Operationale Analyse zur Bestimmung aller relevanter zu Antwortzeiten, Durchsätze und Verzweigungsraten liefert die Beobachtungs-Zeitreihen
- Eine Heuristik zur Eliminierung transienter Beobachtungen aus diesen Zeitreihen,

- Die Batch-Means-Methode zur Schätzung stationärer Erwartungswerte und Varianzen sowie zur Bestimmung der Konfidenzintervalle,
- Eine integrierte Leistungs-Verfügbarkeitsuntersuchung zur Berücksichtigung von Ausfallzuständen bei den Experimenten und die Einbeziehung der Ausfallraten in die betrachteten Leistungsgrößen.

Die wichtigsten Größen im N-Knoten/1-Dienstmodell entsprechend den Definitionen von Kap. 5.1 sind nun:

- **Antwortzeit** erfolgreicher Transaktionen: $RESP_{03,NET}$
- **Durchsatz** erfolgreicher Transaktionen: $DS_{03,NET}$
- **Erfolgsrate** (Commit-Wahrscheinlichkeit):

$$qv_{commit} := \sum_{\theta=1}^{N} TERM_{03}(\theta) \;/\; NT, \text{ mit } NT = \sum_{\theta=1}^{N} TERM_{0}(\theta)$$

(NT ist die Anzahl innerhalb τ beendeter Transaktionen im Gesamtsystem.)

Dieses für den Endbenutzer wichtige Maß gibt an, wie hoch die Wahrscheinlichkeit für den erfolgreichen Abschluß einer einzelnen Transaktion ist.

Als weitere, datenbankorientierte Zielgrößen werden betrachtet:

- **Anzahl Subtransaktionen** pro Transaktion:

$$NSUB := \sum_{i=1}^{NT} NSUB_i \;/\; NT$$

$NSUB_i$ ist die Anzahl Subtransaktionen der i-ten Transaktion. NSUB ist ein Maß für die Komplexität der physischen Transaktionen, die aus dem Transaktionsprofil in Abhängigkeit von der Datenverteilung in der Anfrageauswertungsphase generiert werden.

- **Konfliktrate** pro Transaktion:

$$CONFL := (\text{Anzahl T-Konflikte innerhalb } \tau) \;/\; TERM_{03}$$

Ein **T-Konflikt** tritt auf, wenn bei einer Transaktion T mindestens eine Subtransaktion SUB mit einem SUB-Konflikt existiert.

Ein **SUB-Konflikt** tritt auf, wenn bei einer Subtransaktion SUB mindestens eine Dateneinheit DE,

DE ε W(SUB) υ R(SUB)

in der physischen Basismenge existiert, deren unverzügliches Sperren wegen einer bereits bestehenden, inkompatiblen DE-Sperre verhindert wird.

Eine hohe Konfliktrate bedingt eine zusätzliche Verzögerung von Transaktionen ('serialization delay') durch den Sperr-Scheduler und kann daher zur Erklärung der Ursachen einer ungenügenden Antwortzeit dienen.

Die Blockierungsrate ist ein Maß für die Wartezeit einer (Schreib-) Transaktion für die Reparatur anderer Rechner bzw. den Verbindungen zu anderen Rechnern. Diese Größe wird aber nicht weiter betrachtet: Im ausfallfreien Fall treten keine Blockierungen auf; Blockierungen bei bestimmten Ausfällen gehen in die Leistung nichtblockierter Transaktionen indirekt ein, da blockierte Schreib-Transaktionen Daten gesperrt halten können, die für andere solange unzugänglich sind.

Die benötigten Leistungskenngrößen werden bei der Analyse terminierter Transaktionen auf **allen Rechnerknoten** ermittelt, was bei einer Simulation leicht möglich ist. Daher beziehen sich die erhaltenen Ergebnisse auf das **Leistungsverhalten im gesamten Rechnernetz**.
Eine Verfeinerung des definierten Zielgrößensystems, z.B. in Richtung auf die Unterscheidung von Lese- / Schreib-Transaktionen sowie von Transaktionsklassen (gegeben durch die Kardinalität der Basismenge) wäre leicht möglich; wir betrachten aber lediglich die aggregierten Leistungsgrößen. Alle systemorientierten Leistungsgrößen, wie z.B. Auslastungen der Betriebsmittel, wirken sich auf diese Größen aus.

Zur **integrierten Leistungs-Verfügbarkeit** betrachten wir den Durchsatz erfolgreicher Transaktionen bzw. die Erfolgsrate d_i mit

$d_i := DS_{03}$ im Makrozustand z_i, i=1,2,3, bzw.

$d_i := qv_{commit}$ im Makrozustand z_i, i=1,2,3, bzw.

Dabei werden die wahrscheinlichsten Makrozustände erfaßt:

(z1) **Normalfall,** d.h. alle N Arbeitsrechner und N NIUs intakt, mit Zustandswahrscheinlichkeit π_1,

$\pi_1 = (1-a)^{2N}$ wobei

a = MTTR/(MTTR+MTTF) : stationäre Nichtverfügbarkeit eines Arbeitsrechners AR bzw. eines Netzzugangsrechners NIU

(z2) **Arbeitrechner-Ausfall** eines AR_j, $j \varepsilon [1,N]$ mit Zustandswahrscheinlichkeit π_2,

$$\pi_2 = a(1-a)^{2N-1}$$

(z3) **NIU-Ausfall** einer NIU_j, $j \varepsilon [1,N]$ mit Zustandswahrscheinlichkeit

$$\pi_3 = \pi_2$$

Im Makro-Zustand z3 sind alle Rechner AR_i, $i \neq j$, intakt und wechselseitig verfügbar. Rechner AR_j ist vom übrigen Rechnernetz getrennt, d.h. nur dessen lokale Transaktionen können während des NIU-Ausfalls bearbeitet werden.

Als Approximation des Durchsatzes erfolgreicher Transaktionen bzw. der Erfolgsrate unter Berücksichtigung des ausfallfreien Falls und aller Einzelausfälle wird entsprechend der Definition in Kap. 5.3.5 ein integriertes Durchsatz- bzw. Erfolgsraten-Verfugbarkeitsmaß ILVM gebildet:

$$ILVM = (\pi_1 \cdot d_1 + N \cdot \pi_2 \cdot d_2 + N \cdot \pi_3 \cdot d_3) \; / \; (\pi_1 + N \cdot \pi_2 + N \cdot \pi_3)$$

Nach Einsetzen der π_i-Werte ergibt sich

$$ILVM = ((1-a) \cdot d_1 + Na \cdot (d_2 + d_3) \; / \; (1-a+2Na)$$

Diese Definition ist nur dann anwendbar, wenn

- Die Ausfallereignisse z_1 --> z_2 und z_1 --> z_3 voneinander unabhängig sind (Ann. 5.1, Kap. 5.3.2),
- In jedem betrachteten Makrozustand die Folge der terminierenden Transaktionen einen stationären Mikroprozeß bilden, d.h. daß endliche Erwartungswerte d_1, d_2, d_3 für den Durchsatz bzw. die Erfolgsrate existieren (Ann. 5.2, Kap. 5.3.3),
- Die Erwartungswerte d_2 bzw. d_3 nach einem beliebigen AR_j- bzw. NIU_j-Ausfall ($j \varepsilon [1,N]$) gleich sind.

In unserem Modell wurde Homogenität bezüglich der Verteilung von Daten, Diensten und Transaktionslasten angenommen. Daher sollte sich der Ausfall eines beliebigen Arbeitsrechners im Mittel gleich auf alle Leistungsgrößen auswirken. Entsprechendes gilt auch bei

NIU-Einzelausfall, wegen der 'homogenen' topologischen Rechnernetzstruktur (Ring). Allerdings sind diese beiden Ausfallarten zu unterscheiden, da sie zwar in unserem VDBS-Modell gleichwahrscheinlich sind, aber zu unterschiedlichem Leistungsverhalten führen: Bei NIU-Ausfall sind lokale Transaktionen auf dem zugehörigen Arbeitsrechner noch ausführbar.

Danach sind für eine gegebene Parameterkombination genau drei Experimente erforderlich. Anfangszustand zum Zeitpunkt t=0 sei z_1.

(1) Zur Bestimmung von d_1 bleibt der Anfangszustand im Beobachtungszeitraum $[0,\tau]$ unverändert.

(2) Zur Bestimmung von d_2 wird zu einem beliebigen Zeitpunkt $t'>0$ ein AR-Ausfall eines beliebigen Arbeitsrechners (Zustandsübergang z_1 --> z_2) erzeugt und bis zum Zeitpunkt $t'+\tau$ unverändert gelassen. Beobachtungszeitraum ist $[t',t'+\tau]$.
Wegen der Unabhängigkeit vom Vorzustand wäre es sogar nicht erforderlich, daß z_1 Anfangszustand ist. Wegen der Stationaritätsannahme ist die Wahl des Crash-Zeitpunktes t' beliebig. Wegen der Homogenitätsannahme kann der Arbeitsrechner AR_j, $j\varepsilon[1,N]$, beliebig gewählt werden.

(3) Zur Bestimmung von d_3 wird zum Zeitpunkt $t'>0$ ein beliebiger NIU-Ausfall erzeugt, weiter wie in (2).

Bei allen Experimenten ist im Beobachtungszeitraum eine transiente Anfangsphase zu berücksichtigen.

Die in diesem Kapitel definierten Maße und deren Schätzmethoden bilden die Grundlage zum Entwurf der Auswertungskomponente des im nächsten Kapitel beschriebenen Modellierungswerkzeuges, sowie zum Entwurf der VDBS-Simulationsexperimente in Kapitel 8.

6 VERALLGEMEINERTE FUNKTIONSNETZE ZUR MODELLIERUNG FEHLERTOLERANTER VERTEILTER SYSTEME

Funktionsnetze (FN) /GODB83/ bieten eine mächtige Modellierungssprache auf der Basis von Petrinetzen mit Zeitverbrauch, die mit Erfolg zur Spezifikation und quantitativen Untersuchung diskreter stochastischer Systeme angewendet worden sind /SCHI84, WST82/.
Im Rahmen der vorliegenden Arbeit wurde zunächst diese Sprache zur Modellierung VDBS-typischer Protokolle angewendet. Die dabei gewonnenen Erfahrungen führten sowohl zu Modifikationen der FN-Konstrukte als auch des darauf basierenden Softwaretools "FUN":

- Die Konzeptionen der Funktionsnetze wurde um neue Konstrukte angereichert.

Diese Konstrukte sind meist solche, die sich durch **Netzinterpretation** auf ein (komplexeres) verhaltensäquivalentes Funktionsnetz zurückführen lassen, wie noch gezeigt wird. Einige Konstrukte beinhalten andererseits eine **Verallgemeinerung** der Schaltregel von Funktionsnetzen, die an der Schaltregel der Prädikats-Transitions-Netze angelehnt ist.
Die Modifikationen sind auf die Modellierung von **Rechensystemen** zugeschnitten, während die ursprüngliche Konzeption von der generellen Anforderung ausging, beliebige diskrete Systeme modellieren zu können. Es wird an Hand von VDBS-Modellierungsproblemen gezeigt, daß die vorgeschlagenen Konstrukte zu sehr kompakten und gut strukturierten Modellen führen. Andererseits erweist sich zur Modellierung aller VDBS-Protokolle (Kap. 7) und ihres Umgebungsmodells (Kap. 4) bereits eine Teilmenge der FN-Sprache, zusammen mit den vorgeschlagenen Konstrukten, als ausreichend. (Für andere Systemklassen können alle FN-Elemente sinnvoll bzw. notwendig sein.) Das Ergebnis der modifizierten Funktionsnetze sind die **DAEMON**-Netze
(**D**istributed System **A**vailability **E**valuation and **MO**deling using **N**ets).
Die DAEMON-Netzsprache ist Gegenstand von Kapitel 6.1. In Kap. 6.2 wird die Vorgehensweise bei der Spezifikation von DAEMON-Netzen beschrieben und an einem einfachen verteilten Systemmodell in Kap. 6.3 exemplarisch erläutert. In Kapitel 6.4 wird die Analysierbarkeit von DAEMON-Netzen sowohl im Sinne der Allgemeinen Netztheorie als auch im

Sinne der Markovtheorie untersucht.

- Das Softwaresystem FUN wurde erweitert.

Das resultierende Werkzeug **DAEMON** unterstützt die Eingabe, Manipulation, Simulation und Analyse von sowohl Funktions- als auch DAEMON-Netzen. Bei der Größe und Komplexität der erstellten VDBS-Modelle (insgesamt ca. 300 Kanäle und 400 Instanzen) war die **Effizienz** und der **Komfort** der Modellierumgebung eine wesentliche Anforderung an den DAEMON-Systementwurf; bei früheren FN-Anwendungen bestand diese Anforderung nicht in diesem Ausmaß. Das Werkzeug besitzt nun eine graphische Eingabeschnittstelle, eine interaktive Schnittstelle zur Experimentüberwachung ("net debugger") sowie verbesserte und erweiterte statistische Ergebnisanalysemoduln.
Architektur und Benutzerschnittstellen von DAEMON werden in Kapitel 6.5 erläutert. Das DAEMON-Netzkonzept und dessen Realisierung wurde in Teilen bereits in /LEG085/ vorgestellt.

6.1 Definitionsgerüst der DAEMON-Netze

Bei der Modellierung fehlertoleranter verteilter Rechensysteme im Rahmen dieser Arbeit traten folgende Probleme mit der Verwendung von Funktionsnetzen (FN) als Modellierungssprache auf, die zu einer Weiterentwicklung -den DAEMON-Netzen- führten:

- Der für Rechensysteme zentrale Begriff der **Ressource** (Betriebsmittel) ist bei dem Zeitverhalten dieser Systeme von großer Bedeutung. So ist für die Leistungsanalyse z.B. das Ressourcen-Zuteilungsverfahren, für die Zuverlässigkeitsanalyse z.B. die Ressource-Ausfallrate relevanter Inputparameter. Ressourcen und ihre Parameter sind nicht Teil der FN-Sprache; sie müssen daher bei jedem Modell eines Rechensystems als Subnetz spezifiziert werden.
  In DAEMON sind parametrisierte Ressourcen vorgesehen. Das führt zu besserer Modellstrukturierung und zu wesentlich kompakteren Modellen.

- Bei einer diskreten, detaillierten Modellierung komplexer Systeme durch Netze sind strukturierte, unterscheidbare Marken

unumgänglich. Im FN-Konzept können beliebig strukturierte Marken spezifiziert werden. Diese Spezifikation ist aber in der Netzgraphik nicht sichtbar; die Verantwortung für 'korrekte' Markendeklaration und -manipulation liegt allein bei den (programmiersprachlich formulierten) Instanzalgorithmen. Die Verständlichkeit großer Modelle wird insbesondere dann reduziert, wenn verschiedene unterschiedliche Markeninterpretationen innerhalb eines Modells existieren, ohne daß diese Unterschiede Teil der 'sichtbaren' Modellspezifikation wären. Gerade in VDBS-Modellen können Marken sehr unterschiedliche Semantik haben, z.B.:

VDBS-SUBSYSTEM	MARKEN-SEMANTIK
Transaktionsverwaltung	(Sub-) Transaktionen
lokales DBVS	Dateneinheiten
Recovery-/ Restart-SW	Fehler- / Reparaturereignisse
Task Management	HW-Betriebsmittel (Rechner, Speicher)
	Steuermarken zur Tasksynchronisation
Rechnernetz-SW	Nachrichten zwischen Rechnerknoten und
	zwischen verteilten Prozessen

Wir schlagen daher eine stärkere **Formalisierung** der Markenstrukturen vor, indem zu der graphischen (und z.T. algorithmischen) FN-Beschreibung eine **Markentyp-Spezifikation** gehört, die sich am Relationenmodell anlehnt.

- Bei indeterministischen Modellen, in denen einzelne Aufträge unterscheidbar sind, führt die Verwendung von FIFO- und LIFO-**Bearbeitungsstrategien** zu unnötigen Modellbeschränkungen. Z.B. ist damit keine prioritätsorientierte Auftragsbearbeitung modellierbar, da bei FIFO bzw. LIFO die Ankunftsreihenfolge bereits die Bearbeitungsreihenfolge der Aufträge festlegt. DAEMON erlaubt die Modellierung beliebiger Strategien durch die explizite Spezifikation eines **Markenselektions-Verfahrens**, sodaß die Reihenfolge evtl. erst "zur Laufzeit" eines Modells bestimmt wird.

Durch die explizite Formulierung von Ressourcen, Markentypen und Markenzugriffsverfahren in DAEMON-Netzen konnten **Strukturiertheit**, **Netz-Kompaktheit** und **Flexibilität** des Funktionsnetz-Ansatzes wesentlich gesteigert werden, wenn auch diese Verbesserungen durch umfangreichere Netzsprache und Effizienzeinbußen beim Simulator erkauft wurden.

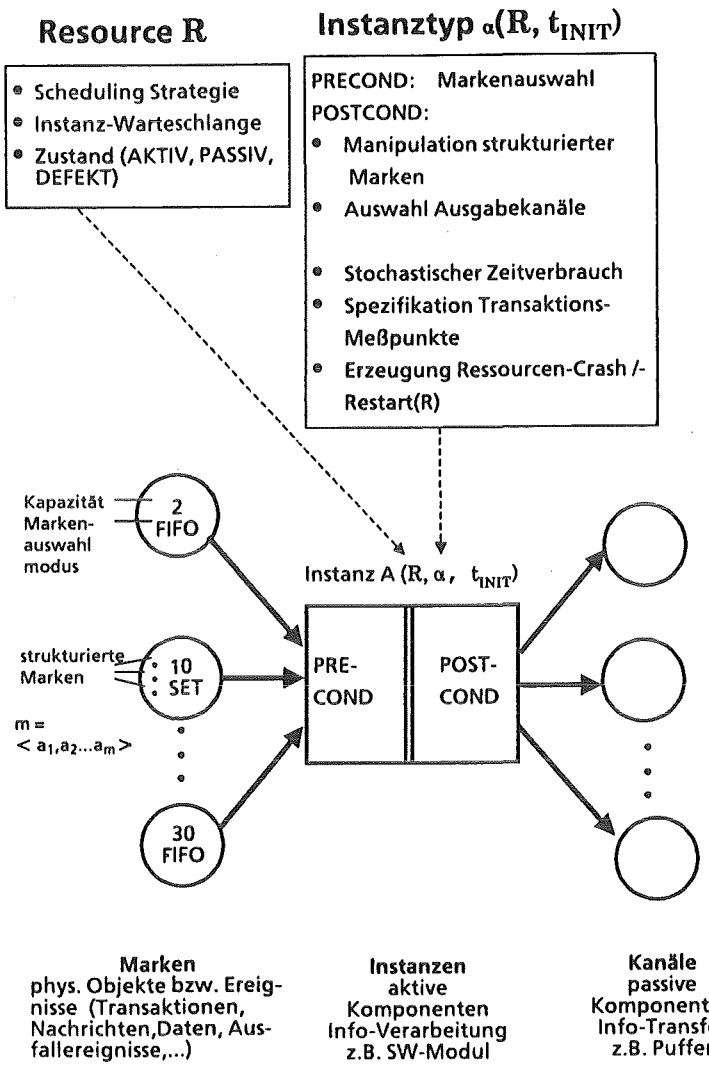


Abb. 6.1: Überblick über die Elemente der DAEMON-Netzsprache

Ein Überblick über die wesentlichen graphischen und algorithmischen Sprachelemente der DAEMON-Netze wird in Abb. 6.1 vermittelt. Im folgenden werden diese Konstrukte informell und formal beschrieben.

6.1.1 Spezifikation von Zeitverhalten mit Ressourcen

Das dynamischen Verhalten eines diskreten Systems über die **Zeit** ist wesentlich zur Bestimmung seines quantitativen Verhaltens. Speziell bei Rechensystemen mit Hardware- und Softwarekomponenten sind zwei unterschiedliche Interpratationen des Zeitbegriffs möglich:

- Softwarekomponenten (z.B. Tasks) **verbrauchen Zeit** einer Hardware-Ressource wie Arbeitsrechner (AR) oder Plattenspeicher,

die ihnen vom Betriebssystem-Kern zugeteilt wird. Da i.d.R. nur Ressourcen beschränkter Kapazität existieren, ist die **Ressourcenzuteilung** (allocation) bei nebenläufigen Anforderungen wechselseitig auszuschließen (mutual exclusion) und zu überwachen (d.h. zu synchronisieren), z.B. mit dem Ziel der optimalen Ressourcen-Auslastung oder der minimalen Wartezeit auf Zuteilung.

- Das Konzept der **Zeitverzögerung** beinhaltet Phänomene wie z.B. die zyklische oder zeitlich versetzte Task-Beauftragung (z.B. Restart von Transaktionen nach deren Abbruch durch das Concurrency Control Verfahren, Zeitschrankenüberwachung von Protokoll-Nachrichten durch Timeouts), Ausfall- und Reparaturzeiten von HW-Ressourcen.

Diese zwei Konzepte wollen wir nun näher charakterisieren.
Zeitverbrauch einer SW-Komponente wird charakterisiert durch eine zugeordnete **physische (HW-)Ressource** r mit den Attributen

- R_NO(r) : Anzahl der Ausprägungen einer Ressource. Diese Anzahl bestimmt die maximale Aktivierbarkeit nebenläufiger Ressource-Anforderungen. Eine Anzahl >1 beschreibt z.B. Anforderungen an Mehrprozessorsysteme, Terminal-Pools oder Disk-Pools.
- R_SCHED(r) : Scheduling-Strategie von nebenläufigen Anforderungen. Die Wahl der Strategie hat einen bedeutenden Einfluß auf das Performance-Verhalten der darauf realisierten Anwendungstasks, wie warteschlangentheoretische Analysen /KLCK76/ als auch Messungen in existierenden Betriebssystemen zeigen /PESI83/.
- R_STATE(r) : Interner Zustand der Ressource r. Das einfachste Modell unterscheidet zwischen einem passiven und R_NO(r) aktiven Zuständen. Da wir jedoch auch fehlertolerante Systeme modellieren wollen, ist auch ein Zustand "ausgefallen" zu berücksichtigen.

Zeitverzögerung von Instanzen wird charakterisiert durch eine zugeordnete **logische Ressource** ohne Attribute. Im Gegensatz zu physischen Ressourcen gilt:

(1) Nebenläufige Anforderungen und damit die Anzahl von Ausprägungen der Ressource werden nicht beschränkt
(2) Alle Anforderungen werden stets ohne Wartezeit erledigt (folgt bereits aus (1))

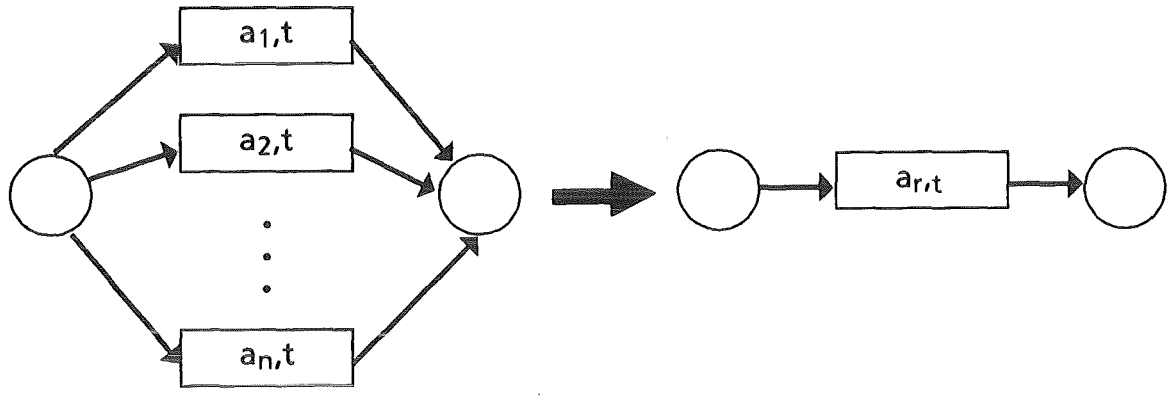
Wegen diesen Eigenschaften sind logische Ressourcen bei der Modellierung von Rechensystemen geeignet, um auch "zeitlose" (koinzidente) Aktivitäten zu beschreiben. Das sind Aktivitäten mit vernachlässigbarem Zeitverbrauch oder 'logische' Aktivitäten, bei denen ein Zeitverbrauch mehr einem Teilnetz als einer einzelnen Instanz zuzuordnen ist.

Ferner können sie im Rahmen der **hierarchischen Modellierung** sinnvoll verwendet werden:

> Von einem Subsystem S sei lediglich sein aggregiertes (stochastisches) Zeitverhalten bekannt, d.h. die Verteilungsfunktion V der Verweilzeiten von Aufträgen, die S bearbeitet. Die Verweilzeiten sollen die Wartezeiten von S einschließen. Nun kann die Modellierung von S mit der tatsächlich vorhandenen physischen Ressource, auf der S abläuft, zu unrealistischen Zeitverhalten führen, da zusätzliche Wartezeiten entstehen können, die aber bereits in V enthalten sein sollen! Das Problem besteht darin, daß in einem detaillierten Modell von S der Bedienzeit-Anteil von V gegeben ist, und V selbst meist als Zielgröße fungiert. Modelliert man S aber mittels einer logischen Ressource, ist wegen Eigenschaft (2) eine sinnvolle Modellierung möglich, obwohl das reale Subsystem auf einer physischen Ressource läuft.
>
> Typisches Beispiel: S sei ein Nachrichtentransportsystem (ISO-Schichten 1 bis 4), von dem lediglich die Verteilung V der 'End-to-End'-Verzögerungszeiten V pro Nachricht bekannt ist.

Wie können diese Zeitkonzepte mit stochastischen Petrinetzen (SPN) modelliert werden? Wir fordern

- Zeitverbrauch und -verzögerung sind **Instanzattribute**, wie in Funktionsnetzen (FN)
- Die zugehörigen Ressourcen sollen ebenfalls explizit als Instanzattribute gelten, was sich als vorteilhaft erweist (s.u.).
- **'Nebenläufige' Schaltregel**. Die übliche (SPN-)Schaltregel läßt kein nebenläufiges Schalten derselben Instanz zu. Benötigt wird aber ein allgemeineres "reentrant"-Schaltverhalten einer Instanz, wobei die Anzahl nebenläufiger Aktivierungen bei phys. Ressourcen durch deren Ausprägungsanzahl, bei log. Ressourcen jedoch unbeschränkt ist. (Bemerkung: Für phys. Ressourcen ist das Schaltverhalten auch durch



(a) SPN -Sicht
$a_{i,t}$: i-te Instanz mit Zeitverbrauch t

(b) DAEMON -Sicht
ar,t: Instanz mit Zeitverbrauch t,
Ressource r mit R . NO (r) = n

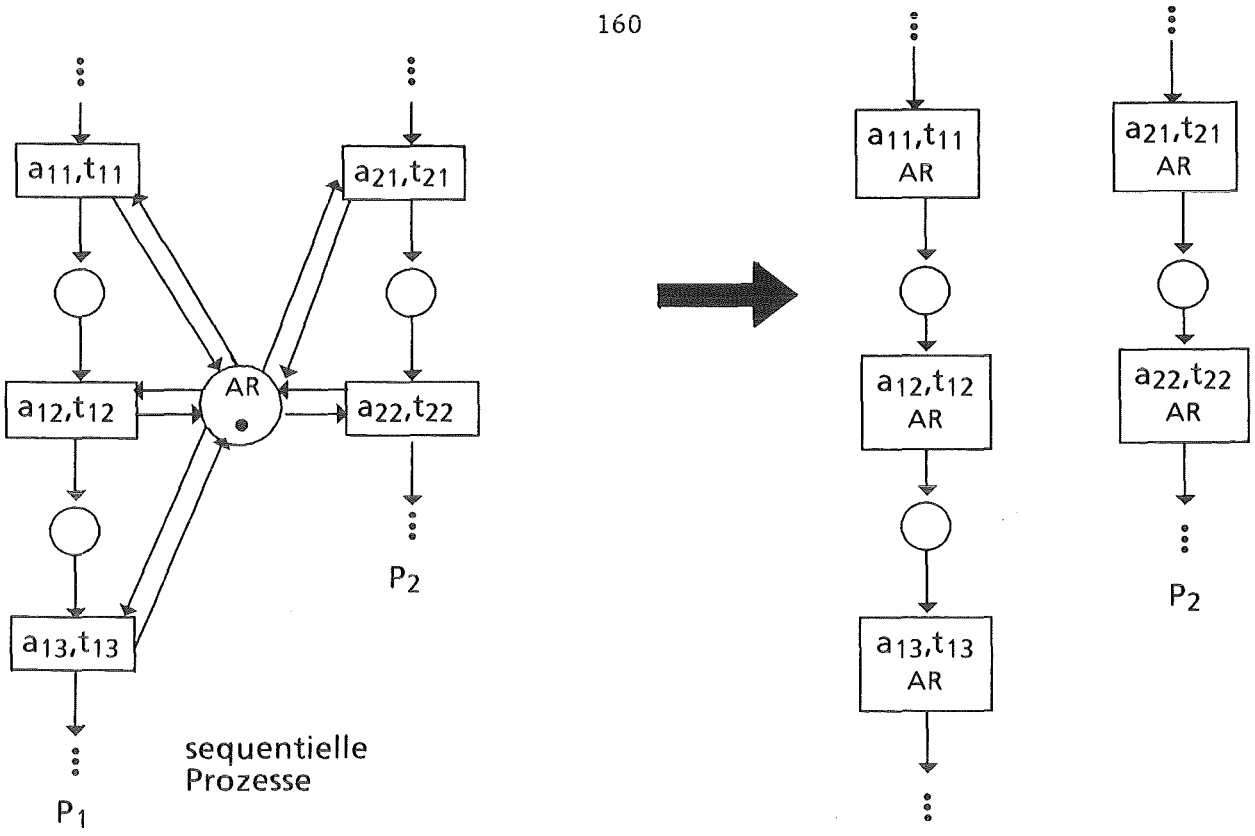
Abb. 6.2: Modellierung des n-nebenläufigen Schaltens von Instanzen in Stochastischen Petrinetzen (SPN) und in DAEMON

eine Verfeinerung der jeweiligen Instanz beschreibbar, wie Abb. 6.2 zeigt).

Wechselseitiger Ausschluß konkurrierender Tasks bei Zugriff auf phys. Ressourcen wird in SPN (auch in FN) 'implizit' modelliert, wie in Abb. 6.3(a) gezeigt. Es ist folgendes festzustellen:

- Der Begriff der Ressource kann nur als eine der vielen möglichen Interpretationen von Netzen gelten; diese sind 'semantikfrei'.
- Die Größe des Netzes hängt stark von der mittleren Anzahl Instanzen ab, die eine Ressource beanspruchen. Die Darstellung von Tasks als höhere Modellebene kann von der Darstellung der Tasksynchronisation als tiefere, den Kontrollfluß spezifizierende Modellebene nicht getrennt werden.
- Die Ausprägungsanzahl kann in SPN durch eine entsprechende Anfangsmarkierung im Ressourcenkanal modelliert werden; jedoch bleibt damit die Ressourcenzuteilungs-Strategie **undefiniert**: Der AR-Kanal in Abb. 6.3(a) führt zu einer Konfliktsituation, wenn z.B. mehr als eine Instanz AR-Zeit verbrauchen will, aber nur eine AR-Ausprägung existiert. Die Netzstruktur eines SPN (auch eines FN) macht aber keine Aussage darüber, **wie** Konflikte aufzulösen sind.

Man könnte nun die gewünschte Konfliktauflösungs-Methode (sprich:



(a) SPN - Sicht **(b) DAEMON - Sicht**

Abb. 6.3: Multitasking-Modell auf Ressource "Arbeitsrechner" (AR)
t_{ij} : Zeitverbrauch der Instanz a_{ij} auf Task i

Scheduling-Strategie) selbst als Subnetz modellieren, indem man den Ressourcen-Kanal verfeinert. Einige Strategien sind auf diese Weise in Beiträgen zu /SPN85/ modelliert worden. Eine andere Möglichkeit besteht darin, bei Kanälen mit mehr als einer Outputkante diesen Kanten Transfer-Wahrscheinlichkeiten als Attribut zuzuordnen /RAZ084/. Wir wollen hingegen die Netzstruktur nicht unnötig mit Submodellen niederer Systemschichten oder mit zusätzlichen Attributen belasten und definieren deshalb die Strategie selbst als **Attribut** jeder physischen Ressource. Die Wahl der Strategie hat dann Konsequenzen für die Schaltregel von Instanzen, denen eine physische Ressource zugeordnet wurde.

Das verhaltensgleiche DAEMON-Netz zu Abb. 6.3(a) zeigt Abb. 6.3(b). Zur einfacheren Netzstruktur kommt nun die Ressourcen-Spezifikation hinzu. Insgesamt hat diese attributbezogene Anreicherung der FN-Variante folgende Vorteile gegenüber früheren Ansätzen:

- Die graphische Netzkomplexität wird reduziert
- DAEMON-Netze bekommen eine Semantik, die an Rechensystemen orientiert ist.
- Die Strukturiertheit und Flexibilität von Modellen verteilter Systeme wird gesteigert. Z.B. kann durch einfache Änderung des Ressourcenattributs einer Instanz verschiedene Zuordnungen von Tasks zu Rechnern bewertet werden, ohne die graphische Netzstruktur zu ändern.
- Die rechnergestützte Modellauswertung hat vollständige Information über alle Ressourcen, sodaß eine automatische Sammlung und Ausgabe ressourcen-orientierter Statistiken erfolgen kann, wie etwa deren Auslastungen und Bedienraten.
- Die Modellierung von Ressourcen-Ausfall und -Wiederanlauf und deren Auswirkungen auf ein Software-Systemmodell ist auf einfache Weise möglich (s. nächsten Abschnitt).

Bemerkung:

Zuordnungen von Instanzen zu Ressourcen sind mit Sorgfalt vorzunehmen, da nicht alle Zuordnungen reale Rechensysteme beschreiben. Es ist auch denkbar, die möglichen Zuordnungen bewußt einzuschränken, indem z.B. für spezielle Subsysteme wie 'Rechnerknoten', 'Schicht', 'Dienst', 'Task' Zuordnungsregeln definiert werden, etwa daß alle Instanzen einer Task nur auf höchstens einer phys. Ressource ablaufen dürfen, oder daß die phys. Ressourcen von Instanzen auf unterschiedlichen Rechnerknoten stets disjunkt sein müssen. Auf den Einbau dieser Semantik offener Kommunikationssysteme in die DAEMON-Definition wurde im Rahmen der vorliegenden Arbeit verzichtet; ein Sprachansatz in dieser Richtung findet sich bei Bauerfeld /BAUE84/.

Andere Arbeiten zum Thema Ressourcen in Netzmodellen

Die Idee der expliziten Ressourcen-Spezifikation in SPN geht auf /NUTT72/ zurück. Im Rahmen der 'Wiederentdeckung' von SPN zur Leistungsanalyse zustandsdiskreter Systeme haben Ressourcen in mehrere SPN-Varianten Eingang gefunden /TPN85/.

6.1.2 Spezifikation des Ausfallverhaltens von Ressourcen

Die integrierte quantitative Bewertung von Leistung und Verfügbarkeit fehlertolerierender Systeme ist ein Hauptziel dieser Arbeit. Die Bewertungsmethode hierzu wurde in Kap. 5.3 erarbeitet; in Kap. 4 wurde ein **Ausfall-Submodell** (ASM) der wesentlichen physischen Ressourcen (Arbeits- und Kommunikationsrechner) eines verteilten Systems sowie das **Operations-Submodell** (OSM) der zu modellierenden VDBS-Protokolle und deren Ablaufumgebung zur Transaktionsverarbeitung aufgestellt.
Zur Unterstützung der integrierten Bewertung fordern wir :

> ASM und OSM sollen durch eine **einheitliche Modellierungsprache** in demselben Modell formulierbar sein.

Bei der Modellierung eines ASM beschränken wir uns auf den einfachsten Fall der voneinander unabhängigen fail-stop Ressourcen mit den in Kap. 4.2 definierten Eigenschaften. Würde man ein ASM allein zum Zweck der Verfügbarkeitsanalyse aufstellen, sind alle SPN-Varianten dazu geeignet. Schwierigkeiten treten jedoch bei der Modellierung der ASM-OSM Interaktionen auf, d.h. konkreter bei der Modellierung der Auswirkungen eines Ausfalls einer phys. Ressource auf alle Instanzen, denen diese Ressource als Attribut zugeordnet ist.
Es liegt hier ein ähnliches Problem wie vor der expliziten Einführung von Ressourcen in DAEMON-Netze. Prinzipiell ist es möglich, Effekte von HW-Ausfällen auf SW-Instanzen z.B. in Funktionsnetzen zu modellieren. Das führt aber zu sehr komplexen und unübersichlichen Darstellungen, wie das Beispiel eines integrierten ASM-OSM Modells mit der Netzsprache 'stochastic activity networks' /MEYE85/ zeigt. Als Abhilfe ist in DAEMON-Netzen (incl. deren Unterstützungs-Software) folgendes realisiert.
Ein Ausfallsubmodell ASM ist ein Teilnetz, welches das "Makroverhalten" aller phys. Ressourcen beschreibt, die ausfallen können. Sei ρ eine solche Ressource, deren fail-stop Makroverhalten durch die Ressourcenzustände R_STATE(ρ) ε {PASSIVE, DOWN} mit exponentieller Verteilung der Zustandsübergänge charakterisiert ist. In DAEMON können nun diese Übergänge von Instanzalgorithmen gesetzt werden, was zu der Beschreibung des ρ-Makroverhaltens in Abb. 6.4

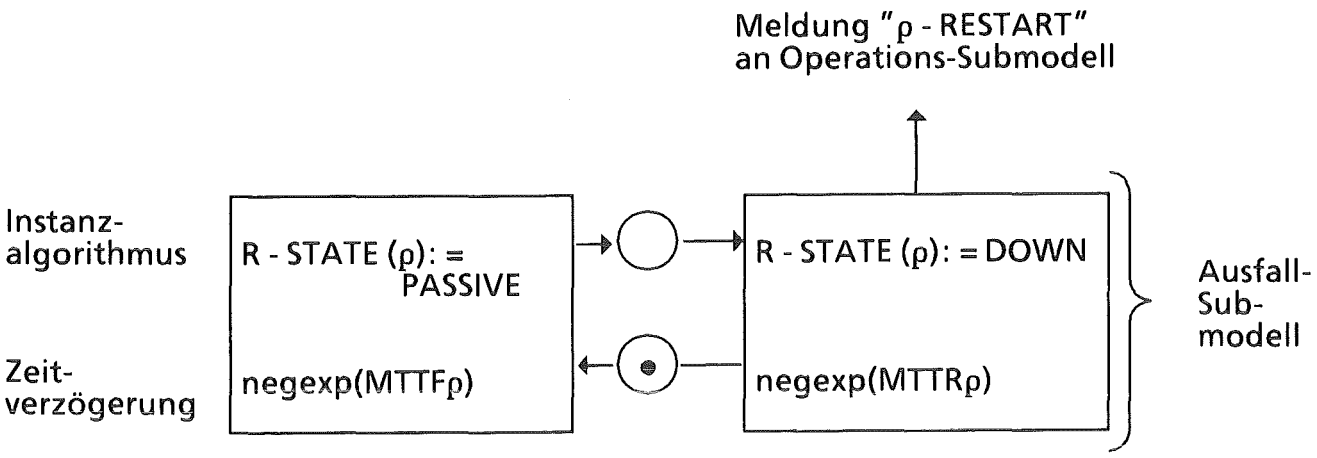


Abb. 6.4: Ausfall-/Wiederanlaufmodell einer phys. Ressource ρ (der Instanzalgorithmus wird zu Beginn der Instanzaktivierung ausgeführt. Danach erfolgt eine Zeitverzögerung gemäß der neg. Exponentialverteilung mit gegebenem Mittelwert der Ausfallzeit $MTTR_\rho$ bzw. der Intaktzeit $MTTF_\rho$)

führt. Dem Übergang in den DOWN-Zustand unterliegt folgende Semantik, die die Effekte des ASM auf ein OSM beschreibt:

- Die Menge aller Inputkanäle von ρ-Instanzen in OSM wird als der flüchtige **Pufferspeicher** der Ressource ρ interpretiert, dessen Inhalt nach ρ-Ausfall undefiniert ist. Ein Übergang nach DOWN bewirkt daher, daß der Inhalt aller dieser Kanäle zerstört wird, d.h. daß alle darauf befindlichen Marken verloren gehen. Dasselbe passiert mit Marken, die während der DOWN-Phase auf einen dieser Kanäle gelangen. (Eine Belegung dieser Kanäle kann nur von Instanzen einer intakten phys. Ressource erfolgt sein.)
- Das Schalten aller ρ-Instanzen wird während der ρ-DOWN-Phase verhindert. Bereits begonnene Schaltvorgänge solcher Instanzen zum Zeitpunkt des Übergangs in den DOWN-Zustand werden abgebrochen, d.h. das Schalten dieser Instanzen hat keinerlei Auswirkungen.

Der Übergang von DOWN zu PASSIVE kann als ρ-Wiederanlauf interpretiert werden, der die Aufhebung der Blockade aller ρ-Instanzen zur Folge hat. Die "Effekte" eines Wiederanlaufs sind stets anwendungsabhängig; in VDBS ist dafür ein spezielles Restart-/Recoveryprotokoll vorzusehen. Daher muß ein innerhalb des ASM modellierter ρ-Wiederanlauf ein solches Protokoll innerhalb des OSM 'anstoßen', wie

in Abb. 6.4 gezeigt. (Damit 'erkennt' z.B. das Betriebssystem eines ausgefallenen Rechners den Restart der eigenen Hardware.)
Das Konzept in DAEMON zur Beschreibung von Ressource-Crash und -Restart sieht also eine 'explizite' graphische Beschreibung der Crash- und Restart-Zeiten im ASM sowie der Restart-Folgen auf das OSM vor; die Folgen eines Ausfalls während der Ausfallzeit auf das OSM werden 'implizit' über die Erweiterung der Schaltregel beschrieben.
Das integrierte ASM-OSM Modellkonzept wird in der vorliegenden Arbeit für folgende Problemkreise eingesetzt:

(1) Zur integrierten Leistung-Verfügbarkeitsanalyse fehlertoleranter Systeme nach der Methode von Kap. 5.2. Dazu wird jeder zu untersuchende Makrozustand der Arbeits- und Kommunikationsrechner im ASM durch eine geeignete Anfangsmarkierung vorbesetzt. Dieser Zustand wird für die Dauer der OSM-Simulation bis zum Erreichen der Stationarität der gewünschten Outputgrößen beibehalten. Der große Vorteil der ASM-OSM Modellierung gegenüber früheren Ansätzen liegt nun darin, daß die OSM-Struktur für **jeden Makrozustand** unveränderlich bleiben kann. Auch wenn z.B. ein einzelner Rechner ausfällt, kann das zugehörige Operations-Submodell der auf diesem Rechner ablaufenden Software in seiner statischen Netzstruktur unverändert bleiben. Wir kommen darauf später bei der Beschreibung der VDBS-Modellexperimente im Kapitel 7 zurück.

(2) Zum **Protokolltest** von Fehlertoleranz-Verfahren, z.B. für Datenbank-Recovery oder für globale Zustandserkennung. Um die Auswirkungen von ASM- auf OSM-Ereignisse verfolgen zu können, sind zwei Maßnahmen sinnvoll:
   -- Die relative Häufigkeit von Ausfall/Restart-Ereignissen im ASM gegenüber den OSM-Ereignissen wird stark erhöht. Als Folge sollen unterschiedliche ASM-Ereignisse bei einer OSM-Simulation genügend oft auftreten.
   -- Die relative Häufigkeit von Restart- gegenüber Ausfall-Ereignissen wird stark erhöht. Als Folge sollen unterschiedliche Mehrfach-Ausfälle bei einer OSM-Simulation genügend oft auftreten.

Ein komplettes ASM-OSM Modell wird in Kapitel 7 beschrieben.

6.1.3 Markentyp-Spezifikation

Individuelle, strukturierte Marken sind sowohl in Funktionsnetzen als auch in Prädikats/Transitionsnetzen (PrT-Netze) /GELA81/ vorgesehen. Als mögliche Interpretation der Wertebereiche strukturierter Marken sehen wir für Funktionsnetze ein **explizites Markentypkonzept** vor, wie es in ähnlicher Form auch mit dem Ziel der rechnergestützten Codegenerierung aus PrT-Netzspezifikationen /BRMA86/ vorgeschlagen wurde.

Das Konzept sieht folgendes vor:

- Zu jedem Modell gehört neben der Netzdefinition eine **Markentyp-Deklaration.**
- Ein Markentyp ist eine Struktur, bestehend aus **Markenattributen.** Ein Markenattribut besteht aus Attributbezeichner und -typ. Als Attributtypen sind Standarddatentypen, wie in Hochsprachen üblich, ausreichend. (Spezialfall: Steuermarken sind Objekte eines attributlosen Typs.) Der Objektbereich aller Attributtypen muß endlich sein, da sonst viele klassische Netzeigenschaften nicht mehr entscheidbar sind (vergl. /PETE81/). Man kann einen Markentyp als Relation im Sinne des Relationenmodells auffassen; Marken wären dann Tupel der Relation (s. /CODD70/).

Die Markendeklaration hat neben der besseren Strukturierung eines Modells den Vorteil, daß Einhaltung und Überprüfung der **Schnittstellen-Konsistenz** zwischen Subsystemen (Teilnetzen) unterstützt werden kann. Z.B. kann dadurch die korrekte Verwendung eines verteilten Dienstes innerhalb des Modells erzwungen werden.

6.1.4 Kanaltyp- und Kantentyp-Spezifikation

Neben den bekannten Kanal-Zugriffsmodi LIFO und FIFO in Funktionsnetzen wird in DAEMON-Netzen ein weiterer Modus **SET** eingeführt: Damit wird die aus PrT-Netzen bekannte Grundform in DAEMON-Netzen übernommen. Aus einem SET-Kanal kann eine bestimmte Marke 'wahlfrei' mittels **Markenselektion** (s.u.) entfernt werden.
Während bei FIFO bzw. LIFO die Reihenfolge der aus einem Kanal

entnommenen (individuellen) Marken durch die zeitliche Reihenfolge des Markentransports in diesen Kanal eindeutig bestimmt ist, gestattet ein SET-Kanal die Entnahme in **beliebiger** (in der Selektionsprozedur festgelegter) Reihenfolge. Damit erreichen wir ein flexibleres, mächtigeres Modellierungskonzept. Typische Anwendungen sind:

(1) Erzwingung der Verarbeitung von Nachrichten in einem lose gekoppelten, verteilten System in Absende-Reihenfolge, z.B. für Reassemblierung von Paketen zu Nachrichten, Einsammeln von Nachrichten für n:1-Empfangsoperationen in der Interprozeßkommunikation (Kap. 4.3.3), in zeitstempel-orientierten Concurrency-Control Verfahren und in Commit-Protokollen.
Ursache für die Ankunft von Nachrichten bzw. Paketen in anderer als der Sendereihenfolge -also von **Überholvorgängen**- ist das inhärente, **stochastische Verhalten** lose gekoppelter Systeme. Als Ursachen dafür kommen in Frage:
- Zeit- und verbindungsabhängige Verzögerungszeiten,
- Fehler im Nachrichten-Transportsystem, die zu Verlust bzw. mehrfachem Senden einer Nachricht führen,
- Unterschiedliche Verarbeitungszeiten von Subsystemen auf verschiedenen Knoten, die an der Ausführung eines verteilten Algorithmus beteiligt sind (sowohl Software- als auch Hardware-Ursachen möglich)

(2) Bearbeitung von Aufträgen (Jobs, Transaktionen, Tasks) durch das Betriebs- bzw. Datenbanksystem entsprechend statischen oder dynamischen Auftrags-Prioritäten. In der Regel besteht zwischen der zeitlichen Reihenfolge der Auftragsankünfte und deren Prioritäten keinerlei Korrelation, sodaß ggf. spätere ankommende Aufträge frühere wegen höherer Priorität überholen können. Auch diese Bearbeitungsstrategie ist nicht mit einer FIFO- oder LIFO-Disziplin formulierbar.

6.1.5 Instanz-Spezifikation

Die Mächtigkeit der Funktionsnetze zeigt sich insbesondere in der Möglichkeit, recht komplexe Aktivitäten eines dynamischen, diskreten

Systems in der Spezifikation einer Instanz und deren Schaltverhalten darzustellen. **Instanzen** erlauben u.a.

(1) beliebige Markentransformationen von strukturierten Input- auf Outputmarken einer Instanz auszuführen (JOB-Attribut in /GODB83/),

(2) der damit zusammenhängenden Aktivität einen beliebigen deterministischen oder stochastischen Zeitverbrauch zuzuordnen (TIMECONSUME-Attribut in /GODB83/),

(3) die Outputmarken auf eine beliebige Teilmenge der Outputkanäle der Instanz selektiv abzubilden (SOME_OUT-Attribut in /GODB83/).

Abbildungen zwischen strukturierten Marken (1) zeichnet z.B. auch die PrT-Netze aus; Zeitverbrauch (2) wurde mittlerweile in viele Netzvarianten eingebaut /TPN85/. Die Fähigkeit der **selektiven Ausgabe** (3) einer Funktionsnetz-Instanz liefert einen Abstraktionsmechanismus, der in 'klassischen' Netzen nur mit wesentlich komplexeren Teilnetzen modelliert werden kann. Ein gutes Beispiel liefert die bereits in Kap. 4.3.3 innerhalb des IPC-Protokollmodells (Abb. 4.12) beschriebene Instanz "1:N-MULTICAST". Eine exakte Beschreibung dieser Instanz zeigt Abb. 6.5. Die Multicast-Operation adressiert eine dynamisch durch den Nachrichteninhalt bestimmte Teilmenge aller N Zielknoten, wobei **jede Teilmenge** der Potenzmenge mit der Mächtigkeit 2^N als Zielmenge der zu sendenden Nachricht in Frage kommt, mit den Spezialfällen (n sei die Zielmengen-Mächtigkeit):

- n = N : Broadcast-Operation
- n = 1 : 1:1-Sendeoperation
- n = 0 : zu sendende Nachricht geht verloren

Die Multicast-Operation repräsentiert also den allgemeinsten Fall einer selektiven Ausgabeinstanz. Ein verhaltensgleiches Netz ohne die Möglichkeit der Outputselektion kann konstruiert werden, indem für jede Teilmenge der N Knoten eine eigene Instanz zuständig ist. Die Auswahl einer Instanz erfolgt per deterministischer Konfliktauflösung, indem für jede Instanz eine Vorbedingung konstruiert wird, die die jeweilige Teilmenge der Potenzmenge adressiert. Die Outputkanal-Menge einer solchen Instanz ist dann genau diese Teilmenge. Wir kommen daher zu folgender Vermutung (ohne Beweis):

Zu einer Instanz mit selektiver Markenausgabe auf X Kanäle, bei

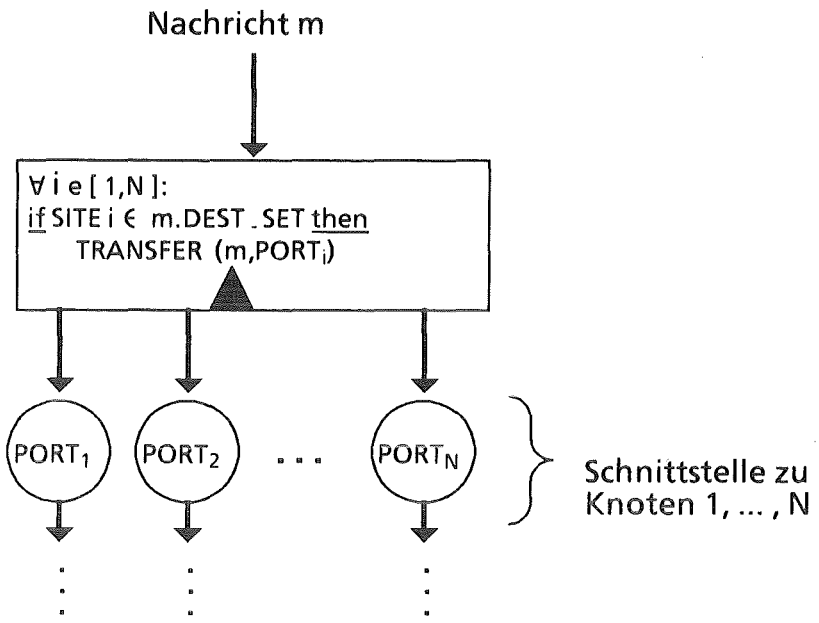


Abb. 6.5: Modell einer 1:N-MULTICAST-Operation

```
m     : Markenvariable über dem Markentyp MSG

MSG   : Markentyp  "Nachricht" mit
MSG = <ID          : Auftrags-Kennung,
       DEST_SET    : Menge der Zielknoten>

TRANSFER(m,c) : Gibt Marke m auf Kanal c aus
```

der jeder Kanal zur Ausgabe ausgewählt werden kann, hat jedes verhaltensgleiche Teilnetz, in dem alle Instanzen nur totale Markenausgabe zulassen, **mindestens** $2^X + 1$ Instanzen

(Verhaltensgleichheit von Netzen wird in/GODB83, S. 36/ definiert.)
Den Vorteilen bei der Kompaktheit eines Funktionsnetzes stehen allerdings Probleme mit dessen Analysierbarkeit gegenüber. Das wird in Kap. 6.4 diskutiert.

DAEMON-Netze bieten eine Interpretion von Funktionsnetzen durch die zusätzliche Möglichkeit der **Inputmarken-Selektion**. Wie bereits oben begründet, erfolgt die Selektion einer Marke durch einen booleschen Ausdruck über Marken eines SET-Inputkanals einer Instanz. Die Selektion erfolgt entsprechend bestimmter, im Ausdruck vorgegebener **Schlüsselattribute** dieser Marken. (Das impliziert, daß Marken auf SET-Kanälen stets einem Markentyp mit mindestens einem Attribut angehören.) Eindeutigkeit von Schlüsselattributen wird nicht

gefordert, sodaß ggf. eine indeterministische Selektion erfolgt. Unser Selektionskonzept ist bezüglich der Beschreibungsmächtigkeit äquivalent mit der Relationenalgebra /CODD70/.

Als Folge ist die **FN-Schaltregel** zu **verallgemeinern**: Findet die Selektion keine Marke, die den booleschen Ausdruck erfüllt, darf die zugehörige Instanz nicht schalten, obwohl evtl. alle Eingangsknoten mit Marken belegt sind! Benötigt wird also eine **bedingte Schaltregel**, wie sie von PrT-Netzen bekannt ist /GELA81/. Die Selektionsfunktion bildet die **Schaltvorbedingung**, während der Instanzalgorithmus, zusammen mit der partiellen Ausgabefunktion, als **Schaltnachbedingung** bezeichnet wird. Die Vorbedingung ist also eine boolesche Funktion über den Eingangskanälen einer Instanz. Ist die Vorbedingung leer, wird die FN-Schaltregel angewendet.

In Abb. 6.6 ist ein typisches Beispiel zur Anwendung von SET, Markenselektion und partiellem Ausgabeverhalten demonstriert. Die gezeigte Instanz realisiert eine selektive n:1-Empfangsoperation, die weniger exakt bereits in Abb. 4.12 bei der Modellierung des IPC-Protokolls für VDBS beschrieben wurde:

> Auf den SET-Kanal AWAIT gelangen Marken, die Aufträge repräsentieren. Jeder Auftrag wartet auf den Erhalt aller Ergebnisse von einer Menge (vorher abgesetzter) Unteraufträge. Die Anzahl erwarteter Ergebnisse ist ein Auftragsattribut. Ein Auftrag und alle seine Unteraufträge werden über den gleichen Schlüssel "ID" identifiziert. Ergebnisse kommen im FIFO-Kanal MAILBOX an. Überholvorgänge sind möglich: Obwohl z.B. zwei Aufträge A1, A2 in dieser Reihenfolge nach AWAIT gelangen, kann ein Unterauftrags-Ergebnis U(2,i) für A2 **vor** U(1,j) für A1 in MAILBOX ankommen, sodaß die Anwendung des Selektionskonzepts notwendig ist, um diesen Vorgang exakt zu modellieren. Die gezeigte Instanz hat die Aufgabe, entsprechend dem Unterauftrags-ID den darauf wartenden Auftrag aus AWAIT zu holen, das Ergebnis in die Auftragsmarke zu speichern, und den Auftrag weiter in AWAIT warten zu lassen, bis alle Unteraufträge eingetroffen sind (partielle Ausgabe auf Outputkanäle).
>
> In der Abbildung kommen Markenvariablen vor, um den Zugriff auf das Schlüsselattribut ID einer Auftrags- bzw. Ergebnis-Marke zu unterscheiden.

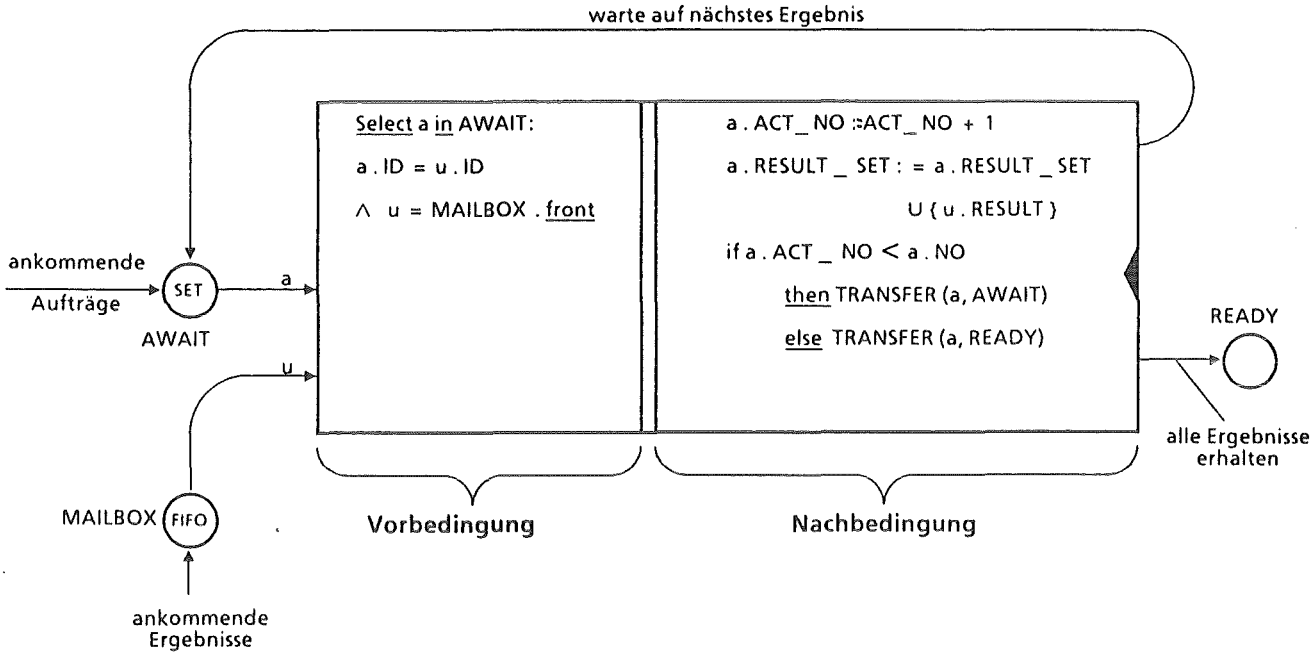
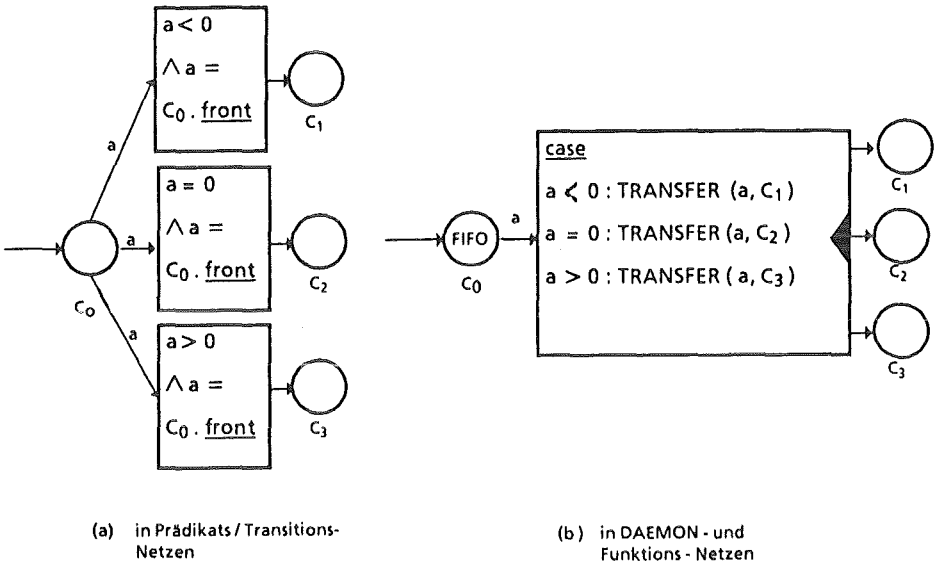


Abb. 6.6: Modell einer selektiven n:1-Empfangsoperation

```
A    : Markentyp "Auftrag" mit
A = <ID          : Auftrags-Kennung,
     NO          : Anzahl erwarteter Unterauftraege
     ACT_NO      : Anzahl bereits erhaltener Unterauftraege
     RESULT_SET  : bereits gespeicherte Unterauftrags-Info>

U    : Markentyp  "Unterauftrag" mit
U = <ID          : Auftrags-Kennung,
     RESULT      : Ergebnis des Unterauftrags>

a, u : Markenvariablen über den Markentypen A, U

front: Funktion, die Marke aus FIFO-Kanal beschafft

TRANSFER(m,c) : Gibt Marke m auf Kanal c aus
```

Die Benutzung der Markenselektion wird auf Instanzen mit (mindestens) einem SET-Inputkanal beschränkt. Eine Verwendung zur deterministischen Konfliktauflösung wie in PrT-Netzen erscheint in DAEMON-Netzen unnötig, da stets ein verhaltensgleiches Modell existiert, in dem die Selektion von Inputmarken bis zur selektiven Ausgabe hin 'verschoben' wurde. (Diese Möglichkeit ist in PrT-Netzen nicht vorgesehen.) Beide Alternativen sind in Abb. 6.7 angedeutet.

Abschließend diskutieren wir die möglichen Fehlerfälle bei der Modellierung von Instanzbedingungen. Zunächst ist klar, daß eine



(a) in Prädikats / Transitions-Netzen

(b) in DAEMON - und Funktions - Netzen

Abb. 6.7: Beispiel einer Inputmarken-Selektion
a : Markenvariable mit a ε R

Instanzbedingung so gewählt werden kann, daß ihr Wert von keiner möglichen Markierung der Inputkanäle einer Instanz erfüllt werden kann, d.h. die Instanz würde nie schalten. Offensichtlich läge dann ein Modellierungsfehler vor. Wegen dieser Gefahr, die bei Funktionsnetzen nicht auftreten kann, ist die Wahl der Bedingungen mit Sorgfalt vorzunehmen.

Im Beispiel der selektiven Empfangsinstanz in Abb. 6.6 können zwei weitere Fehlertypen auftreten:

(1) Es kommt ein Ergebnis im MAILBOX-Kanal an, auf das zur Ankunftszeit kein Auftrag wartet. Da dieser Kanal vom Typ FIFO ist, würden nachfolgende Ergebnisse nie mehr bearbeitet. Es liegt also (zumindest lokal) eine Verletzung der **Lebendigkeit** des IPC-Protokolls vor.

(2) Ein Auftrag wartet auf Ergebnisse, die nie ankommen. Unter der Annahme eines unbeschränkten Auftragsflusses kann die Kapazität des AWAIT-Kanals nicht beschränkt werden. Es liegt also eine Verletzung der **Sicherheit** /GODB83/ des IPC-Protokolls vor.

Beide Fälle basieren auf einem unvollständigen oder inkorrekten Modell. Typische Ursachen für (1) sind logische Protokollfehler,

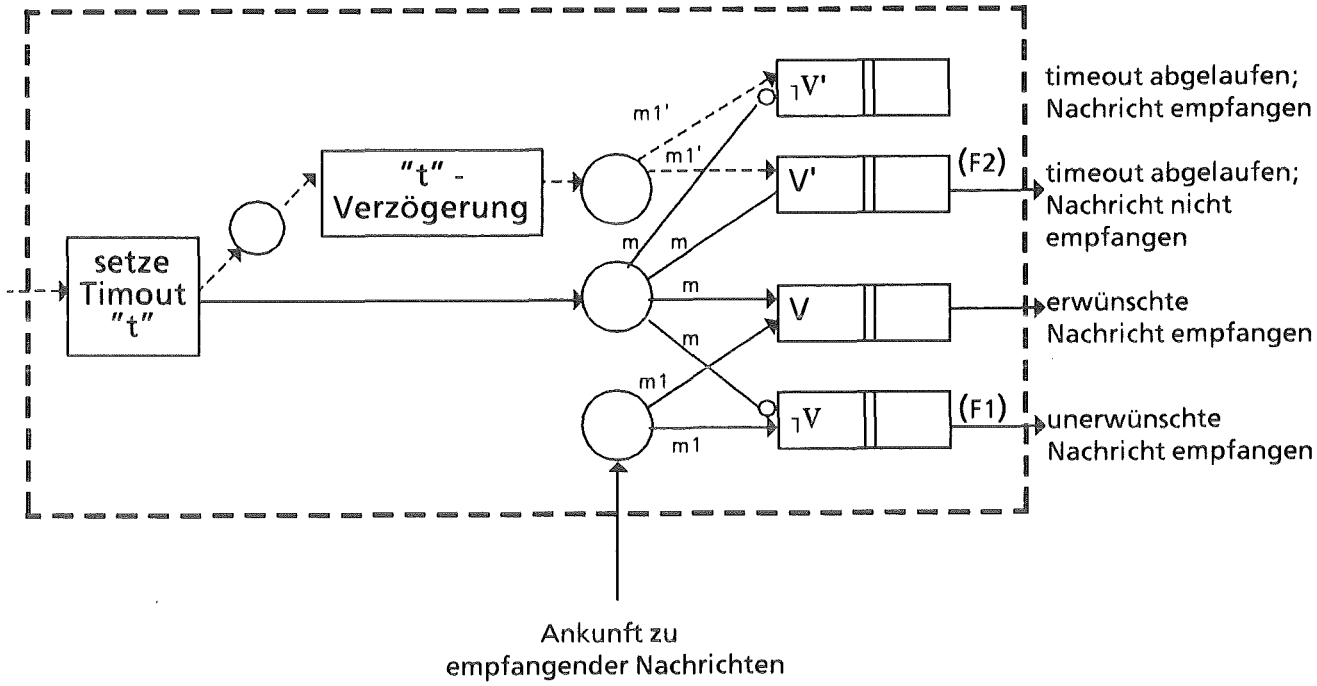


Abb. 6.8: Modellierung des selektiven Nachrichtenempfangs mit TIMEOUT

⟶	Fluß der Warte- und Empfangsnachrichten vom Markentyp <ID, ... >
- - -▸	Fluß der TIMEOUT-Kontrollnachrichten
⟶o	Inhibitor-Kante vom Markentyp <ID, t: TIMEOUT-Verzögerung>
V	Vorbedingung ∃ m : m.ID = m1.ID
V'	Vorbedingung ∃ m : m.ID = m1'.ID

Nachrichtenverdoppelung oder Nachrichtenverfälschung bei der Adressierung des Zielprozesses. Als Ursache für (2) kommt z.B. in Frage, daß ein Unterauftrag nie abgesetzt wurde, auf dessen Ergebnis gewartet wird, oder daß Nachrichten verloren gegangen sind (unvollständiges Protokoll). Ist ein Nachrichtenverlust im Modell vorgesehen, dann müssen auf der Empfängerseite auch Vorkehrungen zu deren Entdeckung existieren, da sonst eben Fall (2) auftritt. Üblich ist hierbei die Überwachung durch **Timeouts**.

Abb. 6.8 zeigt das DAEMON-Modell einer 1:1-Receive-Operation, die durch eine Erweiterung um ein Timeout robust gegen Fehler des Typs (1) und (2) gemacht wurde: Kommt eine nicht erwartete Nachricht an, wird das Teilnetz durch Fehlerausgang (F1) verlassen, d.h. Fehlertyp (1) wird erkannt. Kommt die erwartete Nachricht nicht vor Ablauf des

Timeouts an, wird das Teilnetz durch (F2) verlassen, d.h. Fehlertyp (2) wird erkannt.

In der Abbildung werden **logische Inhibitor-Kanten** zwischen einem SET-Kanal c und einer Instanz α eingeführt: α kann nur dann schalten, wenn die von der α-Selektionsbedingung auf c verlangte Marke dort **nicht vorhanden** ist. Das ist ein verallgemeinertes Konzept der Inhibitor-Arcs der Netzvarianten in /PETE81, ZUBE85/, da sich in diesen Varianten überhaupt keine Marke auf c befinden darf, um ein α-Schalten zu ermöglichen.

6.1.6 Formale Definition von Struktur und Schaltregel der DAEMON-Netze

Def. 6.1: DAEMON-Netz

Ein DAEMON-Netz ist ein 7-Tupel

$DN = (A, C; F, R, D, ATT, M_0)$ wobei

- (A, C; F) ist ein Petrinetz (s. Def. 6.2)
  mit einer Attributstruktur (s. Def. 6.4,7,8)
- R ist die Menge der Ressourcen (s. Def. 6.3)
- D ist die Menge der Markentypen über den Attributtypen ATT (s. Def. 6.5)
- M_0 ist die Menge der Anfangsmarkierungen (s. Def. 6.6)

Def. 6.2: Petrinetz

N = (A, C; F) ist ein Petrinetz (nach /BEFE86/):

$A = \{a_1, \ldots, a_{na}\}$ Menge der **Instanzen** (agencies)

$C = \{c_1, \ldots, c_{nc}\}$ Menge der **Kanäle** (channels)

$F \subseteq (C \times A) \cup (A \times C)$ Menge der **Kanten** (flow relation)

sodaß

- $A \cap C = \phi, \quad A \cup C \neq \phi$
- $domain(F) \cup range(F) = C \cup A$

Es sei $\forall a \in A$:

- $*a := \{ci \mid (a,ci) \in F\}$ **a-Inputkanäle**
- $a* := \{cj \mid (cj,a) \in F\}$ **a-Outputkanäle**

Es sei $\forall c \in C$:

- $*c := \{ai \mid (ai,c) \in F\}$ **c-Inputinstanzen**
- $c* := \{aj \mid (c,aj) \in F\}$ **c-Outputinstanzen**

Def. 6.3: Ressourcen

$R = \{r_1, \ldots, r_{nr}\}$ Menge der Ressourcen

wobei gilt :

$$R \cap A \cap C \cap D \cap ATT = \phi$$

Eine Ressource $r_i \varepsilon R$ ist ein 4-Tupel:

r_i = (R_TYPE, R_NO, R_SCHED, R_STATE) sodaß:

- R_TYPE: R --> {LOGICAL, PHYSICAL}
  log. SW-Ressource (Zeit-Verzögerung) bzw.
  phys. HW-Ressource (Zeit-Verbrauch)
- R_NO: R --> N ∪ {ω} Ressourcen-'Multiplizität'
- R_SCHED: R --> {First-Come First-Served, Shortest-Job-First, Highest-Priority ... } Scheduling-Disziplin
- R_STATE: R --> {PASSIVE, 1-ACTIVE, 2-ACTIVE, ... ,(R_NO)-ACTIVE, DOWN} Ressourcen-Zustand

$\forall r \varepsilon R$ gilt:

- R_TYPE(r) = LOGICAL => R_NO(r) = ω (unbeschränkte Anzahl), R-SCHED(r), R-STATE(r) sind undefiniert
- R_TYPE(r) = PHYSICAL => R_NO(r) ε N (beschränkte Anzahl)

Def. 6.4: Kanäle

Ein **Kanal** c ε C ist ein 2-Tupel
c = (CAP, ACC) sodaß

- CAP: C --> N ∪ {0, ω} Kanal-Kapazität (capacity)
- ACC: C --> {FIFO, SET} Markenzugriffsmodus (token access mode)

Bemerkungen:

- Der Zugriffsmodus LIFO aus Funktionsnetzen wird nicht verwendet
- Der Zugriffsmodus ADDRESS aus Funktionsnetzen ist konzeptionell mit SET identisch

Def. 6.5: Markentypen

$D = \{d_1, \ldots, d_{nd}\}$ Menge der Markentypen
über
$ATT = \{att_1, \ldots, att_{ne}\}$ Menge der Attributtypen
(endliche skalare Datentypen)

wobei gilt:
$R \cap A \cap C \cap D \cap ATT = \phi$

Ein **Markentyp** d_i, i ε [1,nd] ist ein i(ni) - Tupel :

$\exists \{i(1), \ldots, i(ni)\} \subset \{1, \ldots, ne\}$:
$d_i = \langle att_{i(1)}, \ldots, att_{i(ni)} \rangle$

Def. 6.6: Markierungen

Menge der Anfangsmarkierungen M_0:

M_0 = { m_0: C --> Bag(D) **Anfangsmarkierung** eines Netzes DN,
die die Kanalkapazität respektiert:
$\forall c \in C : |m_0(c)| \leq CAP(c)$ }

Dabei ist Bag(D) die Multimenge über D, d.h. jede Markierung kann von einem Markentyp d ε D keine, eine oder mehrere Marken enthalten (vergl. Bag-Definition in Anhang C)

Menge der Markierungen M:
M = {m | m: C --> Bag(D) **Markierung** eines Netzes DN }

M entsteht aus M_0 durch "Schalten" von Instanzen
(s. Def. 6.9)

Es sei $\forall a \in A$:
m(*a) := {m(c) | c ε *a} Markierung der a-Inputkanäle
m(a*) := {m(c) | c ε a*} Markierung der a-Outputkanäle

Schreibweisen:

Sei x eine **Marke** (token) vom Typ d_i, i ε {1, ... ,nd}

- Dann bezeichnet
  $x = \langle v_{i(1)}, \ldots, v_{i(ni)} \rangle$ den **Markenwert**,
  $x.att_j = v_j$ den **Attributwert** von att_j
- Bei i(nk) = 0 bezeichnet x = <>
  die einzig zulässige Marke des Typs d_k
- x ε c : "die Marke x befindet sich auf Kanal c ε C"

Bemerkung:
Da Attribute einen endlichen Wertebereich haben, ist für gegebene Mengen D und ATT die Menge aller möglichen Marken ebenfalls endlich

Def. 6.7: Kanten

F = {in, co} ∪ {ou} ∪ {hb} ist die Menge der Kanten, sodaß

- in: Menge der Input-Kanten (zerstörendes Lesen)
- co: Menge der Copy-Kanten (nicht-zerstörendes Lesen)
- ou: Menge der Output-Kanten
- hb: Menge der Inhibitor-Kanten (s. Kap. 6.1.6)

Bemerkung:
Folgende Kantensorten aus Funktionsnetzen werden nicht verwendet:
- st / fi : Kontrollfluß-Kanten. Diese werden in DN durch in / ou und den Markentyp <> 'simuliert'
- ca : Lesen Markenanzahl aus Kanal
- tp : Lesen Zeitparameter

Def. 6.8: Instanzen

Eine **Instanz** a ε A ist ein 6-Tupel
a = (PRECOND, POSTCOND, T_FUNCTION, MEMO, T_PAR, R_PAR) sodaß:

- PRECOND: $A \times M \rightarrow \{true, false\}$

  Schalt-Vorbedingung und Marken-Selektionsfunktion,
  wobei $\forall a \in A$ gilt:
  - $m1(*a) = m2(*a) \Rightarrow PRECOND(a,m1) = PRECOND(a,m2)$
    (PRECOND hängt nur von der Markierung der a-Inputkanäle ab)
  - $\forall c \in *a : ACC(c) \neq SET \Rightarrow (\forall m \in M : PRECOND(a,m) = true)$
    (PRECOND ist nur dann markierungsabhängig,
    wenn *a einen SET-Kanal enthält)

- POSTCOND: $A \times M \times MEMO \rightarrow (D \cup \phi) \times 2^C$
  (2^C stellt die Potenzmenge über der Kanalmenge C dar)

  Marken-Transformation (activity) und Outputkanal-Selektion,
  wobei $\forall a \in A$, $\mu = MEMO(a)$ gilt:

  - $m1(*a) = m2(*a) \Rightarrow POSTCOND(a,m1,\mu) = POSTCOND(a,m2,\mu)$
    (POSTCOND hängt nur von der Markierung der a-Inputkanäle
    und vom MEMO-Wert ab)

  - Sei $\forall c1, c2 \in *a$:
    $\exists d',d'' \in D \cup \phi$, $\exists a',a'' \in 2^C$:
    $(d', a') = POSTCOND(a,m(c1), \mu)$,
    $(d'', a'') = POSTCOND(a,m(c2), \mu)$
    Dann gilt:

    1) POSTCOND ist definiert nur für $a',a'' \subset a^*$ $(a^* \in 2^C)$

    2) $d' \neq d'' \neq \phi \Rightarrow a' \cap a'' = \phi$
       (POSTCOND legt ≤1 Marke auf einen a-Outputkanal ab)

    3) $d' = \phi \Rightarrow a' = \phi$
       (Nur 'echte' Marken werden auf a-Outputkanäle abgelegt)

- MEMO: A --> R 'Gedächtnis'-Zuordnung

- T_PAR: A --> R^+ Zeitparameter-Zuordnung

- R_PAR: A --> R surjektive Ressourcen-Zuordnung

- T_FUNCTION: $A \times R^+ \times M \rightarrow R^+$ Zeitverbrauchs-Funktion
  Es gilt $\forall a \in A$:

  - $T\_FUNCTION(a,t,m) = undef$ für $t \neq T\_PAR(a)$

  - $m1(*a) = m2(*a) \Rightarrow T\_FUNCTION(a,t0,m1) = T\_FUNCTION(a,t0,m2)$
    (T_FUNCTION hängt nur von der Markierung der a-Inputkanäle
    und vom Wert t0 := T_PAR(a) ab)

  - $t0 := T\_PAR(a) = 0 \Rightarrow \forall m \in M: T\_FUNCTION(a,t0,m) = 0 \wedge R\_PAR(a) = 'COINCID'$
    (Bei Zeitparameter = 0 verbraucht die Instanz grundsätzlich
    keine Zeit. Dieses 'sofortige' Schalten wird durch die
    spezielle Ressource 'COINCID' charakterisiert.)

- Eine Instanz a ε A hat eine **leere Vorbedingung**
  <=> ∀m ε M : PRECOND(a,m) = true

- Eine Instanz a ε A hat ein **partielles Ausgabeverhalten**
  <=> ∀m ε M, d ε Dυ{0} :
  range(POSTCOND(a,m,MEMO(a))) = (d,a') ∧ a'≠a*

- Ein **Instanztyp** (job) ist das 3-Tupel

  (PRECOND, POSTCOND, T_FUNCTION)

  Instanzen des gleichen Typs unterscheiden sich also in der Zuordnung der (unveränderlichen) Parameter MEMO, T_PAR und R_PAR
  Schreibweise :
  α = J(memo, tpar, rpar)
  Die Instanz α ist vom Typ J mit Parameterwerten memo, tpar, rpar

Bemerkung:
Die Instanzattribute JOB und ALL bzw. SOME_OUT in Funktions-Netzen bilden zusammen das POSTCOND-Attribut. Insbesondere entspricht SOME_OUT einem partiellen Ausgabeverhalten einer Instanz.
Das Instanzattribut SOME_IN in Funktionsnetzen wird nicht benötigt.

Def. 6.9: Instanz-Schaltverhalten

Sei α ε A eine Instanz mit Ressource-Attribut r ε R,
m die aktuelle Markierung des zugehörigen DAEMON-Netzes, dann

(1) α ist **schaltbereit** (enabled) <=>
- ∀c ε *α : |m(c)| > 0 ∨
  (ACC(c) = SET ∧ (c,a) ε hb)
  (auf jedem α-Inputkanal befindet sich mind. eine Marke, es sei denn, es ist ein SET-Kanal, der mit α durch eine Inhibitor-Kante verbunden ist)
- PRECOND(a,m) = true
- R_TYPE(r) = LOGICAL ∨
  (R_TYPE(r) = PHYSICAL ∧ R_NO(r) = n ∧
  R_STATE(r) ¬ε {DOWN, n-ACTIVE})

(2) α ist **aktiviert** (activated) <=>
- α ist schaltbereit
- α gewinnt Konfliktauflösung, falls ein α-Konflikt existiert:
  α-Konflikt <=> ∃β ε A : *β ∩ *α ≠ ϕ
  Die Konfliktauflösung erfolgt entsprechend der in R_SCHED(r) definierten Scheduling-Strategie

(3) α **schaltet** (fires) <=>

(3.1) zum Zeitpunkt T:
- falls α aktiviert ist, dann
  ∀c ε *α :
  Es wird genau eine Marke entsprechend dem c-Zugriffsmodus und dem Ergebnis der PRECOND(α, m(c))-Auswertung beschafft.
  Bei F(c,α) ε 'in' wird die Marke entfernt,

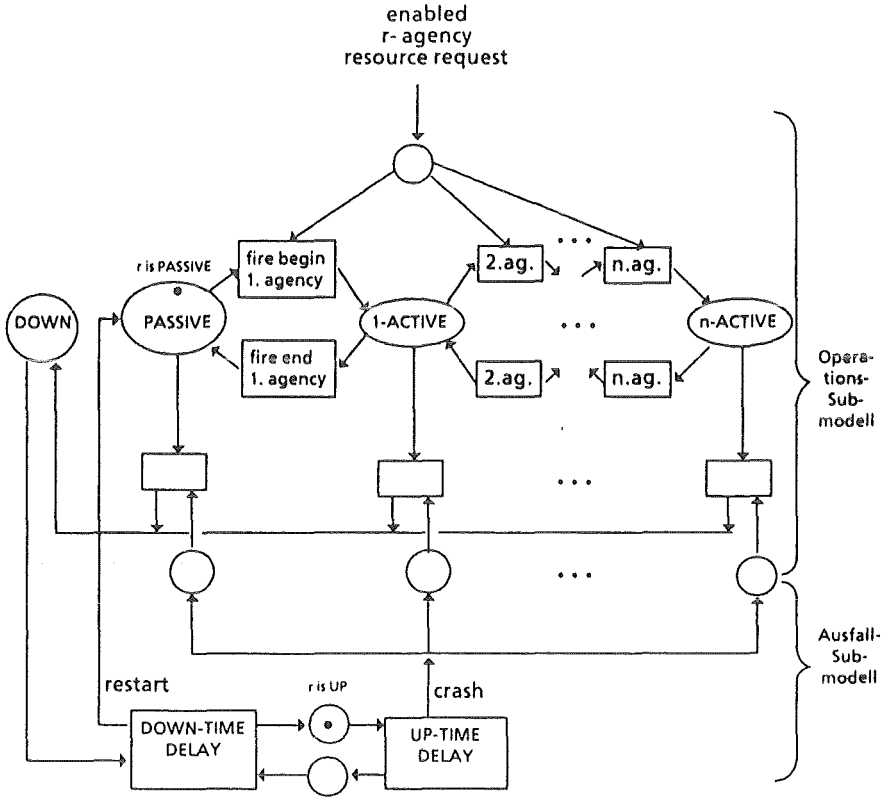


Abb. 6.9: Zustandsdiagramm einer phys. Ressource $\rho \in R$ mit $R\_NO(\rho) = n$

bei $F(c,\alpha) \in$ 'co' wird eine Kopie beschafft

Sei m_α die so erhaltene Markenmenge.

- $\delta(\alpha)$:= T_FUNCTION(α, T_PAR(α), m_α) wird berechnet

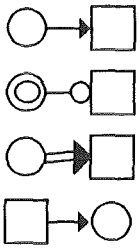
(3.2) zum Zeitpunkt T + $\delta(\alpha)$:
Die Funktion POSTCOND(α, m_α, MEMO(α)) wird ausgeführt;
die resultierenden Ausgabemarken werden auf die selektierten Outputkanäle transportiert.

Die Folge (3.1), (3.2) wird als **Schaltvorgang** bezeichnet.
(Def. Ende)

Die Definition des Schaltverhaltens ist noch um das Verhalten von Instanzen bei Ausfall physischer Ressourcen zu ergänzen, wie in Kap. 6.1.2 beschrieben.

Zur Erläuterung des komplexen Schaltverhaltens mit phys. Ressourcen ist in Abb. 6.9 das Zustandsdiagramm einer solchen Ressource gezeigt. Es zeigt auch einen Teil des Ausfallverhaltens: Bei Ressource-Ausfall

Kanten



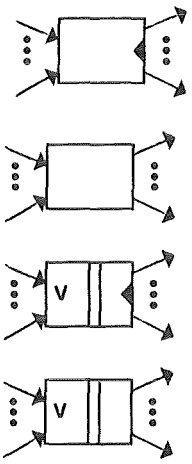
in: zerstörendes Lesen von Marke aus Kanal

hb: Inhibitor-Kante aus SET-Kanal

co: nicht zerstörendes Lesen von Marke aus Kanal

ou: Schreiben von Marke nach Outputkanal

Instanzen



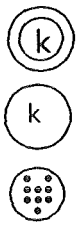
Partielle Ausgabe, leere Vorbedingung

Totale Ausgabe, leere Vorbedingung

Partielle Ausgabe, nicht leere Vorbedingung V

Totale Ausgabe, nicht leere Vorbedingung .V

Kanäle

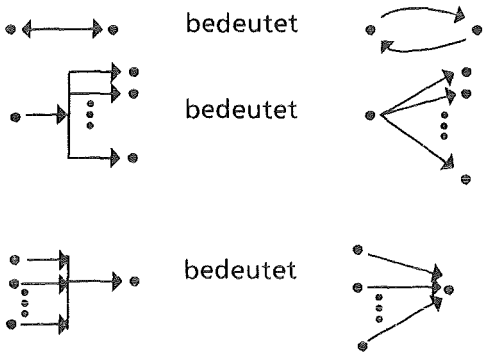


Kanal mit Zugriffsmodus SET und Kapazität = k

Kanal mit Zugriffsmodus FIFO und Kapazität = k

Kanal mit Anfangsmarkierung

Notationen



bedeutet

bedeutet

bedeutet

"Nebenbedingung": Kanal ist Input- und Outputkanal einer Instanz

stellenberandetes Teilnetz oder Schnitt

Netz wird vergröbert bzw. verfeinert zu

Abb. 6.10: Graphische Basiselemente von DAEMON-Netzen

erfolgt ein Übergang in den DOWN-Zustand, in dem keine Instanz mehr Zeit dieser Ressource beanspruchen darf (vergl. Abschnitt 6.1.2).
Abb. 6.10 definiert die graphischen Grundelemente von DAEMON-Netzen.

6.2 Zur Konstruktion von DAEMON-Modellen

Die im vorigen Kapitel definierte DAEMON-Netzsprache kann verwendet werden, um in einem integrierten Ansatz sowohl Hardware / Software als auch Operations- / Ausfallverhalten komplexer Rechensysteme zu spezifizieren und zu bewerten. Die wesentlichen Schritte verdeutlicht Abb. 6.11. Ausgehend von einem Modell der Hardware (phys. Ressourcen), der Software (Protokolle, Prozesse, Datenstrukturen) und des Lastprofils (Transaktionsankunft und -auswertung) werden **Ressourcen**, **Markentypen** und **Instanztypen** spezifiziert, die in einem DAEMON-Netz referenzierbar sind. Die Kanäle und Kanal/Instanzbeziehungen des Netzes ergeben sich aus dem Operations- und Ausfall-Submodell (im Bild nicht gezeigt). Dieses 'Phasenmodell' zur Konstruktion von DAEMON-Netzen läßt sich als Verfeinerung der Komponente "Modellentwurf und -implementierung" in Abb. 2.1 in unsere Methodik der VDBS-Modellierung einordnen.
Das Phasenmodell wird im nächsten Abschnitt an einem Beispiel erläutert.

6.3 DAEMON-Modell eines einfachen Commitprotokolls

Im folgenden wird exemplarisch ein DAEMON-Modell eines verteilten 2-Phasen-Commitprotokolls spezifiziert. In diesem Modell werden fast alle Elemente von DAEMON-Netzen verwendet. Insbesondere ist eine Aufteilung in Operations- und Ausfall-Submodell vorhanden. Das Modell beinhaltet auch eine Spezifikation der Protokollumgebung. Diese basiert in Teilen auf dem VDBS-Basisreferenzmodell (Kap. 4), sodaß das Modell als Grobversion eines kompletten VDBS-Modells (Kap. 7) anzusehen ist.

Hardware-Modell

Ausgangspunkt sei das Modell einer Rechnernetz-Architektur, wie bereits in Abb. 4.1 beschrieben. Das Rechnernetz besteht aus N homogenen Knoten, wobei jeder Knoten aus den phys. Ressourcen

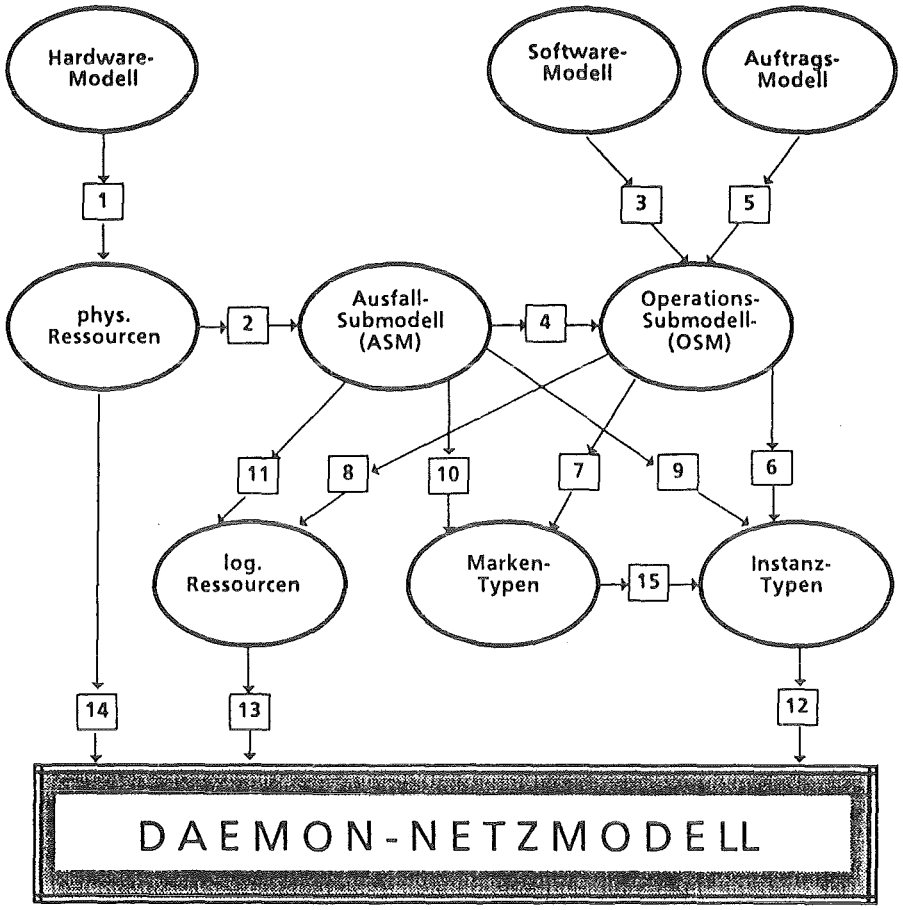


Abb. 6.11: Spezifikation von DAEMON-Netzen für Rechensysteme

1 Spezifikation der phys. Ressourcen
2 Spezifikation Struktur und Zeitverhaltens des Ausfallmodells

3 Spezifikation der Protokolle und Prozesse im 'Normalfall'
4 Spezifikation der Protokolle und Prozesse im 'Fehlerfall'
5 Modellierung Lastgeneration und Auftragsauswertung
6 Spezifikation Struktur und Zeitverhaltens des Operationsmodells (Realisierung der Instanztypen mittels der Markentypen (15))
7 Spezifikation Markentypen für Nachrichten, Datenstrukturen, Aufträge
8 Spezifikation der log. Ressourcen (z.B. timeouts)

9 Spezifikation Struktur und Zeitverhaltens des Ausfallmodells (Realisierung der Instanztypen mittels der Markentypen (15))
10 Spezifikation Markentypen für Ausfall- /Wiederanlauf-Events
11 Spezifikation der log. Ressourcen (z.B. Ausfall- und Reparaturzeiten)

12 - 14 Konstruktion eines DAEMON-Netzes, wobei der Typ jeder Netzinstanz in (6, 9) definiert wurde. Referenzierbare Ressourcen wurden in Schritten (1, 8, 11) spezifiziert

$TERMINALS_i$, AR_i, $DISK_i$, NIU_i ($i \varepsilon [1,N]$)

besteht. Untere Schichten des Rechnernetzes werden durch eine l. g. Ressource EE_DELAY (end-to-end delay) repräsentiert, d.h. es wird ein Kommunikationssystem unendlicher Kapazität angenommen, das jede nichtlokale Nachricht an ihren Zielknoten transportiert und dabei um ein bestimmtes 'delay' verzögert. Ferner existiert an jedem Knoten ein Puffer, der die Anzahl nebenläufig bearbeitbarer Aufträge begrenzt. Die Puffergröße ist genau gleich der Anzahl $R\_NO(TERMINALS_i)$, damit kann jeder Auftrag sofort von einem Terminal bedient werden.

Parameter des Hardware-Modells sind:

Name	Wert	Verteilung	Bedeutung
N	7		Knotenanzahl
TCO	[1,50]		Pufferkapazität pro Knoten
tEED	0.020	ERLANG-2	End_To_End Delay pro Nachricht in [se:]

Ausfall-Submodell

Es wird ein Fehlermodell angenommen, in dem jeder AR und jede NIU mit identischer Rate

$$\lambda = 1/100 \qquad \mu = 1/2.5 \ [1/h]$$

ausfällt bzw. wiederanläuft.

Auftrags-Modell

Aufträge (Transaktionen) können an einem Terminal jedes Knotens eingegeben werden. Befindet sich ein aktivierbarer Auftrag im Puffer, dann wird er von einem Terminal bedient, indem er um eine gewisse Denkzeit verzögert und dann dem Arbeitsrechner zur Bearbeitung weitergegeben wird. Aufträge sind voneinander unabhängig und können an jedem Rechnernetz-Knoten 1,...,N aktiviert werden, solange der Auftragspuffer noch nicht voll ist, d.h. seine Kapazität TCO noch nicht erreicht hat. Ist diese an einem Knoten k erreicht, d.h. sind zu diesem Zeitpunkt TCO Transaktionen mit Heimatknoten k aktiv, dann kann die nächste Aktivierung erst nach Terminierung einer der aktiven Aufträge erfolgen. Ebenso können während eines Ausfalls des AR_k oder der NIU_k keine Aktivierungen erfolgen, auch wenn TCO noch nicht erreicht wurde. Es handelt sich also um ein **geschlossenes Modell**, in dem maximal TCO • N Transaktionen nebenläufig aktiv sein können.

Das Auftragsprofil wird durch die Anzahl Unteraufträge (Subtransaktionen) bestimmt. Diese Anzahl ist durch die Knotenanzahl N begrenzt. Dabei ist die Zugriffshäufigkeit jedes Knotens $i \varepsilon [1,N]$

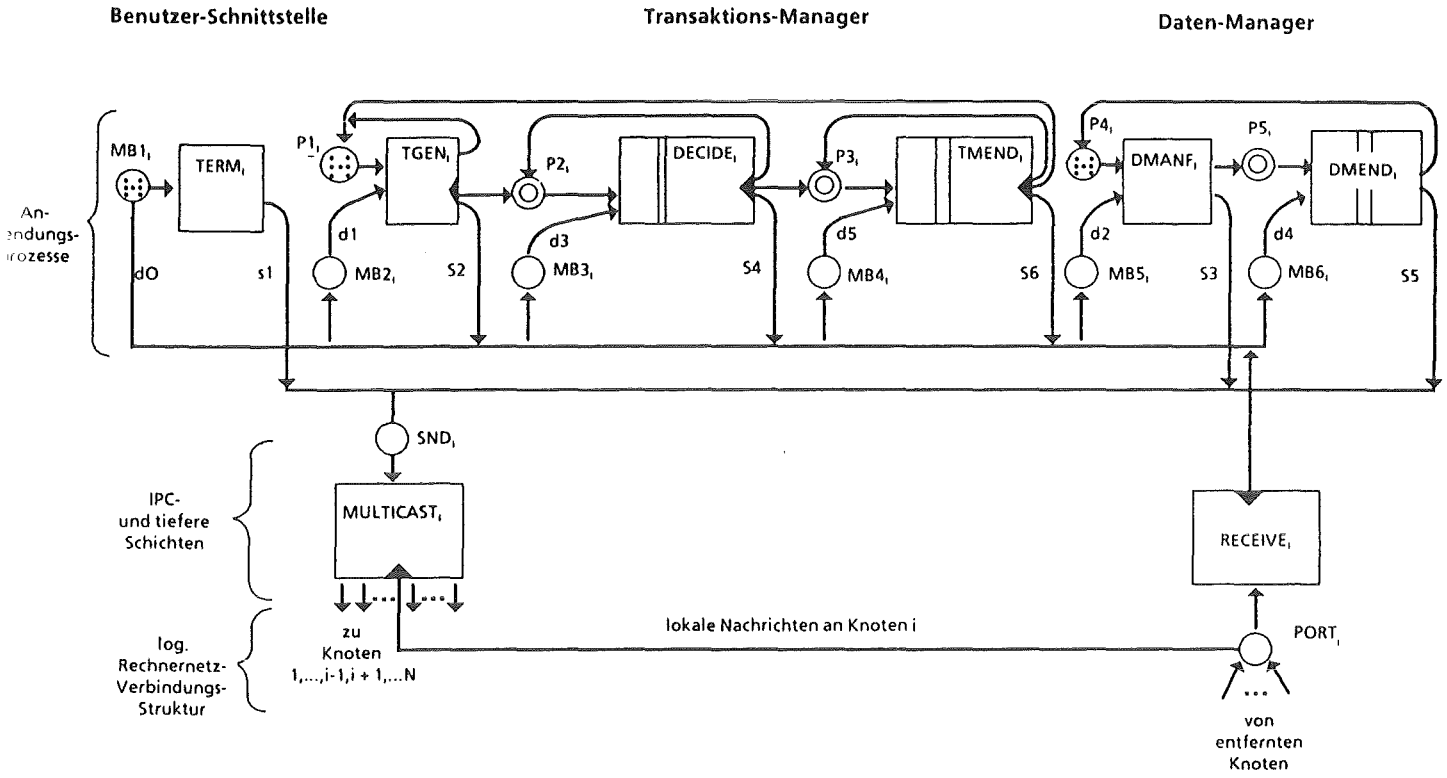


Abb. 6.12: DAEMON-Netz eines verteilten Commitprotokolls auf Knoten i ε [1,N] eines homogenen N-Knoten-Rechnernetzes

gleich.

Parameter des Auftrags-Modells sind:

Name	Bereich	Verteilung	Bedeutung
D	1/15	Negexp	'Denkrate' pro Terminal [1/sec]
NSUB	1,3,5,7		Max. Anzahl Subtr. pro Transaktion

Aufträge und Unteraufträge werden als Marken im DAEMON-Netz modelliert. Diese Marken sind vom strukturierten Typ d:

d = <TID, KQ, DEST_SET, MZ, T_STATE, DATA, ARR_TIME> mit :

Attribut	Wertebereich	Bedeutung
TID	(1,0),..,(1,max), . . . (N,0),..,(N,max), max ε N	Transaktions-ID
KQ	[1,N]	Quell-Knoten
DEST_SET	2^N - {0}	Ziel-Knotenmenge
MZ	MB1,...,MB6	Ziel-Mailbox
T_STATE	OPEN,COMMITTED,ABORTED	Transaktions-Zustand
DATA	ACK, NACK	Ergebnis einer Subtransaktion
ARR_TIME	[0,tmax], tmax ε R^+	Transaktions-Ankunftszeit

Operations-Submodell

Dieses Submodell ist in Abb 6.12 als DAEMON-Netz dargestellt. Es beschreibt ein 2-Phasen Commit-Protokoll und dessen Umgebung. Das Netz zeigt die Schichtenarchitektur auf einem Knoten. Das Gesamtmodell wird durch N identische solcher Netze beschrieben, die durch logische Punkt-zu-Punkt Verbindungen miteinander gekoppelt sind, wie in Abb. 6.12 angedeutet. Instanzen werden durch J_i referenziert, wobei J der Instanztyp und i die Knotennummer ε [1,N] angibt, dem diese Instanz zugeordnet ist.

Den Instanzen des Modells sind folgende Instanzattribute und Parameter zugeordnet (i ε [1,N]) :

Instanz	Ressource (log/phys)	Zeit-Param.	Bereich [sec]	Bedeutung
TERM(i)	TERMINALS(i)	D	1/15	Denkrate [1/sec]
TGEN(i)	AR(i)	tTM1	0.2	Transaktions-Generation
DECIDE(i)	AR(i)	tTM2	0.05	Ergebnis-Einsammlung
TMEND(i)	AR(i)	tTM3	0.1	Transaktions-Terminierung
DMANF(i)	DISK(i)	tDM1	0.06,0.12	Datenbankzugriff
		AB_RATE	0.05	Abbruchrate Subtransaktion
DMEND(i)	AR(i)	tDM2	0.01	Subtr.-Terminierung
MULTICAST(i)	NIU(i)	tNIU	0.001	Nachr.-Bearbeitungszeit
RECEIVE(i)	EE_DELAY	tEED	0.020	globale Nachr.-Verzögerung
		tLOCAL	0.008	lokale Nachr.-Verzögerung

Das Modell besteht aus zwei aggregierten Schichten. Ein verteiltes System wird als eine Hierarchie interagierender Protokollinstanzen auf diesen Schichten modelliert, wobei Instanzen einer Schicht den Dienst für die darüberliegende Schicht erbringen. Jede Instanz einer Schicht wird einem Server-Prozeß dieser Schicht zugeordnet. Das Modell der unteren Schicht enthält die IPC- und alle darunterliegenden Schichten des Rechnernetzes. Ein fehlerfreier Nachrichtenaustausch findet über zwei Instanzen statt. Die selektive MULTICAST-Instanz leitet eine Nachricht an die von einer Transaktion individuell adressierte Menge der Zielknoten weiter, wie bereits in Abb. 6.5 beschrieben. Eie Nachricht kommt in Kanal $PORT_i$ eines Zielknotens an und wird dann von der selektiven RECEIVE-Instanz an eine der Ziel-Mailboxes MB1 bis MB6 der lokalen 'Anwendungsprozesse' weitergeleitet. Diese Instanz

verzögert Nachrichten von entfernten Knoten um das end-to-end delay tEED; lokale Nachrichten werden lediglich um tLOCAL Zeiteinheiten verzögert (s.u.).
Das Modell der oberen Schicht enthält die Prozesse "Terminal-Manager", "Transaktions-Manager", "Daten-Manager", deren Instanzen mit Hilfe von Nachrichtenaustausch über die IPC-Schnittstelle das Protokoll zur verteilten Transaktionsverarbeitung realisieren. Nachrichten werden durch Marken repräsentiert. Alle drei Prozesse eines Knotens können durch maximal TCO nebenläufige Transaktionen aktiviert werden, was durch die Anfangsmarkierung M_0 der 'initialen' Kanäle $MB1_i$, $P1_i$ und $P4_i$ dieser Prozesse auf jedem Knoten i ε [1,N] gesichert wird:

$M_0(MB1_i)$ = TCO <>
$M_0(P1_i)$ = TCO <>
$M_0(P4_i)$ = TCO <>

Auf jedem dieser Kanäle liegen also anfangs genau TCO strukturlose Marken.

Vor- und Nachbedingung der Instanzen der Anwendungsprozesse werden nun in der Reihenfolge der Bearbeitung einer Transaktion beschrieben.

(1) Instanz **TERM(i)**:

Der Terminal-Manager initiiert eine Transaktion durch Generierung einer Marke T vom oben definierten Strukturtyp:

- T.TID : netzweit eindeutiger Transaktions-ID
- T.DEST_SET := {i}
- T.MZ := MB2 (die Transaktion soll von der Instanz TGEN mit Empfangsmailbox MB2 weiterverarbeitet werden)
- T.ARR_TIME : Ankunftszeit der Transaktion wird gespeichert, um daraus später Antwortzeiten zu bestimmen
- T.T_STATE := OPEN

(2) Instanz **MULTICAST(i)**:

Selektiver 1:N-Nachrichtentransport entsprechend Abb. 6.5. Zusätzlich wird in m.KQ der Quellknoten jeder gesendeten Nachricht vermerkt.

(3) Instanz **RECEIVE(i)**:

Nachrichtenempfang und -Verzögerung, unterschieden nach lokaler bzw. nichtlokaler Nachricht.

(4) Instanz **TGEN(i)**:

Repräsentiert Übersetzung von T und Generierung von Subtransaktionen. Aus dem Intervall [1,NSUB] wird eine Zufallszahl z gezogen, die der individuellen Anzahl T-Subtransaktionen entspricht. Jede Subtransaktion soll auf einem unterschiedlichen Rechner ausgeführt werden, wobei die Auswahl jedes Rechners gleichwahrscheinlich ist. Das stellt ein 'random sampling'-Problem dar: Es werden z Rechner aus der Menge {1,...,N} erwartungstreu ausgewählt, d.h. jeder Rechner kann mit gleicher Wahrscheinlichkeit 1/N ausgewählt werden. Diese Menge definiert die Partnerknoten von T, die als Zielknotenmenge in T.DEST_SET gespeichert wird. Subtransaktionen sollen von den DMANF-Instanzen ausgeführt werden, die diese Anforderung an ihrer Mailbox MB5 erwarten. Daher bekommt T.MZ als Wert MB5 zugewiesen. Ist irgendein Knoten in DEST_SET nicht verfügbar, wird T sofort abgebrochen und die Kontrollmarke an Kanal P1 zurückgegeben. Andernfalls wird T lokal im SET-Kanal P2 gespeichert und die Aufforderung zur Ausführung der Subtransaktionen an alle Rechner in DEST_SET gesendet.

(5) Instanz **DMANF(i)**:

Diese Instanz führt eine Subtransaktion SUB(T) aus. SUB(T) ist im Rahmen dieses Modells nicht genauer spezifiziert. Es wird genau ein log. Datenbankzugriff durch einen stochastischen Zeitverbrauch von DMANF modelliert. Die SUB(T)-Ausführung wird mit Wahrscheinlichkeit AB_RATE abgebrochen. Die Instanz schickt folgende Nachricht RESULT an die DECIDE-Instanz auf T's Heimatknoten zurück:

```
RESULT.TID := T.TID
RESULT.DEST_SET := {T.KQ}
RESULT.MZ := MB3
RESULT.DATA := ACK (Ausführung erfolgreich),
               NACK (sonst)
```

(6) Instanz **DECIDE(i)**:

Realisierung einer selektiven N:1-Empfangsoperation, genau wie bereits in Abb. 6.6 erläutert. Wenn die Ergebnisse aller Subtransaktionen vorliegen, wird T.T_STATE auf COMMIT (alle Ergebnisse waren ACK) bzw. ABORT (sonst) gesetzt. Dieses Ergebnis wird an die Instanzen DMEND in T.DEST_SET gesendet, damit der

Daten-Manager ein ABORT bzw. COMMIT auf den zugegriffenen Daten der Transaktion anwenden kann. Daher gilt als Nachbedingung der DECIDE-Instanz:

T.TID, T.DEST_SET : invariant

T.MZ := MB6

T.T_STATE = COMMIT bzw. ABORT

(7) Instanz **DMEND(i)**:

Ausführung von Subtransaktions-Commit bzw. -Abort. Rücksendung eines "ACK" an den Heimatknoten der Transaktion T, Instanz TMEND. Rückgabe der Steuermarke an Eingangskanal P4 des Daten-Managers, da die Subtransaktion nun fertig ist.

(8) Instanz **TMEND(i)**:

Diese Instanz sammelt die Ergebnisse der Subtransaktionen von T ein. Wenn alle Ergebnisse eingetroffen sind, erfolgt die Endebehandlung für T. Ankunftszeit und Endzustand von T werden zur späteren Auswertung von Transaktions-Antwortzeiten und -Durchsätzen festgehalten. Das "Ergebnis" der T-Ausführung wird an den Benutzer zurückgeschickt, der darauf an einem Terminal wartet. (In diesem einfachen Modell besteht das Ergebnis lediglich in der unstrukturierten Meldung, daß T terminiert hat.)

Das damit spezifizierte Operationale Submodell OSM ist unvollständig, da keine Recovery- und Restartverfahren für beliebige Zustandswechsel im Ausfallmodell ASM spezifiziert sind. Ein komplettes OSM wird in Kap. 7 modelliert.

6.4 Zur strukturellen, dynamischen und Markovschen Analyse von DAEMON-Netzen

DAEMON-Netze können durch geeignete Einschränkungen der Netzklasse und durch geeignete Transformationen auf klassische bzw. stochastische Platz-Transitions-Netze (PN bzw. SPN) abgebildet werden, um sie analytischen Untersuchungen zugänglich zu machen. Die Klasse der SPN bzw. PN umfaßt diejenigen Netze, für die zur Zeit bekannte und ausgereifte Analyseverfahren zur Verfügung stehen.

Bei einer Abbildung von der 'höheren' DAEMON- auf die 'tiefere' PN-

bzw. SPN-Netzklasse gehen daten- und markenabhängige Informationen verloren. Von geeigneten Abbildungen wird daher gefordert, daß sie sowohl qualitative, aus der Netztheorie ableitbare Eigenschaften, als auch quantitative, z.B. aus der Markovtheorie ableitbare quantitative Eigenschaften -letztere zumindest angenähert- konservieren. Der Nachweis qualitativer Eigenschaften dient der Überprüfung der theoretischen Modellannahmen (Verifikation (A), Kap. 2.1). Der Nachweis quantitativer Eigenschaften kann zur Überprüfung der Simulationsergebnisse des detaillierten DAEMON-Modells durch die Markovsche Analyse eines daraus abgeleiteten approximierten SPN-Modells dienen (Verifikation (C)). Dabei ist im Auge zu behalten, daß dieses SPN-Modell evtl. nur von theoretischem Interesse ist, wenn durch die Verfeinerung des DAEMON-Modells eine Zustandsexplosion auftritt, die eine praktische Analyse sehr erschwert (vergl. Kap. 2.4.5).

Im folgenden wird ein mehrstufiger Lösungsansatz vorgeschlagen, der unterschiedliche Methoden für die qualitative bzw. quantitative Analyse kombiniert. Dabei sind wir zunächst an solchen eingeschränkten DAEMON-Netzklassen interessiert, die Analysierbarkeit im Sinne der Netztheorie (NET-Analyse) bzw. im Sinne der Markovtheorie (PERF-Analyse) erlauben. Davon ausgehend werden Transformationen konstruiert, deren resultierende Netze sowohl eine NET- als auch eine PERF-Analyse erlauben. Dazu dienen die folgenden Definitionen.

6.4.1 Spezielle Klassen von DAEMON-Netzen

Def. 6.10 Spezielle Klassen von DAEMON-Netzen (DN)

(a) Sicheres DAEMON-Netz (S-DN)
Ein (DAEMON-)Netz DN = (A,C; F,R,D,ATT,M_0) ist ein S-DN

$$\Leftrightarrow \exists k \in N : \forall m \in M \; \forall c \in C : |m(c)| < k$$

(b) Einfach-zeitattributiertes DAEMON-Netz (EZ-DN)
Ein DAEMON-Netz DN = (A,C; F,R,D,ATT,M_0) ist ein EZ-DN

$$\Leftrightarrow \forall a \in A \; \forall m \in M :$$
$$T\_FUNCTION(a, T\_PAR, m) = T\_PAR(a) \;\vee$$
$$T\_FUNCTION(a, T\_PAR, m) = negexp(T\_PAR(a))$$

(c) Erweitertes SET-Auswahl DAEMON-Netz (ES-DN)
Ein DAEMON-Netz DN = (A,C; F,R,D,ATT,M_0) ist ein ES-DN

$$\Leftrightarrow \forall c \in C \; \forall a \in c^* :$$

$$
\begin{aligned}
ACC(c) = SET \wedge\ & (c,a) \in in \\
\Rightarrow\ & \exists a' \in A-\{a\} : \\
& (c,a') \in hb \\
\wedge\ & {}^{*}a = {}^{*}a' \\
\wedge\ & \forall m \in M : PRECOND(a,m) \Leftrightarrow \neg PRECOND(a',m)
\end{aligned}
$$

(d) Performance-DAEMON-Netz (P-DN)

Ein DAEMON-Netz $DN = (A,C; F,R,D,ATT,M_0)$ ist ein P-DN

$\Leftrightarrow$ DN ist ein S-DN $\wedge$
DN ist ein EZ-DN $\wedge$
DN ist ein ES-DN $\wedge$

(Def. Ende)

Ein **sicheres Netz** (S-DN) ist die wesentliche Voraussetzung bei der PERF-Analyse des **Erreichbarkeitsgraphen** eines DN. Zusätzlich sind auch semantische Modelleigenschaften aus dem Graph ableitbar. Dieser Graph enthält **alle Markierungsfolgen**, die aus einer gegebenen Anfangsmarkierung M_0 eines Netzes durch Schalten von Instanzen entstehen können. (Im Gegensatz dazu liefert ein Simulationslauf lediglich **eine mögliche** Markierungsfolge.) Eine Markierung entspricht einem Zustandsknoten im Graph. Ein Zustandsübergang entspricht dem Schalten genau einer Instanz, die zu einer neuen Markierung führt. Der Erreichbarkeitsgraph ist genau dann endlich, wenn aus M_0 nur endlich viele Folgemarkierungen entstehen können. Zu beachten ist, daß Sicherheit für zeitattributierte Netze nicht entscheidbar ist /BEME83/, wohl aber für ein gegebenes k, z.B. der maximalen Kanalkapazität: Wegen Def. 6.5 gibt es in einem DN nur endlich viele unterschiedliche Marken; wegen Def. 6.6 ist daher die Anzahl möglicher M_0 eines DN beschränkt. Damit ist auch die Anzahl möglicher M_0-Folgemarkierungen bei gegebenem k endlich. In diesem Fall spricht man von "k-Sicherheit" anstatt von "Sicherheit" /BEFE86/.

Ein **Erweitertes SET-Auswahl DAEMON-Netz** (ES-DN)
garantiert die Vollständigkeit aller Vorbedingungen in Instanzen eines DAEMON-Netzes, d.h. Vorbedingungen eines ES-DN zerstören nicht dessen Lebendigkeit, da aus Def. 6.10 (b) folgt:

> Zu jeder Instanz a mit einem SET-Inputkanal c und einer Vorbedingung V existiert eine andere Instanz a', die mit c durch eine Inhibitorkante verbunden ist und deren Vorbedingung genau ¬V ist. Da die Inputkanal-Menge für a und a' gleich ist, kann entweder a oder a' stets unabhängig von der c-Markierung

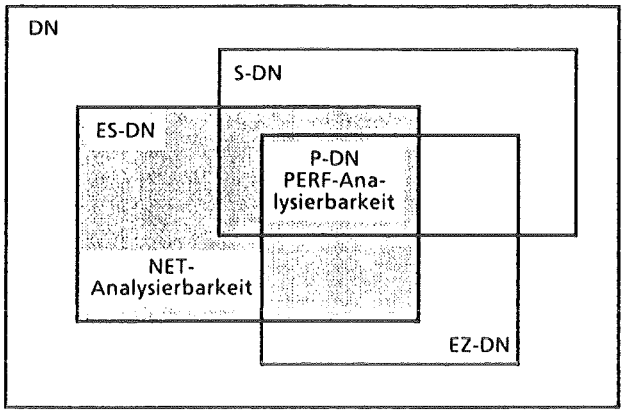


Abb. 6.13: DAEMON-Netzklassen und ihre Analysierbarkeit

schalten.

Das Beispielnetz des Nachrichtenempfangs mit Timeout in Abb. 6.8 ist ein ES-DN. Man sieht dort, daß bei Ankunft einer zu empfangenden Nachricht stets eine der Bedingungen V, ¬V erfüllt ist. Ensprechendes gilt bei Ablauf eines Timeouts für V', ¬V'.

Schließlich erleichtert die Einschränkung auf ein **Einfach-zeitattributiertes Netz** (EZ-DN) die PERF-Analyse, da der Erreichbarkeitsgraph eines solches Netzes isomorph zu einer kontinuierlichen homogenen Markovkette ist. Ist das Netz zusätzlich sicher (S-DN), dann ist die Markovkette wegen der Isomorphie endlich. Für die NET-Analyse sind diese Einschränkungen nicht erforderlich.

Bemerkung:

Eine größere, noch PERF-analysierbare Netzklasse erhält man durch die Erweiterung der Zeitverteilungsfunktion auf 'Phase-Type' Verteilungen, die z.B. die Exponential-, Erlang-, Hypoexponential- und Hyperexponential-Verteilungen enthalten. Ein Netz mit solchen Verteilungen von Zeitverbrauch bzw. -verzögerung der Instanzen kann per Verfeinerung auf ein EZ-DN bei Erhaltung der zeitlichen Eigenschaften transformiert werden /TPN85/. Z.B. wird eine Instanz a mit Erlang-2 Zeitverbrauchsverteilung mit Mittelwert μ verfeinert zu zwei 'hintereinander geschalteten' Instanzen a', a'' mit exponential verteiltem Zeitverbrauch mit

gleichem Mittelwert μ.

Die definierten DAEMON-Netzklassen und deren noch zu zeigende Transformation auf NET- bzw. PERF-analysierbare, 'niedere' Netzklassen sind in Abb. 6.13 dargestellt.

6.4.2 Transformation von DAEMON-Netzen

Ausgehend von den Netzklassen ES-DN und P-DN, werden folgende Schritte (S1) bis (S4) vorgeschlagen, die in Abb. 6.14 zusammengefaßt sind:

(S1) Für ES-DN bzw. P-DN: Verhaltensäquivalente Transformation TRAFO1 auf die entsprechende Funktionsnetz-Klasse ES-FN1 bzw. P-FN1. Dabei werden die Attribute "Ressource", "SET-Kanal" und "Vorbedingung" eines DAEMON-Netzes elimiert. Alle zeitlichen sowie netztheoretischen Eigenschaften bleiben dabei erhalten.

(S2) Für ES-FN1 bzw. P-FN1 : Verhaltensäquivalente Transformation TRAFO2 auf die Klassen ES-FN2 bzw. PS-FN2, wie in /GODB83/ beschrieben. Dabei werden strukturierte auf anonyme Marken abgebildet; Copy-Kanten, Instanzgedächtnis und Kanalzugriffsmodi entfallen. Dabei ggf. entstehende isolierte Netzelemente werden gestrichen.

(S3) Für ES-FN2 bzw. P-FN2 : Annähernd verhaltensäquivalente Transformation TRAFO3 auf die Klassen ES-FN3 bzw. P-SPN. Dabei werden alle Instanzen, deren (markenabhängige) Nachbedingung partielle Ausgabe aufweist, auf ein Teilnetz verfeinert, in dem alle Instanzen totale (markenunabhängige) Ausgaben aufweisen.

(S41) Netze der Klasse P-SPN sind sichere Stellen/Transitionsnetze mit deterministischem u/o negativ-exponentiell verteiltem Zeitverbrauch von Transitionen und konstanten Auflösungsraten für Schaltkonflikte. Aus einem FN3-Netz und dessen Anfangsmarkierung M_0 kann daher ein endlicher Erreichbarkeitsgraph erzeugt werden. Dieser Graph gestattet

- eine direkte Markovanalyse. Diese liefert stationäre Wahrscheinlichkeiten für jeden Zustand des Graphen und damit für jede von M_0 aus erreichbare Netzmarkierung. Daraus lassen sich Aussagen über mittlere Durchsätze und Antwortzeiten einfach ableiten, die allerdings lediglich für das jeweilige M_0 gültig sind. Die Methodik findet sich in /MCB84, TPN85,

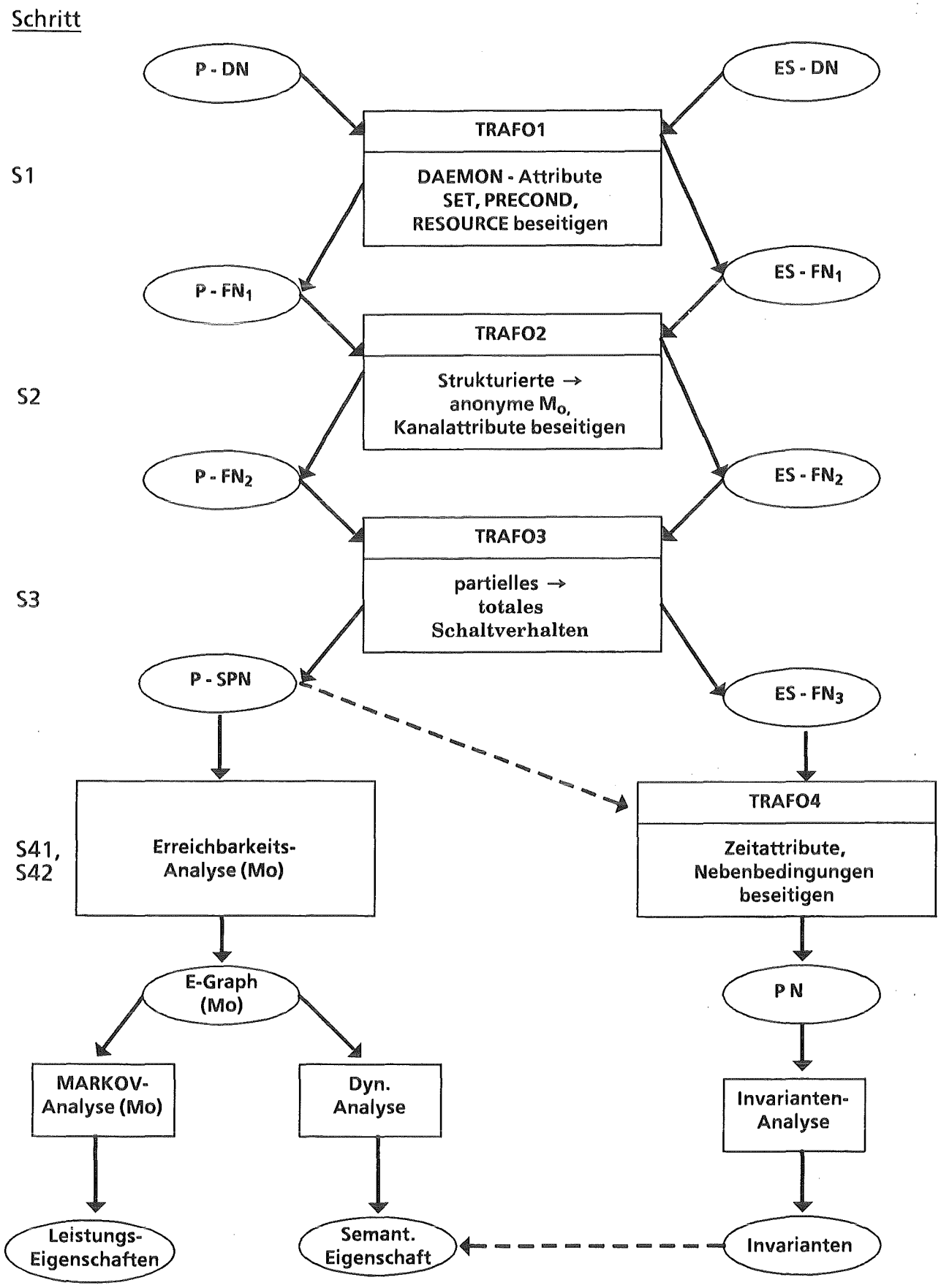


Abb. 6.14: Schritte zur Transformation und Analyse von DAEMON-Netzen

ZUBE85/ detaillierter beschrieben.

- Nachweis semantischer Eigenschaften, z.B. Lebendigkeit, k-Sicherheit, Erreichbarkeit und Reproduzierbarkeit bestimmter Zustände (vergl. Definitionen in /GODB83/). Z.B. gilt Lebendigkeit, wenn jeder Zustand im Erreichbarkeitsgraph mindestens einen Folgezustand hat. Die erzielten Aussagen gelten ebenfalls nur für M_0. Diese "dynamische Analyse" von Modelleigenschaften wurde in /BEME83/ vorgeschlagen. Erreichbarkeitsanalysen zum Nachweis von Eigenschaften von Protokollen, die als Petrinetze modelliert werden, finden sich auch in /BAGA81, WALT82/.

(S42) Für ES-FN3 oder für P-SPN : Transformation auf Stellen-Transitions-Netze (PN). Dabei werden die Zeitverbrauchs-Attribute von Instanzen gelöscht, sodaß die resultierenden Netze frei von allen in DAEMON- bzw. Funktionsnetzen eingeführten Attributen sind. Auf PN sind dann matrixalgebraische Methoden zur Invariantenanalyse ausführbar, die strukturelle Modelleigenschaften bestimmen können /GODB83, SCNK84/. Allerdings geht durch die Ignorierung der Zeitattribute Information verloren, sodaß nur bestimmte Eigenschaften von PN auch im ursprünglichen, zeitbehafteten Netz gelten. Eine stärkere Aussage ist möglich, wenn man die eingeschränkte Klasse der P-SPN ohne Zeitattribute betrachtet: Nach /MCB84/ haben Netze beider Klassen für gegebene Anfangsmarkierungen **denselben Erreichbarkeitsgraph**, sodaß alle Aussagen der netztheoretischen Analyse eine PN-Netzes auch für das entsprechende SPN-Netz gelten.

Wir gehen nun auf die einzelnen Transformationen TRAFO1 bis TRAFO4 genauer ein.

TRAFO1: (s. Abb. 6.15)

Inhibitor-Kanten, SET-Kanäle und Instanz-Vorbedingungen können einfach eliminiert werden, wenn das DAEMON-Netz zur Klasse der ES-DN (erweiterte SET-Auswahl) gehört. In einem solchen Netz ist das Schaltverhalten von Folgeinstanzen eines SET-Kanals c **unabhängig** von der Markierung dieses Kanals. Es bietet sich daher die in Abb. 6.15(a)

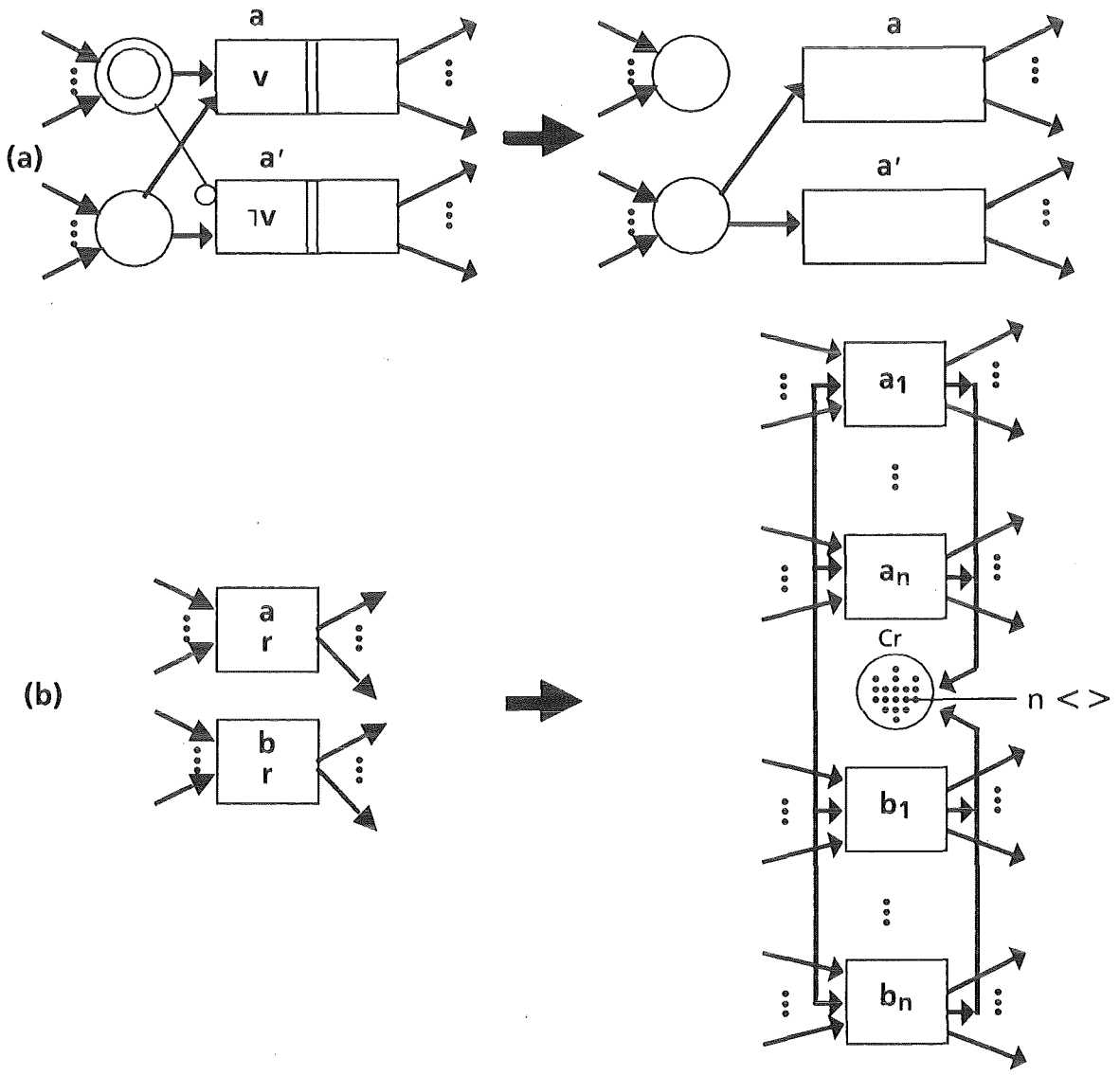


Abb. 6.15: Transformation TRAFO1 von ES-(P-)DN nach ES-(P-)FN
(a) Eliminierung von Instanz-Vorbedingungen und Inhibitor-Kanten (Beispiel)

(b) Verfeinerung von Instanzen, die eine n-ausgeprägte physische Ressource r benutzen

gezeigte Transformation an: Output-Kanten von c werden gestrichen. Das SET-Auswahlverhalten und die Instanzvorbedingungen sind damit irrelevant und können ebenfalls eliminiert werden. Falls dabei isolierte Netzelemente entstehen, sind diese zu entfernen, da sie nicht zum dynamischen Netzverhalten beitragen.

Ressourcenattribute in DAEMON-Netzinstanzen lassen sich durch einen Netzmorphismus ebenfalls eliminieren. Das ist beispielhaft in Abb. 6.15(b) für zwei Instanzen a, b mit demselben phys. Ressource-Attribut r erläutert (vergl.. auch Abb. 6.2,6.3). Ressource r habe dabei n Ausprägungen. Nach der Verfeinerung können a und b insgesamt n mal nebenläufig aktiviert werden. Bei einer FCFS-Schedulingstrategie für r-Instanzen reicht die gezeigte Verfeinerung mit einem zusätzlichen Kanal r_c aus. Anderfalls wäre dieser Kanal entsprechend zu verfeinern (im Bild nicht gezeigt).

Ist r eine logische Ressource, ist eine ähnliche Verfeinerung zu konstruieren. Ein Kanal $c_{r(a)}$ kontrolliert den Ressourcenzugang für jede beanspruchende Instanz a. Die Anzahl der möglichen nebenläufigen r-Aktivierungen ist bei einem k-sicheren Netz durch k beschränkt: Daher ist jede r-Instanz durch k identische Instanzen mit gleichem Vor- und Nachbereich zu verfeinern. Auf jedem $c_{r(a)}$-Kanal kommen k Steuermarken. (Bemerkung: k muß vor der Transformation bekannt sein.)

Als Ergebnis von TRAFO1 wurde ein DAEMON- auf ein (i.d.R. komplizierteres) Funktionsnetz abgebildet. TRAFO1 kann rechnergestützt ausgeführt werden.

TRAFO2:

Strukturierte Marken der Anfangsmarkierung M_0 werden durch eine gleiche Anzahl anonymer Marken ersetzt. Copy-Kanten werden gestrichen, da sie nicht zum Schaltverhalten beitragen. Dabei entstehende isolierte Elemente sind zu entfernen. Kanalzugriffsmodi werden gestrichen: Nach TRAFO1 blieben lediglich FIFO-Kanäle übrig, die einen evtl. nichtleeren Nachbereich haben. TRAFO2 kann rechnergestützt ausgeführt werden.

TRAFO3:

Hierbei erfolgt eine Abbildung der markierungsabhängigen, partiellen DAEMON-Schaltregel auf eine markierungsunabhängige, totale Petrinetz-Schaltregel. Dabei muß zwangsläufig Information verloren gehen. Die Abbildung ist nicht aus der statischen Netzstruktur, sondern allein aus der Instanz-Nachbedingung ableitbar. Es besteht das Problem, daß diese allein bestimmt, welche Teilmenge der Ausgabekanäle jeweils adressiert wird (vergl. Kap. 6.1.5). Dabei kann sogar Absorption auftreten, d.h. Marken können verloren gehen. Ferner werden

bestimmte Teilmengen evtl. nie ausgewählt. Bei einer sinnvollen Abbildung von partiellem auf totales Schaltverhalten muß daher bekannt sein

- Welche Teilmengen nie ausgewählt werden (relevant für PERF- und NET-Analyse)
- Mit welcher **Transferrate** welche Teilmenge ausgewählt wird (nur für PERF-Analyse relevant)

Zur Bestimmung der Transferraten wird vorgeschlagen, dazu **Simulationsergebnisse** zu verwenden: So können stationäre, mittlere Raten geschätzt werden. Ist eine Rate gleich 0, wird die ensprechende Teilmenge der Ausgabekanäle (wahrscheinlich) nie ausgewählt. Als Alternative zur Simulation könnten diese Raten vom Modellierer nur 'manuell' ermittelt werden. Ausgaberaten werden wie folgt ermittelt:

Def. 6.11: Instanz-Ausgaberaten und ihre Approximation

Sei
- $\alpha \in A$ eine Instanz mit nichtleerem Nachbereich α^*, $\alpha^* = \{c_1, \ldots, c_{n(\alpha)}\}$
- $p(\alpha) := 2^{n(\alpha)}$
- Beobachtungszeitraum $[0,\tau]$, $\tau \in R^+$

dann

(a) $ACT_\tau(\alpha)$: Anzahl α-Aktivierungen im Beobachtungszeitraum

(b) $TF_\tau(\alpha,i)$ Anzahl α-Aktivierungen im Beobachtungszeitraum mit Ausgabe auf Outputkanal-Teilmenge $C_i \subseteq \alpha^*$ für $i \in [1, p(\alpha)]$

(c) $\Pi(\alpha,i) := \lim_{\tau\to\infty} TF_\tau(\alpha,i) \,/\, ACT_\tau(\alpha)$ für $i \in [1, p(\alpha)]$

stationäre α-Transferraten

sodaß
- $C_1 = \phi$, $C_2 = \{c_1\}$, $C_3 = \{c_2\}$, ... $C_{p(\alpha)} = \{c_1, \ldots, c_{n(\alpha)}\}$

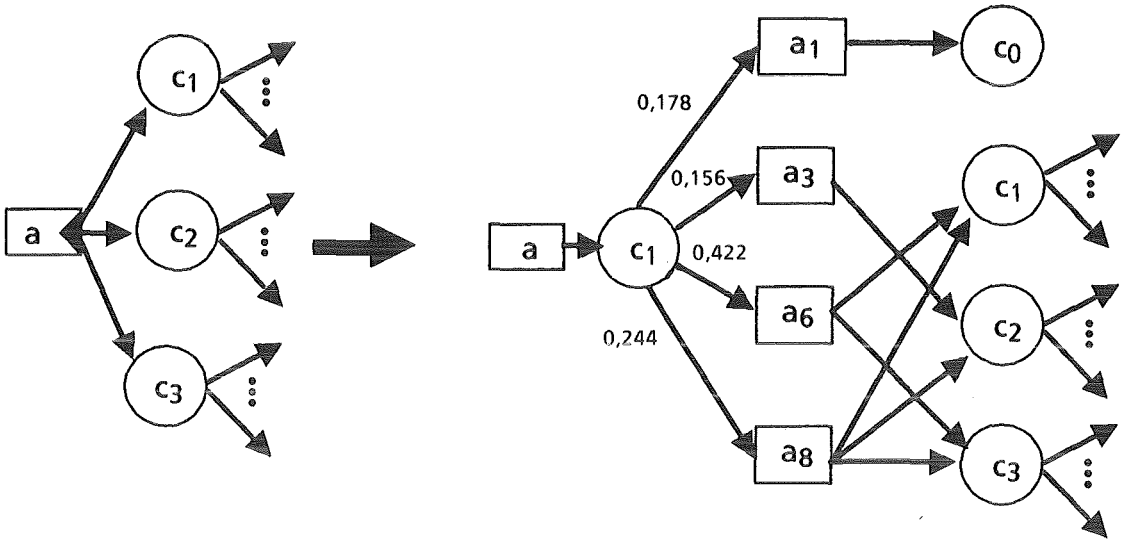
(Def. Ende)

Falls die stationären Transferraten existieren, gilt

$$\sum_{i=1}^{p(\alpha)} \Pi(\alpha,i) = 1$$

Interessant sind folgende speziellen Instanz-Ausgabefunktionen:

- AND $\Pi(\alpha,1) = \ldots = \Pi(\alpha,p(\alpha)-1) = 0$, $\Pi(\alpha,p(\alpha)) = 1$
- XOR $\Pi(\alpha,1) = 0$, $\Pi(\alpha,2) = \ldots = \Pi(\alpha,n(\alpha)+1) = 1/n(\alpha)$, $\Pi(\alpha,n(\alpha)+2) = \ldots = \Pi(\alpha,p(\alpha)) = 0$
- Absorption $\Pi(\alpha,1) = \ldots = \Pi(\alpha,p(\alpha)) = 0$



Folgende Werte wurden im Zeitraum [0,τ] gemessen :

i	a*-Teilmenge	Anzahl Aktivierungen	Transferraten-Schätzung
1	ϕ	24	0.178
3	c2	21	0.156
6	c1,c3	57	0.422
8	c1,c2,c3	33	0.244

Abb. 6.16: Approximation der Instanz-Transferraten zur Transformation TRAFO3 (Beispiel)

In Abb. 6.16 ist ein Beispiel der Transferraten-Ermittlung durch Simulation und ihre Verwendung bei der Abbildung von der partiellen DAEMON- auf die totale Petrinetz-Schaltregel gezeigt.
Zu bedenken ist, daß die Schätzung der Transferraten nur dann zu sinnvollen Resultaten führt, wenn die Raten sich nicht um Größenordnungen unterscheiden. Würde z.B. durch eine Instanz ein Fehlerausgang mit Rate 10^{-6} spezifiziert, dann würde diese Rate nach einem Simulationslauf wahrscheinlich als 0 geschätzt werden, was sicherlich nicht dem spezifizierten Verhalten entspricht.

TRAFO4:

Hierbei werden die verbleibenden Zeitattribute gestrichen und evtl. vorkommende Nebenbedingungen per Verfeinerung beseitigt, um die strukturelle Netzanalyse /SCNK84/ zu ermöglichen. Das Ergebnis ist ein Stellen-Transitionsnetz.

Zur Konstruktion des Erreichbarkeitsgraphen:
Hierbei verweisen wir auf die reichhaltige Literatur, z.B. /TPN35/. Anzumerken ist, daß zur Markovanalyse des Graphen erforderlich ist, daß für jeden 'Free-Choice' Kanal im Netz Konfliktauflösungsraten angegeben werden.

Ein Kanal $c \varepsilon C$ ist ein Free-Choice Kanal, wenn c mehr als eine Folgeinstanz hat ($|c^*|>1$), und wenn alle Inputkanäle dieser Instanzen übereinstimmen:

$a1,a2 \varepsilon c^* \Rightarrow *(a1) = *(a2)$

Auflösungsraten können das Zeitverhalten des Netzes stark beeinflussen, da daraus die Zustandsübergangsraten im Graph hergeleitet werden. Diese Raten können wie folgt bestimmt bzw. definiert werden:

- Instanz-Transferraten werden wie angegeben bestimmt
- Transferraten bei der Auflösung von deterministischen Vorbedingungen können ebenfalls mit Hilfe einer Simulation approximiert werden.
- Alle anderen Raten eines DAEMON-Netzes sind markierungsunabhängig. Daher kann z.B. eine faire Strategie definiert werden, d.h. bei einem Konflikt-Kanal wäre dann die Aktivierungswahrscheinlichkeit für jede Folgeinstanz des Kanals gleich.

Obwohl bei der Abbildung auf analysierbare Netze bestimmte Kanten und isolierte Knoten wegfallen, wird durch Verfeinerung bei der Transformation des partiellen Schaltverhaltens und der Ressourcen das resultierende Netz i.d.R. wesentlich komplexer sein. Netze sind in der Praxis lediglich bis einer bestimmten Größe analytisch behandelbar. Die bisher entwickelten Netzreduktionsverfahren (s. z.B. /LEFA85/) sind zu einer Komplexitätsreduktion ungeeignet, da sie zwar auf die Bewahrung wesentlicher struktureller Eigenschaften hinzielen, dabei aber Performance-Eigenschaften eines stochastischen Netzes i.d.R. **nicht konservieren**. Daher ist die Entwicklung neuer Verfahren sehr wichtig, wenn man nicht allein auf die simulative Untersuchungsmethode vertrauen möchte.
Wir wenden uns als nächstes der Beschreibung von Aufgaben und Funktionen des DAEMON-Systems zu.

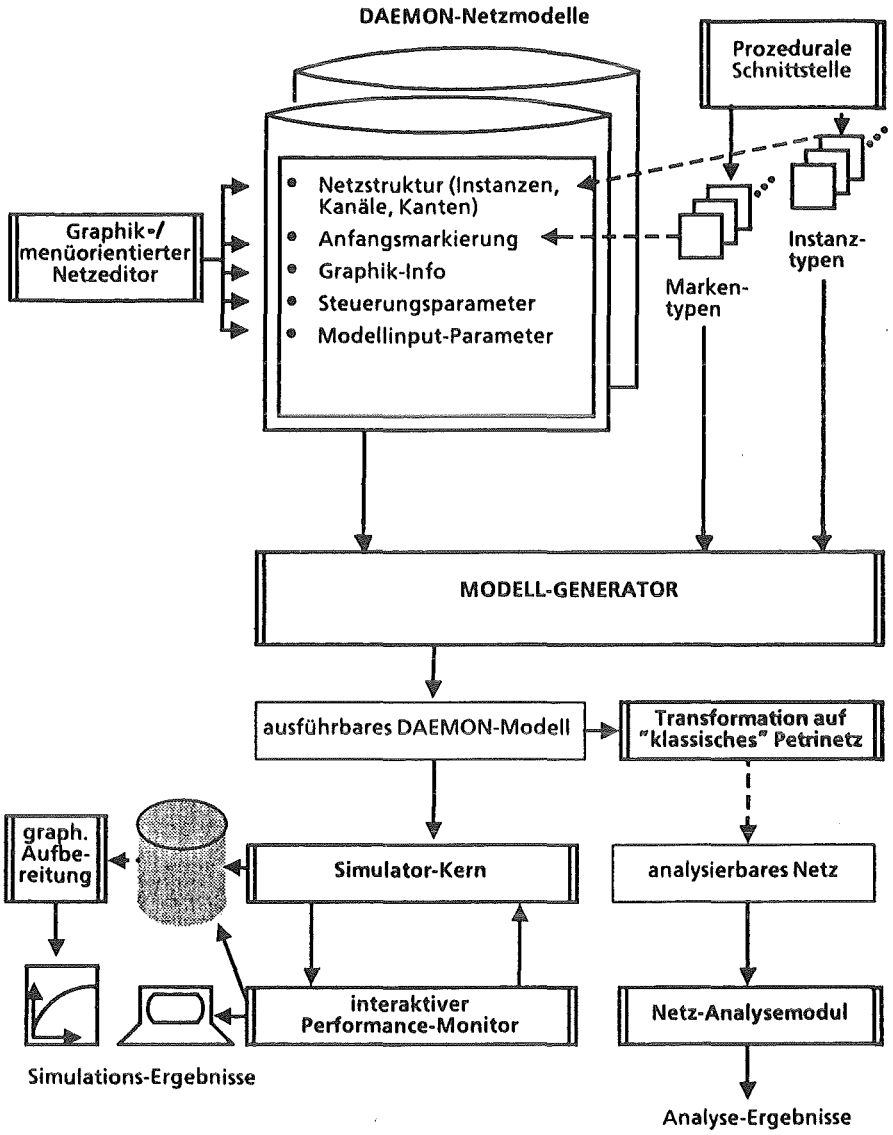


Abb. 6.17: Struktur von DAEMON-Netzmodellen und Komponenten des DAEMON-Systems

6.5 Das DAEMON-System zur rechnergestützten Modellerstellung, -Simulation, -Test und -Analyse

Das vorliegende Kapitel berichtet über das im Rahmen dieser Arbeit entwickelte Modellierungs- und Simulationswerkzeug DAEMON. Es werden Aufgaben, Funktionen und technische Eigenschaften der wesentlichen Komponenten des DAEMON-Systems beschrieben, die das Architekturgerüst des Systems bilden. Einen Überblick über die interne Struktur von DAEMON-Netzmodellen und die Komponenten des DAEMON-Systems vermittelt

Abb. 6.17. Die wesentlichen DAEMON-Komponenten sind:

(1) graphischer Netzeditor
(2) Netzcompiler
(3) Realisierungsunterstützung für Marken- und Instanztypen
(4) Netztransformation und -Analyse
(5) Netzsimulator
(6) Statistische Ergebnisanalyse
(7) Interaktiver Netzdebugger und Performance-Monitor

In Abb. 6.18 wird eine detailliertere Sicht durch die Beschreibung der Schnittstellen zwischen den Systemkomponenten und zu den Benutzern vermittelt.

Alle Subsysteme wurden auf einer IBM3081 unter MVS realisiert und dann auf VM portiert. Als Programmiersprachen wurden PASCAL (Netzeditor) und SIMULA (Subsysteme (2) bis (7)) verwendet. Bei den Subsystemen (2) bis (6) handelt es sich um eine wesentlich erweiterte bzw. verbesserte Version des existierenden Funktionsnetz-Werkzeugs FUN /GODB83/, dessen Laufumgebung von dem Betriebssystem VM auf MVS portiert wurde /LESZ83/.

Die Vorstellung der einzelnen Subsysteme orientiert sich an folgendem Darstellungsschema:

AUFG.: Beschreibung der Aufgaben, die das Subsystem im Modellierungsprozeß übernimmt

F+E.: Beschreibung aller Funktionen, Eigenschaften und verwendeten Konzepte des Subsystems

PROB.: Besondere Probleme bei der Realisierung und deren Lösungen

ERF.: Erfahrungen mit der Benutzung des Subsystems im Modellierungsprozeß und daraus abgeleitete Ansatzpunkte für zukünftige Erweiterungsmöglichkeiten

6.5.1 Graphischer Netzeditor und Netzcompiler

AUFG.: Interaktive und graphische Erstellung, Manipulation, Konsistenzprüfung, Speicherung und Ausgabe von DAEMON-Netzen. Übersetzung in eine interne Datenstruktur.

F+E.: Der Netzeditor gestattet zunächst die Neudefinition oder Änderung eines DAEMON-Netzes. Eingaben und Änderungen erfolgen im wesentlichen über ein Haupt- und zwei Untermenüs (s. Abb. 6.19). Im

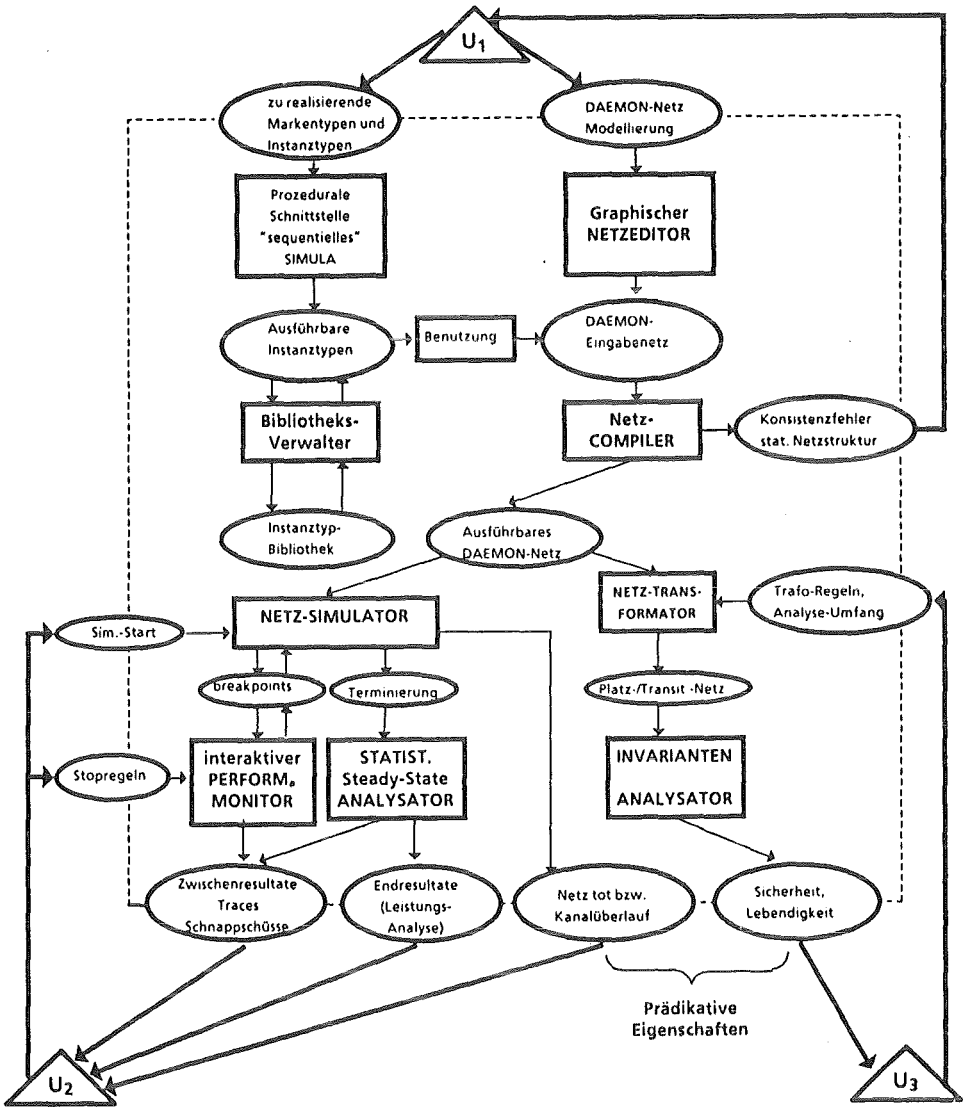


Abb. 6.18: Architektur des DAEMON-Systems

⟷ Kommunikation Benutzer <-> System
- - - Benutzerschnittstellen
⟶ System-Datenfluß

U1: Benutzergruppe "Modellererstellung"
U2: Benutzergruppe "Modellsimulation"
U3: Benutzergruppe "Modellanalyse"

Agency-Menü werden alle Attribute, E/A-Kanten und -Kanäle jeweils einer Instanz spezifiziert. Das Instanzattribut 'partial output' dient der graphischen Kennzeichnung entsprechend den DAEMON-Notationen

(vergl. Abb. 6.10). Graphische Koordinaten können sowohl alphanumerisch als auch mit Hilfe eines Lichtgriffels gesetzt werden. Im Channel-Menü werden alle Attribute jeweils eines Kanals eingegeben bzw. geändert. Die gewünschte Operation wird durch spezielle Funktionstasten ausgelöst. Solche Operationen sind u.a. die graphische Netzausgabe bzw. das Einfügen, Löschen oder Ändern von Instanzen, Kanälen und Kanten. Die möglichen Optionen sind in der Informationszeile jedes Untermenüs gelistet.

Es wird sichergestellt, daß ein auf diese Weise erstelltes Netz korrekt im Sinne der Netzdefinition (s. Kap. 6.1.6) ist. Insbesondere wird auf Vollständigkeit, eindeutige Knotennamen, nichtisolierte Instanzen, Existenz aller im Netz referenzierten Instanztypen in Instanzbibliothek, Schlingen- und Überschneidungsfreiheit der Knoten geprüft.

Ein so erstelltes Netz kann auf folgende Medien ausgegeben werden:

- auf Datei zur Weiterverarbeitung durch den Netzcompiler und zur Archivierung des Netzes
- auf Bildschirm zur interaktiven Inspektion. Es stehen graphische Operationen wie Zooming (Verkleinern) und Blättern (s.u.) zur Verfügung
- auf Plotter (zur Dokumentation)

Der Netzeditor ist in Pascal geschrieben. Die Menue- und Graphikfunktionen verwenden das IBM-Graphikpaket GDDM (Graphical Data Display Manager /IBM84/). Als Arbeitsumgebung kann ein IBM-Bildschirm eines beliebigen Typs dienen. Auch wenn dieser nicht graphikfähig ist, können alle Funktionen des Subsystems außer der Terminalausgabe der Netzgraphik und der graphischen Lichtgriffel-Eingabe aufgerufen werden. Ein so erstelltes Netzmodell ist auch ohne graphische Koordinateninformation durch den DAEMON-Simulator ausführbar, da es in einem GDDM-unabhängigen, portablen Format gespeichert wird.

Der Netzcompiler übersetzt ein von Netzeditor erzeugtes, als korrekt erkanntes DAEMON-Netz in eine interne Darstellungsform, die eine Simulation bzw. Analyse des Netzes gestattet.

PROB.: Die Netzausgabe auf dem Bildschirm wurde durch Positionierungsprobleme und durch die Vielzahl von Instanz- und Kanalattributen erschwert. Das Positionieren erfolgt weiterhin manuell mit Lichtgriffel- oder alphanumerischer Koordinatenvorgabe der

```
                                PETRINET

COMMAND ==> 6     (ENTER A NUMBER)          REVERSE VIDEO ? NO

            1  -  AGENCIES
            2  -  CHANNELS
            3  -  DISPLAY
            4  -  HARDCOPY
            5  -  ERASE NET
            6  -  LOAD NET
            7  -  SAVE NET
            8  -  END
```

```
                               AGENCIES

              NAME : TMEND3                    COORDINATES :
              TYPE : TMEND                       X =   47.0
              TIME :     0.10                    Y =   33.9
          RESOURCE : AR3
            MEMORY :     3.00
    PARTIAL OUTPUT ? YES

  C CHANNELS TYPE                          INTERMEDIATE POSITIONS
    P33      I
    MB43     I
    P33      O
    P13      O

  Commands: I (Insert) D (Delete) M (Move) A (After) B (Before)

  PF:  2=Display  3=End  4=Change Name  5=Find  6=Delete
```

```
                               CHANNELS

              NAME : P33                   Coordinates :
              TYPE : SET                     X = 17
          CAPACITY : 10                      Y = 67.5
    INITIAL MARKING : 10 <>

  PF:  2=Display  3=End  4=Change Name  5=Find  6=Delete
```

Abb. 6.19: Graphischer Netzeditor: wesentliche Menüs (Beispiel)

Netzelemente. Optional können Kanten durch Stützstellen positioniert werden. Attribute sind in der Graphik-Ausgabe nicht sichtbar, sondern können durch Anwählen des Instanz- bzw. Kanal-Menüs manipuliert und inspiziert werden. Weiterhin bestand das Problem, daß DAEMON-Netze ab einer gewissen Größe nicht auf eine Bildschirmseite passen. Es wurde dadurch gelöst, daß der Modellierer ggf. ein Netz über mehrere Bildschirmseiten definiert, die dann durch 'Blättern' selektiert werden können. Dieser Mechanismus wurde unabhängig von den Kanal-/ Instanz-Menüs realisiert.
ERF.: Der Netzeditor eignet sich gut zur effizienten Erstellung 'mittelgroßer' Netze (bis ca. 200 Kanäle und Transitionen). Bei noch größeren Netzen, die gleichartige Teilnetze enthalten -wie z.B. homogenen Rechnernetz-Modellen- wäre ein Ausbau in Richtung auf Kopierfunktionen für Netzelemente und Teilnetze zu wünschen. Ferner ist Rechnerunterstützung bei der Positionierung von Netzelementen anzustreben, z.B. zur annähernden Kreuzungsfreiheit der Netzkanten /SCNK84/, sowie zur Bildung von Netzmorphismen (Vergröberung / Verfeinerung) /JOWI81, VOSS84/.

6.5.2 Realisierung Marken- und Instanztypen

AUFG.: Prozedurale Schnittstelle zur Spezifikation von strukturierten Markentypen sowie von Vor- und Nachbedingungen von Instanzen
F+E.: In der realisierten Version sind Markentypen nur innerhalb Instanzen bekannt; die strenge Typisierung der Instanzschnittstellen wie in PrT-Netzen wird nicht unterstützt. Jeder Markentyp erfordert eigene Zugriffsfunktionen für Markenattribute und eigene Ausgabefunktionen zur mnemotechnischen Ausgabe des Markeninhalts durch den Netzdebugger (s. dort).
Instanztypen werden in 'sequentiellem' SIMULA implementiert, d.h. ohne Verwendung spezieller Prozeß- und Simulationskonstrukte. Die Realisierung wird u.a. durch folgende Funktionen erleichtert:

- Standard-Vorbedingungen zur Selektion von Marken aus einem SET-Kanal entsprechend einem oder zwei Schlüsselattributen.
- Erzeugung individueller Saatkörner für Zufallszahlengeneratoren. Es wird erreicht, daß Zufallszahlenströme annähernd disjunkt sind, d.h. daß sich die entsprechenden Sequenzen bis zu einer bestimmten

Länge nicht überlappen. Das Verfahren wurde nach /BFS83, S.212/ realisiert. (Überlappung kann bei den verwendeten Generatoren vorkommen, da diese periodische Sequenzen liefern /NCC83/.) Insbesondere wird dadurch erreicht, daß unterschiedliche Instanzen mit gleichem Instanztyp ein stochastisch unabhängiges Verhalten haben.

- Random-Sampling Verfahren nach /VITT84/.
- Funktionen zur Erzeugung von Ressource-Crash und -Restart.
- Transaktionsauswertungs-Schnittstelle zur Bestimmung von Antwortzeiten und Durchsätzen.

Instanztypen werden in einer Bibliothek gespeichert und können in verschiedenen DAEMON-Netzen zum Einsatz gelangen.

ERF.: Die Komplexität der VDBS-Modelle erforderte einen Markentyp mit über 30 Attributen (s. Anhang B) und entsprechend komplexen Markentransformationen in Instanztypen. Die angebotene prozedurale Schnittstelle erwies sich dabei zur Realisierung gut geeignet.

6.5.3 Netztransformation und -Analyse

AUFG.: Transformation von sicheren, einfach-zeitattributierten, mit erweiterter SET-Auswahl ausgestattenen DAEMON-Netzen auf klassische und stochastische Stellen-Transitionsnetze. Strukturelle, dynamische und Markovsche Analyse der transformierten Netze, entsprechend den in Kap. 6.4 erarbeiteten Methoden.

F+E.: Die Realisierung befindet sich noch im Entwurfsstadium. Zur Zeit können manuell transformierte DAEMON-Netze mit dem FUN-System strukturell untersucht werden.

6.5.4 Netzsimulator

AUFG.: Ausführung einer Netzsimulation

F+E.: Der Netzsimulator führt das 'Markenspiel' entsprechend der Anfangsmarkierung des Netzes, der Schaltregel-Definition (Def. 6.9) und den realisierten Strategien zur Konfliktauflösung und zum Ressourcen-Scheduling aus, bis eine Stopbedingung erfüllt ist.

Die im Werkzeug "FUN" realisierte, faire Strategie bei Konflikten wurde beibehalten: Konflikte werden per Zufallszahlengenerator so

aufgelöst, daß alle beteiligten Instanzen mit gleicher Wahrscheinlichkeit aktiviert werden können. Sind ferner mehrere, nicht im Konflikt stehende Instanzen 'gleichzeitig' aktivierbar, wird mit einer ähnlichen Methode eine Schalt-Reihenfolge stochastisch festgelegt.

Als Scheduling-Strategien für alle phys. Ressourcen wurden 'fist-come-first-served' (FCFS) und 'shortest-job-first' (SJF) realisiert, die alternativ auszuwählen sind. Sind mehrere Instanzen aktivierbar, die von derselben Ressource bedient werden sollen, wird bei SJF diejenige mit der kleinsten zeitlichen Anforderung zuerst bedient, wobei Unterbrechungen ausgeschlossen sind. Die SJF-Strategie ist optimal bezüglich der mittleren Antwortzeit von Aufträgen, wenn die 'Joblänge' (hier: der aktuelle Instanz-Zeitverbrauch) vor Aktivierung bekannt ist /KLCK76/. Da Transaktions-Antwortzeit eine der wesentlichen Zielgrößen der VDBS-Untersuchung in dieser Arbeit sind, wurde diese Strategie realisiert. Bei anderen Anforderungen, wie Optimierung der Ressourcenauslastung, garantierten Antwortzeiten mit geringer Varianz oder fairem Ressourcen-Zugriff sind andere Strategien vorzuziehen. Z.B. sind Antwortzeit-Varianzen bei der FCFS-Strategie kleiner, entsprechende Mittelwerte jedoch größer als bei SJF.

Zur effizienten Experimentführung sind flexible **Stopregeln** notwendig: Die Vorgabe einer Endzeit ist i.d.R. ungenügend, da die zum Erreichen einer gegebenen Konfidenz und Stationarität erforderliche Anzahl von Beobachtungen selten gut geschätzt werden kann. Andererseits ist bei zu pessimistischer Endzeit-Vorgabe ein Konfidenzgrad evtl. schon vor Terminierung der Simulation erreicht. Daher wurden folgende Stopregeln realisiert. (*)-Regeln sind vom FUN-System übernommen.

- maximale Simulationszeit erreicht (*)
- Netz tot (keine Instanz schaltfähig) (*)
- Kanal-Overflow (optionale Regel) (*)
- minimale Anzahl Transaktionen hat terminiert (optionale Regel)
- Breakpoint des Netzdebuggers erreicht (s.u.)

Bei Breakpoint geht die Kontrolle auf den Netzdebugger über; bei Erreichen jeder anderen Stopregel terminiert die Simulation.

PROB.: Die Laufzeiteffizienz des Simulators kann durch eine aufwendige Auswertung von Instanz-Vorbedingungen beeinträchtigt werden. Daher wurde ein effizientes indexsequentielles Verfahren zum Speichern und

Suchen von Marken in SET-Kanälen entsprechend gegebenem Schlüsselattribut realisiert.
ERF.: Die Stopregeln gestatten eine sehr flexible Steuerung eines Simulationslaufs. Eine weitere sinnvolle Regel ist "simuliere, bis Konfidenzintervall für gegebene Outputgröße X eine Schranke Y unterschreitet". Das bleibt einem späteren Ausbau des Werkzeugs vorbehalten, ebenso wie die Realisierung eines parametrisierten Ressourcen-Schedulers.

6.5.5 Interaktiver Netzdebugger und Performance-Monitor

AUFG.: Interaktive Inspektion, Test und Überwachung von Simulationsexperimenten. Zur 'Verifikation' der Verhaltensähnlichkeit zwischen (DAEMON-)Modell und modelliertem System (vergl. Kap. 2.1) soll die Dynamik des Markenspiels beobachtet werden können. Durch Traces des Markenflusses beim Schalten von Instanzen, durch die Inspektion von Markeninhalten und Zwischenwerten der Leistungsgrößen zu bestimmten Zeitpunkten kann das Modell gegenüber seinen erwarteten Eigenschaften getestet werden. Die dabei anfallende Informationsmenge kann speziell bei detallierten Modellen von verteilten Systemen sehr groß sein. Eine wesentliche Anforderung ist daher eine flexible **Filterung** und **Aggregierung** der Systemausgaben. Ferner soll nach einer Inspektion des Modellzustands die Möglichkeit bestehen, den Systemzustand zu ändern und die Simulation z.B. mit veränderter Markierung oder mit neuen Stopregeln weiterzuführen.
Im Anhang C ist eine detaillierte Beispielsitzung dokumentiert, die insbesondere das Arbeiten mit den interaktiven DAEMON-Schnittstellen erläutert.

F+E.: Folgende Funktionen können über Menüs angesteuert werden:

(1) **Schnappschuß**-Ausgabe des 'Modellzustands', d.h. der aktuellen Markierung, auf Terminal oder Protokolldatei. Es können Kanalinhalte inspiziert werden, die durch eine Menge von Kanalnamen oder durch ein Marken-Attribut und -Wert selektiert werden. Die letztere Möglichkeit ist speziell dann sinnvoll, wenn die aktuellen (Warte-)Zustände einer Transaktion mit allen ihren Subtransaktionen ermittelt werden sollen. Die Ausgabe von Markeninhalt pro Kanal kann auf einen Maximalwert begrenzt

werden.

(2) Selektive Ausgabe der bisher erreichten netz-, ressourcen- und transaktions-bezogenen Performance-Resultate (statistische Ergebnisanalyse, s.u.)

(3) Beobachtung des Simulationsablaufs durch **Traces** des EA-Schaltverhaltens von Instanzen, bis zum nächsten spezifizierten Breakpoint oder dem Erreichen einer anderen Stopregel. Der Benutzer greift dabei nicht in den Simulationsablauf ein, sondern beobachtet die vom Simulator ausgeführte Instanz-Schaltfolge, die einem Zweig im Erreichbarkeitsgraph des Netzes entspricht. Es bestehen folgende Selektionsmöglichkeiten:

- Eine **statische Beobachtung** des Schaltens einer Menge gegebener Instanzen, z.B. aller Instanzen eines Prozesses oder Rechnerknotens oder aller Instanzen, die zu einer bestimmten phys. Ressource gehören. Mit diesem Trace-Typ kann die Korrektheit der realisierten Instanztypen durch Beobachtung ihres EA-Verhaltens überprüft werden.
- **Eine dynamische Beobachtung** des Schaltens derjenigen Instanzen, die als Schalt-Input oder -Output eine strukturierte Marke erhalten, die zu einer spezifizierten Transaktion gehört. (Das setzt voraus, daß ein Markentyp, der Transaktionen und deren Unteraufträge repräsentiert, dem System bekannt sein muß.) Mit diesem Trace-Typ kann die Korrektheit der realisierten Protokolle und Dienste durch Verfolgung einer Transaktion von ihrer Ankunft bis zur Terminierung überprüft werden. Um die Aktivierung bereits vorher überprüfter Instanzen, die zu niederen Systemschichten gehören, nicht wiederholt ansehen zu müssen, kann eine (modellabhängige) Negativliste nicht zu tracender Instanzen spezifiziert werden, die den Transaktionstrace auf die wesentlichen Informationen reduziert. Durch eine geeignete Negativliste kann damit genau der dynamische Ablauf eines **Schichtenprotokolls** beobachtet werden.

  Zur Verdeutlichung zeigt Abb. 6.20 das Verhalten der beobachteten Instanz TGEN15 aus dem Beispielmodell von Abb. 6.12, das durch einen Trace dieser Instanz produziert wurde.

```
-------------------------------------------------------------------------
TGEN-agency 'TM15'  resource CPU_5  worktime  0.2000  ended at  28.727
-------------------------------------------------------------------------

 IN-CHANNEL   P15
 IN-TOKEN    <5  >

 IN-CHANNEL   MB25
 IN-TOKEN    < TRANSACTION-ID: 50000   ARRIVED AT    0.00

                              TRANSACTION MANAGER  DATA MANAGER
                RESTART-STATE:     open / blocked       open / blocked
                RESTART-MAILBOX:   --                   --

                MESSAGE-TYPE: --
                                        SITE(S)       MAILBOX  DATA
                MESSAGE-SOURCE:         5               --     --
                MESSAGE-DESTINATION(S): 5                2     -- >

OUT-CHANNEL   P25 SND5
OUT-TOKEN    < TRANSACTION-ID: 50026   ARRIVED AT   28.53

                              TRANSACTION MANAGER  DATA MANAGER
                RESTART-STATE:     open / blocked       open / blocked
                RESTART-MAILBOX:   --                   --

                MESSAGE-TYPE: --
                                        SITE(S)       MAILBOX  DATA
                MESSAGE-SOURCE:         5               --     --
                MESSAGE-DESTINATION(S): 2 4 7            5     -- >
```

Abb. 6.20: Traceausgabe der Instanz TGEN15 des Beispielmodells

(4) Ändern der aktuellen Markierung

(5) Setzen, Ändern und Löschen von Stopregeln, insbesondere der Simulationsendzeit, der Anzahl terminierter Transaktionen, sowie dem Zeitintervall bis zum nächsten **Breakpoint**, d.h. bis zur Reaktivierung des Monitors, nachdem die Kontrolle auf den Simulator übergeben wird. Die beschriebenen Optionen zur Steuerung eines Simulationslaufs können über das in Abb. 6.21 gezeigte Untermenü ausgewählt werden. Zusätzlich wird die aktuelle virtuelle Simulationszeit und die Anzahl bis dahin terminierter Transaktionen angezeigt.

(6) Übergabe der Kontrolle an den Simulator.

ERF.: Die interaktive Inspektion von Modellzuständen, Leistungsdaten und dynamischem Schaltverhalten hat sich speziell bei komplexen Modellen mit hoher Nebenläufigkeit als sehr nützlich erwiesen.

```
                S I M U L A T I O N   C O N D I T I O N S :

-------------------------------------------------------------------------
SIMULATION-TIME          (CURRENT)        :      7.66
SIMULATION-TIME          (MAXIMAL)        :   9999.90

STARTED    TRANSACTIONS (CURRENT NUMBER):         51
TERMINATED TRANSACTIONS (CURRENT NUMBER):         36
TERMINATED TRANSACTIONS (MINIMAL NUMBER):        500
TRANSIENT  TRANSACTIONS (MINIMAL NUMBER):        120

 CHANGE     CHANGE     CHANGE     SET SCHEDULER   RETURN     CONTINUE
SIM-TIME  T-MINIMAL T-TRANSIENT  FCFS   SJF     MAIN-MENUE  SIMULATION
  <T>        <M>        <N>      <F>   <J>        <R>         <S>
```

Abb. 6.21: Anzeige und Änderung der Simulationsbedingungen (Beispiel)

Zahlreiche Entwurfs- und Modellierungsfehler der VDBS-Protokolle konnten damit schnell entdeckt und beseitigt werden. Insbesondere konnte durch Anwendung des Transaktionstraces ein **tieferes Verständnis** der Modell- und Systemdynamik gewonnen werden, das von statischen Strukturbeschreibungen und von Endresultaten einer Simulation kaum zu erzielen gewesen wäre.

6.5.6 Statistische Ergebnisanalyse

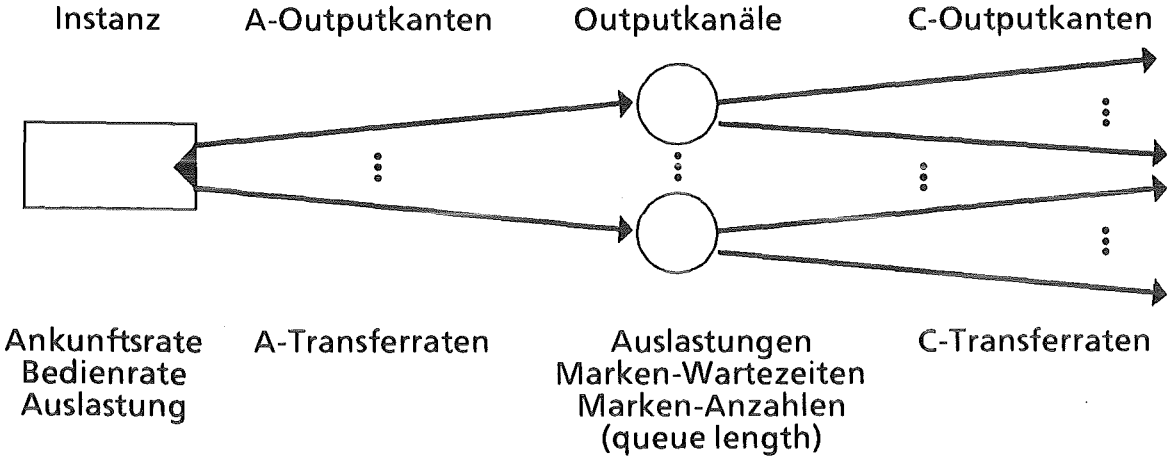
AUFG.: Automatische Leistungs- und Zuverlässigkeitsschätzung

F+E.: Das Subsystem schätzt Leistungsgrößen nach den in Kap. 5.1 und 5.2 beschriebenen Methoden und gibt sie in tabellarischer Form bei Simulationsende oder als Zwischenresultat im Rahmen des Netzdebugging auf Terminal bzw. in eine modellspezifische Ausgabedatei aus. Die ermittelten Größen basieren auf drei Klassen von Beobachtungen:

(1) **Netzbezogene Beobachtungen:**

Instanz-Bedienraten (Feuerungsraten), Kanal-Auslastungen,

Markenwartezeiten und -Anzahlen in Kanälen (wichtig zur Pufferdimensionierung und zur Engpaßanalyse), sowie Markentransferraten pro Outputkanal einer Instanz und pro Kanal mit >1 Outputkante:



A- und C- **Transferraten** sind ein Maß für das partielle Ausgabeschaltverhalten einer Instanz bzw. für die Konfliktauflösungsstrategie (vergl. Definitionen in Kap. 6.4). Außer diesen Raten werden alle Größen als empirische Statistiken wie Mittelwerte, Extremwerte, Varianzen bereits im FUN-System ermittelt. Die interaktiven Ausgabemöglichkeiten des DAEMON-Systems verdeutlicht Abb. 6.22.

(2) **Ressourcenbezogene Beobachtungen:**

Ermittelt und ausgegeben werden Auslastungen, Bedienraten, sowie Ankunftsraten vor einer Ressource. Als Folge der expliziten Ressourcenspezifikation der DAEMON-Netzsprache ist die Bestimmung dieser Größen auf einfache Weise möglich. Der Performance-Monitor zeigt die genannten Größen auf Anforderung entsprechend Abb. 6.23 an.

(3) **Transaktionsbezogene Beobachtungen:**

Beobachtungen werden von Instanzen unter Anwendung spezieller Auswertungsprozeduren (s. Instanztyp-Realisierung) sowie von Transaktionen repräsentierenden Marken (s. Beispielmodell Kap. 6.3) gesammelt und nach den im Kapitel 5.2 beschriebenen Methoden ausgewertet. Der detaillierte, zum Abschluß eines Simulationslaufs erstellte Ergebnisreport enthält folgende Größen:

- Anzahl erfolgreich beendeter Transaktionen; Erfolgsrate
- Anzahl zur Minimierung transienter Effekte eliminierter

A G E N C Y P E R F O R M A N C E

TYPE :	TGEN	NAME :	TM15	RESOURCE: CPU_5
LAST-WORKTIME :	0.200	MEAN-WORKTIME :	0.200	
NO. OF ACTIVATIONS:	9	UTILIZATION (%):	23.49	

IO-CHANNELS

TYPE	NAME	CAPACITY	ACT_SIZE	MEAN_SIZE	MEAN_DELAY	%-UTILIZATION
<--	P15	10	7	8.64	4.33	56.25
<--	MB25	40	0	0.06	0.05	100.00
-->	P25	10	3	1.22	0.07	88.00
-->	SND5	50	0	0.00	0.00	100.00
-->	P15	10	7	8.64	4.33	56.25

A G E N C Y P E R F O R M A N C E

AGENCY	TYPE	RESOURCE	ACTIVATIONS	AVG.SERVICE-TIME
T1	TTERM	TERM_1	9	0.843
T2	TTERM	TERM_2	7	1.072
T3	TTERM	TERM_3	8	0.956
T4	TTERM	TERM_4	9	0.754
T5	TTERM	TERM_5	10	0.730
T6	TTERM	TERM_6	8	0.922
T7	TTERM	TERM_7	9	0.806
TM11	TGEN	CPU_1	7	0.200
TM12	TGEN	CPU_2	6	0.167
TM13	TGEN	CPU_3	6	0.200
TM14	TGEN	CPU_4	8	0.200
TM15	TGEN	CPU_5	9	0.200

C H A N N E L P E R F O R M A N C E

TYPE	NAME	CAPACITY	INIT_SIZE	MEAN_SIZE	MEAN_DELAY
FIFO	MB11	20	20	18.100	4.014
FIFO	MB12	20	20	17.953	4.406
FIFO	MB13	20	20	18.247	3.892
FIFO	MB14	20	20	16.689	3.763
FIFO	MB15	20	20	17.154	3.839
FIFO	MB16	20	20	17.534	4.190
FIFO	MB17	20	20	16.983	4.180
FIFO	P11	10	10	8.843	4.113
FIFO	P12	10	10	9.250	5.243
FIFO	P13	10	10	8.731	3.923
FIFO	P14	10	10	8.339	4.319

Abb. 6.22: Ausgabe netzbezogener Leistungsgrößen (Beispielmodell):
- detaillierte Instanz-Statistik (ohne Transferraten)
- kummulierte Instanz-Statistik (Ausschnitt)
- kummulierte Kanal-Statistik (Ausschnitt)

R E S O U R C E P E R F O R M A N C E

Name	Utilization %	Service Time avg	Arrival Times avg	Activities No.
CPU_1	48.15	0.066	0.136	56
CPU_2	35.37	0.059	0.163	46
CPU_3	48.42	0.074	0.152	50
CPU_4	65.38	0.073	0.111	69
CPU_5	63.81	0.072	0.112	68
CPU_6	52.85	0.070	0.129	58
CPU_7	59.51	0.068	0.113	67
DISK_1	18.76	0.063	0.331	23
DISK_2	21.58	0.066	0.306	25
DISK_3	11.82	0.060	0.510	15
DISK_4	15.65	0.067	0.425	18
DISK_5	19.88	0.069	0.346	22
DISK_6	14.75	0.063	0.415	18
DISK_7	21.39	0.071	0.323	23
NIU_1	21.33	0.017	0.081	94
NIU_2	19.49	0.018	0.095	81
NIU_3	18.39	0.019	0.105	73
NIU_4	22.87	0.018	0.079	96
NIU_5	23.75	0.018	0.074	103
NIU_6	19.83	0.019	0.093	82
NIU_7	23.12	0.018	0.075	100
TERM_1	99.03	0.949	0.949	8
TERM_2	97.90	1.250	1.250	6
TERM_3	99.79	1.092	1.092	7
TERM_4	88.53	0.848	0.848	8

Abb. 6.23: Ausgabe ressourcenbezogener Leistungsgrößen (Beispielmodell)

Transaktionen

Für alle erfolgreichen, nichteliminierten Transaktionen:

- mittlere Anzahl von Unteraufträgen pro Transaktion,
- Durchsatz in [Transaktionen / sec]
- Ankunftsrate in [Transaktionen / sec]

Antwortzeiten für jeden "Dienst" (d.h. Submodell, für das separate Antwortzeiten erwünscht sind, im Sinne von Kap. 5.2):

- Mittelwerte mit 95%-Konfidenzintervallen und Standardabweichungen (empirisch sowie nach der Batch-Means Methode)
- Empirische Verteilungsfunktion der Antwortzeiten

Die Transaktionsanalyse hat als Parameter:

- Anzahl terminierter Transaktionen (meist als Stopregel der

Simulation vorgegeben)

- Anzahl zu eliminierender Transaktionen
- Anzahl Dienste zur Antwortzeiten-Ermittlung
- Blockgröße für Batch-Means Methode
- Zellenanzahl für Histogramm-Ausgabe

Die interaktive Version ermittelt einen Teil dieser Größen, wie in Abb. 6.24 demonstriert wird. Es besteht die Möglichkeit, einen beliebigen Auschnitt der beendeten Transaktionen zu inspizieren.

```
    51 TRANSACTIONS have been started,
    36 TRANSACTIONS    have   terminated.

  give an OBSERVATION WINDOW <from, to> :
10
30
        T R A N S A C T I O N   P E R F O R M A N C E
-------------------------------------------------------------------
TRANSACTION WINDOW:       10      30
   TIME     WINDOW:     1.96    6.56

NODE   NO.STARTED   NO.STARTED    THRUPUT     RESPONSE
                     & ENDED       RATES       TIMES
==================================================
  1          3          4         0.870       0.773
  2          4          4         0.870       0.904
  3          4          3         0.653       0.825
  4          4          2         0.435       1.560
  5          4          3         0.653       1.009
  6          4          4         0.870       0.959
  7          6          3         0.653       0.959
--------------------------------------------------
TOTAL :     29          23        5.004


              R E S P O N S E   T I M E S
-------------------------------------------------------------------
SERVICE  MINIMUM  MAXIMUM   MEAN  95%-Confidence  ST. DEVIATION
   1       0.31     1.63    0.96       +-  0.12          0.33
```

Abb. 6.24: Anzeige transaktionsbezogener Statistiken (Beispielmodell)

ERF.: Konfidenzintervalle sowie die Anwendung der Batch-Means Prozedur wurden lediglich für Transaktions-Antwortzeiten als im Rahmen dieser

Arbeit wichtigsten VDBS-Zielgröße realisiert.
In der realisierten Version werden Auswertungsdaten mehrerer Experimente konsekutiv in einer modellspezifischen Ausgabedatei abgelegt. Wünschenswert wäre eine zukünftige rechnergestützte graphische Ausgabe der jeweiligen Mittelwerte, Konfidenzintervalle und Varianzen aller Outputgrößen bei verschiedenen Experimenten, mehrdimensionale Varianzanalyse u.ä. Im Rahmen der vorliegenden Arbeit wurden die Ergebnisse mehrerer Versuche dem GDDM zur graphischen Ausgabe übergeben (s. Kap. 8).

6.6 Zusammenfassung

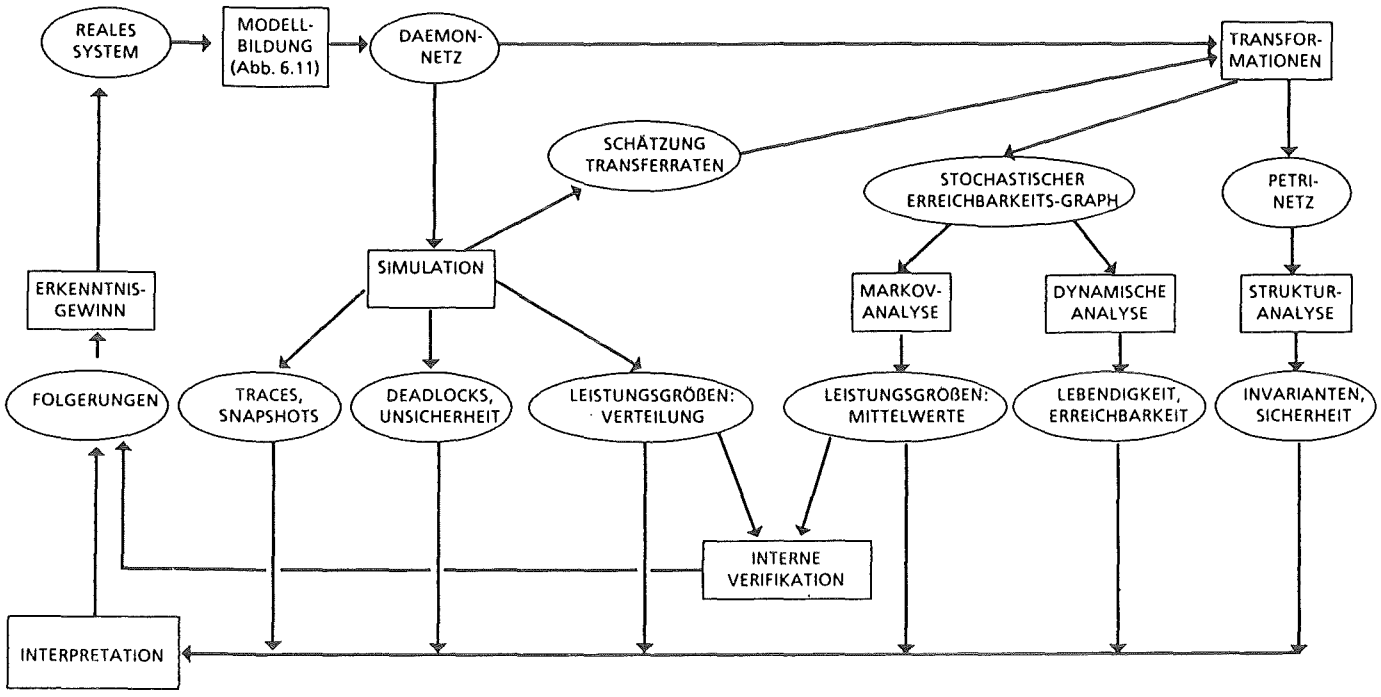


Abb. 6.25: Modellierung mit DAEMON-Netzen

Die DAEMON-Netzsprache und das rechnergestützte DAEMON-Werkzeug bieten einen integrierten Ansatz zur Spezifikation, Test, Analyse und Simulation des Operations- und Ausfallverhalten von komplexen Hardware/Software-Systemen.

Die Sprachkonstrukte gestatten eine flexible, kompakte und gut strukturierte Modellierung aller Subsysteme eines Rechensystems, speziell von fehlertoleranten Kommunikationssystemen mit stochastischem Verhalten.
Das DAEMON-System bietet eine graphische Schnittstelle zur Modellspezifikation. Simulationsexperimente können interaktiv überwacht und die Modelldynamik getestet werden. Neben der Simulation aller DAEMON-Netze kann eine große Unterklasse dieser Netze nach Ausführung größtenteils automatisierbarer Transformationen sowohl nach ihren netztheoretischen als auch nach ihren zeitlichen Eigenschaften formal analysiert werden. Es wurde dabei für einen integrierten Einsatz von simulativen und verschiedenen analytischen Methoden bei der Untersuchung komplexer, stochastischer Systeme plädiert.
Die beschriebenen Transformationen erhalten einerseits diejenigen Eigenschaften, die bereits für Netze ohne Zeitverbrauch gelten, d.h. sie bewahren 'weiche' Netzeigenschaften. Andererseits bewahren sie zeitliche Eigenschaften in dem Sinne, daß die Mittelwerte wesentlicher Leistungsgrößen beim ursprünglichen und transformierten Netz annähernd übereinstimmen. Eine gewisse Unsicherheit bleibt bei der kritischen Abbildung von markenabhängigem auf -unabhängiges Schaltverhalten, die nur manuell oder mittels Simulationsresultaten erfolgen kann. Andererseits können durch Simulation erhaltene Leistungswerte gerade durch die Ergebnisse der Leistungsanalyse intern validiert werden; zur Bestimmung semantischer Modelleigenschaften wie Sicherheit, Lebendigkeit und Reproduzierbarkeit stellt das vorgeschlagene Instrumentarium die strukturelle, netztheoretische und die dynamische, erreichbarkeitsanalytische Analyse zur Verfügung. Zum Teil kann auch die Simulation Aussagen über Verklemmungen und unsicheres Netzverhalten liefern.
Die vorgeschlagenen Untersuchungsmethoden sind in Abb. 6.25 mit ihren wechselseitigen Abhängigkeiten zusammengefaßt.

7 MODELLIERUNG VON INTEGRIERTEN TRANSAKTIONS-VERARBEITUNGSPROTOKOLLEN MIT DAEMON-NETZEN

Die in Kapitel 6 definierten DAEMON-Netze ermöglichen es, komplexe HW-/SW-Systeme, wie etwa Kommunikationsprotokolle und ihre Umgebung, zu spezifizieren und daraus mit Hilfe des DAEMON-Simulators eine automatische Performance-Vorhersage zu erhalten.
Im vorliegenden Kapitel werden zunächst auf Grundlage der in Kap. 3 erarbeiteten Anforderungen geeignete Protokolle zur zuverlässigen und leistungsfähigen Ausführung von Transaktionen auf teilredundant verteilten Datenbeständen ausgewählt (in 7.1). Diese Protokolle benutzen zur Ausführung einer verteilten Transaktion einen sicheren Prozeßkommunikations-Dienst (Kap. 4.4), sowie lokale Dienste zur Fehlerbehandlung (Recovery- und Restart-Komponente, in 7.3 beschrieben). Nach einem Überblick über die Schnittstellen zur Protokollumgebung in Kap. 7.2 werden die ausgewählten Protokolle in 7.4 und 7.5 mit Hilfe der DAEMON-Netze funktional und in ihrem Zeitverhalten spezifiziert. Dabei werden die einzelnen Phasen der Transaktionsverarbeitung sowohl im Normalfall als auch nach Ausfällen beschrieben. Abschließend wird (in 7.6) ein einfaches analytisches Modell der Transaktions-Antwortzeit für die betrachteten Protokolle entwickelt, um diese mit Simulationsergebnissen vergleichen zu können.

7.1 Protokolle zum zuverlässigen und effizienten Zugriff auf verteilte Datenbestände

Gemäß Kap. 3 müssen geeignete Protokolle zur Verwaltung verteilter Datenbestände folgende Anforderungen zur Erhaltung der operationalen Datenkonsistenz erfüllen:

- **Serialisierbarkeit** aller nebenläufiger Datenzugriffe
- **Atomarität** aller Datenbank-Änderungen jeder Transaktion
- **Dauerhaftigkeit** aller Datenbank-Änderungen jeder Transaktion
- **Gegenseitige Konsistenz** aller Kopien einer Dateneinheit

Über diese Korrektheitsanforderungen hinaus, die trotz **beliebiger** globaler Störungen des VDBS-Betriebs wie Knoten- oder Leitungsausfall

zu erfüllen sind, haben wir in Kap. 1 zusätzliche Qualitätsanforderungen gestellt:

- Der Datenbankbetrieb soll auch bei Teilausfall des Systems zumindest eingeschränkt verfügbar sein (**Verfügbarkeit**). Insbesondere ist der Ausfall mindestens einer HW-Komponente zu tolerieren.
- Die **Leistungsfähigkeit** des verteilten Datenbanksystems soll mindestens so hoch sein wie die eines Systems mit vergleichbar großer zentralisierter Datenbank.

In den Kapiteln 3 und 4 haben wir durch Annahme geeigneter **Umgebungsbedingungen** und die Spezifikation **sicherer Software-Subsysteme** für Prozeßkommunikation und Speicherzugriffe die Voraussetzungen zum zuverlässigen und hochverfügbaren Zugriff auf verteilte Datenbestände erarbeitet:

- Störungen des DB-Betriebs werden allein durch Hardware-Ausfälle verursacht. Solche Ausfälle sind 'gutartig' und werden stets erkannt.
- Die IPC-Schicht erkennt und meldet diese Störungen zuverlässig an höhere (VDBS-)Prozesse. Der Nachrichtenaustausch ist sicher.
- Das Hintergrundspeicher-System fällt selbst nie aus und gestattet eine atomare Ausführung von Speicheroperationen, sodaß alle Änderungen der Datenbasis dauerhaft sind.
- Alle Datenzugriffe werden durch das VDBS kontrolliert, das sich dabei der sicheren Dienste der IPC-Schicht und des Speichersystems bedient.
- Die Kontrolle sowohl des Rechnernetz-Betriebs als auch der Datenbankzugriffe ist verteilt, sodaß die Auswirkungen eines Einzelausfalls begrenzt werden können.

Ferner wurde eine Annahme gemacht, die einen effizienten Datenzugriff unterstützen soll:

- Daten sind unter Ausnutzung von Kopien 'fast optimal' im Rechnernetz verteilt, sodaß die meisten Zugriffe lokal (am Heimatknoten der Transaktion) erfolgen können; bei nichtlokalen Zugriffen ist kein aufwendiger Datenaustausch zur Ergebnisermittlung notwendig.

Diese Maßnahmen allein garantieren noch nicht totale Zuverlässigkeit, hohe Verfügbarkeit und gute Performance bei Datenzugriffen. Wesentlich ist auch, daß logisch zusammengehörige Operationen auf verteilte Daten zu **Transaktionen** zusammengefaßt werden, deren Verarbeitung durch geeignete **Protokolle** überwacht wird. Der Entwurf von Protokollen zur Transaktionsverarbeitung (TVP) soll folgender **Anforderung** genügen:

Ein Protokoll soll **robust** sein gegen möglichst viele Fehlerfälle, d.h. daß in allen Protokollzuständen

-- die operationale Datenkonsistenz auch bei Ausfällen gewahrt bleibt.

-- die Auswirkungen von Teilausfällen begrenzt werden, sodaß möglichst viele Transaktionen erfolgreich terminieren.

Bei den in unserem Modell berücksichtigten HW-Teilausfällen kann die (partielle) Korrektheit bei beliebigen Ausfällen durch eine geeignete Concurrency Control Methode zusammen mit einem Commit-Protokoll garantiert werden. Hingegen ist Terminierung ein Problem, da es kein Terminierungsprotokoll geben kann, das gleichzeitig Atomarität und Terminierung verteilter Update-Transaktionen unabhängig von Reparaturzeiten ausgefallener Teilsysteme garantiert, wenn beliebige Netzpartitionierungen vorkommen können /SKEE81/.
Trotzdem kann durch geeignete Strategien beim Protokollentwurf die Wahrscheinlichkeit minimiert werden, daß beliebig lange Blockierungen der Beendigungsentscheidung einer Update-Transaktion vorkommen, und daß als Folge deren blockierte Daten auch zu Wartezeiten für andere Transaktionen führen:

- Durch das 2-Phasen-Commitprotokoll wird die Zeitdauer innerhalb der gesamten Verarbeitung einer Transaktion, in der Blockierungen möglich sind ('Unsicherheits-Fenster'), auf die zweite Commitphase beschränkt.
- Wenn abgesetzte Protokollinstanzen ausschließlich über den sicheren IPC-Dienst kommunizieren, wird eine gegenseitige Nichtverfügbarkeit stets in beschränkter Zeit erkannt. Transaktionen, die die zweite Commitphase noch nicht erreicht haben, können daher schnell beendet werden (wenn auch durch Zurücksetzen).
- Auch die Blockierung lokaler Daten kann dadurch im Protokoll erkannt und so behandelt werden, daß Transaktionen auf diesen Daten

zurückgesetzt werden, anstatt eine ungewisse Zeitdauer auf die Reparatur der Blockierungsursache zu warten.

- Durch Erweiterungen des klassischen Commitprotokolls /GRAY78/ können Netzpartitionierungen schnell erkannt werden. In manchen Fällen kann durch Kooperation der gegenseitig verfügbaren Partnerknoten die Blockierung von Daten aufgehoben werden /BRLE82a, LEBR82/.

Nach der Festlegung der Anforderungen ist die Eignung existierender Protokolle zu untersuchen. Im VDBS-Bereich sind sehr viele Protokolle publiziert worden. Eine umfassende Klassifikation und Bewertung steht noch aus und ist aus verschiedenen Gründen sehr schwierig:

- Viele Arbeiten beschränken sich auf Teilaspekte, wie Anfrageauswertung, Concurrency Control, Kopienbehandlung oder Commit-Protokolle, ohne deren Interaktionen zu berücksichtigen. Maßnahmen zur Ausfallsicherung sind in den Protokollen oft nicht vorgesehen; der Protokollablauf nach realistischen Fehlerfällen wie Netzpartitionierung kann zu inkonsistenter Datenbasis führen.
- Die große Komplexität der zu betrachtenden Ausfall- und Protokoll-Zustände und eine oft nur verbale, z.T. unvollständige Beschreibung erschweren den Korrektheits-Nachweis. Viele Protokolle mit verteilter Kontrolle haben sich im nachhinein als inkorrekt erwiesen.

In Abb. 7.1 wurde der Versuch einer Teil-Klassifikation bekannter Concurrency-Control- und Kopienbehandlungs-Verfahren mit Zuordnung darauf basierender VDBS-Realisierungen gemacht. Eine umfassendere Klassifikation speziell von Zeitstempel-Verfahren findet sich in /BAYE84/ und /BERN81/. Bei den Klassen der Sperr- und Zeitstempel-Verfahren kann weiter in solche mit zentralem und verteilten Scheduler unterschieden werden (nicht in Abb. 7.1). Wir betrachten hier nur die echt verteilten Verfahren. Ferner kann zwischen "pessimistischen" und "optimistischen" Verfahren unterschieden werden /SINH85, WILH84/.
Im Rahmen vorliegender Arbeit werden nur pessimistische (verteilte Sperr-)verfahren modelliert. Bei diesen Verfahren gibt es folgende Optionen zur Lösung des **Verklemmungsproblems** (s. Abb. 7.1):

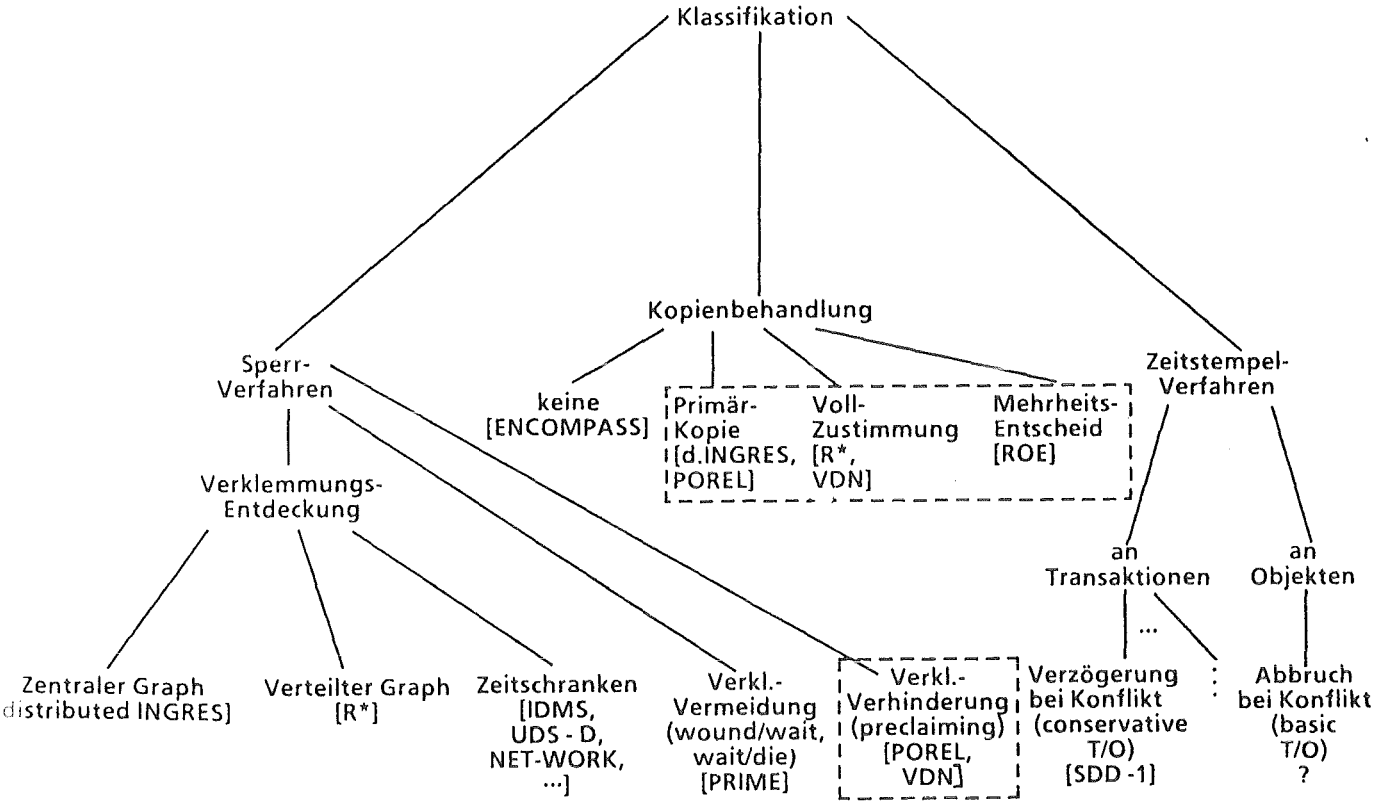


Abb. 7.1: (Teil-)Klassifikation von verteilten Concurrency Control- und Kopienbehandlungs-Verfahren

modellierte Klasse von Verfahren

- Bei statischen Sperrverfahren werden alle Sperren zu Beginn der Transaktion angefordert. Es wird eine totale Ordnung aller Anforderungen erzwungen, sodaß zyklische Anforderungen a priori verhindert werden. Mit dieser Sperr-Preclaiming-Strategie werden Verklemmungen verhindert. Realisierungen finden sich z.B. in den Systemen POREL /WANE84/ und VDN/REFLEX /GMSS81/.
- Alternative zur Verklemmungsverhinderung ist z.B. die Strategie 'Wait-Die' (bzw. 'Wound-Wait'). Dabei wird bei Entdeckung irgendeines Sperrkonflikts zwischen zwei Transaktionen entweder die 'jüngere' (bzw. die 'ältere') zurückgesetzt (Transaction Rollback) und muß, ggf. mehrmals, neu gestartet werden /BERN81/. Das Alter einer Transaktion in einem verteilten System kann z.B. durch netzweit eindeutige Identifikationen definiert werden. Dieses Verfahren wurde in einem VDBS von PRIME Computers realisiert

/DUBO82/.

Während bei Preclaiming-Verfahren Transaktionen in der Sperrphase längere Zeit blockiert sein können, werden bei den anderen Strategien Transaktionen evtl. unnötig und sogar mehrmals abgebrochen und neu gestartet.

- Bei Verklemmungsentdeckungs-Verfahren wird z.B. ein globaler Graph der Wartebeziehungen zwischen konkurrierenden Transaktionen zentral verwaltet und periodisch mit Hilfe von Botschaftenaustausch aktualisiert, um Zyklen zu entdecken und durch Abbruch einer der beteiligten Transaktionen aufzulösen. Dieses Verfahren findet sich in "Distributed INGRES" /STON79/. Alternativen sind die Verwaltung eines verteilten Graphen, wie in R* /LIND83/, oder das Setzen von Zeitschranken, wie in einigen in /ESPO82/ untersuchten VDBS. Das letztere Verfahren ist zwar einfach zu implementieren, bricht aber u.U. auch Transaktionen ab, die nicht an einer Verklemmung beteiligt sind.

Nachfolgend wird die Klasse der **verteilten Zwei-Phasen-Sperrprotokolle** mit integrierter Kopien- und Fehlerbehandlung betrachtet. Diese Klasse ist dadurch charakterisiert, daß

- Die Korrektheit der Serialisierung von Datenzugriffen durch Sperrprotokolle formal nachgewiesen ist /BERN83b/,
- Das Sperrkonzept, verglichen mit alternativen Konzepten zur Datensynchronisation, einfach zu modellieren und um Maßnahmen zur Ausfallbehandlung (crash recovery) zu erweitern ist.
- Viele Implementierungen und Erfahrungen in realen VDBS existieren, sodaß unsere Untersuchungsergebnisse im nachhinein validiert werden können.

Um die oben geforderte Teilverfügbarkeit des transaktionsverarbeitenden Systems bei einem Ausfall zumindest potentiell zu gewährleisten, werden Daten redundant im Rechnernetz als Kopien verteilt. Es gibt im wesentlichen drei Unterklassen verteilter Sperrprotokolle auf redundanten Daten (s. Kap. 3.2.3):

- Vollzustimmungs-Protokoll PROT_ALL, realisiert in den Systemen R* und VDN/REFLEX;
- Primärkopien-Protokoll PROT_PRIM, realisiert in den VDBS POREL und INGRES;

- Mehrheitsentscheid-Protokoll PROT_MAJ, realisiert im verteilten Dateisystem ROE /ELFL83/.

Diese Protokolle basieren alle auf einem verteilten 2-Phasensperren und auf einem verteilten 2-Phasen-Commitprotokoll, unter Berücksichtigung beliebiger Arbeitsrechner- und Leitungs-Ausfälle zur Atomaritäts- und Konsistenzsicherung. Dabei erfolgt eine Beschränkung auf das Preclaiming-Verfahren (Verklemmungsverhinderung).
Die folgende **Modellierung** und **Simulation** mittels DAEMON-Netzen zielt auf die formale Beschreibung und auf einen Leistungsvergleich dieser Protokolle ab. Vergleiche mit anderen Concurrency-Control Verfahren wären auch interessant, müssen aber späteren Untersuchungen vorbehalten bleiben.

VDBS-Modelle mit höheren Petrinetzen als Spezifikationsmittel wurden erstellt u.a. von /BAGA81/ (2-Phasen-Commitprotokoll mit Crash-Behandlung), /BORO82/ (Beschreibung der Protokolle des VDBS SIRIUS-DELTA), /WLS83/ (Beschreibung der Protokolle des VDBS R*). Eine (nach Kenntnis des Autors einzige) simulative VDBS-Untersuchung mit Netzen ist in /OZSU83/ beschrieben.

7.2 Überblick über das Protokollmodell

Ein Protokoll zur Transaktionsverarbeitung (TVP) ist in unserem Modell durch die Funktionalität und die Interaktionen zwischen den TM- und DM-Prozessen (Transaktions- und Daten-Managern) bestimmt. Ausgehend von dem Umgebungsmodell von Kap. 4, kommunizieren diese Prozesse mit Hilfe der sicheren IPC-Dienste. Neben der 'horizontalen' Kommunikation des TM/DM-Schichtenprotokolls werden 'vertikal' auch lokale Dienste wie Daten- und Katalogverwaltung sowie Transaktions-Logging beansprucht. In Abb. 4.3 wurde ein Überblick über diese Protokollstruktur gegeben.
Die ausgewählten Protokolle unterscheiden sich im wesentlichen lediglich in der Kopienbehandlung. Bei einer partitionierten, d.h. redundanzfreien Datenbasis sind alle drei Protokolle funktional identisch.
Auf der Basis dieser Beobachtungen wird im folgenden ein **parametrisiertes VDBS-Protokollmodell** als DAEMON-Netz entworfen. Neben den Simulationsparametern der Protokollumgebung ist das jeweilige TVP

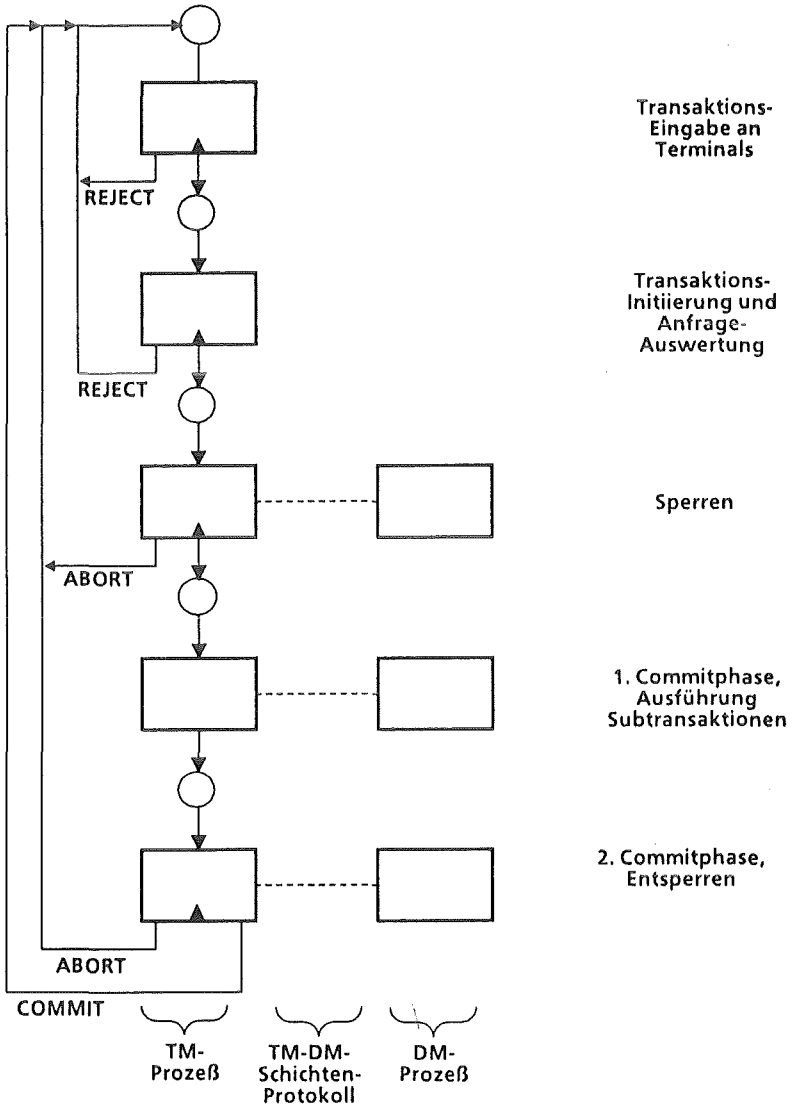


Abb. 7.2: Überblick über die Struktur der Transaktions-Verarbeitungsprotokolle

ebenfalls als Parameter aufzufassen. Es wird daher folgendes **Dokumentationsschema** der Protokolle angewendet. Das Modell wird (in Kap. 7.4) durch das Protokoll auf einer partitionierter Datenbasis beschrieben. Es wird dabei zwischen den Protokoll-Phasen Anfrage-Auswertung, Sperren sowie Transaktions-Ausführung und Terminierung unterschieden. Für jede Phase werden die Protokoll-(Netz-)instanzen, die Struktur der auszutauschenden Botschaften sowie der dynamische Protokollablauf im Normalfall und

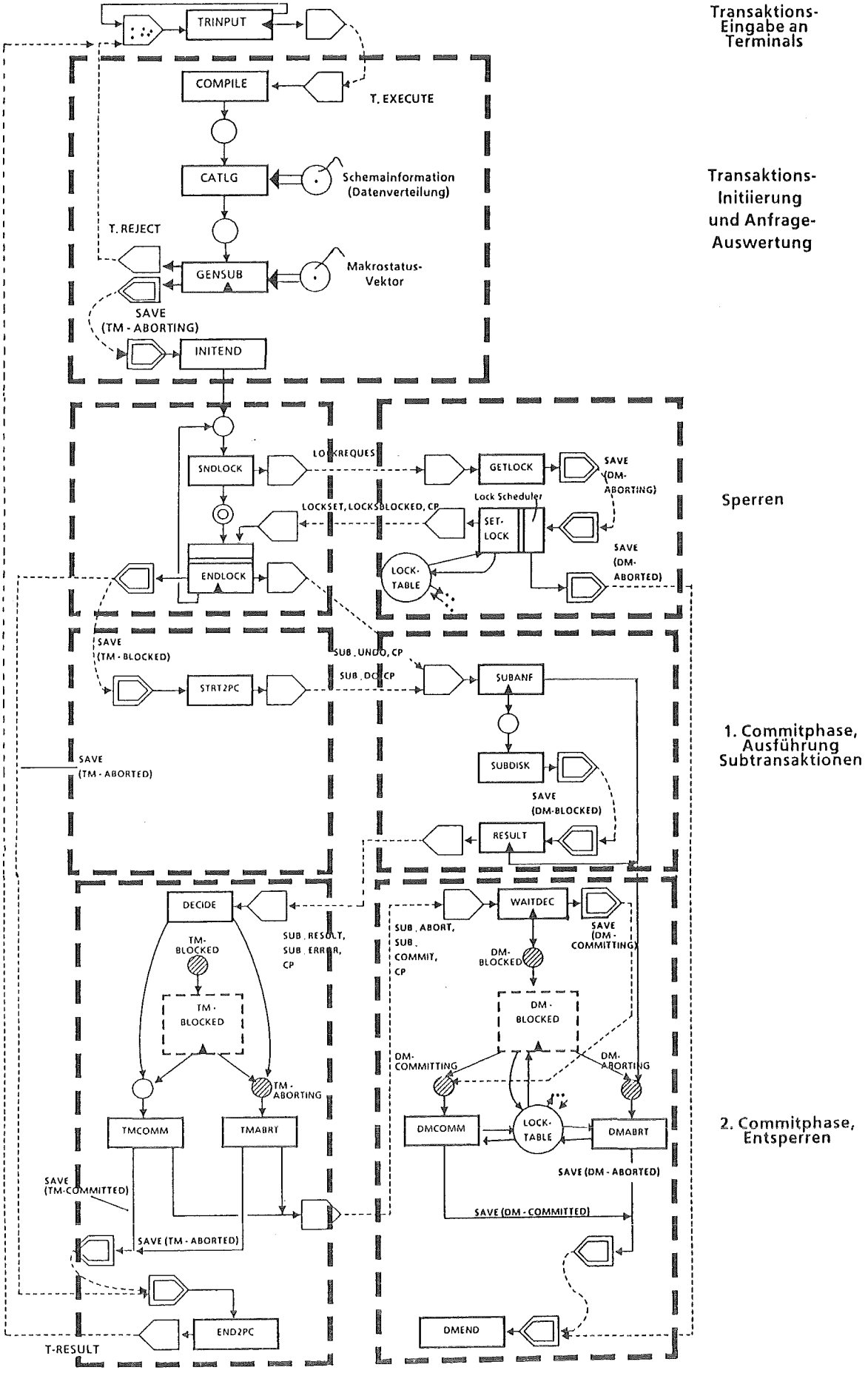


Abb. 7.3: Struktur der Transaktionsverarbeitungs-Protokolle

nach Ausfällen beschrieben. Es folgt (in Kap. 7.5) eine Beschreibung der Protokolle auf (teil-)redundanten Daten.

Eine Überblick über Protokollstruktur und -phasen vermittelt Abb. 7.2. Ein TM-Prozeß als Koordinator der Transaktionsverarbeitung an einem Rechnerknoten kommuniziert dabei mit einer (transaktionsabhängigen) Anzahl $n \geq 1$ von DM-Prozessen. Die Abbildung zeigt auch die möglichen (TM-)Endzustände einer Transaktion in Abhängigkeit von der Bearbeitungsphase (vergl. dazu das VDBS-Auswertungsmodell in Abb. 5.3).

Als Verfeinerung aller in Abb. 7.2 dargestellten Protokollinstanzen ergibt sich in Abb. 7.3 die **Protokoll-Detailstruktur**. Das TVP bestimmt die Abarbeitung einer interaktiven Transaktion ab deren Eingabe an einem Terminal, über die ggf. verteilte Verarbeitung im homogenen Rechnernetz, bis zur Rückmeldung von Ergebnissen an demselben Terminal.
Dieses geschlossene Auftragsmodell (vergl. Beispielmodell in Kap. 6.3) besteht nach Abb. 3.2 und 4.3 aus Sicht des Heimatknotens einer Transaktion aus den Phasen

(P1) Transaktions-Eingabe, -Übersetzung und Anfrage-Auswertung
(P2) Sperrphase
(P3) Transaktions-Ausführungsphase und -Terminierungsphase

In (P1) erfolgt die Transformation der Benutzer-Transaktion auf eine 'physische' Transaktion, d.h. auf eine Menge von Subtransaktionen, die gemeinsam die durch die Transaktion formulierte Anfrage ausführen sollen. (P2) realisiert das Sperren aller dabei erforderlicher Daten nach dem Preclaiming-Verfahren. In (P3) werden die Einzeloperationen der Subtransaktionen ausgeführt, die Sperren aufgehoben und das Endresultat dem Benutzer übermittelt. Diese Schritte werden in das 2-Phasen-Commit-Protokolls (2PC) zur atomaren Transaktionsterminierung integriert.

Die einzelnen Phasen werden durch die statische Struktur und durch den dynamischen Ablauf eines DAEMON-Netzes spezifiziert. Dieses beschreibt das **TM/DM-Schichtenprotokoll**. Instanzen dieses Netzes realisieren sequentielle Prozeß-Schritte eines TM- bzw. DM-Prozesses. Wegen der homogenen HW/SW-Struktur unseres Modells sind diese Instanzen auf allen Knoten des Rechnernetzes vorhanden. TM- und DM-Prozesse bilden

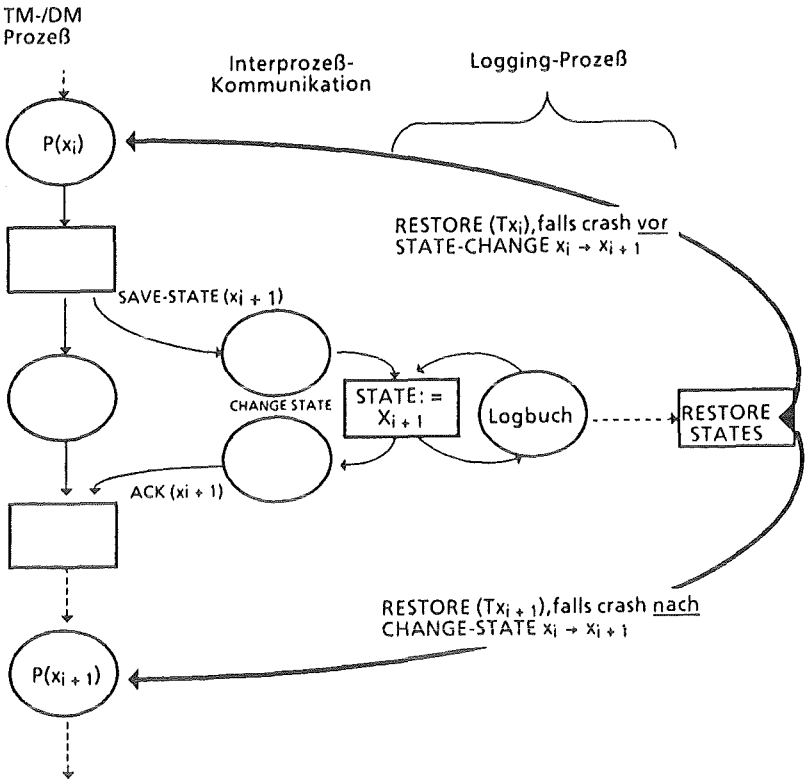
die oberste Stufe der Protokoll-Hierachie unseres Modells. TM und DM sind Dienstprozesse, d.h. diese können nebenläufig durch unterschiedliche Transaktionen aktiviert werden. Kanäle repräsentieren prozeßspezifische Puffer bei der Transaktionsverarbeitung oder Input- und Output-Mailboxes zu den unteren Systemdiensten 'Interprozeß-Kommunikation' und 'Logging'. Mailboxes sind in den folgenden Abbildungen zur Unterscheidung abweichend von der Standard-Notation von Abb. 6.10 dargestellt. Zwischen solchen Mailboxes werden komplexe strukturierte Marken **(Botschaften)** ausgetauscht. Diese, als auch prozeßinterne Information, wird durch einen netzweit gültigen, strukturierten Transaktions-Markentyp repräsentiert, der in Anhang B spezifiziert ist.

Die in Abb. 7.3 spezifizierte Protokollstruktur wird in den nächsten Abschnitten ausführlich erläutert. Vorher werden noch die Dienste zur lokalen Fehlerbehandlung modelliert, die von den Protokollen benutzt werden.

7.3 Modell des lokalen Datenbank-Recovery

Die globale Atomaritätssicherung verteilter Transaktionen nach Rechnerausfall und -Wiederanlauf erfolgt durch ein Protokoll zur Transaktionsverarbeitung. Dieses nimmt hierzu den **Logging-Dienst** der lokalen DBVS in Anspruch. Dieser Dienst ist für fehlertolerantes VDBS-Verhalten von großer Bedeutung. Bei Wiederanlauf eines Arbeitsrechners sind alle offenen Transaktionen, deren letzter Bearbeitungszustand im Logbuch eingetragen wurde **(crashrelevanter Zustand)**, in einen konsistenzsichernden Zustand zu restaurieren. Diese "Zustände" sind durch bestimmte Kanäle innerhalb der Prozesse TM und DM repräsentiert. Somit laufen in unserem Modell lediglich diese zwei Prozesse wieder an, um zum Ausfallzeitpunkt offene (Update-)Transaktionen so zu beenden, daß die Konsistenz der lokalen Datenbasis wieder hergestellt wird, und ggf. Updates auf veralteten Datenkopien 'nachzufahren'.

Das Prinzip des Transaktions-Logging wird in Abb. 7.4 verdeutlicht. Abb. 7.5 zeigt die crashrelevanten Zustände eines TM- bzw. DM-Prozesses sowie das Wiederaufsetzen bei AR-Wiederanlauf: Alle bei



Legende:

x_i:	bereits auf Logbuch gesicherter Zustand der Transaktionsverarbeitung
$P(x_i)$:	Platz, der dem Zustand x_i entspricht
$x_i + 1$:	x_i-Folgezustand
⟶	Transaktionsverarbeitung im Normalfall (Arbeitsrechner AR intakt)
➡	Verarbeitung nach AR - Ausfall/-Wiederanlauf (Ausfall nach x_i -Sicherung im Logbuch)
T_x:	Information, um die Transaktion T im Zustand x nach Wiederanlauf fortzusetzen

Abb. 7.4: Prinzip der Sicherung/Restaurierung crashrelevanter Zustände

AR-Ausfall offenen Transaktionen werden im jeweils zuletzt gespeicherten crashrelevanten Zustand fortgesetzt. Wird dieses Prinzip konsequent auf alle möglichen Zustände der TM- und DM-Prozesse angewendet, dann gibt es nun bei Ausfall eines Rechners nach dessen Wiederanlauf stets einen Übergang zu einem operationalen TM- bzw. DM-'Normalzustand'. Die Existenz solcher Übergänge ist eine notwendige Bedingung zur Eliminierung der durch Rechnerausfall möglicherweise

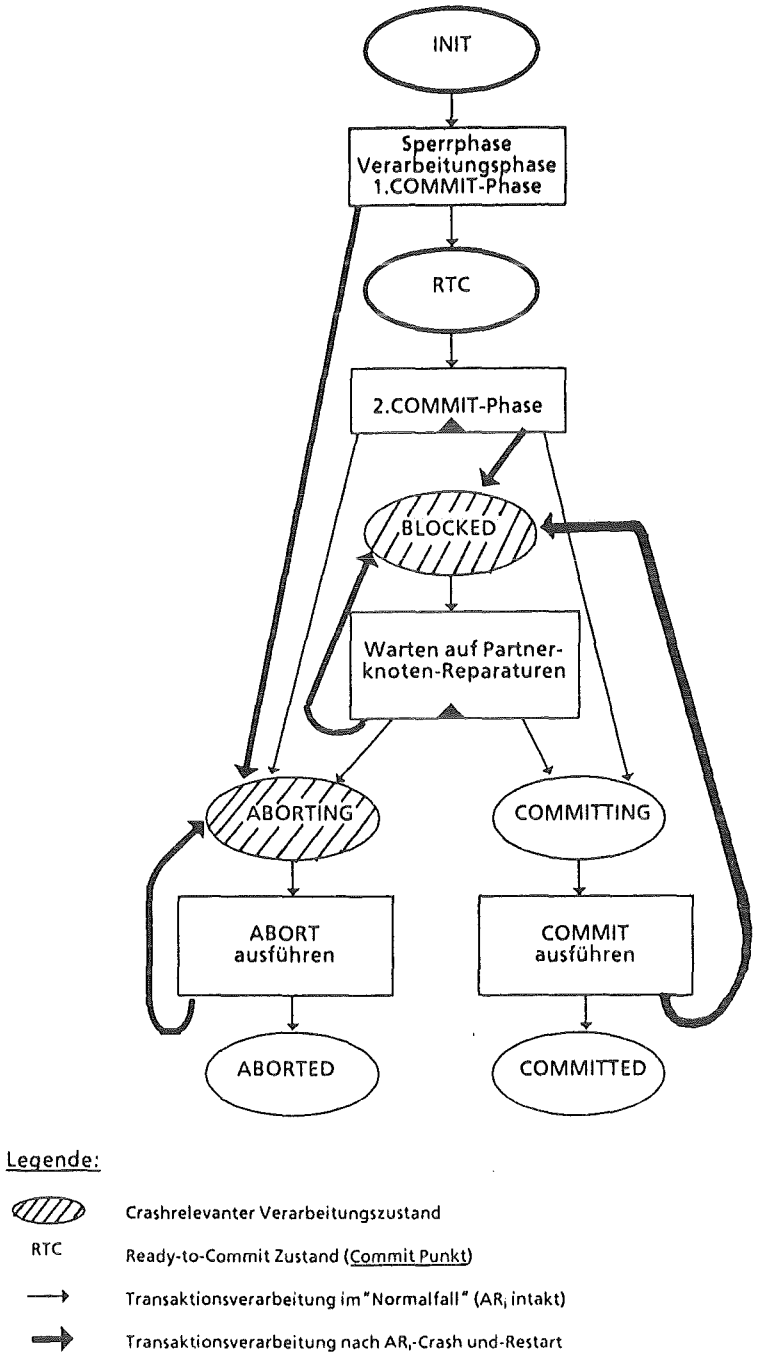


Abb. 7.5: Zustandssicherung und -restaurierung bei TM_i und DM_i

verursachten Inkonsistenzen /MEFA76/. Dort wird dieses Prinzip als 'recoverability' eingeführt.

Der DBVS-Dienstprozeß 'Log manager' LOG_i auf Arbeitsrechner AR_i wird in Anspruch genommen

(a) stets zum Eintrag crash-relevanter Transaktions-Zustände durch die lokalen Prozesse TM_i und DM_i;

(b) stets nach Ausfall von AR_i in dessen Wiederanlauf-Prozedur;

(c) in bestimmten Fällen nach Wiederherstellung der Zugreifbarkeit des Knotens K(i) von einem anderen Knoten aus, um den lokal erreichten Transaktions-Status zu erfahren.

Der Logging-Dienst wird mit Hilfe eines sicheren Hintergrund-Speichersystems (Kap. 4.2.4) realisiert; damit sind alle Log-Operationen unteilbar und dauerhaft.

Das **Logbuch** ist eine Datenstruktur mit folgenden Einträgen für jede offene bzw. terminierte Transaktion:

T: Transaktions-ID
TM_STATE: aktueller crashrelevanter TM-Verarbeitungszustand
DM_STATE: aktueller crashrelevanter DM-Verarbeitungszustand
COHORT(T): Menge der Partnerknoten
$\{SUB_i \mid i \varepsilon COHORT(T)\}$: Menge der Subtransaktionen
UNDO/REDO-Info: Information über zu verändernde Daten (Vor- u/o Nachzustand)

UNDO/REDO-Konzepte werden wir im folgenden nicht detailliert betrachten, da es für den Gesamtablauf in unserem VDBS-Modell unwesentlich ist. In einem Überblick /BERN83c/ wurde festgestellt, daß es sogar Recovery-Verfahren gibt, die weder UNDO noch REDO benötigen. Bewertung und Vergleich dieser Verfahren gehört mehr in den Bereich zentralisierter Systeme und geht über die Ziele dieser Arbeit hinaus. Auf der von uns gewählten Abstraktionsebene sind die Unterschiede zwischen Logging in zentralisierten und verteilten DBVS nicht wesentlich.

Wir definieren nun eine operationale Schnittstelle zum Logging-Dienst:

SAVE-STATE((TM_STATE(T),T-Info) , (DM_STATE(T),SUBT-Info))
returns (Quittung(T))

Diese Operation wird lokal von den TM- und DM-Prozessen i.d.R. mehrfach während der Lebensdauer einer Transaktion angewendet, um einen crashrelevanten Zustand und evtl. Subtransaktions-Information neu ins Logbuch einzubringen bzw. einen Eintrag zu aktualisieren.

Die Operation wird als Anforderungs-Operation an den lokalen Logging-Prozeß geschickt, da eine Quittierung des erfolgten

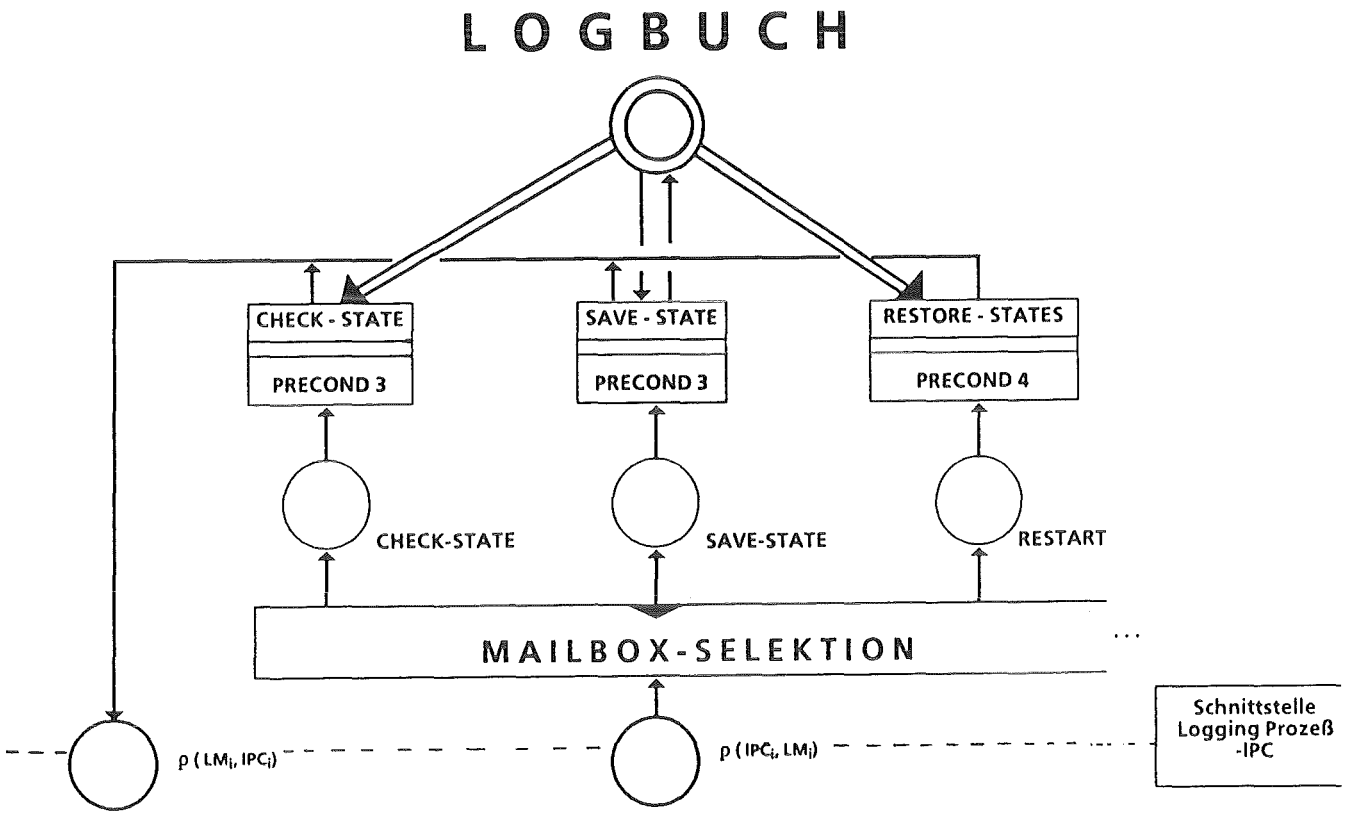


Abb. 7.6: Modell des Logging-Dienstprozesses LM_i

```
PRECOND3: select mεLOGBUCH: m.TID = m'.TID
PRECOND4: select mεLOGBUCH: m.TM_STATE = 'offen'
                          V m.DM_STATE = 'offen'
```

Logbuch-Eintrags entsprechend Gray's WAL-Konzept ("write-ahead log" /GRAY78/) zur Atomaritätssicherung im Rahmen des Commit-Protokolls erforderlich ist. Bei fehlender Quittierung könnte der anfordernde Prozeß nebenläufig zur Sicherungsoperation den angeblich schon erreichten Transaktionszustand als Botschaft seinen Kooperationspartnern mitteilen. Fiele der betreffende Knoten nun nach Abschicken dieser Botschaft, aber vor erfolgter Zustandsicherung aus, wäre der Inhalt der Botschaft und der resultierende globale Zustand der Transaktion inkorrekt.

Nach Terminierung einer Transaktion bleibt der entsprechende Logbuch-Eintrag erhalten, d.h. wir gehen idealisierend von einem unbeschränkten Logbuch-Speicher aus, dessen Länge nicht durch

Checkpoints optimiert wird.

RESTORE-STATES returns DM|TM_STATE_1, T-Menge_1,
DM|TM_STATE_2, T-Menge_2, ...

Ein Wiederanlauf eines Arbeitsrechners AR_i ruft in unserem Modell die Operation RESTORE_STATES auf. Diese löst den Wiederanlauf der Prozesse TM_i und DM_i auf diesem Rechner durch Reaktivierung bestimmter Transaktionen aus, die von diesen Prozessen bearbeitet wurden:

Zu jedem crashrelevanten, offenen Transaktions-Zustand, der im Logbuch vor AR_i-Ausfall gespeichert wurde, wird die Menge der sich in einem solchen Zustand befindlichen Transaktionen mit ihren Logbuch-Einträgen (als "T-Menge" bezeichnet) ermittelt. Diese Transaktionen sind in dem für sie zuletzt gespeicherten crashrelevanten Zustand fortzusetzen. Die Fortsetzung aller dieser Transaktionen ist das Ergebnis der RESTORE_STATES-Operation.

CHECK-STATE(T) **returns** (TM- | DM- STATE(T))

Diese Operation ist bei der Terminierung von Transaktionen wichtig, falls innerhalb der zweiten Commit-Phase, d.h. nach Erreichen des 'Commit-Punkts', eine Entscheidung auf Grundlage der lokal vorhandenen Informationen nicht möglich ist. Zur Behebung dieses sog. "window of uncertainty" werden für jede dieser Transaktionen deren Kommunikationspartner nach dem letzten im Logbuch eingetragenen TM- (DM-) Zustand befragt. Dazu dienen die IPC-Operation WATCH_AV und bei anschließender Knotenzugreifbarkeit die Logging-Operation CHECK_STATE (näheres dazu in 7.4.4).

Abb. 7.6 zeigt ein Netzmodell des Dienstprozesses 'Log Manager' mit den Selektionsbedingungen PRECOND3 zum Lesen und Ändern crash-relevanter Transaktionszustände und PRECOND4 zum Wiederanlauf aller offenen Transaktionen.

7.4 Protokoll auf partitionierter Datenbasis

Das funktionale und zeitliche Verhalten der Netzinstanzen des Protokollmodells wird nachfolgend in der Reihenfolge der Instanzaktivierungen, von der Transaktionseingabe bis zur Transaktionsterminierung, erläutert.

7.4.1 Modell der Anfrage-Auswertungsphase

Das DAEMON-Teilnetz der Anfrage-Auswertung (incl. Transaktions-Eingabe) zeigt Abb. 7.7.

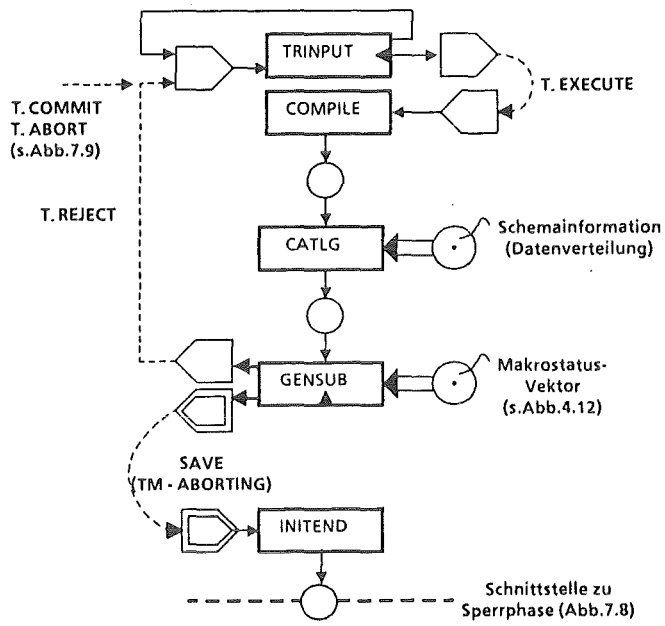


Abb. 7.7: DAEMON-Submodell der Transaktions-Eingabe und Anfrageauswertung

Eine Transaktion T wird an ihrem Heimatknoten HOME(T) an einem Terminal nach einer gewissen Denkzeit eingegeben (Instanz **TRINIT**),ein netzweit eindeutiger Transaktions-ID vergeben und die Ankunftszeit registriert. (Im folgenden steht "T" sowohl zur Identifizierung der Transaktion als auch der Marke, die den Protokollablauf bei Abarbeitung der Transaktion repräsentiert.) Die damit definierte Transaktion wird dann als lokale Nachrichten-Marke an den TM-Prozeß geschickt. Die Kardinalität der initialen Marken auf dem Inputkanal dieser Instanz bestimmt die Terminalanzahl und auch die maximale Anzahl nebenläufig aktiver Transaktionen an diesem Knoten.
In der Empfangsinstanz **COMPILE** wird die Transaktion übersetzt und das Transaktionsprofil entsprechend Kap. 4.4.5 erzeugt. Ergebnis ist die logische Basismenge BS(T), die die Menge aller lokalen/nichtlokalen Lese/Schreib-Dateneinheiten enthält. Die Struktur der Basismenge wird in der T-Marke gespeichert.

In der Folgeinstanz **CATLG** erfolgt ein (gemäß Ann. in Kap. 3.3.2 lokaler) Katalogzugriff. Dabei werden alle Informationen zur Generierung von Unteraufträgen, insbesondere der Speicherort der benötigten Dateneinheiten, beschafft und ein Auswertungsplan produziert. Alle Dateneinheiten der Datenbasis werden dabei entsprechend dem Verfahren von Kap. 4.4.4 zu Rechnern zugeordnet. Im hier diskutierten Fall der partitionierten Datenbasis liefert das Verfahren eine (annähernde) Gleichverteilung aller Dateneinheiten auf Knoten. Auf allen Katalogen im Rechnernetz liegt dann die gleiche Schema-Information über Verteilung und Struktur der Daten vor.
Die Instanz **GENSUB** generiert die Menge der Subtransaktionen von T. Eine Subtransaktion $SUB_i(T)$ enthält

- Die auf einem Knoten Z_i gespeicherten Lese- und Schreib-Dateneinheiten, die von T benötigt werden,
- Den Zielknoten Z_i sowie die Zielknoten aller beteiligten Subtransaktionen,
- Information zur Ausführung der Operationen und zur Interaktion mit anderen Subtransaktionen (in unserem Modell nicht relevant)

Beispiel 7.1:

Gegeben sei
- LOC = 0.6 Lokalität
- WR = 0.5 Update-Häufigkeit

Dann liefern die Schritte (s2) und (s3) folgende Aufteilung der Basismenge BS(T), wenn als Ergebnis von Schritt (s1)

|BS(T)| = 5 geliefert wird:

	Anzahl Primärkopien lokal	Anzahl Primärkopien nichtlokal
Lese-Schreib-Zugriffe	1	1
	2	1

Im Schritt (s4) werden entsprechend diesen Anzahlen lokale bzw. nichtlokale Primärkopien zugeordnet, wobei jede (nicht-)lokale Zuordnung gleichwahrscheinlich ist.

Sei das Ergebnis dieser Zuordnung

(5,A), (5,B), (5,C): lokale Primärkopien
(4,D), (6,E): nichtlokale Primärkopien

wobei (i,d) : Dateneinheit d ε DB ist als Primärkopie auf Knoten K_i, i ε [1,N], gespeichert

HOME(T) = 5 : K_5 ist Heimatknoten der Transaktion T

Bei partitionierter, redundanzfreier Datenbasis werden drei Subtransaktionen zur Ausführung an Knoten 4, 5, 6 mit folgendem Inhalt generiert:

(4, (r,D)),
(5, (r,A), (w,B), (w,C)),
(6, (w,E))

r bzw. w bezeichnet den Zugriffsmodus 'Lesen' bzw. 'Schreiben'
(Beispiel Ende)

Vor der Sperrphase wird für die Transaktion T ein Logbuch-Eintrag angelegt und der crash-relevante Zustand TM-ABORTING gesichert. Nach Ausfall und Restart von HOME(T) kann somit die offene Transaktion T abgebrochen werden, ohne die Konsistenz der Daten verletzt zu haben. Nach erfolgter Quittierung des Eintrags beginnt die Sperrphase.

Die beschriebenen Instanzen sind in Tab. 7.1 mit den Einflußgrößen des Umgebungsmodells (s. Anhang A) und den Ressourcen, auf denen sie realisiert sind, dargestellt.

Instanz	Ressource	Funktions-Inputparameter	Zeit-Inputparameter
TRINIT	TERMINALS	TCO	D
COMPILE	AR	E, BS, LOC, WR	T3
CATLG	DISK	E, Datenverteilung	T4•|BS(T)|
GENSUB	AR	--	0.001 sec

Protokollablauf bei Ausfällen

Zu unterscheiden ist der Ausfall des lokalen Arbeitsrechners $AR_{HOME(T)}$ vom Ausfall anderer Arbeits- bzw. Netzzugangs-Rechner.
Im ersten Fall werden alle Transaktionen an Terminals sofort abgewiesen, d.h. er wird angenommen, daß das Terminal Kenntnis vom Ausfall des lokalen Arbeitsrechners erlangt und dies als 'Ergebnis' dem Terminal-Benutzer mitteilt. Alle Transaktionen, für die zum Crash-Zeitpunkt noch kein Logbuch-Eintrag existierte, sind nach Restart undefiniert, d.h. sind verloren gegangen. Falls der Eintrag schon geschrieben und die Transaktion sich noch nicht in der Sperrphase befunden hat, wird sie bei Restart im Zustand TM-ABORTING

wieder fortgesetzt und anschließend mit ABORT beendet.
Ein Ausfall anderer Rechner führt bei dem diskutierten Protokoll auf partitionierter Datenbasis stets zur Abweisung aller Transaktionen auf HOME(T), bei denen **irgendeine Dateneinheit** durch den Ausfall unverfügbar geworden ist. Im schlimmsten Fall wird der Arbeitsrechner vom restlichen Rechnernetz durch Ausfall von $NIU_{HOME(T)}$ getrennt, sodaß nur noch lokale Transaktionen darauf erfolgreich abgearbeitet werden können. Der globale Verfügbarkeits-Status aus Sicht von HOME(T) wird auf Anforderung der Instanz **GENSUB** gemeldet. Der Ausführbarkeits-Test führt hier zu der REJECT-Botschaft an den Terminal-Benutzer.

7.4.2 Modell der Sperrphase

Das Submodell der Sperrphase wurde nach dem Konzept des Verteilten 2-Phasen-Sperrprotokolls mit Preclaiming von Sperren spezifiziert. Dabei werden die Sperranforderungen an verschiedene Knoten als Subtransaktionen in aufsteigender Reihenfolge einer virtuellen, aber festen Knotennummerierung 1,2,...,N 'sequentiell' an den ggf. abgesetzten Sperr-Scheduler verschickt. Die i+1-te Anforderung wird erst nach positiver Quittierung der i-ten Anforderung übermittelt.
Dieses Verfahren ist nach /BAYE84/ verklemmungsfrei, läßt aber die Strategie des lokalen Sperr-Schedulings offen: Falls mehrere Subtransaktionen in der Warteschlange vor dem Scheduler warten, deren Sperranforderungen erfüllbar sind, muß der Scheduler einer 'geeigneten' Subtransaktion die Sperren gewähren. Zu der Wahl des geeigneten Kandidaten existieren in der Literatur nur spärliche Aussagen.
Ein geeignetes Scheduling-Verfahren sollte **fair** sein, d.h. bestimmte Anforderungstypen nicht einseitig bevorzugen, sodaß im schlimmsten Fall bestimmte Subtransaktionen 'verhungern', d.h. eine unbestimmt lange Zeit auf Erfüllung ihrer Anforderungen warten ('starvation'). Ein unfaires Verfahren würde z.B. lokale vor globalen oder kleine vor großen Subtransaktionen bevorzugen.
Mehrere Verfahren wurden entworfen und deren Fairness, Verhungerungs-Freiheit und Performance durch Simulationsexperimente getestet (ohne Anspruch auf die Verifikation dieser Eigenschaften).

Dabei hat sich folgendes Verfahren als günstig erwiesen:

> Subtransaktionen werden entsprechend ihrem 'Alter', d.h. der Wartezeit in der Sperrqueue, geordnet. Die älteste Subtransaktion hat Priorität. Falls deren Sperranforderungen nicht sofort erfüllbar sind, wird diejenige nächstälteste ausgewählt, deren **Sperrmenge disjunkt** zu der Sperrmenge der ältesten ist.

Wenn keine Konflikte auftreten, werden bei dieser Strategie Subtransaktionen stets in FIFO-Ankunftsreihenfolge vom Scheduler bearbeitet. Andernfalls muß die älteste Anforderung auf das Ensperren von Teilen ihrer Sperrmenge warten; diese kann aber nie durch jüngere Anforderungen vorher gesperrt werden, sodaß ein Verhungern zumindest wegen anderen wartenden Aufträgen nicht stattfinden kann. Allerdings werden dabei zeitweise evtl. überhaupt keine Anforderungen erfüllt, auch wenn es wartende Subtransaktionen gibt, deren Sperrmenge in dieser Zeit frei ist.

Das 'Verhungern' von Sperranforderungen in einem verteilten System kann auch durch folgendes Verhalten verursacht werden:

> In der zweiten Commitphase können Subtransaktionen durch eine Partitionierung vom TM-Prozeß blockiert werden, bis nach Reparatur die Verbindung wieder hergestellt ist. Die dabei gesetzten Sperren würden solange ebenfalls bestehen bleiben.

Dieses Problem wurde gelöst, indem blockierte Dateneinheiten **markiert** werden, sodaß jede Sperranforderung an blockierte Daten zu einem Rücksetzen (ABORT) der anfordernden Subtransaktion führt.

Dieses Sperrkonzept wurde durch das in Abb. 7.8 gezeigte DAEMON-Teilnetz realisiert. Nach Beendigung der Anfrageauswertung wird durch die Instanz SNDLOCK jeweils eine Subtransaktion als Botschaft **LOCKREQUEST** an die Empfangsinstanz GETLOCK im DM-Zielprozeß verschickt. (Bemerkung: Die Menge der zu sperrenden Daten an einem Knoten sind bei dem diskutierten Protokoll identisch mit der dort zu verarbeitenden phys. Basismenge, die wir als 'Subtransaktion' bezeichnen. Grund dafür ist die identische Granularität von Sperr- und Zugriffseinheiten der Datenbasis in unserem Modell.) SNDLOCK benutzt dazu das IPC-Primitiv REQU_MSG (s. Kap. 4.4), da eine Antwort vom Lock-Scheduler erwartet wird. Die Instanz GETLOCK speichert

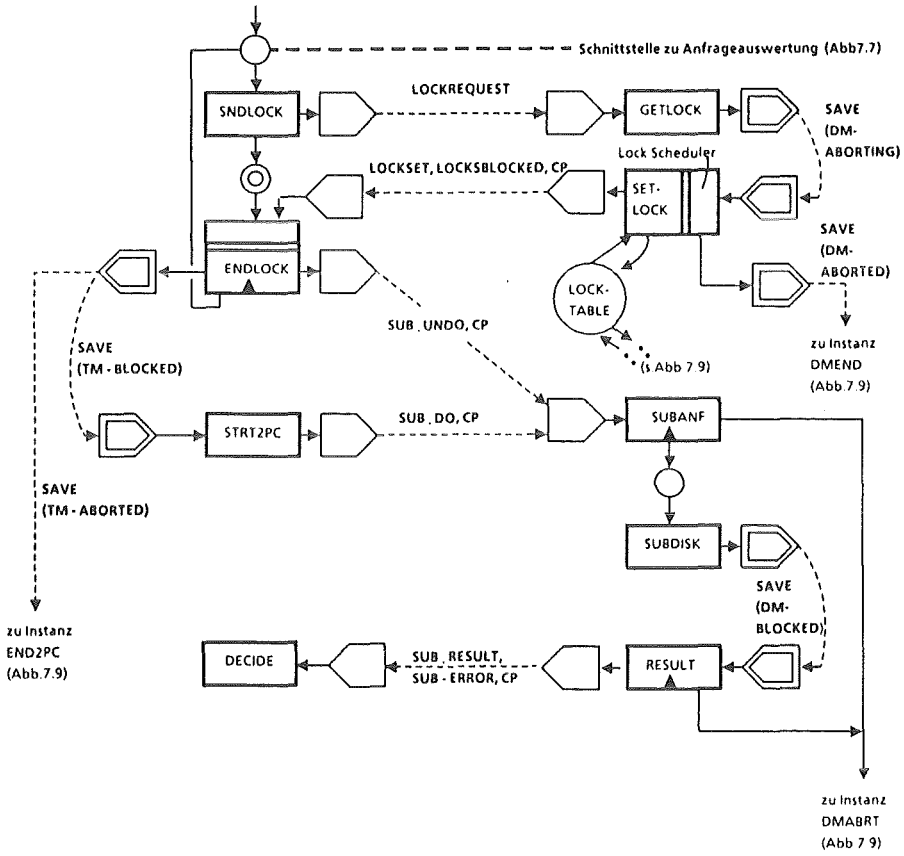
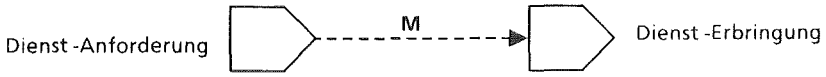
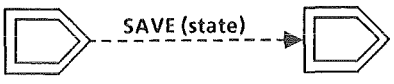


Abb. 7.8: DAEMON-Submodell der Sperrphase und der Subtransaktions-Ausführung

Legende:



Informationsfluß zur Übermittlung der Botschaft **M** zwischen zwei lokalen oder abgesetzten Prozeßinstanzen zur Realisierung eines Anwendungsschicht-Protokolls. Die Botschaft wird durch die darunterliegende IPC-Schicht und evtl. eine Interaktion aller tieferen Schichten übermittelt.



Informationsfluß zur Sicherung des crashrelevanten Zustandes **state** auf Logbuch durch den lokalen Logging-Prozeß und die Quittierung der erfolgten Sicherung. Sicherungs- und Quittierungs-Botschaften werden durch die IPC-Schicht übermittelt. Sicherungsanforderungs- und Quittungsziel-Instanzen gehören zu demselben Prozeß der Anwendungsschicht.

Information über die neuerhaltene Subtransaktion im Logbuch ab, sowie sichert den Zustand DM-ABORTING, sodaß nach Ausfall des Knotens die Subtransaktion stets zurückgesetzt werden kann. Die Quittung vom Logging-Prozeß wird, zusammen mit der Subtransaktion, in die Sperr-Warteschlange vor dem Lockscheduler SETLOCK eingereiht. (Diese Warteschlange ist als SET-Kanal organisiert.)
Der Scheduler benutzt zur Entscheidung eine lokale Sperrtabelle LOCKTABLE. Diese enthält für jede lokal gespeicherte Dateneinheit folgende Einträge:

- Sperrzustand, wobei unterschieden wird zwischen 'frei', 'Schreibsperre' und 'Anzahl n Lesesperren'
- Blockierungszustand (s.o.)
- Versionsnummer (wird für Kopienbehandlung benötigt, s. 7.5)

Eine Sperranforderung einer Subtransaktion SUB ist nur dann erfüllbar, wenn jede Dateneinheit in der Sperrmenge von SUB entsprechend ihrem Zugriffsmodus kompatibel ist mit dem Sperrzustand dieser Dateneinheit. **Kompatibilität** ist wie üblich definiert: Leseanforderungen sind mit Lesesperren kompatibel, aber nicht mit Schreibsperren. Eine Schreibanforderung kann nur bei ungesperrter Dateneinheit erfüllt werden. Das Sperren aller benötigten Daten einer Subtransaktion SUB ist atomar, d.h. entweder werden alle Sperren in einer unteilbaren Operation gesetzt, oder SUB muß weiter warten, oder SUB wird wegen Datenblockierung zurückgesetzt.
Bei erfolgreicher Sperrung wird die Sperrtabelle aktualisiert, und die SETLOCK-Instanz setzt eine REQU_REPLY-Botschaft **LOCKSSET** an den TM-Prozeß ab. Die eventuelle Rücksetzung erfolgt durch die SND_REPLY-Botschaft **LOCKSBLOCKED** (es wird nicht mehr auf eine Fortsetzung von SUB gewartet), sowie durch die Sicherung des Endzustands DM-ABORT auf Logbuch. Die Instanz ENDLOCK emfängt diese Botschaften im TM-Prozeß und setzt die Transaktion wie folgt fort:

- Nach Empfang von **LOCKSSET** wird die Instanz SNDLOCK aktiviert, falls noch weitere Subtransaktionen zu verschicken sind. Andernfalls ist die Sperrphase fertig, und der Zustand TM-BLOCKED wird gesichert (s. nächsten Abschnitt).
- Nach Empfang von **LOCKSBLOCKED** wird die Transaktion bei dem hier diskutierten Fall der partitionierten Datenbasis insgesamt zurückgesetzt. Es wird eine SEND_REPLY-Botschaft **SUBUNDO** als

1:n-Multicast-Operation an alle DM-Prozesse geschickt, die bereits für die bearbeitete Transaktion Sperren gesetzt haben. Die betreffenden Unteraufträge werden dann dort durch die DM-Instanzen SUBANF und DMABRT zurückgesetzt, indem alle Sperren aufgehoben werden. Im TM-Prozeß wird der ensprechende Endzustand TM-ABORT gesichert und die Transaktion mit der Instanz END2PC beendet (genauer in 7.4.4 spezifiziert).

Protokollablauf bei Ausfällen

Alle nach Kap. 4.2 möglichen Ausfälle lassen sich bezüglich ihren Auswirkungen auf die Transaktionsbearbeitung in folgende Klassen einteilen:

(F1) Ausfall und Wiederanlauf des Heimat-Arbeitsrechners HOME(T) einer Transaktion T. Bei Wiederanlauf wird T im vor dem Crash gesicherten Zustand TM-ABORTING bzw. TM-BLOCKED fortgesetzt. Der erste Zustand führt stets zur Rücksetzung von T. Bei TM-BLOCKED sind komplizierte 'Verhandlungen' über den evtl. erreichten Endzustand mit den Partnerknoten nötig; ein Rücksetzen ist ggf. inkorrekt, falls die Partnerknoten eine autonome positive Entscheidung bereits gefällt haben.

(F2) Ausfall und Wiederanlauf eines Partnerknotens führt bei Wiederanlauf zu Fortsetzung der unterbrochenen Subtransaktion in DM-ABORTING bzw. DM-BLOCKED. Im Zustand DM-ABORTING wird diese zurückgesetzt. Es werden alle Sperren aufgehoben. Falls die Subtransaktion schon ausgeführt worden ist, müssen auch alle evtl. erfolgten Datenbasis-Änderungen rückgängig gemacht werden. Bei DM-BLOCKED wird ähnlich wie bei TM-BLOCKED fortgesetzt.

(F3) Die Auswirkungen einer Partitionierung zwischen HOME(T) und einem Partnerknoten K sind vom jeweils erreichten lokalen Zustand des betroffenen TM- bzw. DM-Prozeß ab. Die Nichtverfügbarkeit von K (bzw. von HOME(T)) wird dem TM- (bzw. DM-) Prozeß durch CP-Botschaften (Crashed / Partitioned) von der lokalen IPC-Schicht mitgeteilt, falls ein Reply (Antwortmeldung) von dem unverfügbaren Knoten aussteht (vergl. Kap. 4.4).

Im letzten Fall (F3) kann eine (Sub-)Transaktion je nach Protokollzustand sehr unterschiedlich fortgesetzt werden. Nachfolgend wird daher dieser Fall für alle relevanten Zustände diskutiert. (F1)

und (F2) werden genauer im Rahmen der zweiten Commitphase (in 7.4.4) beschrieben. Wir diskutieren nun die Auswirkungen nach Partitionierungen (F3) in der Sperrphase.

Eine Partitionierung zwischen HOME(T) und einem Partnerknoten K innerhalb der Sperrphase hat dort nur dann Auswirkungen, falls die Botschaft **LOCKREQUEST** noch vor der Partitionierung vom DM-Prozeß auf K empfangen wurde. Die Quittierung des erfolgten Sperrauftrags durch
LOCKSSET führt zu der Rückmeldung **CP** an die Instanz SUBANF, daß der Knoten HOME(T) nicht verfügbar ist. Die Subtransaktion wird anschließend von der Instanz DMABRT zurückgesetzt.
Der TM-Prozeß kann anstatt der erwarteten Sperrquittung die Partitionierungsmeldung **CP** an der Instanz ENDLOCK empfangen. Es wird dann genauso wie bei Empfang eines **LOCKSBLOCKED** verfahren.

7.4.3 Modell der Ausführung von Subtransaktionen (erste Commitphase)

Nach erfolgreicher Beendigung der Sperrphase wird durch die Instanz STRT2PC ein **integriertes 2PC-Protokoll** angestoßen (s. Abb. 7.8 bis 7.10). Dieses in Grundzügen bereits in /LEBR82/ und /BRLE82a/ veröffentlichte Protokoll weist folgende Merkmale auf:

(1) Das Verschicken und Ausführen der Subtransaktionen ist in dessen erste Phase integriert. Die Freigabe der gesperrten Daten (zweite Phase des Sperrprotokolls) ist in dessen zweite Phase integriert. Dadurch wird das Geamt-Protokoll bezüglich der Anzahl von Nachrichten und den dadurch verursachten Verzögerungen minimiert.

(2) Die Dienste der zuverlässigen IPC-Schicht werden in Anspruch genommen, um die (Nicht-)Verfügbarkeit zwischen allen beteiligten Kommunikationspartnern bei der Ausführung einer Transaktion in beschränkter Zeit zu erkennen und dadurch die Transaktion nicht unnötig zu verögern.

(3) Die Ausfallbehandlung wurde so konzipiert, daß die **Atomarität** von T trotz beliebiger Rechnerausfälle und Netzpartitionierungen stets gewahrt wird, d.h. daß nach Reparatur aller ausgefallenen Komponenten der TM-Prozeß auf HOME(T) sowie alle Partnerknoten schließlich in denselben T-Endzustand COMMIT bzw. ABORT gelangen.

(4) Die Einbeziehung von Partionierungen erfordern die Blockierung betroffener Transaktionen. Wo diese unumgänglich ist, wurde die Entwurfsentscheidung gewählt, gesperrte Daten zu blockieren, anstatt den Benutzer über das Schicksal der Transaktion beliebig lange im unklaren zu lassen.

Zu Beginn der ersten Commitphase werden Subtransaktionen mittels einer REQU_REPLY-Botschaft **SUBDO** an die Instanzen SUBANF auf allen Partnerknoten parallel verschickt. Wesentlich dabei ist, daß außer der physischen Basismenge die Liste aller beteiligten Partnerknoten mitgeliefert wird, damit die DM-Prozesse bei HOME(T)-Partitionierung die Transaktion T selbständig fortsetzen können (näheres dazu im nächsten Abschnitt). Eine SUBANF-Instanz initiiert nach Empfang der Botschaft daraufhin die Ausführung der jeweiligen Subtransaktion. Dabei wird folgendes Ablaufmodell unterstellt:

(1) Kopie aller benötigten Dateneinheiten vom Massenspeicher in Arbeitspuffer holen (Instanz SUBDISK),
(2) Darauf Operationen entsprechend dem Ausführungsplan ausführen (Instanz RESULT),
(3) Vorzustände zu ändernder Dateneinheiten von Datenbasis auf Logbuch sichern, um ggf. die Subtransaktion noch rücksetzen zu können (im Rahmen der Sicherung des Zustands DM-BLOCKED),
(4) Änderungen in Datenbasis einbringen (Instanz SUBDISK).

Mit diesen Schritten wird die bekannte 'update-in-place' Strategie /BERN83c/ realisiert. Als wesentliche Modell-Vereinfachung wird der Ergebnisaustausch zwischen Subtransaktionen vernachlässigt. Die beteiligten DM-Prozesse (Instanz RESULT) schicken ihre Zwischenergebnisse als REQU_REPLY-Botschaft **SUBRESULT** an die TM-Instanz DECIDE, die daraus das Endergebnis bestimmt.

Die Ausführung von Subtransaktionen kann in bestimmten Fällen fehlerhaft sein, z.B. wegen Speicherfehlern oder Verletzung der logischen Datenintegrität. Das führt zum Abbruch einer Subtransaktion nach dem Übergang zur Instanz DMABRT. Dabei wird der Zustand der lokalen Datenbasis vor der Ausführung restauriert ('Rückwärts-Recovery', Kap. 4.2.1). Die Vorzustände aller geänderter Dateneinheiten werden aus dem Logbuch in die Datenbasis zurückgeschrieben; alle von der Subtransaktion benötigten Kopien

werden entsperrt. Der bei Ausführung bzw. Abbruch einer Subtransaktion (Operation SUB_DO bzw. SUB_UNDO) den Instanzen SUBDISK bzw. RESULT zugeordnete Speicher- bzw. Arbeitsrechner-Zeitverbrauch kann dem operationalen Modell aus Anhang A entnommen werden.

Falls die Instanz RESULT auf Rücksetzung der Subtransaktion entscheidet, übermittelt sie die Botschaft **SUBERROR** anstatt SUBRESULT an die TM-Instanz DECIDE. Wenn alle Antworten von den Partnerknoten angekommen sind, trifft DECIDE eine **Commit-Entscheidung**, die die Atomarität aller durch die Unteraufträge geänderten Daten sichert:

- Wenn alle Unterauftrags-Ergebnisse per **SUBRESULT** erhalten wurden, kann die Transaktion erfolgreich mit COMMIT beendet werden (Übergang DECIDE --> TMCOMM in Abb. 7.9).
- Wenn mindestens eine **SUBERROR**- oder **CP**-Botschaft angekommen ist, entscheidet DECIDE auf Rücksetzung der Transaktion (Übergang DECIDE --> TMABRT in Abb. 7.9).

Mit dem Senden einer **SUBERROR**-Botschaft nimmt ein Partnerknoten sein 'Veto-Recht' wahr; die gesamte Transaktion muß dann vom TM zurückgesetzt werden. Ist aber die Entscheidung über COMMIT bzw. ABORT lokal vom TM-Prozeß gefällt worden, muß sie zur Erhaltung der atomaren Terminierung auf **allen Partnerknoten** befolgt werden (näheres dazu im nächsten Abschnitt).
Man beachte, daß DECIDE die Entscheidung auch bei beliebigen Nichtverfügbarkeiten von Partnerknoten trifft, ohne auf deren Reaparatur warten zu müssen. Auf den Partnerknoten ist diese nichtblockierende Strategie allerdings nicht möglich, wie noch besprochen wird.

Protokollablauf bei Ausfällen

Bei Partitionierung zwischen HOME(T) und einem Partnerknoten K in der ersten Commitphase empfängt die Instanz SUBANF eine **CP**-Botschaft. Unabhängig davon, ob der TM-Prozeß eine **SUBDO**- oder **SUBUNDO**-Botschaft schicken wollte, kann die Subtransaktion SUB, wie oben beschrieben, zurückgesetzt werden.
Bei Ausfall und Restart eines Partnerknotens ist zu unterscheiden, ob DM-ABORTING oder DM-BLOCKED der zuletzt gesicherte Zustand des

Unterauftrags im Logbuch ist. Liegt letzterer Zustand vor, dann darf SUB nicht zurückgesetzt werden, weil der TM-Prozeß inzwischen möglicherweise die Transaktion erfolgreich beendet hat (TM-COMMIT). Aus Sicht des TM-Prozesses müssen dann **alle** Subtransaktionen ihre erforderlichen Datenbank-Operationen ausgeführt und diese durch **SUBRESULT**-Botschaften an den TM-Prozeß quittiert haben. Der TM-Prozeß muß sich also darauf verlassen können, daß so quittierte Änderungen dauerhaft sind, andernfalls wäre seine COMMIT-Entscheidung nicht korrekt. (Bemerkung: Hier würde sich eine Optimierung des bekannten 2PC-Protokolls für 'read-only' Subtransaktionen anbieten, wie in /MOLI85/ beschrieben. Dies wurde allerdings nicht modelliert.)

7.4.4 Modell der zweiten Commitphase

Nachdem der TM-Prozeß eine COMMIT- (bzw. ABORT-) Entscheidung getroffen hat, wird diese allen Partnerknoten durch die Instanz TMCOMM (bzw. TMABRT) in der Botschaft **SUBCOMMIT** (bzw. **SUBABORT**) mitgeteilt, s. Abb. 7.9. Eine Rücksetzentscheidung wird von TMABRT allerdings nur an diejenige Teilmenge der Partnerknoten übermittelt, die vorher durch Schicken von **SUBRESULT** positiv votiert haben, da die anderen Partner selbst bereits die entsprechenden Unteraufträge abgebrochen haben. Die Instanz TMCOMM (bzw. TMABRT) erteilt auch den Sicherungsauftrag des Endzustandes TM-COMMIT (bzw. TM-ABORT) der Transaktion T. Nach erfolgter Quittierung wird T beendet, indem die Instanz END2PC das Endergebnis aufbereitet und an den Benutzer übermittelt. Nach Empfang am Terminal (Instanz TRINPUT) beginnt eine neue Denkzeit für die nächste Transaktion von diesem Terminal.

Die Entscheidungsnachrichten werden von den Partner-DM-Prozessen durch deren WAITDEC-Instanzen empfangen. Bei **SUBCOMMIT** wird DM-COMMITTING gesichert. Anschließend gibt Instanz DMCOMM alle gesetzten Sperren frei: Damit wird die zweite Phase des Sperrprotokolls realisiert. Wegen der angewandten Update-in-place Strategie (Kap. 7.4.3) sind hier keine Operationen auf der Datenbasis erforderlich. Nach Sicherung des Endzustandes DM-COMMITTED wird die Subtransaktion von der Instanz DMEND beendet, d.h. es existiert nur noch der soeben gesicherte Logbuch-Eintrag.
Bei Empfang einer Rücksetz-Entscheidung **SUBABORT** erfolgt ein Über-

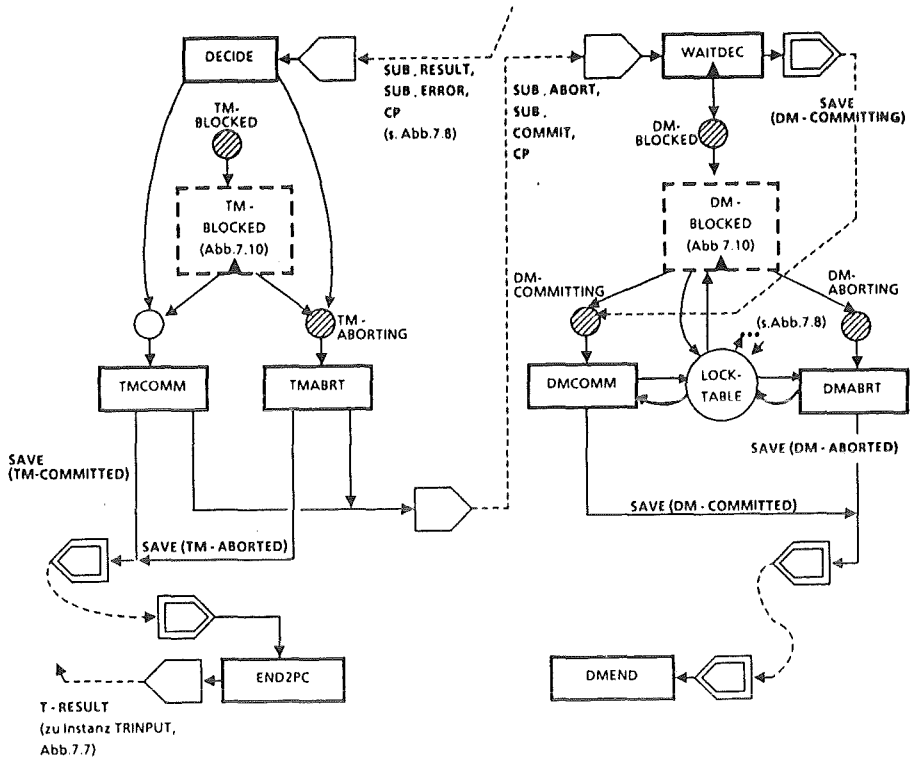


Abb. 7.9: DAEMON-Submodell der Transaktions-Terminierung
(2. Commit-Phase, Behandlung nach Partitionierung des Heimatknotens vergröbert)

Notation für crashrelevanten Kanal

gang zur Instanz DMABRT. Diese macht alle vorher ausgeführten Datenbasis-Änderungen rückgängig, gibt dann alle gesetzten Sperren frei und sichert den Endzustand der Subtransaktion DM-ABORTED. Abschließend wird diese von der Instanz DMEND vernichtet.

Zu beachten ist, daß die hier eingeführte effiziente Version des Commit-Protokolls im Gegensatz zum klassischen Ansatz keine explizite Quittierung der TM-Entscheidung durch die DM-Prozesse zur atomaren Terminierung von Transaktionen benötigt, da das Protokoll eine vollständige Ausfallbehandlung enthält, die aber im Normalfall keinerlei Leistungseinbuße zur Folge hat.

Die Ausfallbehandlung im Rahmen der zweiten Commitphase ist komplex. Zu betrachten sind folgende Fälle:

(a) Ausfallbehandlung im TM-Prozeß nach DM-Nichtverfügbarkeit

(b) Ausfallbehandlung im TM-Prozeß nach eigenem Restart
(c) Ausfallbehandlung im DM-Prozeß nach TM-Nichtverfügbarkeit
(d) Ausfallbehandlung im DM-Prozeß nach eigenem Restart

(a) Ausfallbehandlung im TM-Prozess nach DM-Nichtverfügbarkeit

Der Erhalt einer **CP**-Botschaft durch die TM-Instanz DECIDE wird genauso wie der Erhalt einer **SUBERROR**-Botschaft behandelt, d.h. die Transaktion wird zurückgesetzt.

(b) Ausfallbehandlung im TM-Prozess nach eigenem Restart

Liegt nach Ausfall und Restart eines HOME(T)-Rechners der Zustand TM-ABORTING im Logbuch vor, dann kann die Transaktion sicher zurückgesetzt werden, da kein Partnerknoten eine **SUBCOMMIT**-Botschaft erhalten haben kann.
Ist im Logbuch TM-COMMITTED oder TM-ABORTED gespeichert, dann sind keinerlei Aktionen des TM-Prozesses notwendig: Die Transaktion hat hier vorher schon terminiert; evtl. blockierte Unteraufträge erfragen diesen Endzustand von sich aus (s Fall (c)).
Ist im Logbuch TM-BLOCKED gespeichert, dann weiß der TM-Prozeß zunächst nicht, welche der folgenden Fälle vorliegt:

- Kein Partnerknoten hat eine endgültige Terminierungsentscheidung erhalten. Der HOME(T)-Ausfall muß also vor der zweiten Commitphase erfolgt sein. Die Partnerknoten haben dann keine autonome Entscheidung treffen können, die Unteraufträge sind noch blockiert.
- Mindestens ein Partnerknoten hat die Botschaft über die Enscheidung erhalten; der TM-Prozeß hat aber diese Information noch nicht auf Logbuch gespeichert. Die Partnerknoten können mittlerweile autonom die entsprechende Unteraufträge beendet haben.

Der TM-Prozeß erfragt nun das Schicksal der Transaktion T, bevor er die endgültige Entscheidung dem Benutzer mitteilen kann (s. Abb. 7.10-a). Mit einer WATCH_AV-Operation (Kap. 4.4.3) wird auf die Wieder-Verfügbarkeit aller beteiligten Partnerknoten gewartet. (Da die Namen der Partnerknoten zusammen mit dem Zustand TM-BLOCKED gesichert wurden, kann der TM-Prozeß die entsprechenden Anfragen gezielt abschicken.) Verfügbarkeits-Botschaften **AV** (AVailable) werden von der Instanz TMUNBLCK gesammelt. Nach jeder Meldung erfolgt eine CHECK_STATES-Operation an den betreffenden Knoten, um den

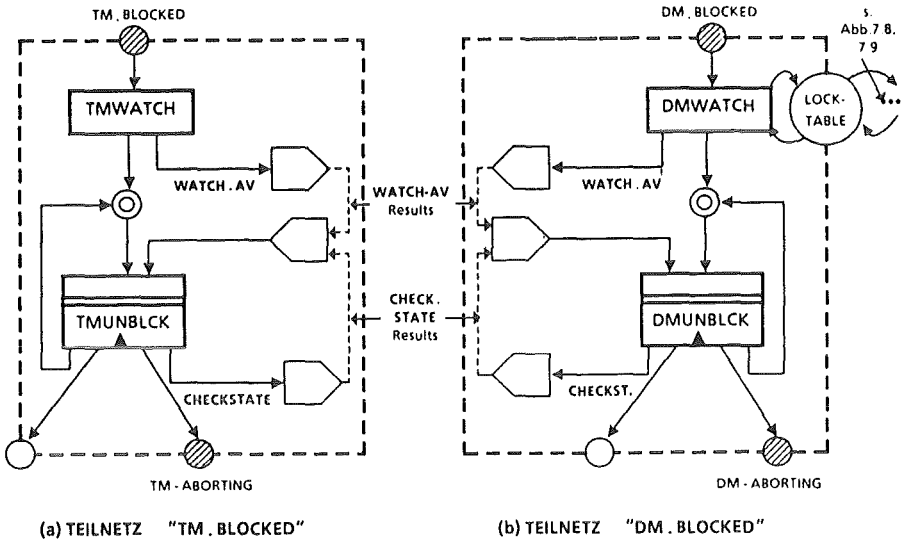


Abb. 7.10: DAEMON-Submodell der Transaktions-Terminierung (nach Partitionierung des Heimatknotens)

Unterauftrags-Zustand aus dem entfernten Logbuch zu erfahren. Anschließend wird versucht, eine Terminierungs-Entscheidung auf der Basis aller bis dahin bekannten Zustände zu erzielen:

- Falls mindestens ein Zustand DM-ABORTING oder DM-ABORTED existiert, erfolgt eine ABORT-Entscheidung. Die noch ausstehenden Verfügbarkeitsmeldungen werden mit einer STOP_WATCH-Operation zurückgenommen.
- Falls alle angefragten DM-Knoten bezüglich T im Zustand DM-BLOCKED oder DM-COMMITTED vorgefunden werden, erfolgt eine COMMIT-Entscheidung.

Ist noch keine Entscheidung möglich, wartet TMUNBLCK auf noch ausstehende **AV**-Botschaften von der lokalen IPC-Schicht.

(c) Ausfallbehandlung im DM-Prozess nach TM-Nichtverfügbarkeit

Der Erhalt einer **CP**-Botschaft durch die DM-Instanz WAITDEC führt zunächst zur **Blockierung** der betreffenden Subtransaktion SUB im DM-Teilnetz DM-BLOCKED (Abb. 7.10-b). Eine Entscheidung allein auf der Basis dieser Botschaft ist nun unmöglich. Die von SUB gesperrten Dateneinheiten werden sofort als blockiert markiert, sodaß andere Subtransaktionen nicht auf Beendigung der Partitionierung warten brauchen. Der DM-Prozeß wendet nun ein ähnliches Verhandlungs-Schema

an wie der TM-Prozeß im Fall (a). Die Information über die in den Logbüchern gespeicherten Zustand der Verarbeitung von T wird allerdings sowohl von allen beteiligten DM-Prozessen als auch vom TM-Prozeß auf Heimatrechner HOME(T) eingeholt. Eine Entscheidung über die Terminierung von SUB kann bei Vorliegen folgender Zustände getroffen werden:

- Eine COMMIT-Entscheidung ist möglich, wenn die Meldung TM-COMMITTED vorliegt, oder wenn mindestens ein beteiligte Subtransaktion im Zustand DM-COMMITTED oder DM-COMMITTING vorgefunden wird.
- Eine ABORT-Entscheidung ist möglich, wenn die Meldung TM-ABORTED vorliegt, oder wenn mindestens ein beteiligte Subtransaktion im Zustand DM-ABORTED oder DM-ABORTING vorgefunden wird.

Schlimmstenfalls finden sich alle Subtransaktionen von T im Zustand DM-BLOCKED vor. Da nun vor Auftreten der Partitionierung der TM-Prozeß eine ABORT-Entscheidung getroffen haben kann, müssen alle Subtransaktionen bis zur TM-Wiederverfügbarkeit blockiert bleiben. Zu beachten ist, daß Protokolle, die Netzpartitionierung ausschließen, hier nichtblockierend sind /MUNZ79/. Ein alternatives Konzept wurde in /LEBR82/ vorgeschlagen: Finden sich alle Subtransaktionen in DM-BLOCKED, dann treffen diese eine autonome COMMIT-Entscheidung. Allerdings führt das zu Blockierung der Transaktion aus TM-Sicht, da dieser bei Partitionierung keine ABORT-Entscheidung treffen darf, ohne die Atomarität zu gefährden. Da diese Strategie zur Blockierung der Terminierungsentscheidung an den Benutzer führt, wurde sie in dieser Arbeit verworfen.

(d) Ausfallbehandlung im DM-Prozess nach eigenem Restart

Liegt nach Ausfall und Restart eines Partnerknotens der Transaktion T der Zustand DM-ABORTING im Logbuch vor, dann kann die Subtransaktion sicher zurückgesetzt werden, da sie keine Entscheidung vom TM-Prozeß auf HOME(T) erhalten haben kann.

Ist DM-COMMITTING der zuletzt gesicherte Zustand, dann muß sich der Ausfall während der Bearbeitung durch die DM-Instanz DMCOMM ereignet haben. Diese Bearbeitung wird nach Restart wiederholt.

Liegt DM-COMMITTED oder DM-ABORTED vor, sind keine DM-Restartaktionen notwendig. Bei Zustand DM-BLOCKED wird genauso verfahren, wie im Fall

(c) beschrieben.

Damit ist das Protokoll auf partitionierter Datenbasis vollständig spezifiziert. Es folgt die Verallgemeinerung auf die Behandlung teilredundant verteilter Daten.

7.5 Protokolle auf (teil-)replizierter Datenbasis

Ausgangspunkt sei eine teilreplizierte Datenbasis mit gleicher Anzahl C>1 Kopien pro Dateneinheit, wobei jede Kopie einer Dateneinheit jeweils einem anderen Rechner zugeordnet wird. Das beschriebene integrierte Protokoll zur Transaktionsverarbeitung auf partitionierter Datenbasis (d.h. C=1) kann dann bezüglich Kopienbehandlung verallgemeinert werden.

Alternative Behandlungsstrategien resultieren in den drei repräsentativen Protokollen PROT_ALL, PROT_PRIM und PROT_MAJ, die im Grundkonzept bereits in Kap. 3.2.3 beschrieben wurden. Die Strategien zielen auf die Nutzung redundanter Kopien zur Leistungs- bzw. Datenverfügbarkeits-Verbesserung für Lese- bzw. Änderungs-Transaktionen, wobei diese Ziele sich gegenseitig widersprechen und daher nicht gleichzeitig optimiert werden können. Für alle Protokolle werden aber die gleichen Qualitätsanforderungen bezüglich operationaler Datenkonsistenz bei beliebigen Ausfällen und Netzpartitionierungen gestellt, wie an das Protokoll auf partitionierter Datenbasis. Zusätzlich muß gegenseitige Kopien-Konsistenz (Kap. 3.2.1) stets gewahrt werden. Damit ist speziell bei Netzpartitionierung zu verhindern, daß zwei Transaktionen in unterschiedlichen Partitionen Änderungen von Kopien derselben Dateneinheit ausführen.
Der prinzipielle Ablauf aller drei Protokolle entspricht dem oben beschriebenen Protokoll. Daher wurde auch bei der Realisierung durch DAEMON-Netze die obige Netzstruktur beibehalten, während die Unterschiede der Protokolle durch unterschiedliche lokale Verarbeitung in bestimmten Instanzen des Netzes realisiert wurden. In diesen Instanzen geht der Protokollname als Parameter ein.

Im folgenden werden die Unterschiede des Protokollablaufs gegenüber

dem oben diskutierten Protokoll sowie wesentliche Protokoll-eigenschaften besprochen. Unterschiede sind hauptsächlich bei der Verarbeitung durch den TM-Prozeß vorhanden:

- Unterschiedliche Abbildung von logischer auf physische Basismenge im Rahmen der Erzeugung der Subtransaktionen in der Auswertungsphase. Dabei können auch die Menge der zu sperrenden Kopien von der Menge derjenigen Dateneinheiten abweichen, auf denen lesend bzw. schreibend zugegriffen wird.
- Unterschiedliche Terminierungsbedingungen in allen drei Protokollphasen: Eine erfolgreiche Terminierung einer Transaktion ist u.U. auch dann möglich, wenn nicht alle Kopien verfügbar sind.

Im DM-Prozeß besteht insbesondere zwischen Kopiensperren und dem Sperren von Dateneinheiten kein prinzipieller Unterschied; die Korrektheit bezüglich Serialisierbarkeit und Verklemmungsfreiheit bleibt dabei erhalten /BERN81/.

Vollzustimmungs-Protokoll (PROT_ALL)

Sei

$$R(T),\ W(T) \subseteq BS(T)$$

die Menge der Lese- bzw. Schreib-Dateneinheiten der logischen Basismenge einer Transaktion T, mit den Dateneinheiten

$$R \in R(T)\ ,\ W \in W(T)$$

Lese- bzw. Schreib-Kopien werden von PROT_ALL wie folgt behandelt:

- $\forall$ W $\in$ W(T): Sperren sind auf **allen** W-Kopien zu setzen; W-Änderungsoperationen auf allen W-Kopien auszuführen.
- $\forall$ R $\in$ R(T): Sperren sind auf **irgendeiner** R-Kopie zu setzen. Ist eine Kopie lokal auf HOME(T) vorhanden, wird diese (in der Auswertungsphase von der TM-Instanz GENSUB) ausgewählt. Andernfalls wird eine andere Kopie genommen, die auf einem verfügbaren Knoten gespeichert ist. Die R-Leseoperation ist auf dieser Kopie auszuführen.

Daraus ergibt sich folgende **Terminierungsbedingung** für T:
T kann nur dann erfolgreich mit COMMIT beendet werden, wenn für alle Dateneinheiten R, W $\in$ BS(T) :

(1) in der Auswertungphase alle W-Kopien und mindestens eine R-Kopie verfügbar sind (von der Instanz GENSUB entschieden),

(2) in der Ausführungsphase keine Subtransaktion mit **SUBERROR** zurückgesetzt wird (von der Instanz DECIDE entschieden).

Andernfalls wird T bei (1) abgewiesen bzw. bei (2) zurückgesetzt.

Primärkopien-Protokoll (PROT_PRIM)

Bei PROT_PRIM ist genau eine Kopie jeder Dateneinheit als Primärkopie gekennzeichnet. Diese Kopien werden hauptsächlich zur Erhaltung der gegenseitigen Kopienkonsistenz nach Ausfällen benutzt; im Normalfall ist die Kopienbehandlung bei PROT_PRIM sehr ähnlich wie bei PROT_ALL:

- $\forall$ W ε W(T): Sperren sind auf **allen** W-Kopien zu setzen; W-Änderungsoperationen auf allen W-Kopien auszuführen.
- $\forall$ R ε R(T): Sperren sind auf der R-Primärkopie zu setzen. Die R-Leseoperation kann in der Ausführungsphase auf **irgendeiner** R-Kopie erfolgen. Ist eine Kopie lokal auf HOME(T) vorhanden, wird diese (von der TM-Instanz STRT2PC) ausgewählt. Andernfalls wird eine andere Kopie genommen, die auf einem verfügbaren Knoten gespeichert ist.

Folgende **Terminierungsbedingung** für T wurde angewendet:

T kann nur dann erfolgreich mit COMMIT beendet werden, wenn für alle Dateneinheiten R, W ε BS(T) :

(1) in der Auswertungphase die R- und W-Primärkopie verfügbar ist (von der Instanz GENSUB entschieden),

(2) in der Ausführungsphase keine Subtransaktion mit **SUBERROR** zurückgesetzt wird, die Zugriffsanforderungen an eine W-Primärkopie oder auf die zum Lesen ausgewählte R-Kopie enthält (von der Instanz DECIDE entschieden).

Andernfalls wird T bei (1) abgewiesen bzw. bei (2) zurückgesetzt.

Bei dieser Strategie können Ausfälle dazu führen, daß Subtransaktionen, die lediglich Schreib-Sekundärkopien enthalten, zurückgesetzt werden, werden alle anderen erfolgreich terminieren. Diese Kopien müssen nach Ausfallbehebung vor einem weiteren Zugriff aktualisiert werden, um Datenkonsistenz zu wahren. Dazu ist ein **Kopienabgleichsprotokoll** erforderlich, das im Rahmen vorliegender Arbeit nicht realisiert wurde: Dieses Protokoll bewirkt eine Leistungseinbuße nach Behebung von Ausfällen; diese Einbuße wird gemäß den Annahmen des Simulationsmodells vernachlässigt (Kap. 5.3.3).

Das Konzept der Kopienbehandlung wurde hier vom VDBS JASMIN /WILA84/ übernommen. Das Konzept des POREL-Systems /WANE84/ ist unterschiedlich. Dort werden Schreib-Sekundärkopien durch einen am Ende der Benutzer-Transaktion initiierten Auftrag 'im Hintergrund' auf den Stand der Primärkopien nachgefahren. Die Auswirkungen dieser Design-Alternative auf die Performance sind unklar, werden aber im Rahmen vorliegender Arbeit nicht untersucht.

Mehrheitsentscheid-Protokoll (PROT_MAJ)

Dieses Protokoll wendet folgende Kopienbehandlung im Normalfall an:

- $\forall$ W ε W(T): Sperren sind auf **allen** W-Kopien zu setzen; W-Änderungsoperationen auf allen W-Kopien auszuführen.
- $\forall$ R ε R(T): Sperren sind auf **allen** R-Kopien zu setzen. Die R-Leseoperation kann in der Ausführungsphase auf **irgendeiner aktuellen** R-Kopie erfolgen. Es wird eine R-Kopie (von der TM-Instanz STRT2PC) ausgewählt, die verfügbar ist und und unter allen verfügbaren R-Kopien die maximale Versionsnummer aufweist (s.u.). Ist eine solche Kopie lokal auf HOME(T) vorhanden, wird diese genommen.

Folgende **Terminierungsbedingung** für T wurde realisiert:
T kann nur dann erfolgreich mit COMMIT beendet werden, wenn für alle Dateneinheiten R, W ε BS(T) :

(1) in der Auswertungphase mindestens eine R- und eine **W-Kopienmehrheit,** von jeweils C Kopien pro Dateneinheit, verfügbar ist (von der Instanz GENSUB entschieden)

(2) in der Ausführungsphase die Instanz DECIDE soviele positive **SUBRESULT**-Quittungen erhalten hat, daß diese Menge der erfolgreich ausgeführten Subtransaktionen sowohl die R-Kopienmehrheit als auch die W-Kopienmehrheit besitzen.

Andernfalls wird T bei (1) abgewiesen bzw. bei (2) zurückgesetzt.

Bei dieser Strategie kann bei Ausfällen kann eine Minderheit von zu ändernder Kopien veralten, sodaß ebenfalls ein Nachführung von Kopien in das Protokoll zu integrieren wäre. Dazu wird eine **Versionsnummer** für jede Kopie benötigt, die nach erfolgter Änderung der Kopie durch die DM-Instanz DMCOMM innerhalb des Commit-Protokolls inkrementiert

wird. Da vor Lesen oder Schreiben einer Kopie stets mindestens die Kopienmehrheit der Dateneinheit gesperrt worden sein muß, ist jedenfalls sichergestellt, daß in dieser Kopienmenge die aktuellste Kopie mit maximaler Versionsnummer enthalten ist /BRLE82a/.

Bei der Kopienbehandlung der drei Protokolle wird versucht, sich **adaptiv** an die Verfügbarkeits-Situation anzupassen. Das wird im folgenden Beispiel deutlich.

Beispiel

Die bei den Protokollen PROT_ALL, PROT_PRIM und PROT_MAJ unterschiedliche Menge generierter Unteraufträge sowie das Verhalten bei wesentlichen Ausfall-Scenarios soll durch das folgende Beispiel erläutert werden.

Ausgangspunkt sei die für eine Transaktion T generierte logische Basismenge von Beispiel 7.1

BS(T) = {A, B, C, D, E}

Die fünf Primärkopien A bis E sind nach dem Verfahren von Kap. 4.4.4 mit ihren Sekundärkopien (mit kleinen Buchstaben bezeichnet) den Knoten im Rechnernetz zugeordnet:

Knoten	Kopien
1	e
2	
3	
4	D
5	A B C d
6	a b c d E
7	a b c e

Wie im obigen Beispiel sei

HOME(T) = 5
A, D: von T zu lesende Dateneinheiten
B, C, E: von T zu schreibende Dateneinheiten

Dann werden im ausfallfreien **Normalfall** folgende Subtransaktionen generiert:

für PROT_ALL: (5, (r,A), (w,B), (w,C), (r,d)),
(6, (w,b), (w,c), (w,E)),
(7, (w,b), (w,c), (w,e)),

(1, (w,e))

für PROT_PRIM: (4, (r,D)),
(5, (r,A), (w,B), (w,C)),
(6, (w,b), (w,c), (w,E)),
(7, (w,b), (w,c), (w,e)),
(1, (w,e))

für PROT_MAJ: (4, (r,D)),
(Sperren) (5, (r,A), (w,B), (w,C), (r,d)),
(6, (r,a), (w,b), (w,c), (r,d), (w,E)),
(7, (r,a), (w,b), (w,c), (w,e)),
(1, (w,e))

für PROT_MAJ: identische Menge Subtransaktionen wie
(Ausführen) bei PROT_ALL

Bei **Ausfällen** müssen folgende Teilmengen der Partnerknoten vor und während der Verarbeitung der Transaktion T intakt und gegenseitig verfügbar bleiben, damit T problemlos mit COMMIT terminieren kann:

für PROT_ALL: {5, 6, 7, 1}

für PROT_PRIM: {4, 5, 6}

für PROT_MAJ: {5, 6, 7} oder {5, 7, 1, 4} oder {5, 6, 1}

In diesem Beispiel gilt also, daß PROT_ALL keine Nichtverfügbarkeit von Partnerknoten der Transaktion T toleriert. Hingegen tolerieren die anderen Protokolle mindestens eine Partitionierung zu zwei Knoten, z.B. PROT_PRIM zu Knoten K_7 und K_1.

Die Charakteristika der drei Protokolle sind in Tabelle 7.1 zusammengefaßt.

7.6 Analyse der mittleren Antwortzeit der Protokolle

Es folgt eine einfache Analyse der voraussichtlichen Kosten der Protokolle, gemessen an der mittleren Antwortzeit. Das Ergebnis der analytischen Abschätzung wird im nächsten Kapitel mit Simulationsergebnissen verglichen, um zu deren interner Validierung beizutragen.

Betrachtet werden

- Das Protokollverhalten unter Ausschluß von Ausfällen,

Kopienbehandlungs-Protokoll	Sperren		Operationen ausführen		Commit-Bedingung		
	R-Kopien	W-Kopien	Lesen R-Kopie	Ändern W-Kopie	Verfügbarkeit R-Kopien	Verfügbarkeit W-Kopien	Tolerierung SUBT-Fehler (in 1. Commit-Phase)
Voll-Zustimmungs-Protokoll	irgendeine verfügbare (mögl. lokale)	alle	gesperrte Kopie	alle	gesperrte Kopie	alle	nein
Primärkopien-Protokoll	Primärkopie	alle	irgendeine verfügbare (mögl. lokale)	alle	Primärkopie	Primärkopie	nur wenn keine Primärkopie betroffen
Mehrheitsentscheid-Protokoll	alle	alle	irgendeine verfügbare u. aktuelle (mögl. lokale)	alle	mindestens Kopien-Mehrheit	mindestens Kopien-Mehrheit	nur solange Kopienmehr-heiten auf allen Fragmenten erhalten bleibt

Tabelle 7.1: Eigenschaften der drei modellierten Protokolle

- Ein Transaktions- und Datenprofil, daß zu U>1 Unteraufträgen pro Transaktion führt

Die mittlere Antwortzeit einer Transaktion RESP ergibt sich grob zu

RESP = R1 (mittl. Zeitdauer der Anfrage-Auswertung)
+ R2 (mittl. Zeitdauer der Sperrphase)
+ R3 (mittl. Zeitdauer der Commitphase)

Die einzelnen Zeitdauern R1, R2, R3 ergeben sich aus

- Summe der mittleren Zeitverbräuche aller beteiligten Instanzen
- Summe der Nachrichtenübermittlungszeiten zwischen Instanzen auf unterscheidlichen, lokalen oder globalen Prozessen
- Summe der Zeitverzögerungen in Warteschlangen (queueing delays), die sich aus Betriebsmittel- und Sperranforderungen ergeben

Gegeben seien die Parameter (vergl. Anhang A):

TL mittl. Übermittlungszeit einer Nachricht zwischen lokalen Instanzen
TG mittl. Übermittlungszeit einer Nachricht zwischen globalen Instanzen

T1 CPU-Zeitverbrauch zum Kataloglesen eines Fragments
T2 CPU-Zeitverbrauch zur Subtransaktions-Ausführung pro Fragment
T3 CPU-Zeitverbrauch zur Transaktions-Übersetzung und Anfrageauswertung
T4 Plattenspeicher-Zugriffszeit zum Lesen / Ändern pro Fragment

Aus der Charakteristik der Protokolle in Abhängigkeit vom einem gegebenen Transaktions- und Datenprofil lassen sich folgende Größen

ermitteln:

U	mittl. Anzahl Subtransaktionen pro Transaktion, U > 1
R	mittl. Mächtigkeit der Lesemenge einer Transaktion
W	mittl. Mächtigkeit der Schreibmenge einer Transaktion
R'	mittl. Mächtigkeit der Lesemenge einer Subtransaktion
W'	mittl. Mächtigkeit der Schreibmenge einer Subtransaktion

Die Bestimmung der Zeitverzögerungen in Warteschlangen ist sehr schwierig , da sie komplexe Berechnungen im Rahmen der Warteschlangen-Theorie erfordert. Wir verzichten auf die analytische Herleitung dieser Größen, verweisen aber auf /GARC79/ für eine Abschätzung der Sperr-verzögerungszeiten, die aber nicht ohne weiteres auf das Preclaiming-Verfahren übertragbar ist. Die wichtigsten Größen sind:

Qlog	mittl. Wartezeit zum Plattenzugriff bei Transaktions-Logging
Qlock	mittl. Wartezeit vor Sperrscheduler (locking delay)

Daraus ergibt sich zunächst die mittlere Zeitdauer TLOG des Transaktionslogging:

```
TLOG = 2*TL + Qlog + T4

     = (Übertragungszeit Transaktionszustand zum Logging-Prozeß)
     + (Wartezeit vor Platte)
     + (Plattenzugriff)
     + (Übertragungszeit Quittung zum Quell-Prozeß)
```

Die mittlere Zeitdauer R1 der Anfrage-Auswertung ergibt sich zu:

```
R1 =  TL           (Übertragung Transaktion von Terminal zu TM-Prozeß)
    + T3 + T1(R+W) (CPU-Zeitverbrauch Transaktionsübersetzung)
    + T4(R+W)      (Plattenzeitverbrauch          "          )
```

Die mittlere Zeitdauer R2 der Sperrphase ergibt sich wie folgt. O.B.d.A. wird angenommen, daß die erste verschickte Sperranforderung an den lokalen DM-Prozeß gerichtet ist:

```
R2 =  TL           (Übertragung  Sperranforderung an 1. DM-Prozeß)
    + Tlog         (1. DM-Prozeß sichert Zustand DM-ABORTING)
    + TL           (Übertragung Quittung an TM-Prozeß)

    + TG           (Übertragung  Sperranforderung an 2. DM-Prozeß)
    + Tlog         (2. DM-Prozeß sichert Zustand DM-ABORTING)
    + TG           (Übertragung Quittung an TM-Prozeß)

    . . .

    + TG           (Übertragung Sperranforderung an U-ten DM-Prozeß)
    + Tlog         (U-ter DM-Prozeß sichert Zustand DM-ABORTING)
    + TG           (Übertragung Quittung an TM-Prozeß)

    + Tlog         (TM-Prozeß sichert Zustand TM-BLOCKED)
```

Die mittlere Zeitdauer R3 der Commitphase ergibt sich wie folgt. Es wird angenommen, daß die Übertragungszeiten bei 1:n-Multicast-Senden sowie bei n:1-Empfangen gleich sind wie das Senden/Empfangen einer Einzelnachricht.
Da alle U DM-Prozesse auf autonom arbeitenden Rechnern ablaufen, fallen die Zeitdauern zur Ausführung der Subtransaktionen im Mittel nur einmal an.

```
R3 =  TG            (Übertragung Subtransaktionen an U DM-Prozesse)
    + T4(R'+2W')    (Platten-Zeitverbrauch zum Ausführen einer Subtr.)
    + T2(R'+W')     (CPU-Zeitverbrauch zum Ausführen einer Subtr.)
    + Tlog          (DM-Prozeß sichert Zustand TM-BLOCKED)
    + TG            (Übertragung Ergebnisse an  TM-Prozess)

    + Tlog          (TM-Prozeß sichert Zustand TM-COMMIT)
    + TL            (Übertragung Ergebnisse an  Benutzer am Terminal)
```

Insgesamt ergibt sich als Schätzung der mittleren Antwortzeit

```
RESP =  R1 + R2 + R3
     =  2U*TG + (U+7)TL + (U+3+R'+2W'+R+W)T4 + T3 + (R'+W')T2 + (R+W)T1
      + (U+3)Qlog + U*Qlock
```

Betrachtet man die Größenordnung der im Anhang A definierten Parameter T1, T2, T3, T4, TL, dann kann diese Formel für U » 1 weiter vereinfacht werden, indem die entsprechenden Terme vernachlässigt werden:

```
RESP' = U*(2TG + Qlock + Qlog) + 3Qlog
```

Daraus können folgende **Vorhersagen** getroffen werden:

(V1) Die Antwortzeit steigt (mindestens) linear mit der Anzahl Unteraufträge U an.

(V2) In Wide-Area Networks (hohes TG) dominieren die Kommunikationskosten, solange Sperrkonflikte selten (kleines Qlock) und Plattenspeicher wenig ausgelastet sind (kleines Qlog).

(V3) Bei hohen Konfliktraten und/oder hohen Plattenauslastungen dominieren die Queue-Wartezeiten; Kommunikationskosten werden unwesentlich.

(V4) In Local-Area Networks (kleines TG) können die lokalen Verarbeitungszeiten T1 bis T4 einen erheblicher Kostenfaktor darstellen, solange Sperrkonflikte selten und Platten wenig ausgelastet sind.

Wir wenden uns nun der Beschreibung der VDBS-Simulationsexperimente und deren Ergebnissen zu.

8 VDBS-SIMULATIONSEXPERIMENTE UND -ERGEBNISSE

Nachfolgend werden die Ergebnisse der Leistungs- und Verfügbarkeitsanalyse der in Kap. 7 modellierten Protokolle zur verteilten Transaktionsverarbeitung beschrieben. Die Vorgehensweise bei Entwurf, Ausführung und Auswertung der Simulationsexperimente folgt der in Kapitel 2.1 entwickelten Methodik.
Zunächst werden im Abschnitt 8.1 die wichtigsten, in den Kapiteln 3 und 4 eingeführten Modellannahmen und -Einflußgrößen zusammenfassend dargestellt. Es folgt im Abschnitt 8.2 eine Beschreibung der Experimententwurfsmethodik, deren Anwendung zu einem Versuchsplan für das VDBS-Simulationsmodell geführt hat. Die Abschnitte 8.3 und 8.4 enthalten eine detaillierte Beschreibung, Auswertung und Interpretation der auf der Basis dieses Plans ausgeführten Simulationsexperimente. Die Leistungen der modellierten Protokolle werden für alle in Kap. 5 festgelegten Zielgrößen miteinander verglichen, wobei die Einflüße der relevantesten Entwurfsparameter diskutiert werden. Abschließend wird im Abschnitt 8.5 über Modelluntersuchungen mit dem Ziel der Gültigkeits(teil-)überprüfung des VDBS-Simulationsmodells berichtet.

8.1 Zusammenfassung der Modellannahmen und Einflußgrößen

Im folgenden werden für jedes Submodell des VDBS-Simulationsmodells jeweils alle **Annahmen, primäre Eingabeparameter** (die in den Experimenten variiert werden), sowie **sekundäre Eingabeparameter** (deren Wert festgelegt wird) beschrieben. Sekundär-Parameter können in zukünftigen Studien variiert oder auch verfeinert werden. Für eine detailliertere Beschreibung der gewählten VDBS-Entwurfs-Parameter mit einer Motivierung der gewählten Werte sei auf Anhang A verwiesen.

8.1.1 Hardware-Architektur

Annahmen (s. Kap. 4.1)

(A1) Homogenes Rechnernetz aus N Knoten. Jeder Knoten besteht aus den Hardware-Ressourcen Arbeitsrechner, NIU, Terminals und Plattenspeicher.

(A2) Alle Systemressourcen können nebenläufig aktiv sein.

(A3) Nachrichten zwischen disjunkten Knotenpaaren können nebenläufig ausgetauscht werden (d.h. es existiert keine zentrale Netzkontrollinstanz).

(A4) Das Rechnernetz ist als Ring strukturiert.

Primär-Parameter

(P1) TCO Terminalanzahl pro Knoten

Sekundär-Parameter

N Knotenanzahl
D Denkrate pro Terminal

8.1.2 Kommunikations- und Betriebssystem-Software

Annahmen

(A5) Alle Protokollschichten des Kommunikationssystems bis incl. der Transportschicht sind auf den NIUs realisiert.

(A6) Die darüberliegende, auf den Arbeitsrechnern realisierte Kommunikations-Steuerungsschicht bietet insbesondere den Dienst 'zuverlässiger 1:n/n:1-Nachrichtenaustausch mit lokaler Erkennung des Systemausfallzustands' an.

(A7) Übertragungszeit zum Senden einer 1:n-Nachricht ('Multicast') ist im Mittel gleich wie die Zeit zum Senden einer Einzelnachricht. Alle Nachrichten haben gleiche Länge.

(A8) Prozeßscheduling-Strategie des Betriebssystems ist 'shortest-job-first / no preemption'

Primär-Parameter

(P2) TG Nachrichtenverweilzeit zwischen entfernten Anwendungsprozessen. Diese legt den Rechnernetz-Typ fest.

Sekundär-Parameter

TL Nachrichtenverweilzeit zwischen lokalen Anwendungsprozessen

8.1.3 Datenprofil

Annahmen (s. Kap. 4.4.3)

(A9) Die Datenbasis besteht aus einer Menge von Fragmenten, die auf allen Rechnerknoten verteilt sein können.
Fragmente bilden die atomare Einheit zur Datenverteilung sowie

zum Sperren, Lesen und Ändern der Daten.

(A10) Kataloginformation ist vollredundant auf allen Knoten vorhanden. Alle anderen Dateneinheiten können teilredundant mit konstanter Kopienanzahl vorliegen.

(A11) Die Verteilungsstrategie ist unabhängig vom Transaktionsprofil. Fragmente (bzw. deren Kopien) werden entsprechend der Gleichverteilungsstrategie PAR (bzw. TRG, s. Kap. 4.4.1) den Rechnerknoten zugewiesen.

(A12) Die Datenbasis ist statisch, d.h. Operationen, die zur Änderung der im Katalog gespeicherten Schemainformation führen würden, kommen nicht vor oder sind so selten, daß sie sich nicht auf das Leistungsverhalten des Systems auswirken.

Primär-Parameter

(P3) E Anzahl Dateneinheiten in Datenbasis

(P4) C Anzahl Kopien pro Dateneinheit

Sekundär-Parameter

Datenverteilungstrategie

8.1.4 Anwendungs- und Transaktionsprofil

Annahmen (s. Kap. 4.4.5)

(A13) Das Systemmodell ist geschlossen. Eine feste Population interaktiver Benutzer, die genau der Terminalanzahl in Rechnernetz entspricht, gibt nach einer gewissen Denkzeit (s.o.) am Terminal eine Transaktion ein und wartet auf die Ergebnis-Rückmeldung. Danach beginnt wieder eine Denkphase.

(A14) Die Systemlast wird allein aus der Transaktionslast gebildet; andere Benutzer-Aktivitäten werden ausgeschlossen.

(A15) Bei Ankunft der Transaktion ist diese bereits vollständig spezifiziert (keine dynamische Anforderung von Daten und Betriebsmitteln).

(A16) Eine Transaktion enthält aus Benutzersicht eine Folge von Lese- u/o Änderungoperationen auf (nicht notwendig lokal vorhandenen) Dateneinheiten.

(A17) Eine Datenbank-Anwendungsklasse DA

DA := (BS, LOC, WR)

wird durch die Verteilung der drei Größen BS, LOC, WR festgelegt. Eine Transaktionsklasse ist allein durch die Mächtigkeit BS der Basismenge charakterisiert. Die Größen LOC bzw. WR legen den

lokalen bzw. zu ändernden Anteil der Basismenge fest. LOC kann als Gütemaß für die Datenverteilungsstrategie interpretiert werden.

(A18) Selektivität: Der Zugriff auf lokal vorhandene Dateneinheiten ist für alle Elemente des lokalen Anteils der Basismenge einer Transaktion gleichverteilt. Entsprechendes gilt für nichtlokale Dateneinheiten und den globalen Anteil einer Transaktion.

Primär-Parameter

(P5) BS Verteilung der Mächtigkeit der Basismenge

(P6) LOC Lokalität: lokaler Anteil der Basismenge

(P7) WR Update-Häufigkeit: zu ändernder Anteil der Basismenge

Sekundär-Parameter

Selektivität

8.1.5 Datenbanksoftware-Architektur

Annahmen

(A19) Alle VDBS-Prozesse 'Anfrageauswertung', 'globale Transaktions-Koordination' (TM), lokale Transaktions-Koordination' (DM), 'lokale Datenverwaltung', 'Logging' (LM) und 'Data Dictionary / Directory' (DD/D) sind auf den Arbeitsrechnern realisiert (vergl. Abb. 4.3).

(A20) Das VDBS enthält Protokolle zwischen TM-, DM- und LM-Prozessen, die operationale Integrität der Datenbasis und Atomarität aller darauf zugreifenden Transaktionen auch bei AR- und NIU-Ausfällen garantieren.
Die Protokolle unterscheiden sich in der Kopienbehandlungs-Strategie auf verteilten Datenbeständen.

(A21) Alle Zugriffe auf lokale Daten eines Knotens werden vom dort befindlichen DM-Prozeß überwacht; ein 'direkter' Datenzugriff findet nicht statt.

(A22) DBVS-Wiederanlaufkosten werden vernachlässigt.

Primär-Parameter

(P8) PROT Transaktionsverarbeitungs-Protokoll (TVP)

Sekundär-Parameter

T1 AR-Zeitverbrauch für 'einfache' Operationen des DBVS
T2 AR-Zeitverbrauch für Ausführung einer Subtransaktion
T3 AR-Zeitverbrauch für Transaktions-Übersetzung und

Generierung von Subtransaktionen

T4 Platten-Zeitverbrauchsrate zum Lesen bzw. Ändern eines Fragments

8.1.6 Fehlermodell

Annahmen (s. Kap. 4.2)

(A23) Hardware-Ausfälle der Ressourcen 'Arbeitrechner' (AR) und 'Netzvermittlungsrechner' (NIU) werden betrachtet. Diese Ressourcen verhalten sich fail-stop, d.h. eine Ressource ist entweder intakt oder total ausgefallen. Alle Ausfallereignisse sind voneinander unabhängig. Alle ARs bzw. NIUs sind in endlicher Zeit reparierbar. Ein Ausfall führt zu Netzpartitionierung zwischen allen solchen Arbeitsrechnern, die nicht durch einen intakten Weg verbunden sind.

(A24) Alle Ausfälle und Partitionierungen werden von intakten Arbeitsrechnern in endlicher Zeit erkannt. Entsprechendes gilt bei Reparaturen.

(A25) System-Hardware und -Software sind korrekt entworfen und realisiert.

(A26) Plattenspeicher, Terminals und Kommunikationsleitungen fallen nie aus.

(A27) Datenbankfehler: Betrachtet wird Subtransaktions-Abbruch.

(A28) Kommunikationsfehler: Nachrichtenaustausch ist sicher, d.h. nur Nachrichten zwischen nichtintakten Arbeitsrechnern oder NIUs können verloren gehen.

Primär-Parameter

(P9) STATE Globaler System-Ausfallzustand

Sekundär-Parameter

λ_{AR} AR-Ausfallrate

μ_{AR} AR-Reparaturrate

λ_{NIU} NIU-Ausfallrate

μ_{NIU} NIU-Reparaturrate

λ_{SUB} Abbruch-Rate pro Subtransaktion

8.1.7 Folgerungen

Gegenüber vielen bisher durchgeführten VDBS-Simulationsstudien (s. /CHEN81, DANT80, DECA82, GARC79, GATS84, OZSU83, RIES79, SEVC83/) enthält unser Modell eine Reihe von wichtigen Verallgemeinerungen:

- Detaillierte "1:1"-Modellierung komplexer Protokolle zur

Transaktionsverarbeitung mit vollständiger Spezifikation einer operationalen, anspruchsvollen IPC-Dienstschnittstelle. Zusammen mit der uneingeschränkten Ressourcen-Nebenläufigkeit (gegenüber analytischen VDBS-Modellen) kann so ein realistisches Protokollverhalten auch im Hochlastfall (resource contention) erwartet werden.

- Realistisches Datenprofil durch Teilreplikation der Dateneinheiten
- Detailliertes Transaktionsprofil durch Betrachtung der Update-Häufigkeit und der Lokalität, sowie einer (stochastischen) Verteilung der Basismenge.
- Berücksichtigung von Arbeitsrechner- und NIU-Ausfällen, incl. Netzpartitionierung, bei der Leistungsanalyse von Protokollen.

Hingegen wird eine Detaillierung der relativ grob modellierten Architektur der Datenbankverwaltungs- und Kommunikations-Software einer späteren Modellverfeinerung vorbehalten bleiben.
Ein Vergleich unserer Ergebnisse mit anderen Untersuchungen ist als Folge unseres realistischen Modells wenig sinnvoll. Z.B. hat die in den meisten Studien angenommene Vollreplikation aller Daten zur Folge, daß die Anzahl Subtransaktionen stets gleich der Knotenanzahl ist, unabhängig von Umgebungsbedingungen. In unserem Modell kann aber gerade diese Anzahl bei unterschiedlichen Bedingungen stark variieren. Die Anzahl Subtransaktionen hat einen wesentlichen Einfluß auf die Performance (vergl. analytisches Modell in Kap. 7.6).

8.2 Experimententwurf

Nach der Erstellung des (VDBS-)Simulationsmodells ist ein systematischer Experimententwurf notwendig, auf dessen Grundlage die Experimente auszuführen und auszuwerten sind /JOWE84/.
Sei $\{P_1, \ldots, P_n\}$ die Menge der Eingabeparameter des Modells, wobei jeder Parameter P_i durch m(i) zu variierende Werte

$$\text{P-Werte}(i) := \{x_{1(i)}, \ldots, x_{m(i)}\}$$

definiert ist.
Sei $\{Z_1, \ldots Z_m\}$ die Menge der Zielgrößen mit reellem Wertebereich, deren Beziehungen f_p, $p \varepsilon [1,m]$, zu den Parametern

$$Z_p = f_p(x_1, \ldots, x_n)$$

dem Modellierer nicht bekannt und durch Simulationsexperimente zu schätzen sind. Dann ist ein **Einzelexperiment** X_j vorgegeben durch das n-Tupel von Parameterwerten

$$X_j = (x_{j1}, \ldots, x_{jn}), \qquad x_{jk} \in \text{P-Werte}(k),\ k \in [1,n]$$

Bei stochastischer Simulation ist jeder nichtkonstante P-Wert x_{jk} als Mittelwert eines Zufallszahlen-Stroms aufzufassen. Jedem P-Wert wird für ein Experiment genau ein Anfangswert dieses Stroms zugeordnet. Pro Einzelexperiment kann es mehrere Simulationsläufe geben, bei denen lediglich diese stochastischen Anfangsbedingungen geändert werden.
Bei einem Experiment X_j wird in einem Zeitintervall Δt für jede Zielgröße eine Zeitreihe

$$\{Z_{qj}(t) \mid t \in \Delta t\}, \qquad q \in [1,m]$$

beobachtet, deren jeweiliger Mittelwert eine Schätzung der Zielgröße Z_q an dem durch X_j vorgegebenen Parametertupel darstellt.

Ein **Experimentplan** aus r Experimenten

$$\{X_{j1}, \ldots, X_{jr}\}$$

legt die zu untersuchenden Parameterkombinationen fest. Dabei sind zwei extreme Strategien denkbar (vergl. /KLEI74/):

(1) **Fakultativ-Entwurf**: Es wird jede mögliche Parameterkombination untersucht, d.h.

$$\text{Anzahl Einzelexperimente} = \prod_{i=1}^{n} |\text{P-Werte}(i)|$$

(2) **Einzelvariations-Entwurf**: Es wird jeder Parameter gesondert variiert; alle anderen Parameter werden dabei auf "charakteristischen" Werten festgehalten, d.h.

$$\text{Anzahl Einzelexperimente} = \sum_{i=1}^{n} |\text{P-Werte}(i)|$$

Eine Anwendung des Einzelvariations-Entwurfs erfolgte in einigen Untersuchungen im VDBS-Bereich, z.B. in /OZSU83/. Allerdings sind damit Parameter-Interaktionen (ohne zumindest partielle Anwendung der Fakultativ-Methode) nicht nachweisbar. Zudem ist nicht klar, wie "charakteristische" Parameterwerte zu wählen sind, man denke etwa an

die Rechnerknotenanzahl als Parameter. Wenn man in der Entwurfsphase eines Systems noch zu wenig über mögliche Wechselwirkungen zwischen den Eingabeparametern weiß, ist der fakultative Entwurf der statistisch gesehen einzig sinnvolle. Andererseits scheidet er bei komplexen Systemen mit großem Werteraum aus Aufwandsgründen aus.

Tabelle 8.1 zeigt die im vorigen Abschnitt definierten primären Enwurfsparameter unseres Simulationsmodells mit den zu variierenden Werten entsprechend Anhang A.

PARAMETER	WERTEBEREICH	ANZAHL	BEDEUTUNG
(P1) TCO	2,3,5,8,12,20	6	Terminalanzahl pro Knoten
(P2) TG	0.02 , 0.05	2	End-to_end Delay
(P3) E	50, 200, 400	3	Anzahl Dateneinheiten
(P4) C	1 , 3	2	Anzahl Kopien pro Dateneinheit
(P5) BS	klein,mitt.,groß	3	Mächtigkeit der Basismenge
(P6) LOC	0.7, 0.8, 0.9	3	Lokalität
(P7) WR	0.2, 0.5, 0.8	3	Update-Häufigkeit
(P8) PROT	ALL, PRIM, MAJ	3	Transaktionsverarb.-Protokoll
(P9) STATE	Normalfall, CPU-, NIU-Crash	3	Globaler Systemzustand

Tabelle 8.1: Primäre Simulationsparameter des VDBS-Modells

Die gewählten Parameter und deren Werte stellen bereits Minimalforderungen dar, um einen differenzierten Protokollvergleich zu erlauben. Dabei wurde vorab bereits eine Reduktion des Parameterraums erzielt, wie ein Vergleich mit obigen Sekundärparametern ergibt:

- Alle CPU- und Plattenverarbeitungszeiten werden konstant gehalten,
- Datenverteilungs-Strategie wird fixiert,
- Die Anzahl zu betrachtender Crashfälle **wird minimiert** durch die Näherungsmethode (Kap. 5.2) unter Annahme homogener Last-, Daten-Kommunikationsleitungs- und Funktionsverteilungen im Rechnernetz.

Eine Anwendung des fakultativen Entwurfs würde entsprechend obigem Parameterraum zu über 17000 Experimenten führen. Eine weitere Reduktion des Parameterraums erhält man durch folgende Überlegungen:

- In Kapitel 7 sind drei zu untersuchende Transaktions-verarbeitungs-Protokolle spezifiziert worden, die sich in ihrem funktionalen und Leistungsverhalten lediglich in der Behandlung redundanter Daten unterscheiden. Im Fall C=1 ist diese Behandlung bei allen Protokollen gleich, d.h. dann reicht die Untersuchung

irgendeines Protokolls aus.

- Die Protokolle "PROT_ALL" und "PROT_PRIM" unterscheiden sich bei Ausschluß von Ausfällen ebenfalls kaum in ihrem Leistungsverhalten, und zwar unabhängig von der Datenredundanz. (Diese Behauptung hat sich in Pilot-Experimenten bestätigt.)

Der Werteraum für die Parameter PROT, C und STATE wird daher auf folgende Kombinationen reduziert:

STATE	PROT	C	PARAMETER
Normalfall	ALL	1	C1
	ALL	3	C3a
	MAJ	3	C3c
CPU-Crash, NIU-Crash	ALL	1	C1
	ALL	3	C3a
	PRIM	3	C3b
	MAJ	3	C3c

Die angegebenen Parameter C1 bis C3c sind eine Aggregierung der Parameter C und PROT. Der so reduzierte Parameterraum ergibt bei Anwendung des fakultativen Entwurfs ca. 10000 Experimente, was aus Effizienz- und Zeitgründen nicht ausführbar ist.

Folgende Alternativen führen aus unserem Dilemma. Zunächst existieren Kompromisse zwischen der Fakultativ- und der Einzelvariations-Methode, wie etwa in /KLEI74/ gezeigt wird. Dabei wird eine Teilmenge des Parameterraums untersucht, deren Wahl aber Zusatzkenntnisse über den Trend der Parameterwirkungen verlangt. Zudem beschränken sich die Methoden meist auf den Spezialfall von genau zwei Werten pro Parameter.
Vielversprechender für unsere Problemklasse erscheint die Anwendung der **Group-Screening** Methode /KLEI82/, die im wesentlichen eine hierarchische, iterative Vorgehensweise empfiehlt:

(1) Unwesentliche Parameter werden a priori fixiert (wie oben ausgeführt).
(2) Alle diejenigen primären Einzelparameter, deren Einfluß sich in "die gleiche Richtung" bei allen Outputgrößen auswirkt (nach Annahme des Modellierungs-Experten), werden jeweils zu einem aggregierten Gruppen-Parameter zusammengefaßt. Werte dieses

Parameters werden durch **gleichzeitige Änderung** von Werten der Einzelparameter gebildet; typisch sind zwei Werte pro Gruppe.

(3) Ist der Trend eines Parameters nicht bekannt, bildet er selbst einen Gruppen-Parameter der Mächtigkeit 1.

(4) Mit den verbleibenden Gruppenparametern werden alle Experimente ensprechend dem Fakultativ-Entwurf ausgeführt. Stellt sich dabei heraus, daß der Einfluß eines Gruppenparameters auf alle Zielgrößen gering ist, wird dieser von der weiteren Betrachtung ausgeschlossen, unter der Annahme, daß alle darin zusammengefaßten Einzelparameter ebenfalls geringen Einfluß haben.

(5) Nur Gruppenparameter mit signifikantem Einfluß werden wieder zu Einzelparametern verfeinert.

In unserem VDBS-Modell bieten sich die Größen E, BS, WR und LOC zur Gruppenbildung an, auf der Basis folgender

Annahme:

Der mittlere Durchsatz steigt; die mittlere Antwortzeit sinkt bei

- steigender Anzahl Dateneinheiten E (wegen geringer Konfliktwahrscheinlichkeit, wenn die Transaktionslast gleich bleibt)
- sinkender Mächtigkeit der Basismenge BS (wegen kleineren Subtransaktionen)
- sinkender Update-Häufigkeit WR (wegen geringerer Konfliktwahrscheinlichkeit, da konkurrierende Lesezugriffe nebenläufig ablaufen können)
- steigender Lokalität LOC (wegen kleineren Subtransaktionen)

Wir führen daher den Gruppenparameter **Datenbank-Anforderung** ein:

DB-ANFORDERUNG	E	BS	WR	LOC
hoch	50	groß	0.8	0.7
mittel	200	mittel	0.5	0.8
niedrig	400	klein	0.2	0.9

Eine update-intensive Umgebung, in der viele große, z.T. globale Transaktionen auf relativ wenige Daten zugreifen wollen, stellt hohe (Leistungs-)Anforderungen an das Datenbank-Verwaltungssystem. Umgekehrt sind die Systemanforderungen wesentlich geringer, wenn in

einer query-intensiven Umgebung viele kleine, meist lokale Transaktionen auf einer großen Anzahl Dateneinheiten arbeiten.
Der verbleibende Parameter des Datenprofils, die Kopienanzahl C, ist hingegen kein geeigneter Kandidat zur Gruppenbildung, da über den Performance-Trend keine eindeutigen Vorhersagen möglich sind.

Aus dem so reduzierten Parameterraum resultieren 396 Experimente, davon 108 im Normalfall. Die Normalfall-Experimente wurden vollständig ausgeführt und sind im nächsten Abschnitt beschrieben. Bei der Untersuchung der Crash-Fälle erfolgte auch eine Verfeinerung der Datenbank-Anforderungen, um den Einfluß der Einzelparameter in interessanten Einzelfällen aufzuzeigen, ohne daß ein Anspruch auf vollständigen Entwurf erhoben wird.

Alle nachfolgend beschriebenen Experimente wurden entsprechend folgender **Vorgehensweise** ausgeführt und ausgewertet. Zielgrößen sind Antwortzeit, Durchsatz, Erfolgsrate, Konfliktrate und Anzahl Subtransaktionen (vergl. Kap. 5). Gebildet werden Mittelwerte für jeden dieser Leistungsmaße; diese beziehen sich auf **alle Transaktionen** auf **allen Rechnerknoten**. Ferner werden Konfidenz-Intervalle und empirische Verteilungsfunktionen für die wichtigste Größe aus Benutzersicht, die Antwortzeit, gebildet.
Der DAEMON-Simulator liefert meist nichtaggregierte Größen pro Rechnerknoten und zusätzlich netzbezogene Beobachtungen, z.B. von Warteschlangen-Längen und Ressourcen-Auslastungen (in Anhang C ist ein vollständiger Simulations-Output dokumentiert). Wir beschränken uns aber meist auf die aggregierten Größen, um unsere Protokolle einfacher vergleichen zu können.
Um den statistischen Fehler bei der Mittelwertschätzung mit dem Ziel des Protokollvergleichs klein zu halten, wurden alle Experimente unter **gleichen Versuchsbedingungen** ausgeführt:

- Alle Zufallszahlenströme wurden durch unterschiedliche, "weit entfernte" initiale Saatkörner erzeugt, die selbst als Zufallszahlen aus genau einem statischen Saatkorn generiert wurden. Damit lag bei allen Experimenten z.B. dieselbe, stochastische Datenverteilung vor.
- Bei allen Versuchen wurden dieselbe Anzahl T1 terminierter Transaktionen beobachtet, nachdem die ersten T0 terminierten

Transaktionen eliminiert wurden unter der Annahme, daß das System sich danach im stationären Zustand befindet. Nach Auswertung einiger Pilotversuche wurde T0 = 120, T1 = 720 gewählt. T1 Transaktions-Beobachtungen wurden in Blöcke gleicher Länge aufgeteilt, die nach der Batch-Means Methode (Kap. 5.2) zur Berechnung von Konfidenzintervallen herangezogen wurden. Als Blockgröße wurden vierzig Transaktionen gewählt.

Eine abgesicherte Entscheidung über die Rangordnung der Protokolle ist trotz dieser Maßnahmen nur dann möglich, wenn die Mittelwert-Differenzen nicht zu klein sind.

8.3 Experimente unter Ausschluss von Ausfällen

In der ersten Experimentserie wurde das Leistungsverhalten der Protokolle

- C1 : Vollzustimmungs-Protokoll bei nichtredundanter Datenhaltung
- C3a : Vollzustimmungs-Protokoll bei 3 Kopien pro Dateneinheit
- C3c : Mehrheits-Protokoll bei 3 Kopien pro Dateneinheit

im Normalfall (keine Ausfälle) bei unterschiedlicher Last untersucht. Neben der Datenredundanz wird auch das Verhalten bei unterschiedlichem Rechnernetztyp, repräsentiert durch die Verzögerungszeit TG, und unter verschiedenen Datenbank-Anforderungen betrachtet.

Diese Anforderungen führen in der Transaktions-Vorverarbeitungsphase zur Generierung von Subtransaktionen unterschiedlicher Mächtigkeit, unabhängig von der Transaktionslast. Abb. 8.1 zeigt die experimentell bestimmte mittlere Anzahl Subtransaktionen NSUB. Diese Anzahl reicht von einer (lokalen) Subtransaktionen bei geringen DB-Anforderungen und Redundanzfreiheit, bis zu fünf Subtransaktionen bei hohen Anforderungen und drei Kopien.

Bei redundanter Datenhaltung und geringen DB-Anforderungen ist NSUB bei Protokoll C3a kleiner als bei C3c, da C3a im Normalfall wegen der geringen Update-Häufigkeit (WR = 0.2) lediglich auf eine Lese-Kopie zugreift, während C3c alle Lese-Kopien einer Dateneinheit sperrt.

Umgekehrt ist im Fall hoher DB-Anforderungen NSUB bei C3a und C3c in etwa gleich, da die Basismengen fast nur Updates enthalten (WR = 0.8) und beide Protokolle (im Normalfall) alle Update-Kopien sperren und ändern.

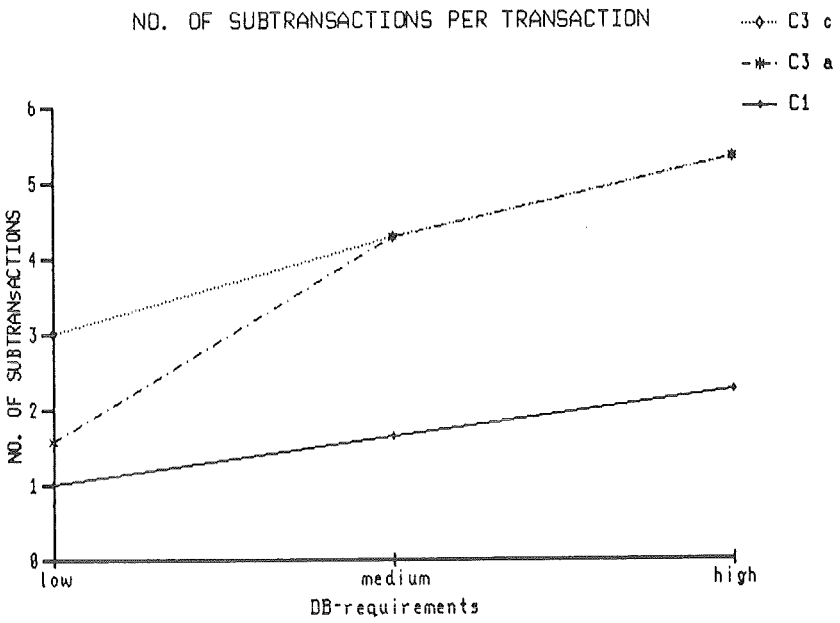


Abb. 8.1: Anzahl Subtransaktionen pro Transaktion bei unterschiedlichen DB-Anforderungen

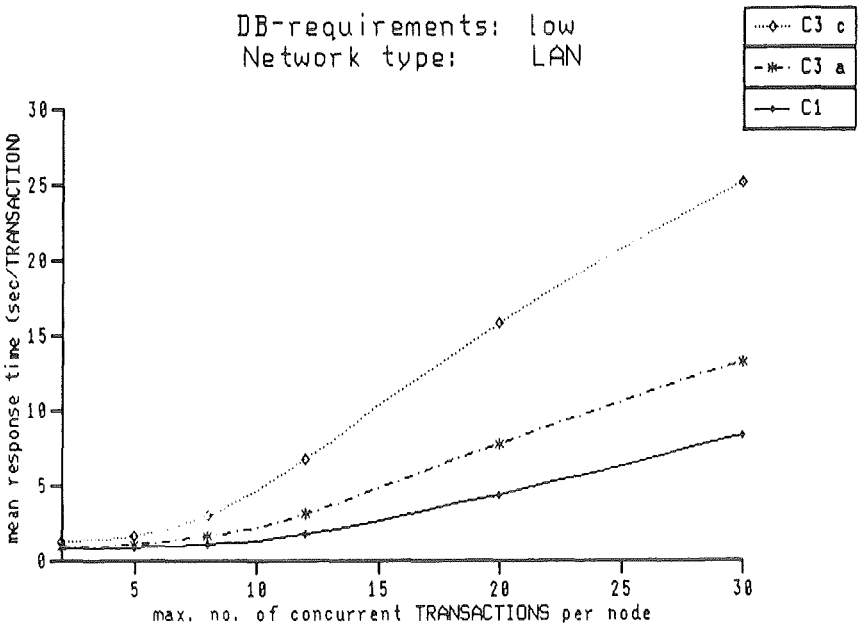
Dieses Verhalten wird zur Erläuterung der folgenden Ergebnisse herangezogen.

8.3.1 Einfluss der Transaktionslast

Zur Untersuchung des Einflusses der Transaktionslast wurde eine konstante Denkzeit D pro Terminal angenommen und die Terminalanzahl TCO pro Knoten in den Versuchen sukzessive gesteigert, bis keine Durchsatzverbesserung mehr zu beobachten war ('saturation'). TCO ist ungefähr proportional zur resultierenden Ankunftsrate von Transaktionen an einem Knoten und umgekehrt proportional zur Denkzeit. TCO beschränkt auch die maximale Anzahl aktiver Transaktionen pro (Heimat-)Knoten (vergl. Kap. 4.1).
Die aus den Beobachtungen geschätzten Mittelwerte von Transaktions-Antwortzeit und -Durchsatz sind in den Abb. 8.2 bis 8.4 dargestellt. Die Mittelwerte aller Zielgrößen wurden jeweils aus einem Experiment im Parameterraum

(TCO, C/PROT, DB-Anforderungen, Netztyp, Makrozustand)

(a)



(b)

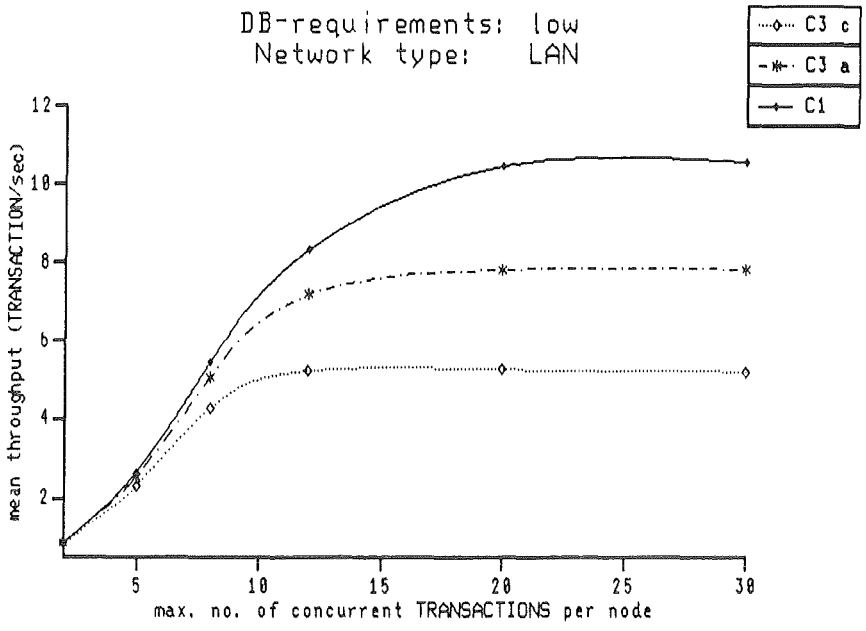
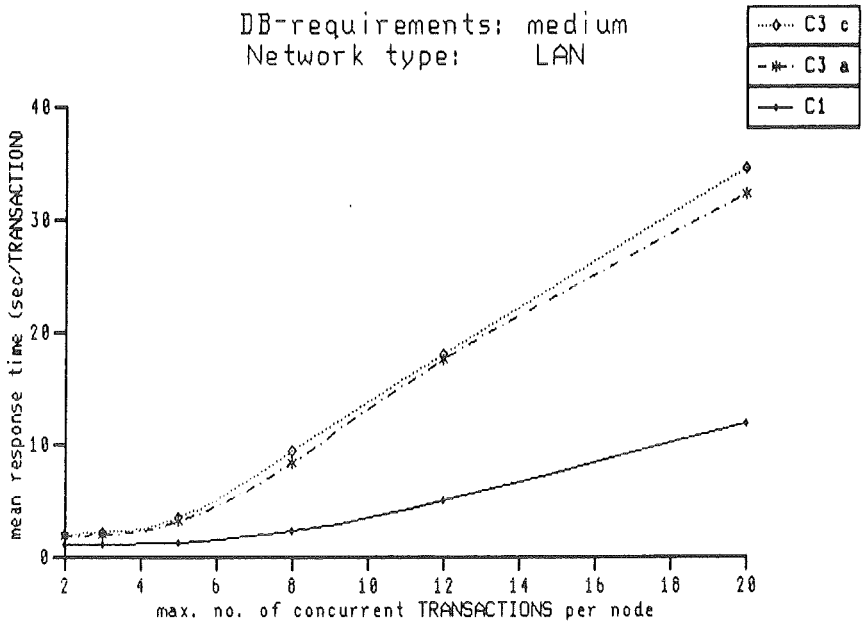


Abb. 8.2: Antwortzeit und Durchsatz bei unterschiedlicher Transaktionslast (DB-Anforderungen niedrig, Netztyp LAN)

(a)



(b)

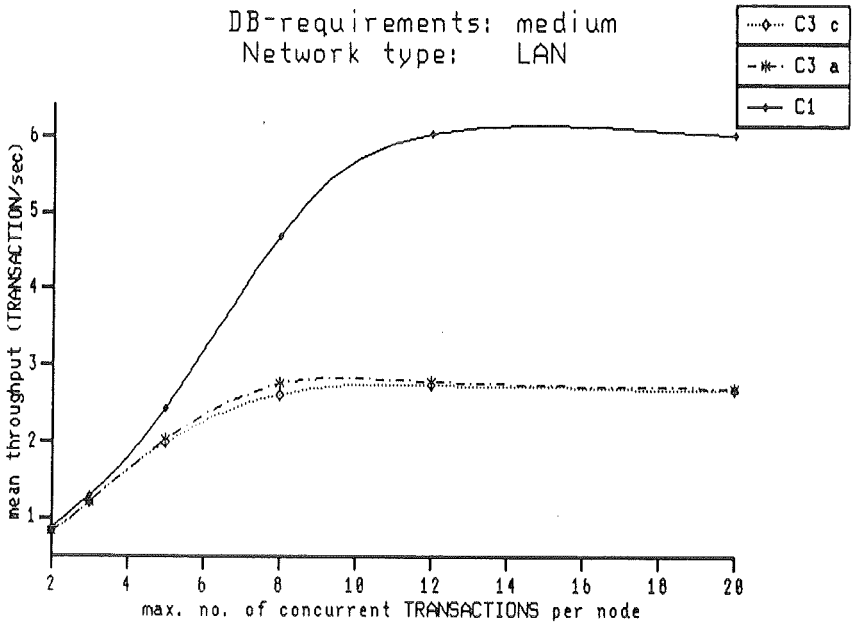
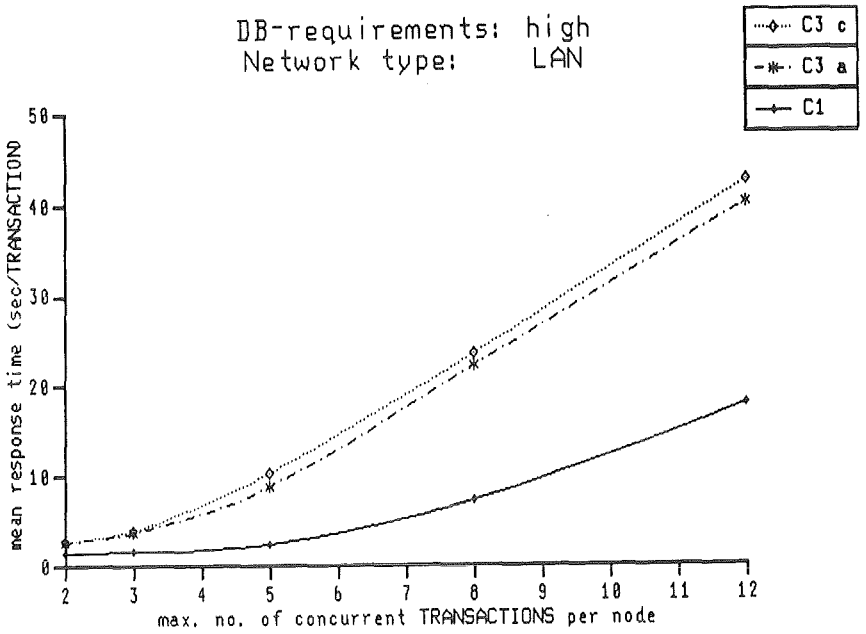


Abb. 8.3: Antwortzeit und Durchsatz bei unterschiedlicher Transaktionslast (DB-Anforderungen mittel, Netztyp LAN)

(a)



(b)

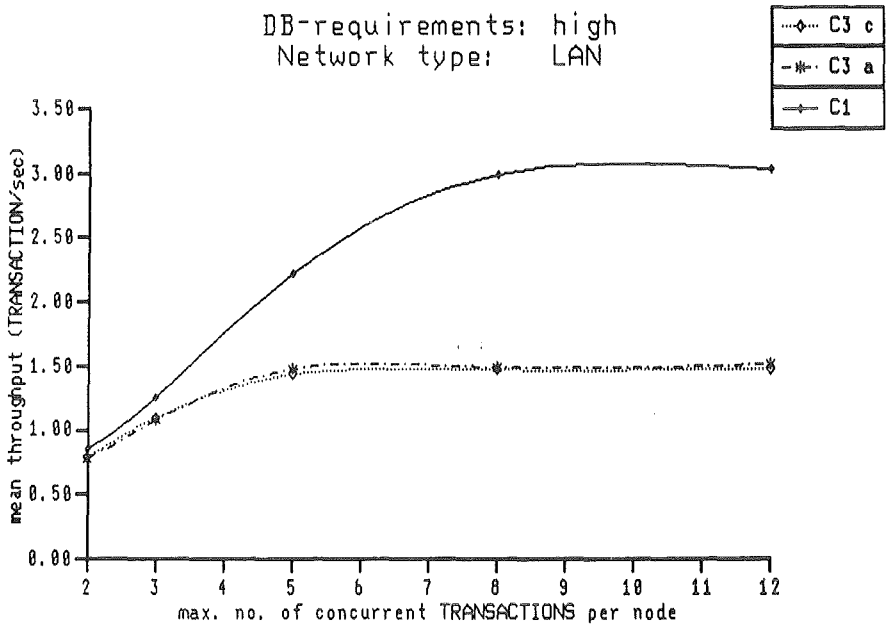


Abb. 8.4: Antwortzeit und Durchsatz bei unterschiedlicher Transaktionslast (DB-Anforderungen hoch, Netztyp LAN)

mit den im vorigen Abschnitt gegebenen Wertemengen bestimmt. Die gezeigten Kurvenfamilien sind durch polynomiale Approximation aus den Mittelwerten für jeweils ein Protokoll bei Variierung der Terminalanzahl TCO entstanden und werden mi Hilfe des IBM-Graphikpackets GDDM /IBM84/ präsentiert. Wir vergleichen zunächst die Protokolle für den Fall eines lokalen Rechnernetzes.

Niedrige DB-Anforderungen, lokales Rechnernetz (Abb. 8.2)

Bei sehr niedriger Transaktionslast (TCO≤5) verhalten sich alle untersuchten Protokolle annähernd gleich bezüglich Antwortzeit und Durchsatz. Die unterschiedliche Anzahl Subtransaktionen NSUB wirkt sich hier also nicht aus. Im Fall sehr hoher Last liefert Protokoll C1 einen doppelt so hohen Durchsatz wie Protokoll C3c. Das System ist bei Protokoll C1 erst bei sehr hoher Last voll ausgelastet (d.h. liefert maximalen Durchsatz), bei redundanter Datenhaltung ist das Maximum bereits bei mittlerer Last erreicht. Die Antwortzeit beträgt nur etwa 30% derjenigen von C3c, wobei der Gradient bei C3c größer ist. Protokoll C3b liegt in etwa zwischen beiden Extremen. Im Hochlastfall ist die Antwortzeit bei verschiedenen Protokollen in etwa proportional zur Anzahl Unteraufträge NSUB; der Durchsatz ist umgekehrt proportional.

Höhere DB-Anforderungen, lokales Rechnernetz (Abb. 8.3, 8.4)

Bei Steigerung der Anforderungen wird die Leistung erwartungsgemäß bei allen Protokollen für alle untersuchten Transaktionslasten schlechter (Durchsatz-Einbuße bei Übergang von niedrigen zu mittleren und von mittleren zu hohen Anforderungen jeweils 40% bis 50%).
Hauptursache dafür dürfte die teilweise dramatische Steigerung der Konfliktraten pro Transaktion sein, die zu längeren Wartezeiten vor dem verteilten Sperrscheduler führt. Dieses Verhalten zeigt Tabelle 8.2; wir gehen darauf unten noch genauer ein.
Es ist auch auch ein ähnliches relatives Verhalten zu beobachten wie bei den niedrigen Anforderungen. Allerdings unterscheiden sich die Antwortzeiten bereits bei fast leerem System (TCO = 2) signifikant, indem diese bei redundanter Datenhaltung ca. 80% höher liegen,

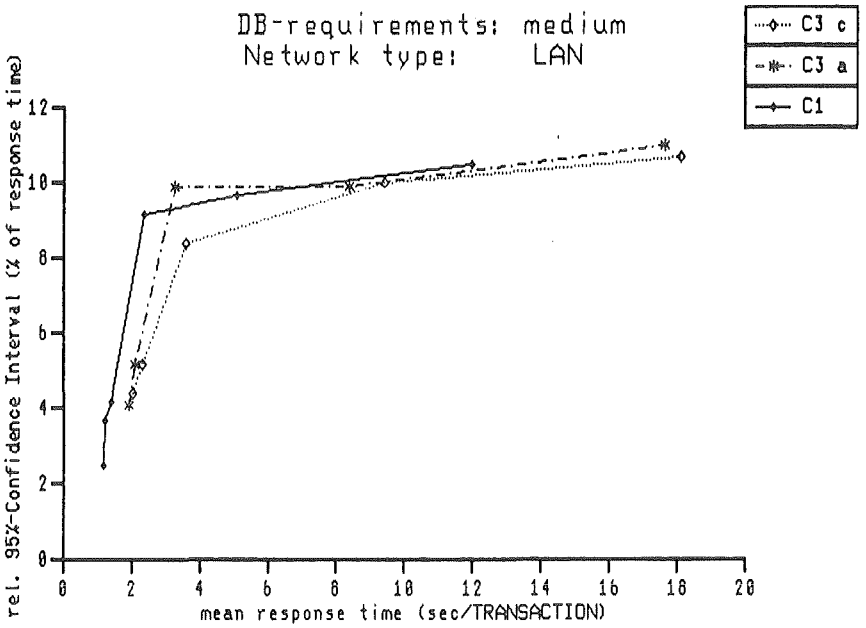
unabhängig davon, welches Protokoll gefahren wird.

Die geschätzten mittleren Antwortzeiten im Hochlastbereich suggerieren, daß Protokoll C3a etwas besser als C3c ist. Diese Aussage ist aber statistisch nicht haltbar, da die jeweilige Wertdifferenz innerhalb des 95%-Konfidenzintervalls beider Werte liegt. Wie Abb. 8.5-a zeigt, liegen die Intervalle um ca. 10% der geschätzten Werte, wenn der Absolutwert mindestens 4 Transaktionen pro Sekunde beträgt, d.h. bereits in einem niedrigeren Lastbereich. Höhere Antwortzeiten ergeben in unserem Modell eine größere Unsicherheit in der Mittelwertschätzung, was sich auch im Histogramm der Antwortzeiten zeigt (s. Beispiel in Abb. 8.5-b; dort ist die Verteilung relativ zum geschätzten Mittelwert aufgetragen). Die Streuung um den Mittelwert der Antwortzeiten nimmt mit dem Mittelwert zu.
Die Ursache dieses Verhaltens läßt sich auf die unterschiedlichen Konfliktraten zurückführen (vergl. Tab. 8.2). Eine geringe Transaktionslast führt zu (relativ) geringen Antwortzeiten und zu kleiner Konfliktrate; die Zeiten streuen nur wenig.Bei hoher Last sind die Konfliktraten wesentlich größer. Ein Teil der Transaktionen terminiert schnell ohne Konflikte, während ein anderer Teil auf das Entsperren von Daten wartet. Da bei hoher Last zusätzlich die Plattenspeicher sehr stark ausgelastet werden (s. Tab. 8.2), warten Transaktionen, die Sperren gesetzt haben, auch auf Plattenzuteilung. Damit erhöhen sich die Zeiten, in denen eine Dateneinheit gesperrt ist, was wiederum die Konfliktrate erhöht.
Andererseits sind die Konfidenzintervalle im Verhältnis zu den Performance-Unterschieden zwischen partionierter und replizierter Datenbasis so klein, daß folgender Schluß zulässig erscheint:

> Im Normalfall liefert Protokoll C1 in einer lokalen Rechnernetz-Umgebung eine wesentlich bessere Leistung als die Protokolle C3a und C3c.

Dieses ist das erwartete Ergebnis, da die auf redundanten Daten arbeitenden Protokolle gerade mit dem Ziel entworfen wurden, im Fehlerfall und evtl. sogar im Gesamtverhalten eine bessere Leistung zu liefern als Protokolle auf einer zwar verteilten, aber nichtredundanten Datenbasis.

a)



b)

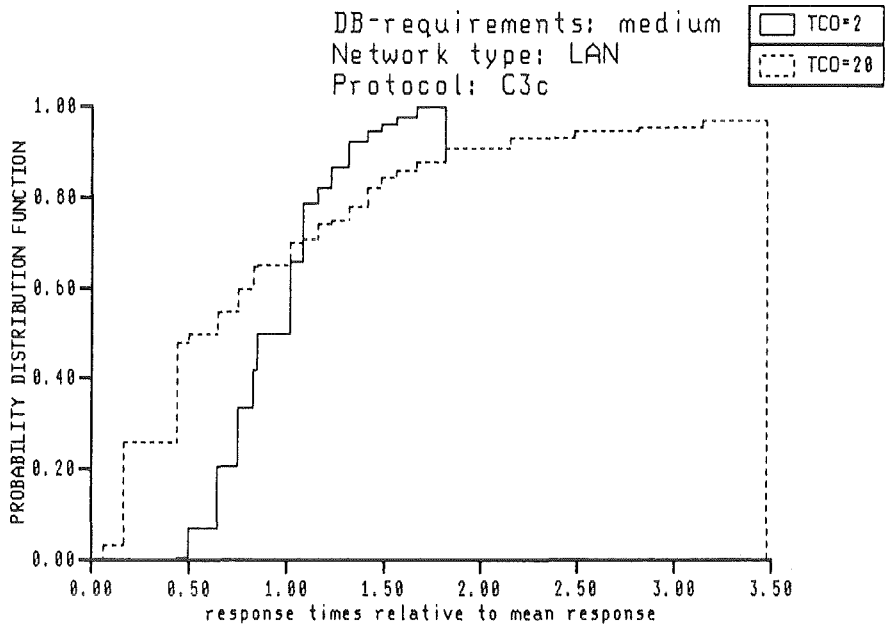


Abb. 8.5: Konfidenzintervalle und Verteilung der Antwortzeiten (DB-Anforderungen mittel, Netztyp LAN)

8.3.2 Einfluss des Daten- und Transaktionsprofils

Die bisherigen Ergebnisse bezüglich den Unterschieden in der Antwortzeit bei verschiedenen Daten- und Transaktionsprofilen, die sich im Parameter 'Datenbank-Anforderungen' widerspiegeln, sind in den Abbildungen 8.6-a bis 8.6-c zusammengefaßt.
Je höher die Anforderungen gesetzt werden, umso weniger zahlt sich redundante Datenhaltung im Normalfall unter den gegebenen Modellannahmen aus. Das relative Antwortzeit-Verhalten zwischen den Protokollen ist im Niederlastfall (3 Terminals pro Knoten, Abb. 8.6-a) ähnlich dem bei hoher Last (12 Terminals pro Knoten, Abb. 8.6-b).
Abb. 8.6-c beantwortet eine typische Frage an den Systemverwalter:

> Wieviele Terminals können höchstens pro Knoten installiert werden, damit die mittlere Antwortzeit pro Transaktion höchstens 5 Sekunden beträgt ?

Diese Anforderung gestattet bei niedrigen Anforderungen und Protokoll C1 ≤22 Terminals pro Knoten, bei hohen Anforderungen und redundanter Datenhaltung aber nur ≤4 Terminals. (Es ist zu beachten, daß diese Werte nur bei einer mittleren Denkzeit pro Terminalbenutzer von 15 sec gelten; bei höheren Denkzeiten würde sich die zulässige Terminalanzahl entsprechend erhöhen.)

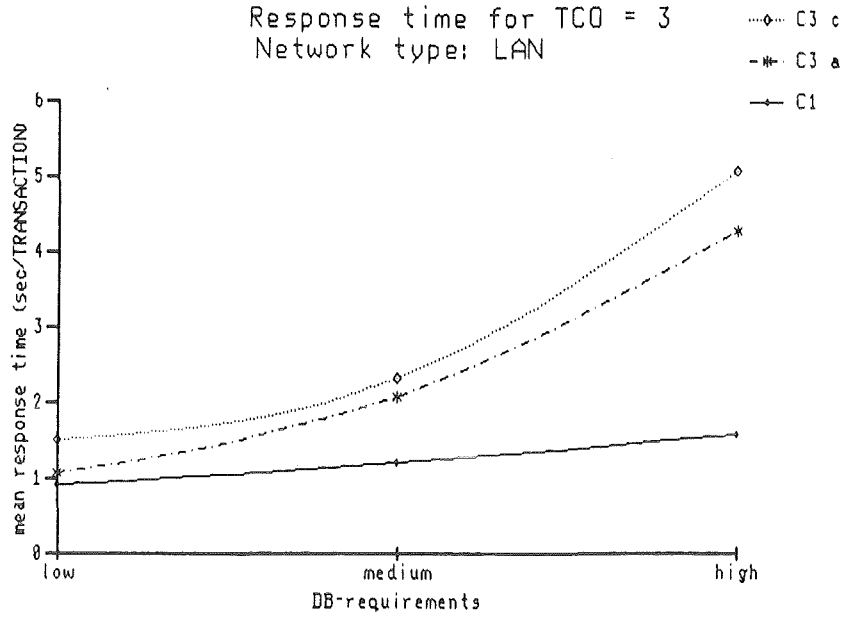
8.3.3 Einfluss des Rechnernetztyps

Das VDBS-Modellverhalten in einer DATEX-P-ähnlichen 'WAN'- (wide-area network) Umgebung bezüglich Durchsatz und Antwortzeit ist in den Abb. 8.7 bis 8.10 zusammengefaßt, die für alle sonstigen Parametern dem in den Abb. 8.2 bis 8.4 und 8.6 dargestellten Verhalten in einer LAN-Umgebung entsprechen.
Ein Vergleich des Protokollverhaltens in beiden Umgebungen ergibt:

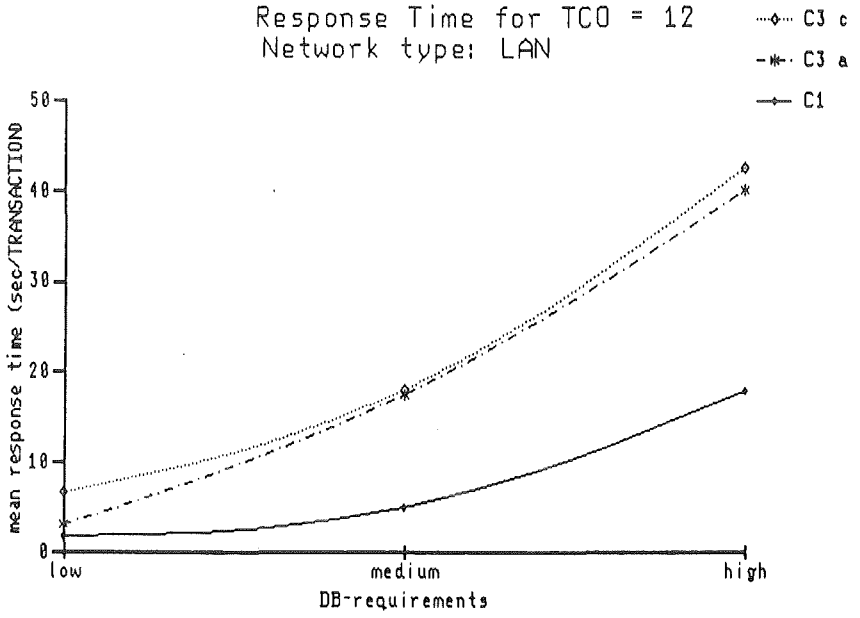
(1) Die relative Performance der Protokolle zueinander ist fast unabhängig vom gewählten Rechnernetz

Unterschiede sind insbesondere im Fall hoher DB-Anforderungen vorhanden, wenn man z.B. den Durchsatz in Abb. 8.4-b mit Abb. 8.9-b vergleicht: Die relative Durchsatz-Einbuße bei der Einführung von Kopien (Protokoll C3a, C3c) beträgt z.T. über 200% , während sie im LAN-Fall nur halb so groß ist. Dieser Unterschied wird bei Reduktion der DB-Anforderungen immer kleiner.

(a)



(b)



(c)

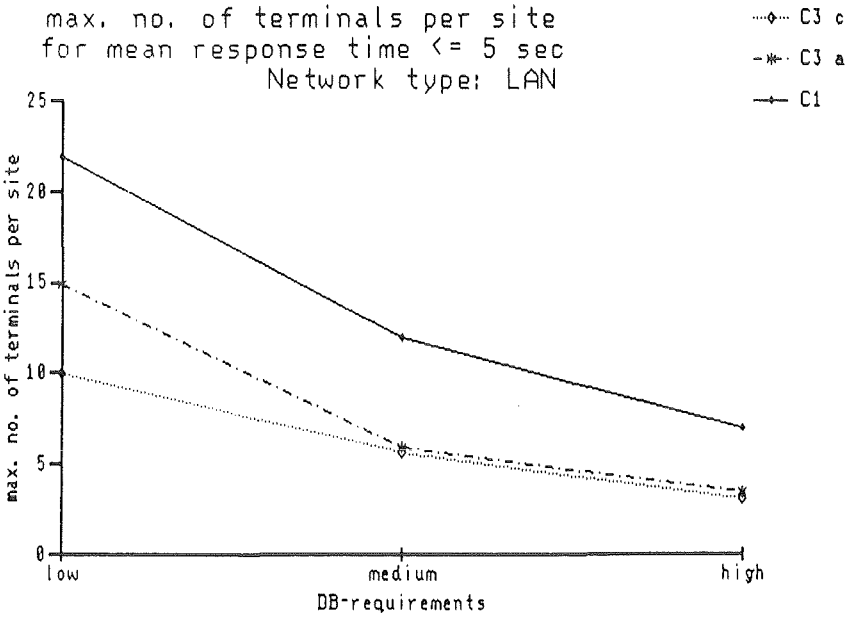
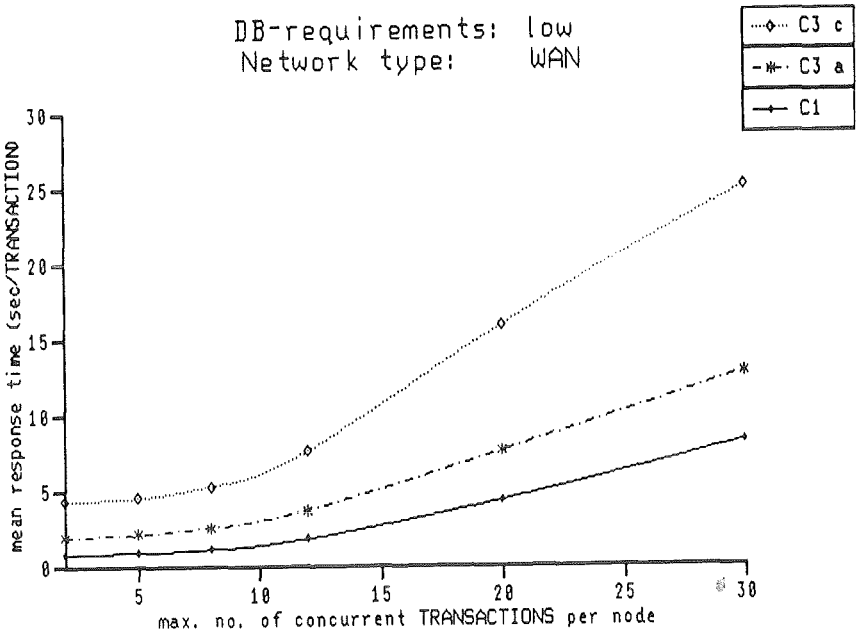


Abb. 8.6: Auswirkung der DB-Anforderungen an die Antwortzeit (Netztyp LAN)

(a)



(b)

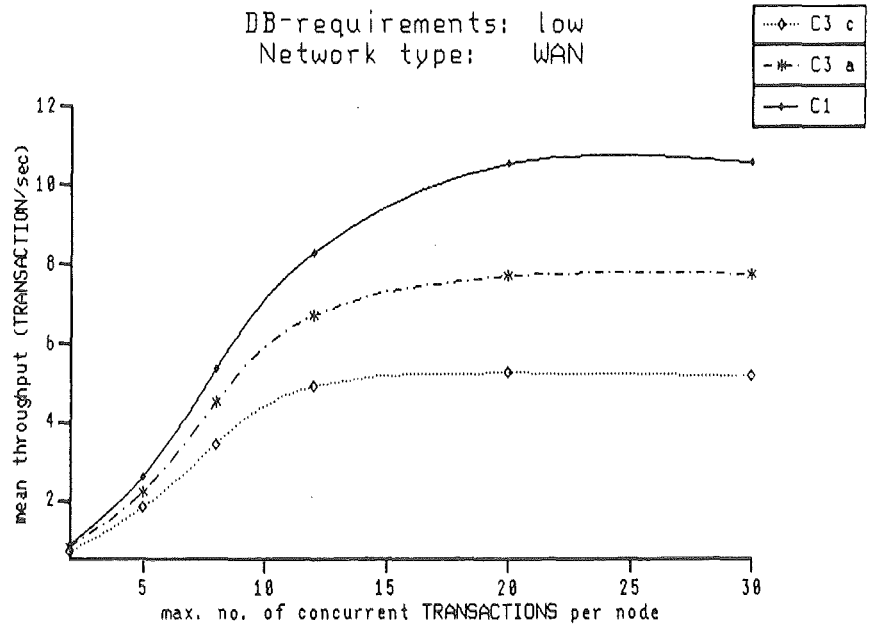
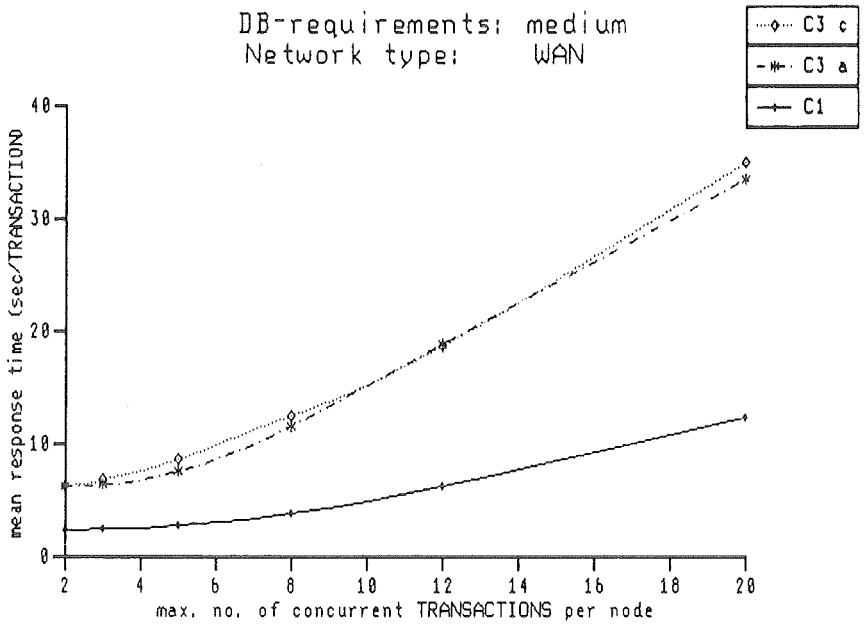


Abb. 8.7: Antwortzeit und Durchsatz bei unterschiedlicher Transaktionslast (DB-Anforderungen niedrig, Netztyp WAN)

(a)



(b)

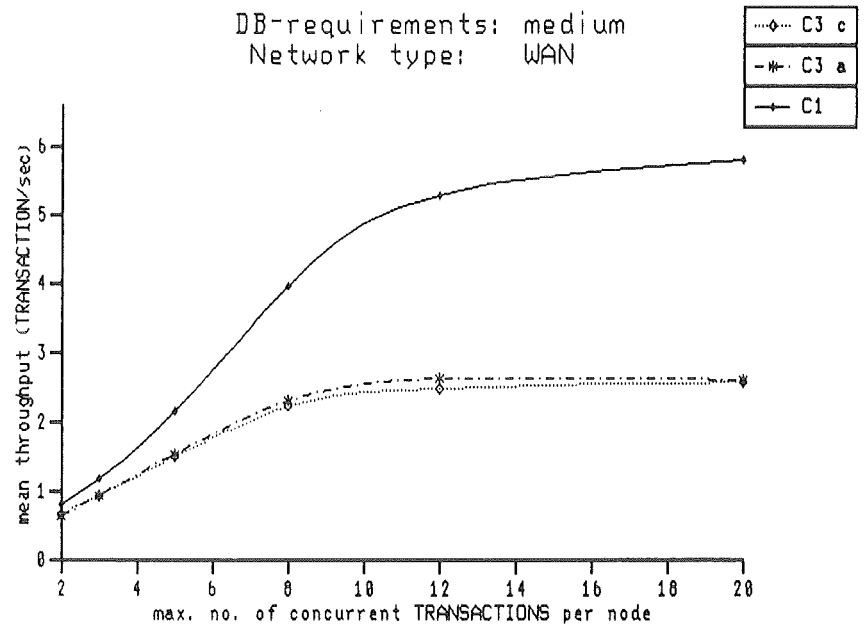
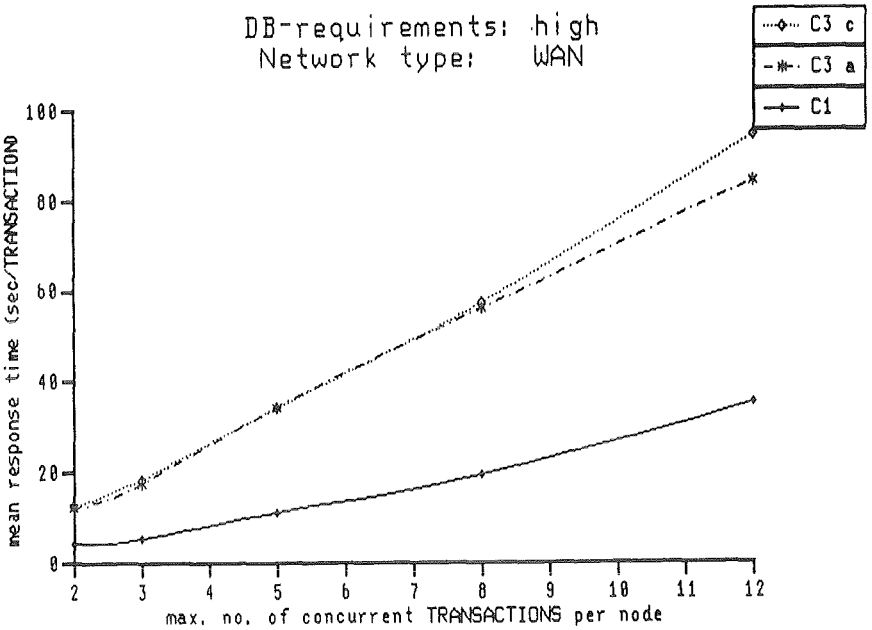


Abb. 8.8: Antwortzeit und Durchsatz bei unterschiedlicher Transaktionslast (DB-Anforderungen mittel, Netztyp WAN)

(a)



(b)

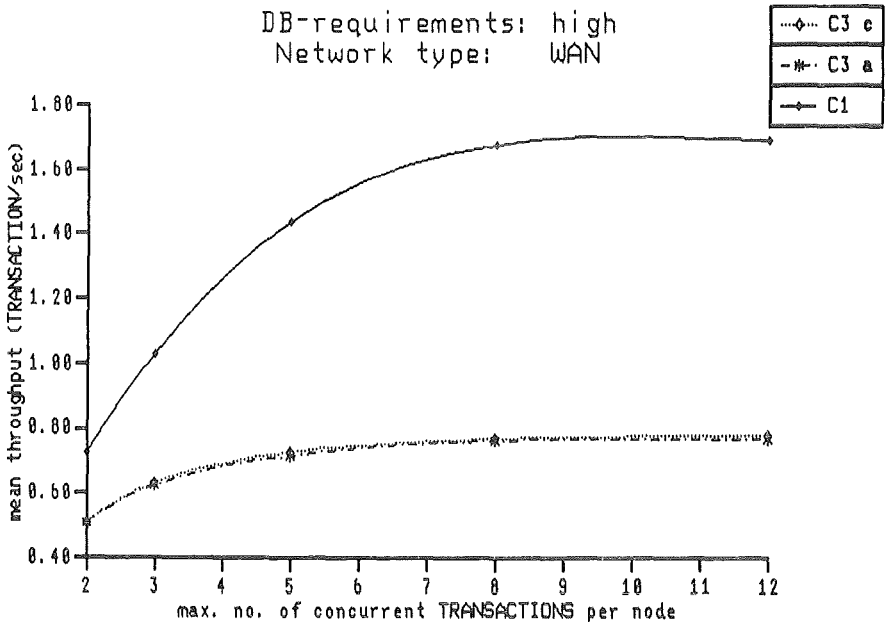
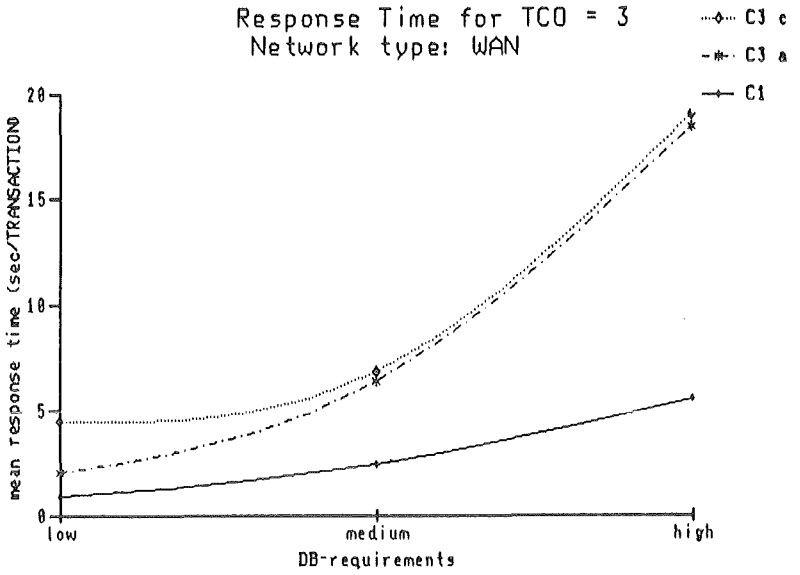
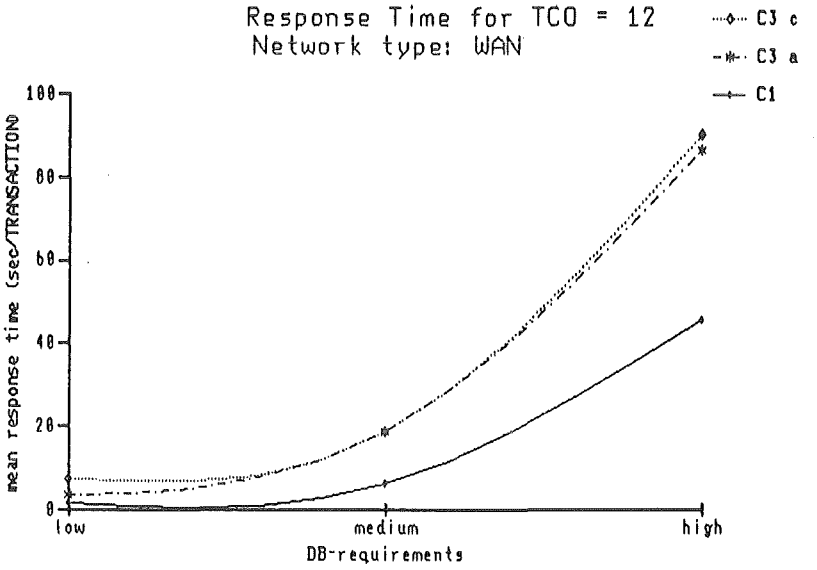


Abb. 8.9: Antwortzeit und Durchsatz bei unterschiedlicher Transaktionslast (DB-Anforderungen hoch, Netztyp WAN)

(a)



(b)



(c)

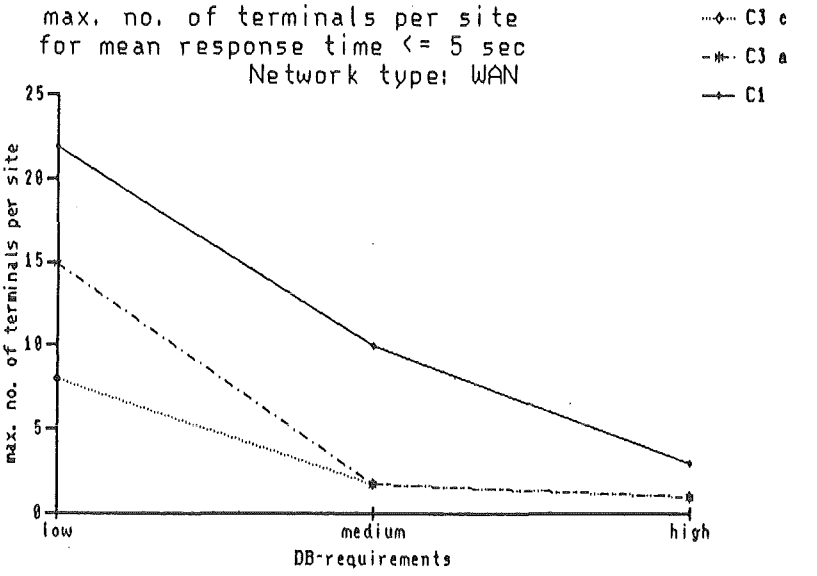


Abb. 8.10: Auswirkung der DB-Anforderungen an die Antwortzeit (Netztyp WAN)

Diese Beobachtungen werden durch die in Tab. 8.2 gezeigten Plattenauslastungen und Konfliktraten bestätigt: Die Unterschiede zwischen LAN- und WAN-Fall werden bei sinkenden DB-Anforderungen immer geringer.

(2) Die absolute Performance eines Protokolls bei niedrigen bis mittleren DB-Anforderungen und bei mittleren bis hohen Lasten ist annähernd unabhängig vom gewählten Rechnernetz.

Dieses Ergebnis ist bei einem Verhältnis der Verzögerungszeiten von 0.02 zu 0.50 sec zwischen LAN- und WAN-Umgebung doch erstaunlich. Als Ursache kommt im wesentlichen wieder die Vollauslastung der Plattenspeicher in Frage, die zu längeren Wartezeiten führt. Diese Wartezeiten werden ab einer gewissen hohen Auslastung größer als jede beschränkte Verzögerungzeit. Hinzu kommt, daß die Plattenauslastungen im LAN- bzw. WAN-Fall bei niedrigen bis mittleren DB-Anforderungen annähernd gleich sind.
Eine schlechtere Leistung im WAN-Fall ergibt sich also in einer Umgebung mit hohen DB-Anforderungen oder bei fast leerem System (im letzteren Fall differieren allerdings die Durchsatzwerte im WAN- wie im LAN-Fall für dasselbe Protokoll kaum, wie ein Vergleich von Abb. 8.6-a mit 8.10-a zeigt).

Die bisher beschriebenen Ergebnisse der Leistungsanalyse können allein aus dem funktionalem Verhalten der Protokolle nicht befriedigend erklärt werden. Im folgenden werden zur Interpretation die systemorientierten Größen **Konfliktrate** und **Auslastung** der Betriebsmittel herangezogen.

8.3.4 Einfluss der Konfliktraten und der Ressourcenauslastungen

Die Transaktions-Konfliktrate bestimmt diejenigen aktiven Transaktionen, die bei mindestens einer ihrer Subtransaktionen auf das Entsperren von Dateneinheiten warten, wenn diese eine mit der aktuellen Sperrmenge inkompatible Sperranforderung stellt. Diese wartenden Transaktionen verrichten keine sinnvolle Arbeit im System. Die Wartezeiten gehen als 'serialization delay' in die Gesamt-Antwortzeiten ein; sie sind nahezu proportional zu der

Konfliktrate, wie sich durch Detailergebnisse bestätigt hat. Eine hohe Konfliktrate ist einer der Faktoren, die für eine Verschlechterung der Antwortzeiten verantwortlich ist.

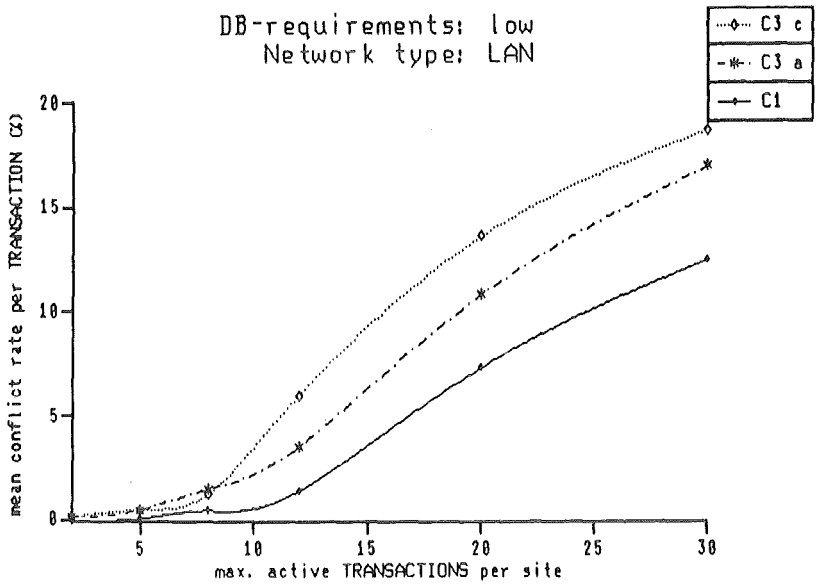
DB-Anf.	Protokoll	Platten-Auslastung im LAN	Platten-Auslastung im WAN	Konfliktraten im LAN	Konfliktraten im WAN
niedrig	C1	8 - 90	8 - 90	<0.1 - 13	0.2 - 15
	C3a	11 - 94	10 - 93	<0.1 - 16	0.2 - 19
	C3c	16 - 97	14 - 96	<0.1 - 18	0.2 - 21
mittel	C1	13 - 86	12 - 84	1 - 63	3 - 67
	C3a	28 - 90	22 - 85	2 - 66	10 - 67
	C3c	29 - 92	22 - 87	3 - 67	8 - 64
hoch	C1	19 - 60	10 - 25	15 - 90	31 - 88
	C3a	22 - 67	23 - 36	26 - 88	62 - 91
	C3c	38 - 67	23 - 35	25 - 88	64 - 88

Tabelle 8.2: Mittlere Plattenauslastungen und Transaktions-Konfliktraten bei niedriger bzw. hoher Last

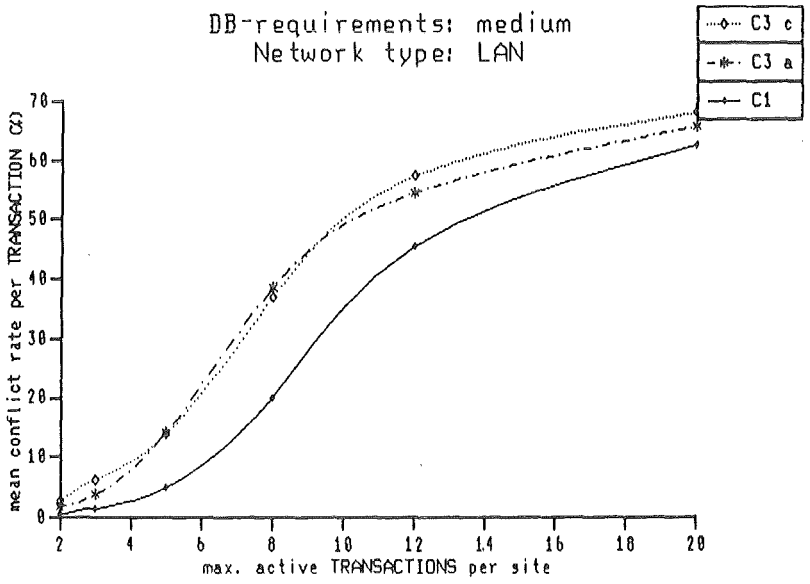
In Tab. 8.2 sind die Extremwerte der Konfliktrate bei fast vollem bzw. stark belastetem System dargestellt. Der Verlauf der Konfliktraten bei steigender Last zeigt Abb. 8.11. Dabei hat eine gleichzeitige Steigerung der Last als auch der DB-Anforderungen ein Ansteigen der Konfliktrate bis z.T. über 90% zur Folge, d.h. daß dann fast alle Transaktionen auf das Freigeben von Sperren warten müssen. (Diese Umgebung stellt daher für reale Systeme zu hohe Anforderungen.) Bei niedrigeren DB-Anforderungen ist die Menge sperrbarer Daten relativ groß; in diesem Fall steigt die Konfliktrate auch bei hoher Last kaum über 20% . Auch bei weiterer Laststeigerung kann dieser Wert kaum überschritten werden, da die Plattenspeicher als Engpaß-Ressourcen (s.u.) bereits voll ausgelastet sind. Bei hohen DB-Anforderungen sind Konflikte schon bei fast leerem System relativ häufig, da große Transaktionen auf einer kleinen Datenbasis arbeiten.

Durch Verteilung von Kopien werden Konflikte pro Transaktion häufiger, da im Mittel mehr Subtransaktionen Sperranforderungen stellen. Die Kopienbehandlungsprotokolle C3a und C3c haben aber im Mittel eine etwa gleiche Konfliktrate. Im Fall globaler Rechnernetze (WAN) und geringer Transaktionslast sind Konflikte häufiger als bei lokalen Netzen, da Sperren wegen den längeren Übertragungszeiten länger gesetzt bleiben,

(a)



(b)



(c)

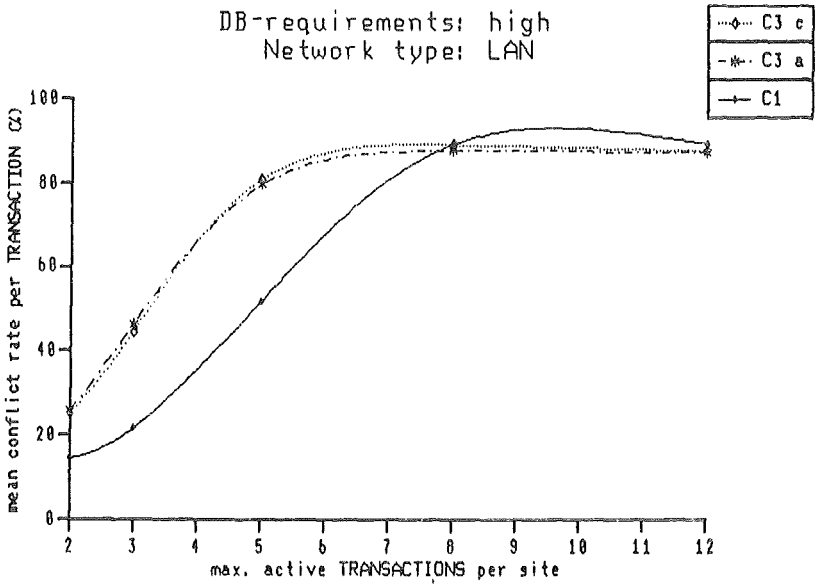
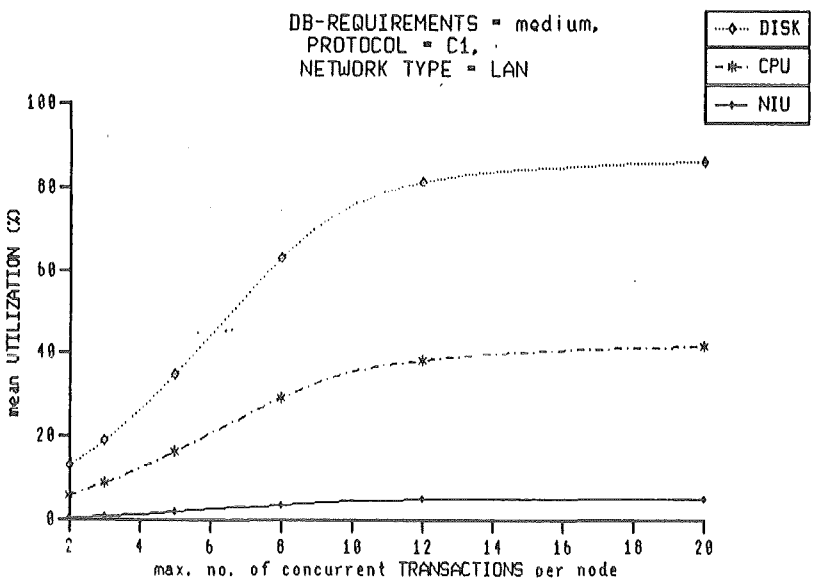
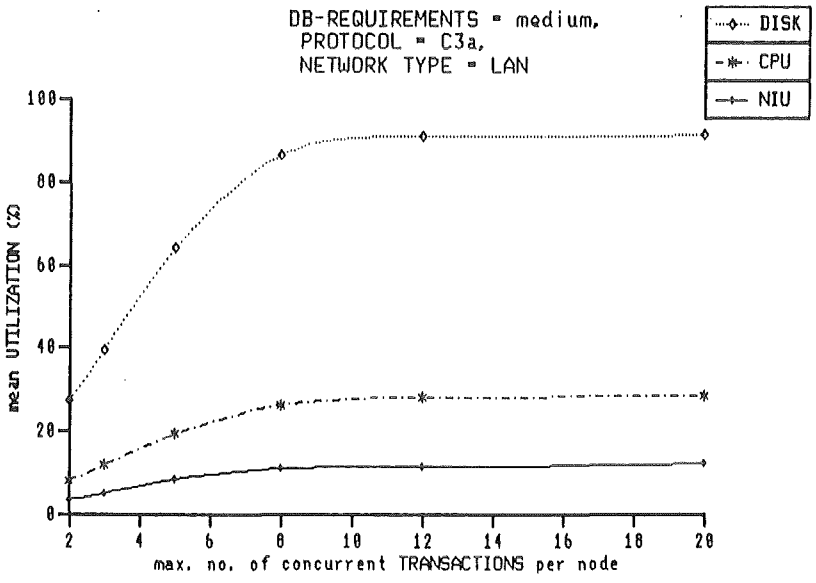


Abb. 8.11: Transaktions-Konfliktrate bei unterschiedlicher Transaktionslast (Netztyp LAN)

(a)



(b)



(c)

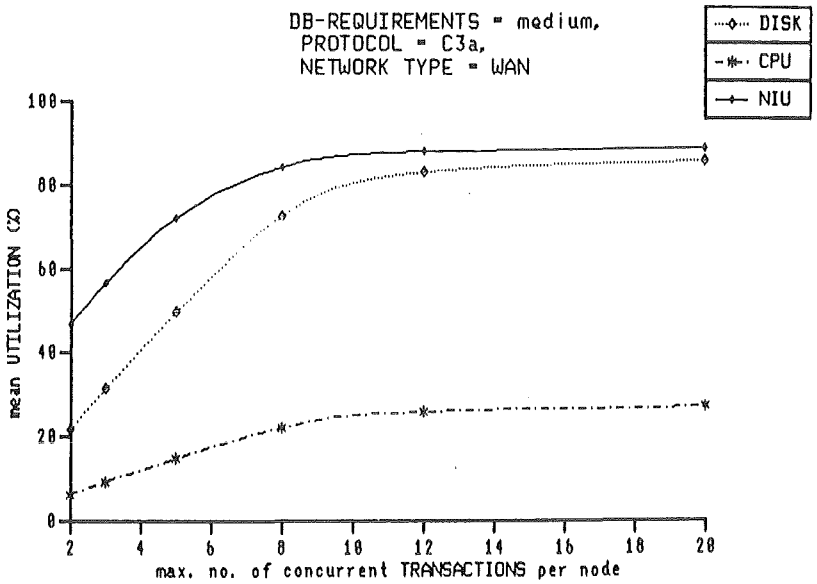


Abb. 8.12: NIU-, CPU- und DISK-Auslastung bei unterschiedlicher Transaktionslast (DB-Anforderungen mittel)

während alle Betriebsmittel wenig ausgelastet sind. Bei höheren Lasten werden diese Unterschiede zunehmend geringer, weil die Sperrdauer pro Dateneinheit durch die im LAN- wie auch im WAN-Fall sehr hohe Plattenauslastung bestimmt wird: Transaktionen, die die Sperrphase erfolgreich absolviert haben, benötigen im weiteren Verlauf mindestens drei Plattenzugriffe (bei großen Subtransaktionen entsprechend mehr) zur Ausführung der Subtransaktionen sowie zur Sicherung ihres Endzustands.

Als Engpaß bei hoher Transaktionslast haben sich bei allen Experimenten die Plattenspeicher (im Mittel über alle Knoten des Rechnernetzes) erwiesen. Dies ist an einigen Beispielen in Abb. 8.12 gezeigt. Im Netztyp WAN dominieren NIU-Auslastungen. Diese führen aber nicht zu höheren Wartezeiten, da die Verzögerungszeit einer Nachrichtenübertragung in unserem Modell lastunabhängig ist (Kap. 8.1.2). Lastabhängigkeit hätte sicherlich andere Ergebnisse zur Folge. Bereits bei niedrigen bis mittleren DB-Anforderungen und hoher Transaktionslast werden die Platten voll ausgelastet (ca. 90%). Bei mittleren Anforderungen ist diese maximale Plattenauslastung für Protokoll C1 bzw. C3a bei TCO = 12 bzw. TCO = 8 Terminals pro Knoten erreicht (Abb. 8.12-a bzw. 8.12-b). Das entspricht exakt der Last, bei der für diese Protokolle die Obergrenze des Durchsatzes erreicht ist (Abb. 8.3-b). Dieses Ergebnis ist ein wichtiges Indiz für die **interne Konsistenz** unserer Beobachtungen.
Tendenziell fällt die Auslastung bei steigenden DB-Anforderungen. Grund dafür dürften die hohen Konfliktraten sein, die zu längeren Transaktions-Blockierungen führen.

8.3.5 Zusammenfassung und Folgerungen

In den präsentierten Ergebnissen haben sich Unterauftragsgrößen, Konfliktraten und Plattenauslastungen als wesentliche Faktoren bei der absoluten Leistung der untersuchten Protokolle unter Auschluß von Ausfällen erwiesen. Die beobachteten, teilweise großen Leistungseinbußen bei steigender Transaktionenlast ließen sich allerdings durch verschiedene Maßnahmen vermeiden, z.B.

- Durch eine größere Granularität bei den zu sperrenden Daten könnte

die Konfliktrate gesenkt werden.

- Durch Erweiterung der Hardware-Konfiguration, z.B. durch zusätzliche Plattenspeicher oder durch Einsatz von Datenbank-Rechnern, könnte das System in seiner Auslastung balanciert und der größte Engpaß bei Plattenzugriffen beseitigt werden.

Solche Maßnahmen würden zu unterschiedlichem Architektur- und Umgebungs-Submodell führen, die im Rahmen dieser Arbeit nicht betrachtet werden.

Das Protokoll auf partitionierter Datenbasis liefert über ein weites Spektrum von Umgebungsbedingungen eine bessere Leistung als die untersuchten Protokolle auf redundanten Daten. Lediglich in einer Umgebung mit geringer Transaktionslast oder niedrigen DB-Anforderungen sind diese Protokolle konkurrenzfähig, wobei die Gültigkeit dieser Aussage sich auf ein ausfallfreies System beschränkt.

Die relative Leistung der Protokolle hat sich unter Ausschluß von Ausfällen als annähernd unabhängig sowohl von der Charakteristik des Rechnernetzes als auch von den Datenbank-Anforderungen erwiesen. Daher reduzieren wir im folgenden den Parameterraum, indem nur noch

- Netztyp: LAN,
- DB-Anforderungen: mittel

bei der folgenden Einbeziehung von Ausfällen untersucht werden. Speziell der Fall der angenommenen hohen DB-Anforderungen erscheint wegen den sehr hohen Konfliktraten für den Entwurf realer VDBS kaum relevant zu sein. Die Übertragung der folgenden Ergebnisse auf den Fall niedriger DB-Anforderungen ist allerdings nicht immer zulässig. Dieses Problem wird umgangen, indem für interessante Einzelfälle die DB-Anforderungen bei gegebener Last verfeinert werden. Insbesondere interessiert dabei der Einfluß der Einzelfaktoren Update-Häufigkeit und Lokalität.

8.4 Experimente mit Einbeziehung von Ausfällen

Bei diesen Experimenten wurden mittlere Durchsatz- und Erfolgsraten-Werte für die wahrscheinlichsten Makrozustände beobachtet und mit analytischen Verfügbarkeits-Berechnungen entsprechend den

Schätzformeln von Kap. 5.4 kombiniert.

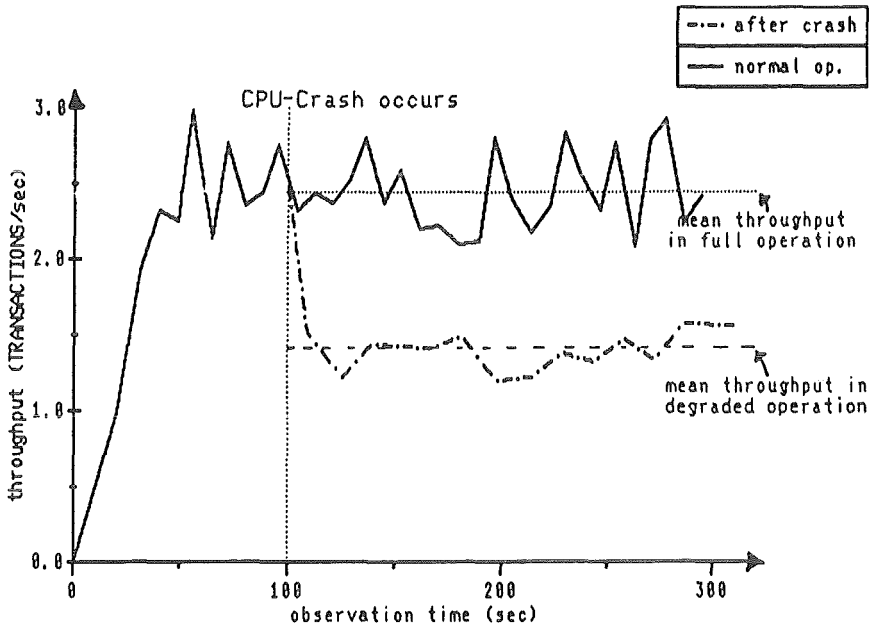


Abb. 8.13: Durchsatzbeobachtung nach jeweils 20 terminierten Transaktionen über der Zeit (Beispiel)
- bei System ohne Ausfall
- bei System mit einem CPU-Ausfall nach 100 sec

(Protokolltyp C3a, DB-Anforderungen mittel, Netztyp LAN)

Den prinzipiellen zeitlichen Verlauf des Transaktions-Durchsatzes im gesamten Rechnernetz veranschaulicht Abb. 8.13. Dabei wurde nach jeweils zehn terminierten Transaktionen ein neuer Durchsatzwert gebildet und als Zeitreihe aufgetragen. Die erste Kurve ergab sich bei einem Experiment unter ausfallfreien Makro-Anfangszustand. Nach einer transienten Anfangsphase stabilisiert sich der Durchsatz; die verbleibenden Beobachtungen weisen ein 95%-Konfidenzintervall von 7% relativ zum geschätzten Mittelwert auf. Bei einem anderen Experiment mit denselben Parametern und Anfangsbedingungen wurde zu einem nach der angenommenen transienten Phase liegenden Zeitpunkt ein CPU-Ausfallereignis erzeugt. Die Abbildung zeigt einen dramatischen Abfall des Gesamtdurchsatzes, aber ebenfalls mit einer schnellen Stabilisierung des Kurvenverlaufs mit relativem Konfidenzintervall von 10% des Mittelwerts.

Damit bestätigt sich das prinzipielle zeitliche Verhalten des modellierten fehlertoleranten Systems, wie es bereits in Abb. 5.2-c postuliert wurde.

8.4.1 Einfluss der Transaktionslast

Zunächst wurden Experimente zur Durchsatz-Verfügbarkeit aller vier Protokolle C1, C3a, C3b und C3c bei Variierung der Last bis zur Sättigungsgrenze und bei Fixierung der DB-Anforderungen auf 'mittel' und des Netztyps auf 'LAN' ausgeführt. Die Ergebnisse sind in Abb. 8.14-a dargestellt. Dabei ist die relative Rangordnung der Protokolle gegenüber dem Durchsatz ohne Ausfallbetrachtung (Abb. 8.3-b) erhalten geblieben. Allerdings ist der Durchsatz-Vorsprung des Protokolls C1 ohne Datenredundanz gegenüber den anderen Protokollen bei sehr hoher Transaktionslast kleiner geworden, was auf eine geringere Durchsatz-Einbuße dieser Protokolle nach Ausfällen zurückzuführen ist.

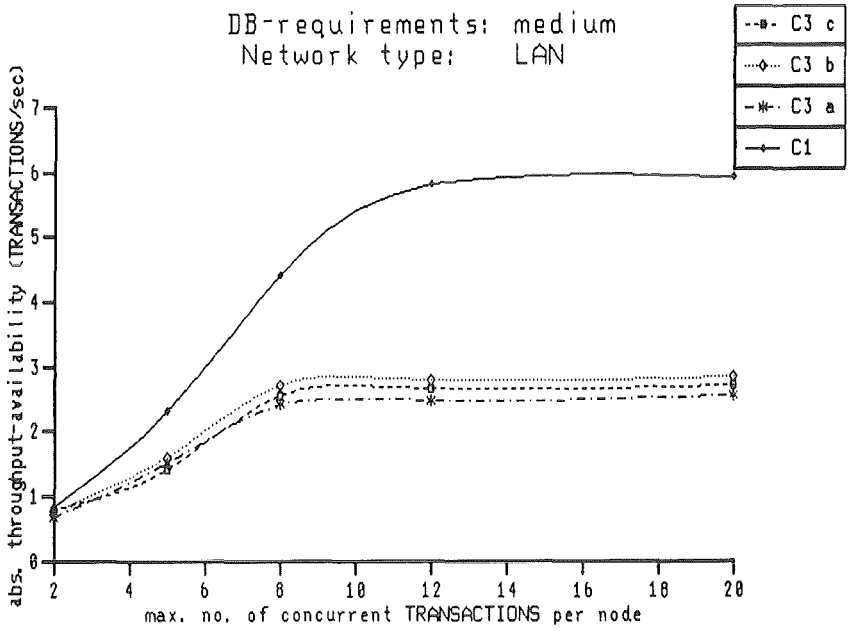
Bei den obigen Experimenten wurde auch die Erfolgsraten-Verfügbarkeit ermittelt (s. Abb. 8.14-b).
Unter Ausschluß von Ausfällen war die Erfolgsrate als Zielgröße nicht interessant, da sie dann lediglich von der Unterauftrags-Abbruchrate λ_{SUB} und der mittleren Anzahl Unteraufträge NSUB abhängt. Transaktionen werden entweder erfolgreich beendet (Enzustand 'COMMIT') oder je nach Zustand und Protokoll ggf. mit 'ABORT' abgebrochen. Nach der Fixierung von λ_{SUB} auf 1% wurde im Normalfall eine Erfolgsrate von 97% bis 99% beobachtet.
Bei Ausfällen kommen Zurückweisungen ('REJECT') von Transaktionen vor, wenn notwendige Daten nicht verfügbar sind. Das wirkt sich bei Ausfällen wesentlich auf die Gesamt-Erfolgsrate qv_{commit}

$$qv_{commit} = \sum_{i=1}^{N} COMMIT_i \; / \; \sum_{i=1}^{N} (COMMIT_i + ABORT_i + REJECT_i)$$

$COMMIT_i$: Anzahl erfolgreich terminierter Transaktionen auf Knoten K_i im Beobachtungszeitraum

im Rechnernetz aus (vergl. Kap. 5.4). In unserem zustandsunabhängigen, geschlossenen Systemmodell passen sich Benutzer nicht adaptiv an den jeweiligen Makrozustand an. Das führt z.B. dazu, daß Transaktionen auch dann weiter an Terminals eingegeben werden, wenn der

a)



b)

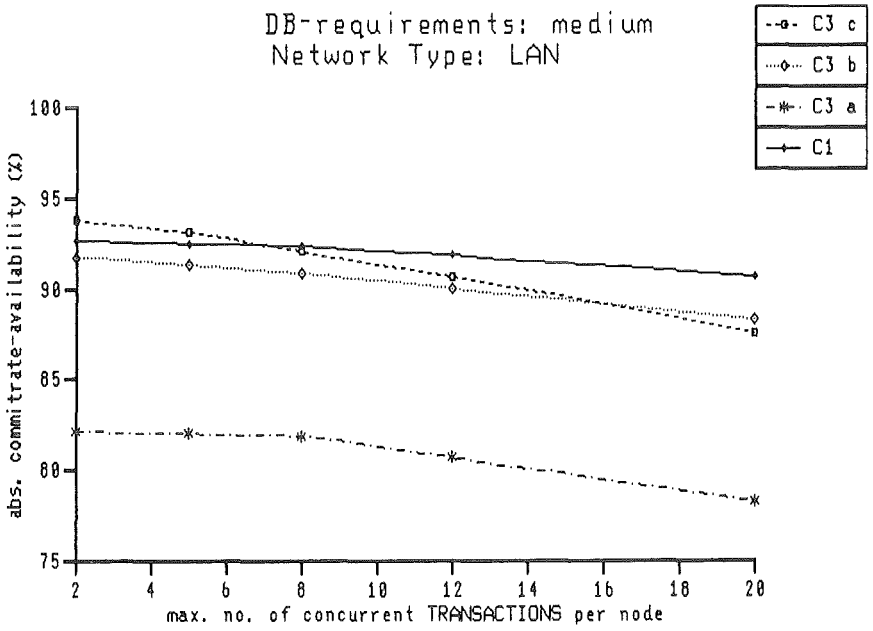


Abb. 8.14: Absolute Leistungs-Verfügbarkeit bei unterschiedlicher Transaktionslast (DB-Anforderungen mittel, Netztyp LAN)

Arbeitsrechner am selben Knoten ausgefallen ist. Diese Transaktionen werden unverzüglich an den Benutzer zurückgewiesen, der aber trotzdem nach einer gewissen Denkzeit eine neue Transaktion startet.
In diesem Modell steigt die Transaktions-Ankunftsrate aus Sicht eines Arbeitsrechners mit sinkender Antwortzeit. Daher ist die Zurückweisungs-Rate an einem ausgefallenen Arbeitsrechner lastabhängig; an einem ausgefallenen Rechner ist diese Rate oft größer als die Terminierungsrate (d.h. der Durchsatz) an anderen Knoten. Das erklärt die mit steigender Last sinkende Gesamt-Erfolgsrate (Abb. 8.14-b).
Bei dem Vergleich des Protokoll-Verhaltens schneidet C3a (Vollzustimmung auf redundanten Daten) mit Abstand am schlechtesten ab. Ursache ist, daß dieses Protokoll eine Transaktion bereits bei Unverfügbarkeit irgendeiner benötigten Schreibkopie zurückweist. Das Mehrheits-Protokoll C3c wäre in einem offenen System wohl relativ am besten: Auf allen N-1 intakten und gegenseitig verfügbaren Knoten liegt die Erfolgsrate konstant bei 100% , auf dem wegen NIU- oder Arbeitsrechner-Ausfall partitionierten Knoten K_i ist diese stets 0% . Dieses stabile, lastunabhängige Verhalten ist in unserem geschlossenen Modell nicht gegeben. Die Erfolgsraten-Verfügbarkeit fällt, wie bei den anderen Protokollen, monoton bei steigender Last.

8.4.2 Einfluss der Update-Häufigkeit

Bei den folgenden Experimenten wurde der Parameter 'DB-Anforderungen' verfeinert, indem gezielt der Einfluß der Update-Häufigkeit WR bei konstanter, mittelgroßer Last und hoher Lokalität variiert wurde. Die Wahl der mittelgroßen Last erlaubt die Untersuchung der Frage, ob der große Durchsatzvorsprung bei nichtredundanter Datenhaltung (Abb. 8.14-a) auch bei unterschiedlicher Update-Häufigkeit bzw. Lokalität erhalten bleibt.
In einer sehr query-intensiven Umgebung (WR nahe bei 0) ist die absolute Durchsatz- als auch die Erfolgsraten-Verfügbarkeit (Abb. 8.15-a, 8.15-b) bei den Protokollen wenig unterschiedlich. Protokoll C3a hat die höchste Performance, da schon die Verfügbarkeit einer beliebigen, evtl. sogar lokal vorhandenen Lese-Kopie für alle Dateneinheiten der Basismenge zu einer erfolgreichen, schnellen Bearbeitung einer Transaktion ausreicht. Das Mehrheits-Protokoll C3c

hat relativ den kleinsten Durchsatz und die kleinste Erfolgsrate, da es als einziges auf mehr als eine Kopie pro Dateneinheit zum Sperren und Lesen zugreift.

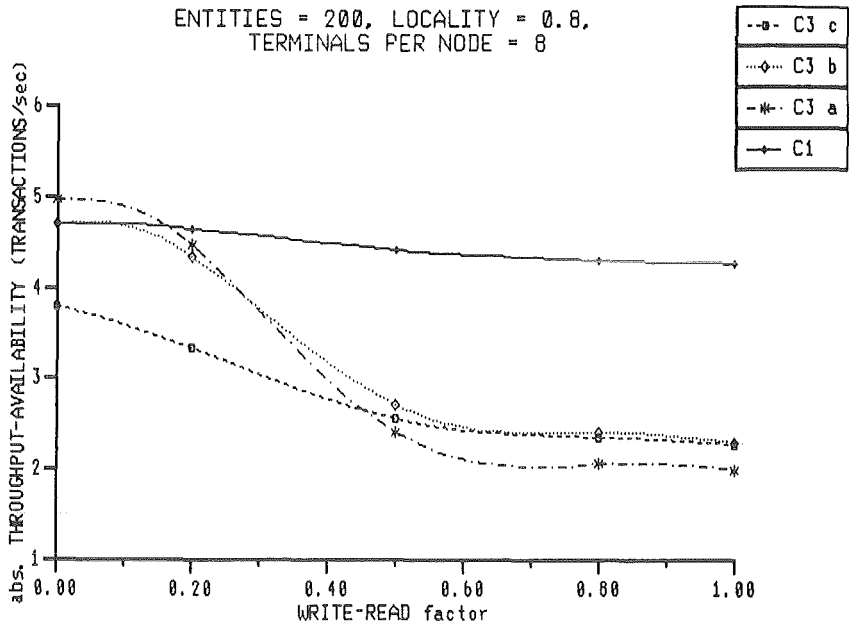
In einer sehr update-intensiven Umgebung (WR nahe bei 1) ist die absolute Durchsatz- als auch die Erfolgsraten-Verfügbarkeit bei Protokoll C3a relativ am schlechtesten wegen der oben beschriebenen Behandlung von Schreib-Kopien. Die Protokolle C3b und C3c sind in der Gesamtleistung fast gleich, da sie Updates auf einer gleichen Anzahl (auf allen) Schreib-Kopien ausführen. Bezüglich dem absoluten Durchsatz wird der redundanzfreie Fall C1 immer günstiger, je höher WR ist, da bei diesem Protokoll die Unterscheidung von Lese- und Schreibkopien kaum Einfluß auf die Performance hat: Lediglich die bei einer WR-Erhöhung steigende Sperrkonflikt-Rate macht sich in einem geringen Abfallen der integrierten Leistungswerte bemerkbar.

Erstaunlich ist, daß die Update-Häufigkeit -außer bei Protokoll C3a- kaum Einfluß auf die Erfolgsraten-Verfügbarkeit hat (Abb. 8.15-b). Die entsprechenden Werte sind bei diesen Protokollen sogar fast gleich. Es kann angenommen werden, daß wegen hoher Lokalität bei diesen Versuchen die nichtredundante Datenhaltung (Protokoll C1) begünstigt ist, wie ein Vergleich mit geringerer Lokalität (Abb. 8.16-c) zeigt.

8.4.3 Einfluss der Lokalität

Bei den folgenden Experimenten wurde der Parameter 'DB-Anforderungen' verfeinert, indem gezielt der Einfluß der Lokalität LOC bei konstanter, mittelgroßer Last und Update-Häufigkeit variiert wurde.
Zunächst ist klar, daß bei sinkender Lokalität und damit steigendem NSUB (Anzahl Unteraufträge pro Transaktion) sowohl Durchsatz- als auch Erfolgsraten-Verfügbarkeit monoton fallen, wie Abb. 8.16 zeigt. Allein das Mehrheitsprotokoll (C3c) auf redundanten Daten bleibt bei der Gesamt-Erfolgsrate nahezu konstant, obwohl bei einer Lokalitäts-Senkung von 90% auf 40% die Unterauftragsanzahl pro Transaktion von 2,8 auf 4,4 steigt. Als Folge hat dieses Protokoll bei einer Lokalität von unter 80% die höchste Erfolgsrate.
Bei sehr hoher Lokalität ist die Erfolgsrate für C1 und C3b relativ am höchsten. Protokoll C3a ist bezüglich der Erfolgsrate wesentlich

a)



b)

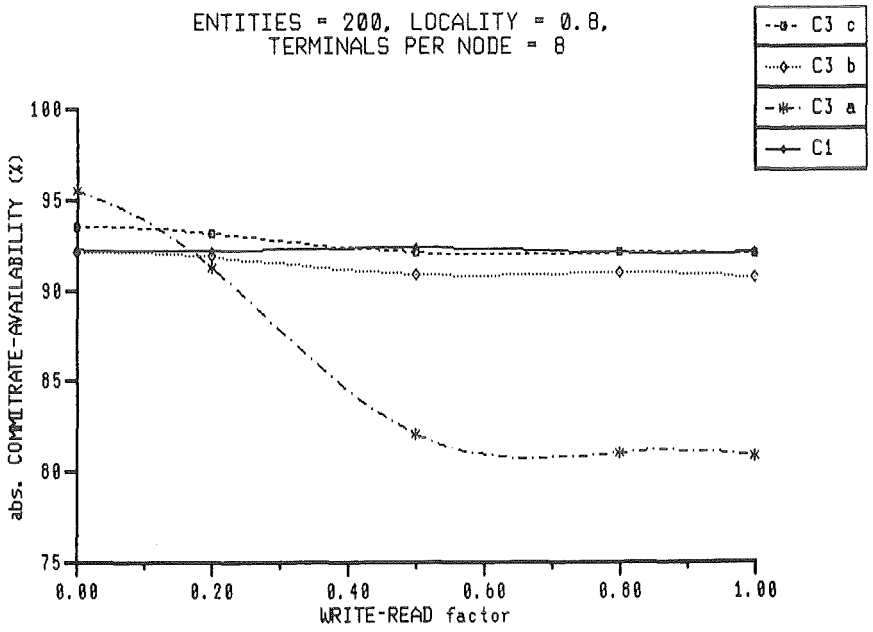
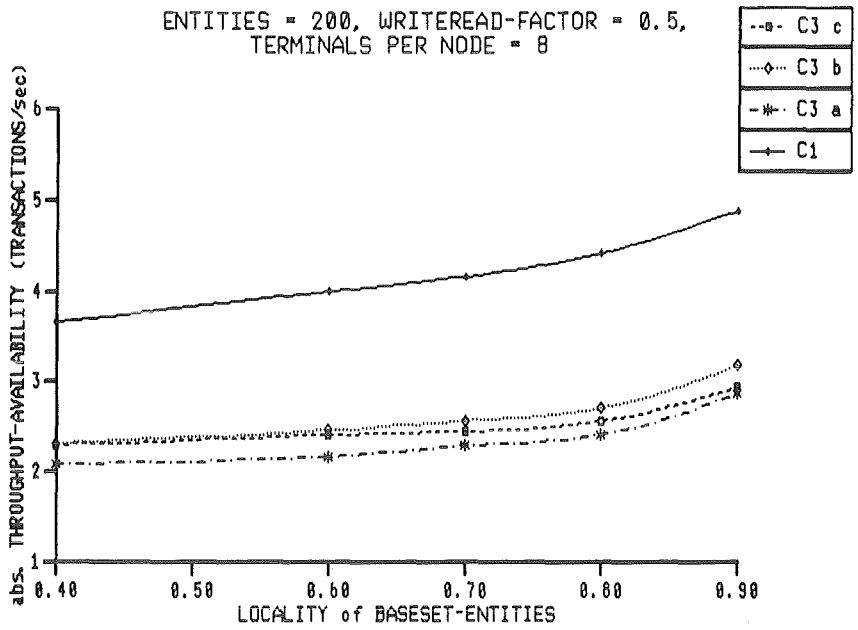


Abb. 8.15: Integrierte Leistung-Verfügbarkeit bei unterschiedlicher Update-Häufigkeit (200 Dateneinheiten, 0.8 Lokalität, 8 Terminals pro Knoten, Netztyp LAN)

(a)



(b)

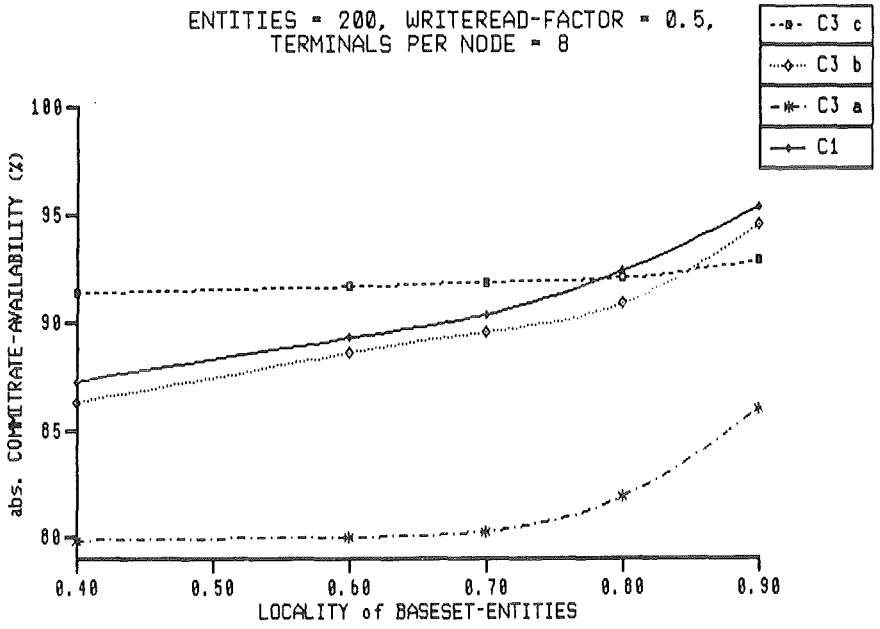


Abb. 8.16: Integrierte Leistung-Verfügbarkeit bei unterschiedlicher Lokalität
(200 Dateneinheiten, 0.5 Update-Häufigkeit,
8 Terminals pro Knoten, Netztyp LAN)

schlechter, was aber nicht notwendig auf den hierbei nicht untersuchten Fall einer query-intensiven Umgebung zutreffen wird (vergl. Abb. 8.15-b).
Eine ähnlich einschränkende Aussage ist für die integrierten Durchsatz-Werte gültig. Die relativen Werte der Protokolle sind bei der untersuchten Update-Häufigkeit unabhängig von der Lokalität, da der Durchsatzvorsprung des Protokolls C1 annähernd konstant bleibt.

8.4.4 Einfluss der CPU- und NIU-Ausfallraten

Der Protokollvergleich bezüglich Durchsätzen und Erfolgsraten ergibt in den meisten oben untersuchten Umgebungsbedingungen sowohl im Normalfall als auch bei CPU- und NIU-Ausfällen dieselbe Rangordnung der Protokolle. Damit ist die jeweilige Rangordnung bei den Normalfall und Fehlerfälle integrierenden Maßen von Reparaturraten und Ausfallraten **unabhängig** (vergl. Relation 'besser-als' in Kap. 5.3), sodaß die Variierung dieser Raten an der relativen Bewertung der Protokolle nichts ändern würde. Auch die im Normalfall bei den Protokollen wenig unterschiedliche Erfolgsrate ergibt stets dieselbe Bewertung.
In einigen wenigen Fällen ist die Rangordnung der Protokolle bezüglich integriertem Durchsatz vom stochastischen Verhalten des Rechnernetzes abhängig, z.B.:

- Bei geringer Update-Häufigkeit (WR = 0.1) ist der C1-Durchsatz im Normalfall besser als der C3a-Durchsatz, bei Ausfällen ist es umgekehrt.
- Bei Lokalität LOC %< 70% ist der C3b-Durchsatz im Normalfall besser als der C3c-Durchsatz, bei Ausfällen ist es umgekehrt.

Protokolle, die eine bessere Leistung lediglich im Normalfall bieten, schneiden im Gesamtverhalten ab einer gewissen hohen Ausfallrate nicht mehr besser ab. Das wird in Abb. 8.17 verdeutlicht. In Abb. 8.16-a wurden Durchsatz-Verfügbarkeiten unter Annahme einer Ausfallrate von 1/100 [1/h] pro CPU bzw. NIU berechnet. Die entsprechenden Ergebnisse bei Variierung der Ausfallrate für feste Lokalität LOC = 0.4 zeigt Abb. 8.17. Das Protokoll C1 wurde aus diesem Vergleich ausgeschlossen, da es bei dieser Lokalität stets den höchsten relativen Durchsatz aufwies.

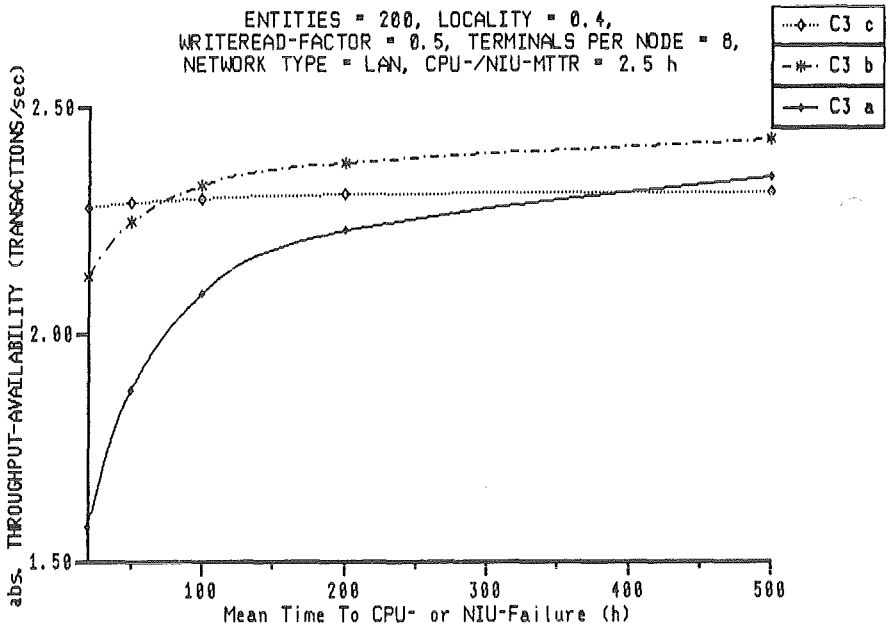


Abb. 8.17: Absolute Durchsatz-Verfügbarkeit bei unterschiedlichen CPU-/NIU-Ausfallraten (200 Dateneinheiten, 0.1 Update-Häufigkeit, 0.8 Lokalität, 8 Terminals pro Knoten, Netztyp LAN, MTTR einer CPU und einer NIU 2.5 Stunden)

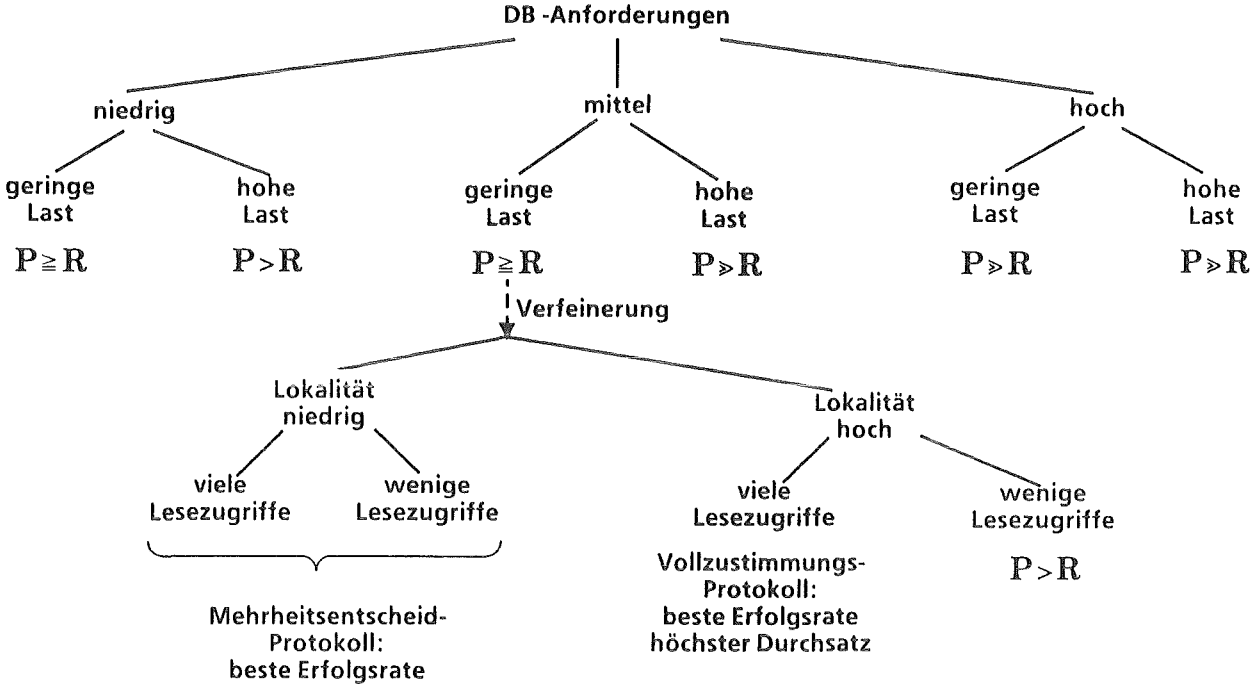
8.4.5 Zusammenfassung und Folgerungen

Der relative Durchsatz- und Erfolgsraten-Vergleich der Protokolle unter Einbeziehung von Normal- und Fehlerfällen fällt in der Mehrzahl der untersuchten Last- und Daten-Profile zugunsten des Protokolls auf partitionierten, nichtreplizierten Daten aus, wie die Ergebnisübersicht in Abb. 8.18 zeigt. Redundante Datenhaltung lohnt sich aber unter gewissen Bedingungen in folgender Umgebung:

- Höchstens mittelgroße Datenbasis und mittelgroße Transaktionen, die durch höchstens mittlere DB-Anforderungen repräsentiert werden, sowie
- relativ kleine Transaktionslast.

In dieser Umgebung folgt aus unseren Ergebnissen:

(1) Alle untersuchten Protokolle auf redundanten Daten versprechen zumindest keine schlechtere Leistung als das Protokoll auf



P>R: Protokoll auf partitionierter DB hat besseren Durchsatz und bessere Erfolgsrate als alle 3 Protokolle auf replizierter DB

Abb. 8.18: Zusammenfassung der VDBS-Simulationsergebnisse

partitionierten Daten bei demselben Transaktionsprofil.

(2) Das Vollzustimmungs-Protokoll verspricht optimalen Durchsatz und optimale Erfolgsrate, falls die meisten Transaktionen nur lokale Daten benötigen und diese nur lesend zugreifen.

(3) Das Mehrheitsentscheids-Protokoll verspricht optimale Erfolgsrate, falls mindestens die Hälfte aller benötigten Daten lokal nicht vorhanden sind.

Speziell das Mehrheits-Protokoll hat sich somit als konkurrenzfähig erwiesen, obwohl alle Protokolle, die eine Kopienmehrheit zum Lesen Sperren, in einer bekannten Untersuchung /BERN81/ wegen angeblicher Ineffizienz für den praktischen Einsatz verworfen wurden.
Hingegen bietet die untersuchte Version des Primärkopien-Protokolls kein solch optimales Verhalten gegenüber den anderen Protokollen, obwohl dessen Durchsatz-Werte oft höher als bei Vollzustimmungs- bzw.

Mehrheits-Protokoll sind. Die modellierte, 'pessimistische' Behandlung von verfügbaren Schreib-Sekundärkopien durch dieses Protokoll könnte verbessert werden, indem diese 'im Hintergrund' geändert werden (s. Kap. 7.5).
Es kann erwartet werden, daß sich diese wichtigen Ergebnisse auf den Fall niedriger DB-Anforderungen bei geringem Lastaufkommen übertragen.

8.5 Experimentelle Gültigkeits(teil-)überprüfung des VDBS-Simulationsmodells

Im Rahmen der VDBS-Simulationsexperimente wurden verschiedene Modelltests mit dem Ziel der internen Modellverifikation sowie der Sensitivitätsanalyse ausgeführt (vergl. Kap. 2.1). Es lassen sich drei Gruppen von Testmethoden unterscheiden:

(1) Überprüfung der dynamischen Modellstruktur und axiomatischer Modellannahmen gegenüber der Entwurfsspezifikation der Submodelle in den Kapiteln 4 und 7, mit dem Ziel der internen Modellverifikation.
(2) Überprüfung der Simulationsausgaben gegenüber analytischen Modellen oder Vergleichsrechnungen, mit dem Ziel der internen Modellverifikation.
(3) Sensitivitätsanalysen ausgewählter Eingabeparameter, mit dem Ziel der Überprüfung der Modellannahmen und der Ermittlung kritischer Einflußgrößen.

Über diese Untersuchungen wird im folgenden berichtet.

Überprüfung der dynamischen Modellstruktur und der Modellannahmen

Das wesentliche Hilfsmittel dabei war der interaktive Performance-Monitor des DAEMON-Systems (Kap. 6.5.5), mit dem sowohl das funktionale als auch das zeitliche Verhalten des Simulationsmodells gegenüber dessen Spezifikation getestet wurde:

- Mit **Transaktionstraces** wurde der dynamische Ablauf, d.h. die zeitliche Folge der aktivierten Netzinstanzen, der Protokolle zur Transaktionsverarbeitung und zur Interprozeß-Kommunikation überprüft. (Vergleiche Beispieltrace in Anhang C.) Die Prüfung

erfolgte sowohl für den Protokollablauf im Normalfall als auch bei Eintritt von Ausfällen. Speziell wurde die Kommunikation zwischen TM- und DM-Prozessen getestet, u.a.

-- Jede auf Antwort wartende (Sub-)Transaktion empfängt diese in endlicher Zeit;
-- Eine Nachricht kommt nur an solchen Mailboxes an, an denen eine (Sub-)Transaktion wirklich darauf wartet.
-- Alle 1:n-Nachrichten gelangen zu den intendierten Empfängern.
-- Alle n:1-Nachrichten werden korrekt eingesammelt und gelangen zu den intendierten Empfängern.
-- Nach Crash bzw. Restart erfolgt eine korrekte Statuserkennung entsprechend dem lokalen Kenntnisstand an jedem Knoten, wie im Zustandsdiagramm von Abb. 4.4 spezifiziert. Statusmeldungen gelangen zu allen auf Antwort wartenden (Sub-)Transaktionen.

- Mit **Instanztraces** wurde das E/A-Verhalten (Markentransformation, Markenauswahl) sowie das durchschnittliche Zeitverhalten getestet. Das E/A-Verhalten auch sehr komplexer Instanzen (z.B. der Instanzen zur Generierung von Subtransaktionen, zur Priorisierung nebenäufiger Sperranforderungen sowie zur Commit-Entscheidung) konnten damit gegenüber der Spezifikation aller Protokolle zur Transaktionsverarbeitung erfolgreich getestet werden.
  Das komplexe Zeitverhalten vieler Instanzen, z.B. der Terminal-Instanzen mit Denkzeit-Verzögerung sowie der Instanzen mit auftragsabhängigem Platten- und CPU-Zeitverbrauch, wurde ebenfalls überprüft.

- Zur Überprüfung axiomatischer Modelleigenschaften wurde das Modell auf **Deadlockfreiheit** getestet. Diese muß bei korrekter Realisierung des Preclaiming-Sperrverfahrens gelten. Zunächst war das Netzmodell bei allen Experimenten lebendig, d.h. daß die Mehrzahl der Transaktionen terminierte und führte im geschlossenen Modell zur Generierung neuer Aufträge. Um zu testen, ob alle aktiven Aufträge tatsächlich in endlicher Zeit terminierten, wurden einige Versuche gemacht, bei welchen die Generierung neuer Aufträge zu einem gewissen Zeitpunkt unterbunden wurde.

Die beschriebenen dynamischen Tests trugen zur Erkennung und

Beseitigung einer Vielzahl von Modellimplementierungsfehlern bei.
Es wurden auch Modellentwurfsfehler aufgespürt:

Z.B. war beim Entwurf der IPC-Schicht vernachlässigt worden, daß nach CPU-Ausfall eine IPC-Instanz auf einem anderen Knoten die Meldung der Nichtverfügbarkeit dieser CPU an auf Antwort wartende Prozesse weitergab, obwohl die Antwort zum Ausfallzeitpunkt bereits im Rechnernetz unterwegs war und auch korrekt am Zielknoten ankam. Auf diese verspätete Nachricht wartete aber kein Zielprozeß mehr, weil der CPU-Status schneller gemeldet wurde.

Als Lösung des Problems wurde das Ignorieren verspäteter Nachrichten in den Enwurf der IPC-Schicht eingebaut und im Modell realisiert.

Überprüfung der Simulationsausgaben

Eine Trendvergleich unserer Simulationsergebnisse mit bekannten Warteschlangenmodellen geschlossener Timesharing-Systeme (vergl. /KLCK76/) bestätigt den erwarteten Kurvenverlauf der wesentlichen Zielgrößen bei steigender Systemlast:

- Antwortzeiten bleiben bis zu einer Lastgrenze G zunächst konstant, steigen aber bei weiterer Lasterhöhung linear an.
- Durchsätze steigen bis zu einer Sättigungsgrenze und bleiben dann in etwa konstant. Die Grenze sollte oberhalb von G erreicht sein.
- Ressourcen-Auslastungen entwickeln sich ähnlich wie Durchsatzwerte. Die Sättigungsgrenze muß wegen dem stochastischen Bedien- und Ankunftsratenprofil unter 100% betragen (s. /FERR78, S.217/), kann im Einzelfall aber weit darunter liegen.

Das in vielen realen Systemen beobachtete Absinken des Durchsatzes und der Auslastungen nach Erreichen der Sättigungsgrenze basiert auf einer Dominanz des System-Overheads gegenüber der Auftragsbearbeitung bei dem Versuch einer Ressourcen-Zuteilung, die dann aber wegen Vollauslastung erfolglos bleiben muß. Typisches Beispiel sind Betriebssysteme mit virtuellem Speicher. Dieses Verhalten konnte bei unseren Versuchen erwartungsgemäß nicht beobachtet werden, da wir den Overhead a priori vernachlässigen.

Stichprobenartig wurden die experimentell ermittelten Antwortzeiten mit den in Kap. 7.6 analytisch vorhergesagten verglichen und eine gute Übereinstimmung erzielt.

Beispiel:

Im Fall DB-Anforderungen mittel, Protokolltyp C3a, Netztyp WAN, 2 Terminals pro Knoten ergaben sich folgende

Experimentergebnisse:

U = 4.32 (mittl. Anzahl Unteraufträge)

R+W = 3.0 (mittl. Mächtigkeit der Basismenge pro Transaktion)

R' = 1.5, W' = 1.5 (mittl. Mächtigkeit der Basismenge pro Subtransaktion)

Qlog = 0.02 sec (Verzögerungszeit für Plattenzugriff bei Transaktionslogging)

Qlock = 0.20 sec (Sperr-Verzögerungszeit pro Subtransaktion)

RESP = 6.33 sec (mittl. Antwortzeit einer Transaktion)

Aus den Parameterwerten (s. Anhang A)

TG = 0.50 sec TL = 0.008 sec

T1 = 0.001 sec

T2 = (0.01+0.05)/2 sec T3 = (0.15+0.30)/2 sec

T4 = 0.05 sec

ergibt sich nach Kap. 7.6 folgende **Analytische Vorhersage:**

RESP' = 5.28 + V',

mit V' = (U+3)Qlog + U*Qlock

(Summe der Logging- und Locking-Verzögerungszeiten)

Der Wert der kritischen Komponenten von V, Qlog und Qlock, ist dabei nicht bestimmt worden. Nimmt man aber den Vorhersagewert V' und setzt die durch Simulation erhaltenen Ergebnisse ein, so ergibt sich V' = 1.01 sec. Es ist nun RESP-V' = 5.32, sodaß RESP'-V' eine sehr gute Schätzung dieser Antwortzeit darstellt.

Sensitivitätsanalysen

Es wurden einige Sensitivitätsexperimente entworfen und ausgeführt, um

die wichtigsten Hypothesen des Simulationsmodells zu testen.

Die Hauptannahme des VDBS-Experimententwurfs (Kap. 8.2) ist, daß eine Vergrößerung der individuellen Faktoren E und LOC bzw. eine Verringerung der Faktoren BS und WR des Daten- und Transaktionsprofils einen positiven Einfluß auf den mittleren Durchsatz und gleichzeitig einen negativen Einfluß auf die mittlere Antwortzeit haben.
Diese Annahme konnte experimentell bestätigt werden, wobei die Sensitivitäts-Experimente auf die Extremwerte TCO=2 und TCO=20 sowie PROT=C1 und PROT=C3a beschränkt wurden, mit dem Netztyp LAN. Die vier Faktoren des Parameters 'DB-Anforderungen' wurden dabei jeweils einzeln um einen relativ kleinen Wert vergrößert bzw. verkleinert, ausgehend von den Werten für mittlere Anforderungen:

(E,BS,WR,LOC) = (200,mittel,0.5,0.8)

Die Experimente ergaben die in Tabelle 8.3 dokumentierten Ergebnisse,

		DB-Anforderungen				Zielgrößen-Mittelwerte		
TCO	PROT	E	BS	WR	LOC	THROUGHPUT	RESP.TIME	NSUB
2	C1	200	mit.	0.5	0.8	0.88	1.18	1.65
		250				0.88	1.15	1.65
			nied.			0.89	0.99	1.30
				0.4		0.88	1.15	1.65
					0.85	0.88	1.14	1.33
2	C3a	200	mit.	0.5	0.8	0.83	1.93	4.31
		250				0.83	1.91	4.26
			nied.			0.85	1.58	3.61
				0.4		0.85	1.55	3.04
					0.85	0.84	1.81	3.65
20	C1	200	mit.	0.5	0.8	6.07	12.0	1.65
		250				6.34	11.6	1.64
			nied.			8.30	7.0	1.29
				0.4		6.95	10.3	1.64
					0.85	7.10	9.6	1.30
20	C3a	200	mit.	0.5	0.8	2.83	32.4	4.31
		250				2.86	31.3	4.30
			nied.			3.78	22.6	3.54
				0.4		4.15	20.7	3.00
					0.85	3.10	28.6	3.62

Tabelle 8.3: Sensitivität gegenüber Änderung individueller Faktoren der DB-Anforderungen

aus denen kein Widerspruch zu obiger Annahme ersichtlich ist.

Interessant ist aber auch, daß das Modell speziell im Fall redundanter Daten bei hoher Last sensibel auf kleine Änderungen der Einzelfaktoren reagiert (bis zu 50% Abweichung). Der Faktor mit kleinstem Einfluß (unter 5% Abweichung) ist dabei die Anzahl Dateneinheiten E.

Bei sehr geringem Lastaufkommen sind hingegen die Abweichungen bei allen Faktoren sehr klein. Damit sind im nachhinein die Detailuntersuchungen des Einflusses von Update-Häufigkeit WR und Lokalität LOC bei höheren Transaktionslasten (Kap. 8.5.2, 8.5.3) gerechtfertigt.

Abschließend wurden die Annahmen für die Versuchsserie mit Einschluß von CPU- und NIU-Ausfällen getestet, die eine Stationarität aller Zielgrößen bei beliebigen Crash-Zeitpunkten $t' \geq 0$ und bei Ausfall einer beliebigen CPU_i bzw. NIU_i, $i \varepsilon [1,N]$, unterstellen. Die Sensitivitätstests gegenüber dem in Kap. 8.5 untersuchten Fall

$t' = 100$ sec, $i = 4$

erbrachten die in Tabelle 8.4 gezeigten Durchsatz-Mittelwerte, wobei eine Fixierung auf

DB-Anforderungen = mittel, TCO = 8, Netztyp = LAN

erfolgte. Die Abweichungen gegenüber dem diskutierten Fall betragen dabei $\leq 7\%$.

PROT	CRASH NODE	CRASH TIME	THROUGHPUT CPU-CRASH	THROUGHPUT NIU-CRASH
C1	4	100	3.53	3.74
	4	200	3.77	3.91
	4	300	3.64	3.89
	2	100	3.60	3.84
	5	100	3.65	3.91
	7	100	3.58	3.87
C3a	4	100	1.40	1.41
	4	200	1.46	1.48
	4	300	1.42	1.45
	2	100	1.47	1.51
	5	100	1.46	1.49
	7	100	1.50	1.52

Tabelle 8.4: Sensitivität gegenüber Änderung von Ausfall-Zeitpunkten und -Knoten

9 ZUSAMMENFASSUNG UND AUSBLICK

9.1 Zusammenfassung der Erkenntnisse der Arbeit

Vorliegende Arbeit hat eine durchgängige Methodik zur Modellierung fehlertoleranter verteilter Systeme unter drei Aspekten vorgestellt:

(1) Einheitliche Sprache zur rechnergestützten Spezifikation und Leistungsvorhersage komplexer Hardware-/Software-Systeme,
(2) Entwurfs- und Simulationsmethodik für robuste Kommunikationsprotokolle,
(3) Anwendung der Methodik bei einer vergleichenden Untersuchung der Leistung und Fehlertoleranz von Protokollen zur verteilten Transaktionsverarbeitung für teilredundant verteilte Datenbestände.

Der Einsatz Verteilter Datenbank-Verwaltungssysteme (VDBS) wurde in letzter Zeit vorgeschlagen, um die Verfügbarkeit und Effizienz von Datenbanksystemen zu erhöhen. Protokolle zur Transaktionsverarbeitung haben einen sehr wesentlichen Einfluß auf das gesamte VDBS-Verhalten. Hingegen hat die Komplexität der Protokolle und die große Zahl von Entwurfsparametern von Rechnernetzen und der darauf realisierten VDBS bis heute eine systematische Entwurfs- und Bewertungsmethodik erschwert.
Mit der im Kapitel 6 der Arbeit vorgestellten Sprache der DAEMON-Netze haben wir VDBS-Modelle spezifiziert und bewertet. Das Einsatzspektrum der Sprache umfaßt komplexe zustandsdiskrete, stochastische Systeme. Die Spezifikationsmethode bietet einen formalen, deskriptiven und graphischen Ansatz, mit höheren stochastischen Petrinetzen als Basiskonzept. Funktionales und stochastisches Verhalten von Software-Komponenten, zeitliches und fehlerhaftes Verhalten von Hardware-Ressourcen sowie Struktur und Ablauf von Transaktionen durch das System können auf unterschiedlichen Detaillierungsgraden spezifiziert werden. Spezifikation und Simulation von Netzmodellen in dieser Sprache werden durch das Software-Werkzeug DAEMON unterstützt. Der DAEMON-Simulator liefert Schätzungen aller relevanten system-, ressourcen- und transaktions-orientierten Leistungsgrößen nach statistischen Methoden. Folgende Hilfsmittel zur internen Verifikation

des Modells wurden erarbeitet:

- Simulationsexperimente können interaktiv überwacht werden. Der Modellablauf kann an logisch definierten Zuständen angehalten, der erreichte Modellzustand zusammen mit Leistungsschätzungen inspiziert, und die Simulation z.B. durch selektive Traces fortgesetzt und getestet werden.
- Durch größtenteils automatisierbare Abbildungen von DAEMON-Netzen auf einfachere Petrinetze, wobei von wertabhängigen Ablaufentscheidungen abstrahiert wird, werden Methoden der Petrinetz- und der Markov-Theorie in beschränktem Umfang anwendbar. Daraus können sowohl prädikative Modelleigenschaften als auch stationäre Leistungsgrößen analytisch ermittelt werden.

In Kapitel 4 wurde ein parametrisiertes Modell der Umgebung eines VDBS mit DAEMON-Netzen spezifiziert, das wesentlich weitergehende Vorhersagen des VDBS-Verhaltens erlaubt als bei früheren Studien. Das Umgebungsmodell charakterisiert zunächst die Hardware-Architektur unterschiedlicher Klassen von Rechnernetzen. Im Hardware-Fehlermodell wurden Rechnerausfall und Netzpartitionierung formal definiert. Um die Komplexität der Protokolle zur Transaktionsverarbeitung zu reduzieren, wurde ein operationales Modell eines robusten Dienstes zur Kommunikation zwischen verteilten Anwendungsprozessen entworfen. Der Dienst enthält Primitive zur Mehrpartner-Kommunikation sowie zur Verfügbarkeitserkennung abgesetzter Rechnerknoten. Es wurde ein Profilschema von Datenbankanwendungen entworfen, das sowohl die Größe, Redundanz und Verteilung der Datenbasis im Rechnernetz, als auch Komplexität, Änderungshäufigkeit und Lokalität von Datenbankzugriffen durch Transaktionen beschreibt.

Auf der Grundlage dieses Umgebungsmodells wurden in Kapitel 7 drei Protokolle entworfen und mit DAEMON-Netzen sehr detailliert spezifiziert. Die Protokolle unterscheiden sich in der Strategie der Behandlung redundanter Daten. Dabei wurden die wichtigsten Verarbeitungsphasen verteilter Transaktionen spezifiziert, die von deren Übersetzung, über die Synchronisation konkurrierender Datenbank-Zugriffe durch eine Zweiphasen-Sperrmethode, bis zur atomaren Terminierung durch ein Commit-Protokoll reichen. Als Erweiterung bisher bekannter Protokolle wird die operationale

Integrität der Datenbasis auch nach Netzpartitionierung gewahrt.

Hauptziel der vergleichenden Untersuchung von unterschiedlichen VDBS war die Anwendung der Simulationsmethodik zur Bewertung der Protokolle auf redundanten Daten in unterschiedlichen Anwendungs- und Rechnernetz-Umgebungen. Dazu wurden in Kapitel 5 Leistungsmaße und statistische Schätzmethoden definiert, die allgemein zur Untersuchung fehlertoleranter verteilter Systeme geeignet sind. Als primäre Leistungsmaße der Transaktionsverarbeitung aus Benutzersicht werden Antwortzeit, Durchsatz und Erfolgsrate von Transaktionen betrachtet.
Um eine einheitliche Vergleichsbasis für verschiedene Klassen fehlertolerierender Rechensysteme zu schaffen, wurden integrierte Leistungs-Verfügbarkeitsmaße definiert. Systeme mit besserer Leistung im ausfallfreien Normalfall können nun mit Systemen verglichen werden, deren Leistung nach System-Teilausfall günstiger ist. Dabei wird in wesentlichen Systemzuständen die stationäre Leistung durch Simulation ermittelt und mit der Verfügbarkeit der Rechnernetz-Komponenten gewichtet. Es wurde eine effiziente Schätzmethode für größere Rechnernetze entwickelt, die einen tolerierbaren Fehler aufweist. Die Anwendung der Methode bei den Protokollmodellen lieferte eine umfassendere Charakterisierung der Performance als frühere Ansätze.

Die experimentelle Leistungsbewertung der drei Protokolle in Kapitel 8 hatte zum Ziel, die unterschiedlichen Kopienbehandlungsstrategien bei jeweils unterschiedlichen Transaktionslasten, Datenbankanwendungen und Rechnernetztypen zu vergleichen. Untersucht wurde auch eine Umgebung mit partitionierter, nichtredundanter Datenbasis.
Die Gültigkeit der komplexen Simulationsmodelle wurde intern überprüft. Die dabei eingesetzten interaktiven Hilfsmittel des DAEMON-Systems erwiesen sich als sehr nützlich. Ferner wurden Outputvergleiche mit vergröbertem analytischen Modell sowie Sensitivitätsanalysen erstellt.
Die Simulationsergebnisse zeigten, daß bei einem weiten Spektrum der untersuchten Umgebungsparameter die Kopienhaltung mit Leistungseinbußen erkauft werden muß. In bestimmten Umgebungen erwiesen sich aber das Vollzustimmungs- bzw. das Mehrheitsentscheid-Protokoll auf redundanten Datenbeständen effizienter. Diese Umgebungen sind z.B. durch überwiegend lokale Lesezugriffe bzw. durch überwiegend

nichtlokalen Zugriffe auf eine größere Datenbasis bei geringem Lastaufkommen charakterisiert.

9.2 Ausblick

Die Ergebnisse der Arbeit bieten vielfältige Ansatzpunkte, das Instrumentarium zur quantitativen Untersuchung fehlertolerierender Rechensysteme weiter auszubauen und anzuwenden:

(1) Das entwickelte rechnergestützte Werkzeug DAEMON ist bereits einsetzbar. Um es in eine komfortable Software-Produktionsumgebung /SCHN81/ zu integrieren, sind noch Verbesserungen notwendig. Zunächst ist die Einbettung in ein Modellbanksystem erforderlich, um Ergebnisse mehrerer Experimente zu verwalten und graphisch aufbereitet auszugeben. Auf der Spezifikations-Schnittstelle ist die Unterstützung von hierarchischer Modellierung mit Teilnetzen anzustreben.

(2) Die in dieser Arbeit gewonnenen Erfahrungen mit der Modellierung komplexer Verteilter Systeme zeigten die Notwendigkeit formaler rechnergestützter Verifikationshilfsmittel. Die vorgeschlagene Transformation von DAEMON-Netzen würde unter gewissen Randbedingungen die Anwendung analytischer Methoden z.B. zum Vergleich mit Simulationsergebnissen erlauben. Hierzu liegen aber noch keinerlei Erfahrungen vor. Der Zusammenhang zwischen prädikativen und quantitativen Eigenschaften von Petrinetzen ist noch vollkommen unerforscht /TPN85/.

(3) Die entwickelten parametrisierten VDBS-Modelle basierten auf empirisch kalibrierten Parameterwerten und aus Sicht realer DB-/DC-Systeme teilweise stark vergröbernden Annahmen. Wüschenswert wäre die Kalibrierung der Parameter und die Validierung unserer Ergebnisse durch Messungen an realen VDBS. Erst wenn hierzu Erfahrungen gesammelt werden, kann die Notwendigkeit möglicher Modellverfeinerungen, z.B. in Richtung auf Datenbank- und Kommunikations-Software, sowie die Untersuchung anderer Protokolle zur Transaktionsverarbeitung, abgeschätzt werden.

(4) Inwieweit das Instrumentarium auch auf andere Systemklassen problemlos anwendbar ist, wird erst der weitere Einsatz zeigen können. Hierzu sollte die vorgeschlagenen Methode zur integrierten Bewertung von Leistung und Verfügbarkeit auf beliebige fehlertolerierende Rechensysteme verallgemeinert werden.

(5) Ein möglicher zukunftsweisender Einsatzbereich des interaktiven Performance-Monitors sind höhere Kommunikationsprotokolle für Lokale und Öffentliche Rechnernetze. Diese werden in zunehmendem Maße entworfen und realisiert, obwohl komfortable rechnergestützte Hilfsmittel bisher kaum angeboten werden.

ANHANG

A Eingabeparameter des VDBS-Simulationsmodells

Jeder relevante Inputparameter des VDBS-Simulationsmodells wird im folgenden beschrieben durch
- das Submodell, in dem er vorkommt;
- einen Parameterbezeichner bzw. '*' bei einem Strukturparameter;
- einen Wertebereich für Simulationsexperimente, bestehend aus einem Wert w oder einer Wertemenge {w1, ... ,wn}.
  Werte sind Parameter einer der folgenden Verteilungsfunktionen:

NEGEXP(λ)
: Negative Exponentialverteilung mit Mittelwert $1/\lambda$ ($\lambda>0$). Wird für **Ankunftszeitverteilung** der Transaktionen benutzt, da diese als unabhängig angenommen werden. Wird auch zur Erzeugung von **Ausfallraten** verwendet, falls Ausfallereignisse voneinander unabhängig sind.

NORMAL(λ, σ^2)
: Normalverteilung mit Mittelwert λ und Varianz σ^2. Wird für Bedienzeiten benutzt: Es wird angenommen, daß diese stets in einem endlichen Intervall [ug, og] liegen, mit (ug+og)/2 als Mittelwert. Das Intervall kann durch entsprechend kleines σ^2 festgelegt werden. Alternative Verteilungen, für die Zufallszahlen stets in einem endlichen Intervall liegen, sind z.B. die Dreiecks- und die Beta-Verteilung. Im Gegensatz zur Normalverteilung sind diese nicht symmetrisch, d.h. der Mittelwert kann innerhalb der Intervallgrenzen beliebig festgelegt werden. Mangels realen Meßdaten wird auf diese Verallgemeinerung verzichtet.

ERLANG(λ, k)
: Erlangverteilung mit Mittelwert $1/\lambda$ und Varianz $1/(\lambda^2 k)$, $\lambda>0$, $k>0$. Für $k>1$ ist der Variationskoeffizient stets <1, was nach /FERR78/ eine realistische Modellierung z.B. von Plattenzugriffszeiten darstellt.

UNIF(ug, og)
: Gleichverteilung im Intervall [ug, og]. Wird für Bedienzeiten benutzt, wenn keine zuverlässige Mittelwert-Schätzung verfügbar ist.

EMPIR
: Empirische Verteilung: Z.B. zur Erzeugung realistischer Transaktionsklassen-Verteilungen.

BERN(p)
: Bernoulli-Verteilung: Zur Erzeugung 'boolescher' Ereignisse. Bei jedem Aufruf tritt ein bestimmtes Ereignis mit Wahrscheinlichkeit p auf.

CHOICE({e1,...,em}, M)	Erwartungstreue Auswahl von m Elementen aus der Menge M.
DET(w)	Deterministische, 'konstante' Verteilung.

- Bedeutung des Parameters mit Motivierung des Wertebereichs.

Rechnernetz-Architektur:

TCO	DET({2,5,8,12,20})	Transaktions-Nebenläufigkeit: Max. Anzahl offener Transaktionen pro Eingabeknoten
TG	NORMAL(λ,σ)	mit λ = {0.02,0.5}, σ^2 = {λ/10,λ/2}. Nachrichtenverweilzeit zwischen Prozessen entfernter Anwendungsschichten (End-To-End Delay) in [sec]. Für lokale Netze gelte TG = 0.02
TL	DET(0.008)	Nachrichtenverweilzeit zwischen lokalen Prozessen einer Anwendungsschicht in [sec]
D	NEGEXP(1/15)	'Denkrate' pro Terminal in [1/sec], nach /FERR78, S. 211/
N	DET(7)	Knotenanzahl in einem kleinen homogenen Rechnernetz
*	DET(Ring)	Ringnetzstruktur mit bidirektionalen NIU-Verbindungen
*	DET(dezentral)	Kommunikations-Nebenläufigkeit: eine autonome NIU pro Knoten
*	UNIF	Prozeßscheduling-Strategie: aktivierbare Prozeß-Instanzen werden entsprechend dem Wert ihrer (zeitlichen) Betriebsmittel-Anforderungen unterbrechungsfrei bedient ('shortest job first / no preemption')

Fehlermodell:

λ_{AR}	NEGEXP(1/100)	AR-Ausfallrate in [1/h]
λ_{NIU}	NEGEXP(1/100)	NIU-Ausfallrate in [1/h]
μ_{AR}	DET(1/2.5)	AR-Reparaturrate in [1/h]
μ_{NIU}	DET(1/2.5)	NIU-Reparaturrate in [1/h]
λ_{SUB}	BERN(0.01)	Jede 100. Subtransaktion wird vom lokal en DBVS abgebrochen /GRAY81/

Die angegebenen Reparatur- und Ausfallraten beziehen sich auf das Verfügbarkeitsverhalten der jeweiligen HW-Komponenten aus Sicht der Benutzer. Die AR-Werte entsprechen gemessenen Werten an einer IBM3081

/KfK83/ und auch an Mikrorechnern /MEIS82/. Die in /SISW82/ angegebenen MTTF liegen wesentlich höher (500 - 2000 h), beziehen sich aber lediglich auf Hardware-Ausfälle. Die NIU-Werte entsprechen Messungen an Virtual Circuits im ARPANET /JOHN77/.

Sind Rechnernetz-Struktur, Reparatur- und Ausfallraten gegeben, so lassen sich daraus Reparatur- und Intaktzeiten aller logischen Rechnernetz-Verbindungen bestimmen. Diese legen die Verteilung der **Rechnernetzpartitionierungen** fest. Ein Verfahren hierzu beschreibt Davies /DAVS79, S. 433 ff/.

Datenprofil:

E	DET({50,200,400})	Anzahl log. Dateneinheiten (Fragmente als Sperr-, Lese-, Updateeinheit) in Datenbasis
C	DET({1,3})	Anzahl Kopien pro Fragment (gleich für alle Fragmente)
*	DET(PAR,TRG)	Fragmentkopien-Verteilungsstrategie: PAR: partitioniert / gleichverteilt (C=1) TRG: teilrepliziert / gleichverteilt (C>1) (mittl. Kopienanzahl pro Knoten = E•C/N)

Anwendungs- und Transaktionsprofil:

BS EMPIR:

	Fragmente pro T. = |BS(T)|					
	1	2	3	4	5	Fall
Transaktions-Anteile	40	30	20	7	3	(a)
	15	20	30	20	15	(b)
	3	7	20	30	40	(c)

Diese empirische Verteilung definiert die relativen Häufigkeiten der 'Transaktionsklassen'. Eine Klasse wird durch die Mächtigkeit der Basismenge charakterisiert. Fall (a) spezifiziert Umgebungen mit vielen 'einfachen', Fall (c) mit vielen 'komplexen' Transaktionen.

LOC	DET({0.7,0.8,0.9})	**Lokalität**: Anteil lokal gespeicherter an allen Fragmenten der Basismenge
WR	DET({0.2,0.5,0.8})	**Update-Häufigkeit**: Anteil zu ändernder an allen Fragmenten der Basismenge
*	CHOICE(BS, E)	**Selektivität**: Jedes der E Fragmente kann bei Bildung der Basismenge mit gleicher Wahrscheinlichkeit ausgewählt werden (gilt jeweils für lokale bzw. nichtlokale Fragmente der Basismenge)

Datenbanksoftware-Architektur:

Prozesse der lokalen Datenbanksysteme verbrauchen Zeit zum physischen Datenzugriff auf Hintergrundspeicher oder zur -i.d.R. nebenläufigen- Weiterverarbeitung von Daten auf Arbeitsrechnern, z.B. zur Datenaggregierung oder Anfrageauswertung. Es werden folgende Zeitverbrauchs-Typen unterschieden:

T1	DET(0.001)	AR-Zeitverbrauch für 'einfache' DBVS-Operationen wie Sperren, Kataloglesen in [sec/Fragment]
T2	UNIF(0.01,0.05)	AR-Zeitverbrauch zur Ausführung einer Subtransaktion (SUB_DO) in [sec/Fragment]
T3	UNIF(0.15,0.30)	AR-Zeitverbrauch zur Transaktions-Vorverarbeitung (syntakt. und semantische Analyse von ad-hoc-Transaktionen und Generierung aller Sub-Transaktionen)
T4	ERLANG(1/0.05, 2)	Plattenspeicher-Zugriffsrate zum phys. Lesen bzw. Ändern eines Fragments in [1/sec], nach /WILD79/.

Daraus wird das **operationale DBVS-Zeitverhalten** abgeleitet als lineares Modell der obigen Zeitverbrauchsvariablen sowie der Anzahl Fragmente, auf die eine Transaktion T bzw. eine Subtransaktion SUB zugreifen. Sei

R := |R(T)| R' := |R(SUB)|
W := |W(T)| W' := |W(SUB)|

Operation	Arbeitsrechner-Zeitverbrauch	Massenspeicher-Zeitverbrauch	Erläuterungen bzw. Annahmen
COMPILE	T3 + (R+W)•T1	(R+W)•T4	T-Auswertung
LOCK	T1	-	Sperrtabellen im
UNLOCK	T1	-	Hauptspeicher
SUB_DO	(R'+W')•T2	(R'+2W')•T4	*)
SUB_UNDO	-	W'•T4	**)
SAVE_STATE	-	T4	AR-Zeitverbrauch
CHECK_STATE	-	T4	vernachlässigt
RESTORE_STATES	-	-	***)

*) 2-fache Schreibkosten angenommen wegen Speichern der Before-Images ins Logbuch

**) Diese Kosten entstehen nur, falls der 'COMMIT-Punkt' von SUB schon erreicht wurde, weil dann die Before-Images vom Logbuch in die Datenbasis zurückzuschreiben sind.

***) Die Kosten für RESTORE_STATES werden meist erheblich höher sein als bei SAVE_STATE und CHECK_STATE, da evtl. eine große Zahl von Logbuch-Einträgen, entsprechend der Anzahl offener Transaktionen, gelesen wird. Das ist aber unerheblich: RESTORE STATES wird nur bei Restart aufgerufen; da unser Ausfallmodell aber MTTF » MTTR annimmt, sind die Recovery-Kosten im Normalfall wesentlich höher. Daher vernachlässigen wir die anfallenden Restartkosten.

B Netzweites VDBS-Datagrammformat

Die Prozesse des VDBS-Simulationsmodells kommunizieren miteinander entsprechend einer global gültigen Vereinbarung bezüglich den Attributen einer Sende- bzw. Empfangs-Nachricht. Damit wird gleichzeitig für unser Modell eine uniforme **Tokenstruktur** (s. Kap. 6.1.3) im VDBS-Simulationsmodell spezifiziert. Eine solch strukturierte Nachricht hat dabei die Länge 100 Bytes. Wegen ihren voll spezifierten Quell- und Zieladressen ist eine Nachricht auch als **Datagramm** aufzufassen.

LFD. NR.	ATTR.-ID	BEREICH	BEDEUTUNG
			Header (Quellinfo) :
1	MTYP	1,99	Nachr.-Typen: 1=snd_m 2=requ_m 3=requ_r 4=snd_r 5=watch_AV 6=stop_watch IPC-intern: 7=AV-/CP-Statusvektor 8=reserviert 9=own_restart
2	HOME	1,9	Eingabeknoten einer Transaktion -> Transaktions-
3	SEQ	0,9999	Sequenz-Nummer -> ID
4	MQ	1,99	'Quell'-Mailbox für zu empfangende Nachricht
5	KQ	1,9	Quell-Knoten (nicht notwendig gleich mit KE)
			Header (Zielinfo) :
6	MZ	1,99	Ziel-Mailbox (bei Multi-MSG für alle KZ gleich)
7	KZ	1,9	(aktueller) Zielknoten
8	NKZ	1,9	Anzahl Zielknoten (≥1)
9	KZi	9*(1,9)	Zielknoten_1 , ... ,Zielknoten_NKZ
			Ergebnis-Info (Sender) :
10	MDAT	1,99	1=ACK,commit 2=NACK,abort 3=AV 4=CP ≥5 : anwendungsabhängig
			Ergebnis-Info (Empfänger) :
11	NRES	1,9	akt. Anzahl erhaltener results (0≤NRES≤NKZ)
12	RESi	9*(1,9)	Results: 1=commit 2=abort (höhere Prozesse) 3=AV 4=CP (von IPC generiert) 0=noch kei result erhalten
			Recovery-Info :
13	TMX	1,99	aktuelle crashrelevante TM-Mailbox
14	DMX	1,99	" " DM-Mailbox
15	TMST	1,9	T-Bearbeitungs-Status durch TM : 0=offen 1=normal-committed 2=blockiert-committed 3=normal-aborted 4=blockiert-aborted
16	DMST	1,9	T-Bearbeitungs-Status durch DM : wie TMST
17	ISCR	0,1	1=HOME war während T-Bearbeitung DOWN, 0=sonst

Basismengen- und Subtransaktions-Info :

18	NLR	1,5	Kardinalität log. Lese/Schreibmenge, es gilt:
19	NLW	1,5	NLR + NLW <= 5
20	Ei	5*(1,400)	Basismenge mit NLR+NLW Fragment-IDs
21	NDM	1,9	Anzahl Partnerknoten ("Subtransaktions-Knoten")
22	DMi	9*(1,9)	Knoten_1, ... , Knoten_NDM
23	NPR	9*(1,5)	Kardinalität phys. Lese/Schreibmenge
24	NPW	9*(1,5)	auf Partnerknoten 1, ... ,NDM
25	COP(j)	5*(1,400)	j-te Fragmentkopie der phys. Lese/Schreibmenge
			auf Partnerknoten 1, ... ,NDM
			(je NPR(i) + NPW(i) Kopien pro Partnerknoten i)

Performance-Info :

26	TARR	0,9999.9	abs. Transaktions-Ankunftszeit
27	TR1	"	Antwortzeit für Dienst_1
28	TR2	"	Antwortzeit für Dienst_2

C Eine kommentierte Beispielsitzung mit dem DAEMON-System

Nachfolgend ist eine Beispielsitzung mit dem DAEMON-System zusammengestellt, die wesentliche Dialoge des Endbenutzers mit dem interaktiven Performance-Monitor (IPM) und Endergebnisse eines Simulationslaufs zeigt. Dabei beziehen wir uns auf das in den Kapiteln 4 und 7 beschriebene VDBS-Modell. Benutzereingaben sind in Kleinschreibung angegeben.

Bei Aufruf des DAEMON-Simulators wird das Netzmodell "VDBMODEL" sowie dessen Eingabeparameter von Datei eingelesen (Tab. C.1). Diese Parameter sind:

- 8 Terminals pro Knoten
- Lokales Rechnernetz
- Mehrheitsentscheid-Protokoll
- 3 Kopien pro Dateneinheit
- mittlere Datenbank-Anforderungen

```
READING VDBMODEL ....
 *** SIM-TIME:            1000.0
 *** INITIAL SEED:          97
 *** SPECS      read, O.K. ...
 *** MODEL      read, O.K. ...   SIZE (CH X AG X ARC) = 366 X 296 X1054
 *** PARAMETERS read, O.K. ...

*******************************************************************************
************** SIMULATION STARTED  AT 27 NOV 1985   8:32:01.14  **************
*******************************************************************************

*******************************************************************************
****  I N T E R A C T I V E     P E R F O R M A N C E     M O N I T O R    ****
****                                                                       ****
**** CURRENT SIMULATION TIME:    0.000 COMMITTED TRANSACTIONS:    0       ****
**** BREAKPOINT  INTERVAL   :    5.00 STARTED   TRANSACTIONS:    0       ****
****                                                                       ****
****                        BREAKPOINT ? <Y | N>                           ****
*******************************************************************************
*
                          O P T I O N S :

 SET    SET    SHOW/SET    SHOW/SET      SHOW/SET    SHOW   HELP  EXIT
BREAK  TRACE   MARKING  STOP-CONDITION  PARAMETERS  RESULTS
 <B>    <T>     <M>          <C>          <P>        <S>    <H>  <X>

b
 NEXT BREAKPOINT TIME        INTERVAL =
35.8
 NO. TERMINATED TRANSACTIONS INTERVAL =
0



**********************************************************************
************** INTERACTIVE CRASH/RESTART GENERATION ****************
**********************************************************************

  CPU-     NIU-      STATUS
1234567  1234567     0=down  1=up
1111111  1111111
 ***  STATUS CHANGE <Y|N> : ***
n
 *** next event TIME (sec): ***
43.0
```

Tab. C.1: Beginn des Simulationslaufs

Zu Beginn des Simulationslaufs wird dieser durch den IPM angehalten. Es wird ein Haltepunkt auf 35.8 sec (virtuelle Simulationszeit) gesetzt. Anschließend meldet sich eine interaktive Instanz der IPC-Schicht, mit der der Makrozustand der Arbeitsrechner und NIUs verändert werden kann. Das erste Ausfallereignis wird zum Zeitpunkt 43.0 vorgesehen.

Bei Erreichen des Haltepunktes wird die Simulation unterbrochen (Tab. C.2). Es wird ein Trace der Multicast-Instanz "MCAST" auf Knoten K_3 über 2 sec initiiert. Als Ausgabeoptionen werden vollständiges EA-Verhalten der Instanz mit minimalem Detaillierungsgrad gewählt, d.h. für jede Eingabe- und Ausgabemarke der Instanz werden lediglich das Transaktions-ID, der Nachrichtentyp, sowie Quell- und Zielknoten bzw. -Mailboxes angegeben.

```
**********************************************************************************
****  I N T E R A C T I V E     P E R F O R M A N C E     M O N I T O R    ****
****                                                                          ****
**** CURRENT SIMULATION TIME:   35.806 COMMITTED TRANSACTIONS:   51          ****
**** BREAKPOINT  INTERVAL   :    35.80 STARTED   TRANSACTIONS:   75          ****
****                                                                          ****
****                        BREAKPOINT ? <Y | N>                              ****
**********************************************************************************
y
                          O P T I O N S :

 SET    SET    SHOW/SET    SHOW/SET      SHOW/SET    SHOW   HELP  EXIT
BREAK  TRACE   MARKING  STOP-CONDITION  PARAMETERS  RESULTS
 <B>    <T>     <M>          <C>          <P>        <S>    <H>  <X>

t
  TERMINAL  | PROTOCOL     NEWTRACE | OLDTRACE
    <T>     |   <P>          <N>    |   <O>
tn
 S H O W  S H O W    T R A C E    T R A C E    T R A C E   RETURN
 AGENCIES RESOURCES  TRANSACTION   AGENCY      RESOURCE  MAIN-MENU
   all      all        select     select all    select
   <A>      <B>        <T>        <X>  <Y>      <Z>       <M>

x
<n,  AGENCY-ID-1, ..., -n>   n in (1,30):
1
mcast3
      S E T  T R A C E  O U T P U T  O P T I O N S :

    FIRING BEHAVIOR          TOKEN ATTRIBUTE MNEMONICS

 out-tokens in/out-tokens   none minimal medium   all
    *           <I>          <1>   <2>    <3>     <4>

i2
  NEXT BREAKPOINT INTERVAL =
2
 NO. TERMINATED TRANSACTIONS INTERVAL =
0
                  A G E N C Y   T R A C E
----------------------------------------------------------------------------
D_MCAST-agency 'MCAST3'  resource CO_INCID  worktime  0.0000  ended at  36.811
----------------------------------------------------------------------------

 IN-CHANNEL   IP13
 IN-TOKEN    < TRANSACTION-ID: 30011   ARRIVED AT   33.33

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          3               8      1
               MESSAGE-DESTINATION(S):  1 2 3 4 5       13     -- >

 IN-CHANNEL   PSTAT3
 IN-TOKEN    < 71111111                                        >
```

```
OUT-CHANNEL   PORT1
OUT-TOKEN    < TRANSACTION-ID: 30011   ARRIVED AT   33.33

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          3                8      1
               MESSAGE-DESTINATION(S):  1 2 3 4 5        13     -- >

OUT-CHANNEL   PORT2
OUT-TOKEN    < TRANSACTION-ID: 30011   ARRIVED AT   33.33

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          3                8      1
               MESSAGE-DESTINATION(S):  1 2 3 4 5        13     -- >

OUT-CHANNEL   LOCAL3
OUT-TOKEN    < TRANSACTION-ID: 30011   ARRIVED AT   33.33

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          3                8      1
               MESSAGE-DESTINATION(S):  1 2 3 4 5        13     -- >

OUT-CHANNEL   PORT4
OUT-TOKEN    < TRANSACTION-ID: 30011   ARRIVED AT   33.33

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          3                8      1
               MESSAGE-DESTINATION(S):  1 2 3 4 5        13     -- >

OUT-CHANNEL   PORT5
OUT-TOKEN    < TRANSACTION-ID: 30011   ARRIVED AT   33.33

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          3                8      1
               MESSAGE-DESTINATION(S):  1 2 3 4 5        13     -- >
```

Tab. C.2: Trace der Instanz MCAST

Am nächsten Haltepunkt soll die Transaktions-Auswertung am Beispiel der Instanzen "COMPILE" und "GENSUB" auf Knoten K_5 über 5.5 sec detailliert getraced werden (Tab. C.3). Instanz COMPILE legt den Transaktions-ID T = (5,14) fest und generiert die logische Basismenge. (Vergl. Anhang B zur Bedeutung der Markenattribute und der Codierung der Attributwerte.) Instanz GENSUB generiert daraus alle Subtransaktionen von T und gibt genauere Information darüber auf Bildschirm aus (TEST-OUTPUT T-GENERATION). Dabei wird für jede Dateneinheit der Basismenge der Bezeichner, die Primärlokation, die Anzahl verfügbarer Kopien aus Sicht des Heimatknotens, sowie der Zugriffsmodus angegeben (r bzw. w für lokale Lese- bzw. Schreib-Primärkopie, R bzw. W für abgesetzte Lese- bzw. Schreib-Primärkopie).

```
**********************************************************************************
****  I N T E R A C T I V E     P E R F O R M A N C E     M O N I T O R     ****
****                                                                          ****
**** CURRENT SIMULATION TIME:   37.807 COMMITTED TRANSACTIONS:   54          ****
**** BREAKPOINT  INTERVAL   :    2.00 STARTED   TRANSACTIONS:   82          ****
****                                                                          ****
****                        BREAKPOINT ? <Y | N>                              ****
**********************************************************************************
y
                          O P T I O N S :

 SET    SET    SHOW/SET    SHOW/SET      SHOW/SET    SHOW   HELP  EXIT
BREAK  TRACE  MARKING  STOP-CONDITION  PARAMETERS  RESULTS
 <B>    <T>     <M>          <C>          <P>        <S>    <H>  <X>

t
  TERMINAL | PROTOCOL     NEWTRACE | OLDTRACE
    <T>    |   <P>          <N>    |   <O>
tn
  S H O W  S H O W     T R A C E     T R A C E     T R A C E   RETURN
 AGENCIES RESOURCES   TRANSACTION     AGENCY       RESOURCE  MAIN-MENU
   all       all         select      select all     select
   <A>       <B>          <T>        <X>  <Y>       <Z>        <M>

x
 <n,  AGENCY-ID-1, ..., -n>   n in (1,30):
2
compile5
gensub5
      S E T  T R A C E  O U T P U T  O P T I O N S :

    FIRING BEHAVIOR          TOKEN ATTRIBUTE MNEMONICS

out-tokens in/out-tokens   none minimal medium   all
    *           <I>         <1>   <2>     <3>    <4>

14
  NEXT BREAKPOINT INTERVAL =
5.5
 NO. TERMINATED TRANSACTIONS INTERVAL =
0
                   A G E N C Y    T R A C E

 ------------------------------------------------------------------------------
 D_TGEN1-agency 'COMPILE5'  resource CPU5  worktime  0.2504  ended at  41.127
 ------------------------------------------------------------------------------

  IN-CHANNEL   MB45
  IN-TOKEN    < TRANSACTION-ID: 50000   ARRIVED AT    0.00

                                TRANSACTION MANAGER  DATA MANAGER
                  RESTART-STATE:     open / blocked       open / blocked
                  RESTART-MAILBOX:   --                   --

                  MESSAGE-TYPE: --
                                          SITE(S)       MAILBOX  DATA
                  MESSAGE-SOURCE:         5               --     --
                  MESSAGE-DESTINATION(S): 5                4     -- >

  IN-CHANNEL   PSTAT5
  IN-TOKEN    < 71111111                                  >

 OUT-CHANNEL   KT15
 OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

                                TRANSACTION MANAGER  DATA MANAGER
                  RESTART-STATE:     open / blocked       open / blocked
                  RESTART-MAILBOX:   --                   --

                  LOGICAL BASESET:       (ENTITY, ACCESS MODE, PRIMARY SITE)
                           < (29,r,5) (93,w,5) (159,w,5) (47,w,4) >

                  SUBTRANSACTION-SITE(S): < 4 5 6 7 >

                  MESSAGE-TYPE: send_msg
                                          SITE(S)       MAILBOX  DATA
                  MESSAGE-SOURCE:         5               --      1
                  MESSAGE-DESTINATION(S): 5                4     -- >
```

```
 TEST-OUTPUT T-GENERATION
 ========================
OWN_NODE IS  5
Base-Set     29   93  159   47
 r/w R/W      r    w    w    W
prim-node     5    5    5    4
AV-copies     3    3    3    3

Generated SUBTRANSACTIONS:
DM-node  NPR  NPW  copies
   4      0    1   47
   5      1    3   29   93  159   47
   6      1    3   29   93  159   47
   7      1    2   29   93  159

------------------------------------------------------------------------------
D_TGEN3-agency 'GENSUB5'  resource CPU5  worktime  0.0000  ended at  41.354
------------------------------------------------------------------------------

 IN-CHANNEL   KT25
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

                                 TRANSACTION MANAGER  DATA MANAGER
                RESTART-STATE:    open / blocked       open / blocked
                RESTART-MAILBOX:  --                   --

                LOGICAL BASESET:       (ENTITY, ACCESS MODE, PRIMARY SITE)
                        < (29,r,5) (93,w,5) (159,w,5) (47,w,4) >

                SUBTRANSACTION-SITE(S): < 4 5 6 7 >

                MESSAGE-TYPE: send_msg
                                        SITE(S)       MAILBOX  DATA
                MESSAGE-SOURCE:         5               --      1
                MESSAGE-DESTINATION(S): 5                4     -- >

OUT-CHANNEL   MB25
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
                                 TRANSACTION MANAGER  DATA MANAGER
                RESTART-STATE:    open / blocked       open / blocked
                RESTART-MAILBOX:   5                   --

                LOGICAL BASESET:       (ENTITY, ACCESS MODE, PRIMARY SITE)
                        < (29,r,5) (93,w,5) (159,w,5) (47,w,4) >

                SUBTRANSACTION-SITE(S): < 4 5 6 7 >

                MESSAGE-TYPE: TM-save-state
                                        SITE(S)       MAILBOX  DATA
                MESSAGE-SOURCE:         5               5       5
                MESSAGE-DESTINATION(S): 5                4     -- >
```

Tab. C.3: Trace der Instanzen COMPILE und GENSUB

```
*********************************************************************
************** INTERACTIVE CRASH/RESTART GENERATION ***************
*********************************************************************

  CPU-     NIU-      STATUS
1234567  1234567     0=down  1=up
1111111  1111111
 ***  STATUS CHANGE <Y|N> : ***
y
 *** new event on NODE-ID : ***
6
 *** CPU or NIU   (1 or 2): ***
2
 *** NIU6-CRASH occurred at TIME    43.01
```

Tab. C.4: Interaktive Erzeugung eines NIU_6-Ausfalls

Zum Zeitpunkt 43.0 wird ein NIU_6-Ausfall interaktiv erzeugt (Tab. C.4). Das nächste Ausfall- bzw. Wiederanlauf-Ereignis wird zu einem sehr späten Zeitpunkt angesetzt, sodaß es für den vorliegenden Simulationslauf irrelevant ist.

Am nächsten Haltepunkt wird ein Transaktions-Trace von T initiiert (Tab. C.5). Dabei soll lediglich das Verarbeitungs-Protokoll der Transaktion auf der Ebene der TM- und DM-Prozesse inspiziert werden; der Protokollablauf in tieferen Schichten wie IPC- und Transport-Schicht und in Logging-Prozessen soll nicht gezeigt werden.
In der ersten beobachteten Instanzaktivierung wird gerade der erste Sperrauftrag an DM-Knoten K_4 verschickt (Partnerknotenmenge ist {4,5,6,7}, s. Tab. C.4). Die Sperraufträge an Knoten 4 und 5 werden ausgeführt und positiv quittiert (C.5-c und g). Der Auftrag an Knoten 6 wird von der lokalen IPC-Schicht mit einer Unverfügbarkeits-Meldung an den Absender beantwortet (C.5-i und j). Nach erfolgreichem Sperren an Knoten K_7 (C.5-m) wird die Sperrphase beendet (C.5-n). Trotz Unverfügbarkeit von Knoten 6 wird T nicht abgebrochen, da an den Knoten 4, 5 und 7 eine Kopienmehrheit für jede Dateneinheit der Basismenge gesperrt werden konnte.
Nach Zustandssicherung auf Logbuch wird für T die erste Commitphase bei entsprechend reduzierter Partnerknotenmenge gestartet (C.5-o). Nach quittierter Ausführung aller Unteraufträge (in Tabelle nicht gezeigt) trifft der TM-Prozeß eine COMMIT-Entscheidung und terminiert die Transaktion T, indem er das Endergebnis an das Benutzerterminal schickt (C.5-p bis r, lokale Terminierung aller Unteraufträge nicht gezeigt.)

```
**********************************************************************************
****  I N T E R A C T I V E     P E R F O R M A N C E     M O N I T O R    ****
****                                                                          ****
**** CURRENT SIMULATION TIME:   43.308 COMMITTED TRANSACTIONS:   71          ****
**** BREAKPOINT  INTERVAL   :    5.50 STARTED   TRANSACTIONS:   96          ****
****                                                                          ****
****                          BREAKPOINT ? <Y | N>                            ****
**********************************************************************************
J
                          O P T I O N S :

 SET    SET    SHOW/SET    SHOW/SET      SHOW/SET    SHOW    HELP  EXIT
BREAK  TRACE   MARKING  STOP-CONDITION  PARAMETERS  RESULTS
 <B>    <T>     <M>          <C>          <P>        <S>     <H>   <X>

t
  TERMINAL  | PROTOCOL     NEWTRACE | OLDTRACE
    <T>     |   <P>          <N>    |   <O>
tn
 S H O W  S H O W    T R A C E    T R A C E    T R A C E   RETURN
 AGENCIES RESOURCES TRANSACTION    AGENCY      RESOURCE  MAIN-MENU
   all      all       select     select all    select
   <A>      <B>       <T>        <X>  <Y>      <Z>        <M>

t
  HOMESITE-ID FOR NEXT TRANSACTION TO BE TRACED in (1,7)
5
   15 TRANSACTIONS HAVE BEEN INITIATED ON THIS SITE.
```

```
  CHOICE OF SEQUENCE NUMBER (0 for NEXT NEW ONE):
14
  YOUR CHOICE IS:   TRANSACTION WITH TID =  50014
 EXCLUDE SOME AGENCY TYPES FOR THIS TRACE ?
   NO   YES: SELECTION    YES: LOWER LAYERS
  <N>      <Y>              <L>
1
      S E T  T R A C E  O U T P U T  O P T I O N S :

    FIRING BEHAVIOR          TOKEN ATTRIBUTE MNEMONICS

 out-tokens in/out-tokens   none minimal medium   all
    *           <I>          <1>   <2>    <3>    <4>

1.2
  NEXT BREAKPOINT INTERVAL =
15.0
 NO. TERMINATED TRANSACTIONS INTERVAL =
0
```

T R A N S A C T I O N T R A C E

(a)

```
D_TM1-agency 'SNDLOCK5'  resource CPU5  worktime  0.0001  ended at  43.324

 IN-CHANNEL   KT35
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: ack-save
                                        SITE(S)      MAILBOX  DATA
               MESSAGE-SOURCE:          5               5      9
               MESSAGE-DESTINATION(S):  5               5      -- >

OUT-CHANNEL   KT45 PTM05
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: requ_msg
                                        SITE(S)      MAILBOX  DATA
               MESSAGE-SOURCE:          5               6     11
               MESSAGE-DESTINATION(S):  4              11     -- >
```

(b)

```
D_DM1-agency 'GETLOCK4'  resource CPU4  worktime  0.0001  ended at  43.331

 IN-CHANNEL   MB114
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: requ_msg
                                        SITE(S)      MAILBOX  DATA
               MESSAGE-SOURCE:          5               6     11
               MESSAGE-DESTINATION(S):  4              11     -- >

OUT-CHANNEL   MB24
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: DM-save-state
                                        SITE(S)      MAILBOX  DATA
               MESSAGE-SOURCE:          5              12      6
               MESSAGE-DESTINATION(S):  4              11     -- >
```

(c)

```
D_DM2-agency 'SETLOCK4'  resource CPU4  worktime  0.0010  ended at  43.860

 IN-CHANNEL   MB124
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: ack-save
                                        SITE(S)      MAILBOX  DATA
               MESSAGE-SOURCE:          4              12      9
               MESSAGE-DESTINATION(S):  4              12     -- >

 IN-CHANNEL   KD14
 IN-TOKEN    <            4 >

OUT-CHANNEL   PDM04 KD14
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: requ_reply
                                        SITE(S)      MAILBOX  DATA
               MESSAGE-SOURCE:          4              13      1
               MESSAGE-DESTINATION(S):  5               6     -- >
```

(d)

```
D_TM2-agency 'ENDLOCK5'  resource CPU5  worktime  0.0001  ended at  43.870

 IN-CHANNEL   KT45
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6    11
               MESSAGE-DESTINATION(S):  4               11    -- >

 IN-CHANNEL   MB65
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               13    11
               MESSAGE-DESTINATION(S):  4                6     1 >

OUT-CHANNEL   KT35
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6    11
               MESSAGE-DESTINATION(S):  4               11     1 >
```

(e)

```
D_TM1-agency 'SNDLOCK5'  resource CPU5  worktime  0.0001  ended at  43.870

 IN-CHANNEL   KT35
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6    11
               MESSAGE-DESTINATION(S):  4               11     1 >

OUT-CHANNEL   KT45 PTM05
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6    12
               MESSAGE-DESTINATION(S):  5               11     1 >
```

(f)

```
D_DM1-agency 'GETLOCK5'  resource CPU5  worktime  0.0001  ended at  43.874

 IN-CHANNEL   MB115
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6    12
               MESSAGE-DESTINATION(S):  5               11    -- >

OUT-CHANNEL   MB25
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: DM-save-state
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               12     6
               MESSAGE-DESTINATION(S):  5               11    -- >
```

(g)

```
D_DM2-agency 'SETLOCK5'  resource CPU5  worktime  0.0010  ended at  50.013

 IN-CHANNEL   MB125
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: send_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               12    99
               MESSAGE-DESTINATION(S):  5               12    -- >

 IN-CHANNEL   KD15
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               13     1
               MESSAGE-DESTINATION(S):  2                6    -- >

OUT-CHANNEL   PDM05 KD15
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               13     1
               MESSAGE-DESTINATION(S):  5                6    -- >
```

(h)

```
-------------------------------------------------------------------------------
D_TM2-agency 'ENDLOCK5'  resource CPU5  worktime  0.0001  ended at  50.019
-------------------------------------------------------------------------------

 IN-CHANNEL   KT45
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     12
               MESSAGE-DESTINATION(S):  5               11      1 >

 IN-CHANNEL   MB65
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               13     12
               MESSAGE-DESTINATION(S):  5                6      1 >

OUT-CHANNEL   KT35
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     12
               MESSAGE-DESTINATION(S):  5               11      1 1 >
```

(i)

```
-------------------------------------------------------------------------------
D_TM1-agency 'SNDLOCK5'  resource CPU5  worktime  0.0001  ended at  50.020
-------------------------------------------------------------------------------

 IN-CHANNEL   KT35

 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     12
               MESSAGE-DESTINATION(S):  5               11      1 1 >

OUT-CHANNEL   KT45 PTM05
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     13
               MESSAGE-DESTINATION(S):  6               11      1 1 >
```

(j)

```
-------------------------------------------------------------------------------
D_TM2-agency 'ENDLOCK5'  resource CPU5  worktime  0.0001  ended at  50.022
-------------------------------------------------------------------------------

 IN-CHANNEL   KT45
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     13
               MESSAGE-DESTINATION(S):  6               11      1 1 >

 IN-CHANNEL   MB65
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     13
               MESSAGE-DESTINATION(S):  6                6      4 >

OUT-CHANNEL   KT35
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     13
               MESSAGE-DESTINATION(S):  6               11      1 1 4 >
```

(k)

```
-------------------------------------------------------------------------------
D_TM1-agency 'SNDLOCK5'  resource CPU5  worktime  0.0001  ended at  50.022
-------------------------------------------------------------------------------

 IN-CHANNEL   KT35
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     13
               MESSAGE-DESTINATION(S):  6               11      1 1 4 >

OUT-CHANNEL   KT45 PTMO5
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     14
               MESSAGE-DESTINATION(S):  7               11      1 1 4 >
```

(l)

```
-------------------------------------------------------------------------------
D_DM1-agency 'GETLOCK7'  resource CPU7  worktime  0.0001  ended at  50.042
-------------------------------------------------------------------------------

 IN-CHANNEL   MB117
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6     14
               MESSAGE-DESTINATION(S):  7               11     -- >

OUT-CHANNEL   MB27
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: DM-save-state
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               12      6
               MESSAGE-DESTINATION(S):  7               11     -- >
```

(m)

```
-------------------------------------------------------------------------------
D_DM2-agency 'SETLOCK7'  resource CPU7  worktime  0.0010  ended at  50.265
-------------------------------------------------------------------------------

 IN-CHANNEL   MB127
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: ack-save
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          7               12      9
               MESSAGE-DESTINATION(S):  7               12     -- >

 IN-CHANNEL   KD17
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          7               13      1
               MESSAGE-DESTINATION(S):  2                6     -- >

OUT-CHANNEL   PDMO7 KD17
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          7               13      1
               MESSAGE-DESTINATION(S):  5                6     -- >
```

(n)

```
-------------------------------------------------------------------------------
D_TM2-agency 'ENDLOCK5'  resource CPU5  worktime  0.0001  ended at  50.284
-------------------------------------------------------------------------------
 IN-CHANNEL   KT45
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                6    14
               MESSAGE-DESTINATION(S):  7               11     1 1 4 >

 IN-CHANNEL   MB65
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_msg
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               13    14
               MESSAGE-DESTINATION(S):  7                6     1 >

OUT-CHANNEL   MB25
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: TM-save-state
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                7     5
               MESSAGE-DESTINATION(S):  7               11     1 1 4 1 >
```

(o)

```
-------------------------------------------------------------------------------
D_TM3-agency 'STRT2PC5'  resource CPU5  worktime  0.0001  ended at  51.276
-------------------------------------------------------------------------------
 IN-CHANNEL   MB75
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: ack-save
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                7     9
               MESSAGE-DESTINATION(S):  5                7    -- >

OUT-CHANNEL   PTM05
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5                8     1
               MESSAGE-DESTINATION(S):  4 5 7           13    -- >
```

(p)

```
-------------------------------------------------------------------------------
D_TM4-agency 'DECIDE5'  resource CPU5  worktime  0.1000  ended at  52.851
-------------------------------------------------------------------------------
 IN-CHANNEL   MB85
 IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               15     1
               MESSAGE-DESTINATION(S):  4 5 7            8     1 1 1 >

OUT-CHANNEL   KT55
OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

               MESSAGE-TYPE: requ_reply
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          5               15     1
               MESSAGE-DESTINATION(S):  4 5 7            8     1 1 1 >
```

```
(q)  ------------------------------------------------------------------------
     D_TM5-agency 'TMCOMM5'  resource CPU5  worktime  0.0001  ended at  52.851
     ------------------------------------------------------------------------

      IN-CHANNEL   KT55
      IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

                    MESSAGE-TYPE: requ_reply
                                          SITE(S)       MAILBOX  DATA
                    MESSAGE-SOURCE:       5               15     1
                    MESSAGE-DESTINATION(S): 4 5 7          8     1 1 1 >

      OUT-CHANNEL   MB25
      OUT-TOKEN   < TRANSACTION-ID: 50014   ARRIVED AT   40.88

                    MESSAGE-TYPE: TM-save-state
                                          SITE(S)       MAILBOX  DATA
                    MESSAGE-SOURCE:       5               10     5
                    MESSAGE-DESTINATION(S): 4 5 7          8     1 1 1 >

      OUT-CHANNEL   PTM05
      OUT-TOKEN   < TRANSACTION-ID: 50014   ARRIVED AT   40.88

                    MESSAGE-TYPE: send_reply
                                          SITE(S)       MAILBOX  DATA
                    MESSAGE-SOURCE:       5               15     1
                    MESSAGE-DESTINATION(S): 4 5 7         15     1 1 1 >


(r)  ------------------------------------------------------------------------
     D_TM7-agency 'END2PC5'  resource CPU5  worktime  0.0001  ended at  53.077
     ------------------------------------------------------------------------

      IN-CHANNEL   MB105
      IN-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88
                    MESSAGE-TYPE: ack-save
                                          SITE(S)       MAILBOX  DATA
                    MESSAGE-SOURCE:       5               10     9
                    MESSAGE-DESTINATION(S): 5             10     -- >

     OUT-CHANNEL   MB35
     OUT-TOKEN    < TRANSACTION-ID: 50014   ARRIVED AT   40.88

                    MESSAGE-TYPE: ack-save
                                          SITE(S)       MAILBOX  DATA
                    MESSAGE-SOURCE:       5               10     9
                    MESSAGE-DESTINATION(S): 5              3     -- >
```

Tab. C.5: Trace der Transaktion (5,14)

Am nächsten Haltepunkt werden die TM-Instanzen zur Transaktions-Übersetzung und Unterauftrags-Generierung am Knoten K_6 getraced (s. Tab. C.6). Dieser Knoten ist wegen NIU-Ausfall von allen anderen partitioniert, was die COMPILE-Instanz im Statusvektor PSTAT erfährt. Da für keine Dateneinheit der Basismenge eine verfügbare Kopienmehrheit zustande kommen kann, wird die Transaktion zurückgewiesen. Dieses Ergebnis wird von GENSUB an die Terminals geschickt.

```
**********************************************************************************
****  I N T E R A C T I V E     P E R F O R M A N C E     M O N I T O R     ****
****                                                                          ****
**** CURRENT SIMULATION TIME:   53.702 COMMITTED TRANSACTIONS:   97          ****
**** BREAKPOINT  INTERVAL   :    0.04 STARTED   TRANSACTIONS:  123          ****
****                                                                          ****
****                          BREAKPOINT ? <Y | N>                            ****
**********************************************************************************
y
                              O P T I O N S :

  SET    SET    SHOW/SET    SHOW/SET      SHOW/SET    SHOW   HELP  EXIT
 BREAK  TRACE   MARKING  STOP-CONDITION  PARAMETERS  RESULTS
  <B>    <T>     <M>          <C>          <P>        <S>    <H>  <X>

t
  TERMINAL | PROTOCOL     NEWTRACE | OLDTRACE
    <T>    |   <P>          <N>    |   <O>
tn
  S H O W  S H O W     T R A C E     T R A C E    T R A C E   RETURN
 AGENCIES RESOURCES   TRANSACTION     AGENCY      RESOURCE  MAIN-MENU
   all       all         select     select all     select
   <A>       <B>          <T>       <X>  <Y>       <Z>        <M>

x
 <n,  AGENCY-ID-1, ..., -n>   n in (1,30):
2
compile6
gensub6
       S E T  T R A C E  O U T P U T  O P T I O N S :

    FIRING BEHAVIOR          TOKEN ATTRIBUTE MNEMONICS

 out-tokens in/out-tokens   none minimal medium   all
    *            <I>         <1>   <2>     <3>    <4>

14
  NEXT BREAKPOINT INTERVAL =
0.8
 NO. TERMINATED TRANSACTIONS INTERVAL =
3


                                A G E N C Y   T R A C E

 ------------------------------------------------------------------------------
 D_TGEN1-agency 'COMPILE6'  resource CPU6  worktime  0.1855  ended at  55.423
 ------------------------------------------------------------------------------

  IN-CHANNEL   MB46
  IN-TOKEN    < TRANSACTION-ID: 60000   ARRIVED AT    0.00

                                  TRANSACTION MANAGER  DATA MANAGER
                 RESTART-STATE:     open / blocked       open / blocked
                 RESTART-MAILBOX:   --                   --

                 MESSAGE-TYPE: --
                                            SITE(S)       MAILBOX  DATA
                 MESSAGE-SOURCE:            6               --     --
                 MESSAGE-DESTINATION(S):    6                4     -- >

  IN-CHANNEL   PSTAT6
  IN-TOKEN    < 70000010                                    >

 OUT-CHANNEL   KT16
 OUT-TOKEN    < TRANSACTION-ID: 60026   ARRIVED AT   55.24

                                  TRANSACTION MANAGER  DATA MANAGER
                 RESTART-STATE:     open / blocked       open / blocked
                 RESTART-MAILBOX:   --                   --

                 LOGICAL BASESET:        (ENTITY, ACCESS MODE, PRIMARY SITE)
                          < (24,r,6) (100,w,6) (141,w,6) (63,w,2) >
                 SUBTRANSACTION-SITE(S): --

                 MESSAGE-TYPE: --
                                            SITE(S)       MAILBOX  DATA
                 MESSAGE-SOURCE:            6               --      2
                 MESSAGE-DESTINATION(S):    6                4     -- >
```

```
 TEST-OUTPUT T-GENERATION
 ========================
OWN_NODE IS  6
Base-Set    24  100  141   63
 r/w R/W     r    w    w    W
prim-node    6    6    6    2
AV-copies    1    1    1    0

 T rejected ...

-------------------------------------------------------------------------------
D_TGEN3-agency 'GENSUB6'  resource CPU6  worktime  0.0000  ended at  55.690
-------------------------------------------------------------------------------

 IN-CHANNEL   KT26
 IN-TOKEN    ( TRANSACTION-ID: 60026   ARRIVED AT   55.24

                                TRANSACTION MANAGER  DATA MANAGER
               RESTART-STATE:     open / blocked       open / blocked
               RESTART-MAILBOX:   --                   --

               LOGICAL BASESET:       (ENTITY, ACCESS MODE, PRIMARY SITE)
                       ( (24,r,6) (100,w,6) (141,w,6) (63,w,2) )

               SUBTRANSACTION-SITE(S): --

               MESSAGE-TYPE: --
                                        SITE(S)       MAILBOX  DATA
               MESSAGE-SOURCE:          6                --     2
               MESSAGE-DESTINATION(S):  6                 4     -- )

OUT-CHANNEL   MB36
OUT-TOKEN    ( TRANSACTION-ID: 60026   ARRIVED AT   55.24
```

Tab. C.6: Trace der Instanzen COMPILE und GENSUB am partitionierten Arbeitsrechner CPU_6

In den Tabellen C.8 bis C.12 sind einige Zwischenresultate der Leistungsanalyse aufgeführt. Diese können durch Direktkommandos am Terminal inspiziert werden (Tab. C.7).

```
                              O P T I O N S :

 SET    SET    SHOW/SET    SHOW/SET      SHOW/SET    SHOW   HELP  EXIT
BREAK  TRACE  MARKING  STOP-CONDITION  PARAMETERS  RESULTS
 (B)    (T)    (M)          (C)           (P)        (S)    (H)  (X)

h
            DIRECT  COMMAND  LIST  FOR  PERFORMANCE MONITOR
-------------------------------------------------------------------------------

H, HELP     this info

AGOUT       agency specification to terminal
ROUT        resource specification to terminal

SNOPT       set snapshot options
(P)SNAP     full channel-snapshot (into file)

TROPT       set trace    options
AGTR        complete agency trace to terminal
(P)TRTR(L)  transaction trace for home-node = 1
            P = (into file)    L = (no lower layer info)

   Intermediate statistical output :
AGSTA       selective   agency     statistics
AGCOM       complete    agency     statistics
CHCOM       complete    channel    statistics
RCOM        complete    resource   statistics
TRCOM       complete  transaction  statistics
```

Tab. C.7: Liste der Direktkommandos des Performance-Monitors

Tabelle C.8 entält eine Statistik über alle Arbeitsrechner, Plattenspeicher, NIUs und Terminals. Die logische Ressource MACRO dient zur Modellierung der Verzögerungszeit bis zum nächsten Ausfall- bzw. Wiederanlauf-Ereignis.

Der Auszug aus der Kanal-Statistik (Tab. C.9) zeigt mittlere Queue-Größen und -Verzögerungen pro Marke, jeweils für die Terminal-Warteschlangen (MB3-Kanäle), die Queues vor der TM-Instanz ENDLOCK (Warten auf erfolgtes Sperren, KT4-Kanäle), sowie die Warteschlangen vor den Sperr-Schedulern (MB12-Kanäle).

In Tabelle C.10 ist ein Auszug aus der Instanz-Statistik dokumentiert. Detaillierte Angaben enthält die individuelle Statistik in C.11 am Beispiel der Sperrscheduler-Instanz SETLOCK. Dabei ist interessant, daß nur jede dritte Sperrversuch erfolgreich war. (Die Vorbedingung der Instanz wird bei jeder neu ankommenden Subtransaktion mit Sperranforderungen und bei jeder Freigabe von Sperren auf Erfüllbarkeit geprüft.)

```
                              O P T I O N S :

 SET     SET     SHOW/SET     SHOW/SET        SHOW/SET     SHOW     HELP  EXIT
 BREAK   TRACE   MARKING      STOP-CONDITION  PARAMETERS   RESULTS
 <B>     <T>     <M>          <C>             <P>          <S>      <H>   <X>

room
                R E S O U R C E   P E R F O R M A N C E   at time:    53.70
-------------------------------------------------------------------------------
Name      Utilization   Service Time    Arrival Times    Activities
             %              avg             avg             No.
-------------------------------------------------------------------------------
CPU1         21.95          0.006           0.029           1854
CPU2         21.66          0.007           0.030           1752
CPU3         21.35          0.007           0.031           1714
CPU4         21.79          0.006           0.029           1830
CPU5         22.88          0.007           0.029           1853
CPU6         21.84          0.008           0.035           1526
CPU7         24.07          0.007           0.029           1845
DSK1         63.86          0.074           0.116            461
DSK2         63.22          0.077           0.122            440
DSK3         60.15          0.077           0.127            422
DSK4         67.55          0.077           0.114            469
DSK5         66.86          0.078           0.117            459
DSK6         52.68          0.077           0.145            367
DSK7         63.27          0.078           0.123            438
MACRO        80.07         43.000          43.000              1
NIU1          8.59          0.018           0.206            260
NIU2          8.03          0.018           0.222            241
NIU3          7.86          0.018           0.226            237
NIU4          8.23          0.017           0.203            263
NIU5          8.49          0.017           0.205            261
NIU6          6.11          0.017           0.217            195
NIU7          8.62          0.018           0.207            260
TRM1         94.77          2.827           2.827             18
TRM2         95.51          2.849           2.849             18
TRM3         98.56          2.940           2.940             18
TRM4         94.42          2.983           2.983             17
TRM5         90.38          2.855           2.855             17
TRM6         99.03          2.216           2.216             24
TRM7         97.97          2.505           2.505             21
```

Tab. C.8: Ressourcen-Zwischenstatistik

chcom

C H A N N E L P E R F O R M A N C E at time: 53.70

TYPE	NAME	CAPACITY	INIT_SIZE	MEAN_SIZE	MEAN_DELAY
FIFO	MB31	40	8	4.810	13.277
FIFO	MB32	40	8	5.005	13.518
FIFO	MB33	40	8	4.839	13.219
FIFO	MB34	40	8	4.626	13.431
FIFO	MB35	40	8	5.103	13.743
FIFO	MB36	40	8	5.515	11.317
FIFO	MB37	40	8	5.245	12.494
• • •					
SET	KT41	40	0	2.296	0.877
SET	KT42	40	0	2.037	0.775
SET	KT43	40	0	2.180	0.883
SET	KT44	40	0	2.381	1.066
SET	KT45	40	0	1.761	0.577
SET	KT46	40	0	1.444	0.439
SET	KT47	40	0	1.803	0.513
• • •					
SET	MB121	90	0	2.025	0.672
SET	MB122	90	0	1.349	0.166
SET	MB123	90	0	1.451	0.345
SET	MB124	90	0	1.769	0.537
SET	MB125	90	0	1.330	0.247
SET	MB126	90	0	0.875	0.034
SET	MB127	90	0	1.026	0.035
• • •					

Tab. C.9: Kanal-Zwischenstatistik (Auszug)

agcom

A G E N C Y P E R F O R M A N C E at time: 53.70

AGENCY	TYPE	RESOURCE	ACTIVATIONS	AVG.SERVICE-TIME
TRINPUT1	D_TERM	TRM1	19	2.678
TRINPUT2	D_TERM	TRM2	19	2.699
TRINPUT3	D_TERM	TRM3	19	2.786
TRINPUT4	D_TERM	TRM4	18	2.817
TRINPUT5	D_TERM	TRM5	18	2.696
TRINPUT6	D_TERM	TRM6	25	2.127
TRINPUT7	D_TERM	TRM7	22	2.391
COMPILE1	D_TGEN1	CPU1	18	0.226
COMPILE2	D_TGEN1	CPU2	18	0.222
COMPILE3	D_TGEN1	CPU3	18	0.223
COMPILE4	D_TGEN1	CPU4	17	0.230
COMPILE5	D_TGEN1	CPU5	17	0.233
COMPILE6	D_TGEN1	CPU6	24	0.225
COMPILE7	D_TGEN1	CPU7	21	0.226
CATLG1	D_TGEN2	DSK1	18	0.122
CATLG2	D_TGEN2	DSK2	18	0.127
• • •				

Tab. C.10: Instanzen-Zwischenstatistik (Auszug)

```
agsta
  AGENCY - NAME :
setlock7
                A G E N C Y   P E R F O R M A N C E   at time:    53.70
-------------------------------------------------------------------------------
TYPE              :    D_DM2    NAME           : SETLOCK7    RESOURCE: CPU7
LAST-WORKTIME     :    0.001    MEAN-WORKTIME  :    0.001
NO. OF ACTIVATIONS:       64    UTILIZATION (%):     0.12
NO. OF PRECOND-CALLS:    177
-------------------------------------------------------------------------------

                                IO-CHANNELS

TYPE  NAME    CAPACITY ACT_SIZE MEAN_SIZE  MEAN_DELAY  %-UTILIZATION
===============================================================================
 <--  MB127      90        1        1.03        0.03        98.46
 <--  KD17        1        0        0.06        0.01       100.00
 -->  PDMO7      40        0        0.00        0.00       100.00
 -->  MB27      120        8        1.35        0.24        97.37
 -->  KD17        1        0        0.06        0.01       100.00
```

Tab. C.11: Einzelinstanz-Zwischenstatistik

```
trcom
   123 TRANSACTIONS have been started,
    97 TRANSACTIONS    have   terminated.

  give an OBSERVATION WINDOW <from, to> :
20
80
        T R A N S A C T I O N   P E R F O R M A N C E
-------------------------------------------------------------------------------
TRANSACTION WINDOW:      20     80
   TIME     WINDOW:   17.54  46.62

NODE   NO.STARTED   NO.STARTED    THRUPUT     RESPONSE
                     & ENDED       RATES       TIMES
================================================
  1         13          9         0.309       6.232
  2         13          9         0.309       7.283
  3         13          8         0.275       5.670
  4         10          9         0.309       9.273
  5         13         10         0.344       5.525
  6         13          8         0.275       5.657
  7         13         12         0.413       4.675
------------------------------------------------
TOTAL :     88         65         2.235


              R E S P O N S E   T I M E S
-------------------------------------------------------------------------------
SERVICE  MINIMUM  MAXIMUM   MEAN  95%-Confidence  ST. DEVIATION
   1       0.55    13.87    3.87      +-  0.45          2.82
   2       1.18    17.59    6.26      +-  0.59          3.68
```

Tab. C.12: Transaktions-Zwischenstatistik

Eine transaktionsbezogene Statistik wird in Tab. C.12 gezeigt. Dabei kann ein Ausschnitt aller bisher termininierter Transaktionen gewählt werden. Gezeigt werden

- Mittlere Durchsatz und Antwortzeit pro Heimatrechner,
- Mittlere Antwortzeit für Auswertungs- und Sperrphase (Dienst_1), sowie für die gesamte Verarbeitung einer Transaktion.

Abschließend folgt in Tab. C.13 ein Auschnitt aus der am Ende eines Simulationslaufes erstellten Ergebnisdatei.

```
T R A N S A C T I O N - STATISTICS: Response Times & Thruputs
------------------------------------------------------------------

    No. of SERVICES =   2

TOTAL SIMTIME  313.204

 EMPIRICAL ESTIMATIONS, HISTOGRAMMS, BATCH MEANS
 -----------------------------------------------

T's started     777
T's committed   721
T's aborted      34

T's rejected    219

T's transient   120 at time   65.657 (abs.)
T's stationary  600          260.985 (stationary SIMTIME)

NODE   NO.STARTED   NO.STARTED   THRUPUT     RESPONSE
                    & ENDED      RATES       TIMES
=================================================
  1        110         106        0.406       7.425
  2         94          91        0.349       6.821
  3        122         107        0.410       6.769
  4          0           0        0.000       0.000
  5        120         116        0.444       6.370
  6        115         113        0.433       7.081
  7        102          99        0.379       8.566

TOTAL THRUPUT                              2.422 (T/sec)


T-COMMIT RATE                             95.50 (%)
T-BLOCK RATE                               0.26 (%)
T-CONFLICT RATE      per Subtransaction   12.08 (%)
T-CONFLICT RATE      per Transaction      39.06 (%)
T-REJECTION RATE     per Transaction      21.99 (%)
NO. SUBTRANSACTIONS per Transaction        3.68 (1/T)

RESPONSE TIME by SERVICE: SUMMARY, HISTOGRAMM, BATCH MEANS
----------------------------------------------------------


SERVICE 2    MIN         1.40
---------    MAX        23.33
             EMP_MEAN    7.15 +-  0.17 with 95% Confidence
             EMP_DEV     3.40

          BATCH MEANS METHOD: RESULTS   (BATCH SIZE =   40)

          BATCH MEAN  1 =  10.46
          BATCH MEAN  2 =  10.19
          BATCH MEAN  3 =   6.28
          BATCH MEAN  4 =   6.71
          BATCH MEAN  5 =   7.92
          BATCH MEAN  6 =   9.40
          BATCH MEAN  7 =   8.24
          BATCH MEAN  8 =   6.93
          BATCH MEAN  9 =   7.82
          BATCH MEAN 10 =   7.78
          BATCH MEAN 11 =   6.38
          BATCH MEAN 12 =   4.10
          BATCH MEAN 13 =   5.40
          BATCH MEAN 14 =   5.36
          BATCH MEAN 15 =   6.20

          TOTAL MEAN    =   7.28 +-  0.82 with 95% Confidence
          STANDARD DEV. =   1.81

             H I S T O G R A M M
             underflow / overflow in cell 0 / 11        CUT-OFF =  3.0 %

             CELL MIN  Ts  FREQU |
             ------------------- | ----------------------------------------
               0  1.4   6   1.0  | **
               1  1.8  40   7.7  | ************
               2  3.3 109  25.8  | **********************************
               3  4.8 131  47.7  | ****************************************
               4  6.3 123  68.2  | ***************************************
               5  7.8  76  80.8  | ************************
               6  9.3  40  87.5  | ************
               7 10.8  22  91.2  | *******
               8 12.3  13  93.3  | ****
               9 13.8  17  96.2  | *****
              10 15.3   8  97.5  | ***
              11 16.7  15 100.0  | *****
```

Tab. C.13: Transaktionsbezogene Endstatistik

D Abkürzungen und Symbole

AM	Arbeitsrechner-Menge $\subseteq$ {AR(1), ... , AR(N)}
AR	Arbeitsrechner, 'Host'
ASM	Ausfall-Submodell
AV(AR)	Meldung "AR wieder verfügbar" (AVailable)
AVA(n)	Systemverfügbarkeit bei $\leq$n nebenläufig auftretenden Fehlern
BRM	VDBS-Basisreferenzmodell
BS(T)	Basismenge einer Transaktion T mit BS(T) = W(T) $\cup$ R(T)
C	Anzahl Kopien pro Dateneinheit
COHORT(T)	Knotenmenge der Transaktion T, die Subtransaktionen verarbeitet
CP(AR)	Meldung "AR nicht verfügbar" (Crashed / Partitioned)
D	mittl. Denkzeit pro Terminal
DB	Datenbank
DBVS	Datenbankverwaltungssystem
DD/D	Data Dictionary / Directory
DE	Dateneinheit, Fragment
DN	DAEMON-Netz
DM(i)	Datenmanager-Prozeß auf Rechner AR(i)
DS	mittl. Durchsatz (Anzahl beendete Aufträge pro Zeiteinheit)
E	Anzahl Dateneinheiten in einer verteilten Datenbank
E(X)	Erwartungswert der Zufallsgröße X
FN	Funktions-Netz
f(i,j)	j-te Funktion bzw. Instanz des Prozesses P(i)
HOME(T)	Arbeitsrechner auf dem Heimatknoten der Transaktion T
ILVM	integriertes Leistungs-Verfügbarkeits-Maß
IPC	Interprozeß-Kommunikation (Inter-Process Communication)
K	Rechnerknoten
K_Q, K_Z	Quell-, Zielknoten
LAN	Lokales Rechnernetz (Local Area Network)
LOC	mittl. Anteil lokal gespeicherter an allen Fragmenten der Basismenge von Transaktionen; 'Lokalitätsfaktor'
LOC(T)	lokal auf HOME(T) gespeicherte Primärkopien von Fragmenten der Basismenge; LOC(T) = RL(T) $\cup$ WL(T), LOC(T) $\subseteq$ BS(T)
λ	Erwartungswert der Exponential- bzw. Normalverteilung
Λ	Leistungsmaß
M	Anzahl 'ausfallbarer' Rechnernetz-Komponenten (= 2N)
M_Q, M_Z	Quell-, Ziel-Mailbox
MTTF	Mean Time To Failure (=mittl. Intaktzeit)
MTTR	Mean Time To Repair (=mittl. Reparaturzeit)
N	Gesamtanzahl Arbeitsrechner bzw. NIUs im Rechnernetz
NIU	Network Interface Unit (Netzanpassungs- / Vermittlungsrechner)
NSUB	Mittlere Anzahl Subtransaktionen pro Transaktion
OSM	Operations-Submodell
p	Wahrscheinlichkeit
P, P(i)	Prozeß (im Sinne des Betriebssystems)
PAR	Partitioniertes Subsystem (bei Zuordnung Komponenten zu Rechnern)
PROT_ALL	Vollzustimmungs-Transaktionsverarbeitungs-Protokoll
PROT_PRIM	Primärkopien-Transaktionsverarbeitungs-Protokoll
PROT_MAJ	Mehrheitsentscheids-Transaktionsverarbeitungs-Protokoll
PrT-Netz	Prädikats-Transitions-Netz
qv_{ij}	Verzweigungswahrscheinlichkeit von i-ter zu j-ter Funktion
R(T)	Lesemenge einer Transaktion T
REL(n)	Systemzuverlässigkeit bei $\leq$n nebenläufig auftretenden Fehlern

REP	Vollrepliziertes Subsystem (bei Zuordnung Komponenten zu Rechnern)
RESP	mittl. Antwortzeit (Zeiteinheiten pro Auftrag)
RL(T)	Lokale Lesemenge einer Transaktion T, RL(T) $\subseteq$ R(T)
RNP(AM,AM')	Rechnernetzpartitionierung zw. disjunkten AR-Mengen AM, AM'
ρ(Si, Sj)	FIFO-Schnittstellenport zwischen Schicht i und Schicht j (Nachrichtentransport von Schicht i nach Schicht j)
SPN	Stochastisches Petri-Netz
ST-Netz	Stellen-Transitions-Netz
$SUB_i(T)$	i-te Subtransaktion der Transaktion T
σ^2	Varianz der Normalverteilung
τ	Beobachtungszeitraum der Zielgrößen eines Simulationslaufs
t(i,j)	Zeitverbrauch (-Verteilung) der Funktion f(i,j)
T	Transaktions-ID
T1	AR-Zeitverbrauch für 'einfache' DBVS-Operation
T2	AR-Zeitverbrauch zur Ausführung einer Subtransaktion
T3	AR-Zeitverbrauch zur Transaktions-Vorverarbeitung
T4	Plattenspeicher-Zugriffsrate zum Fragment-Lesen bzw. -Ändern
TCO	Transaktions-Nebenläufigkeit
TG	Nachrichtenverweilzeit zwischen entfernten Prozessen (End-To-End Delay)
TL	Nachrichtenverweilzeit zwischen lokalen Prozessen
TM(i)	Transaktionsmanager-Prozeß auf Rechner AR(i)
TR	Teilrepliziertes Subsystem (bei Zuordnung Komponenten zu Rechnern)
TRG	Teilrepliziert-gleichverteiltes Subsystem (bei Zuordnung Komponenten zu Rechnern)
TVP	Transaktionsverarbeitungs-Protokoll
VAR(X)	Varianz der Zufallsgröße X
VDB	Verteilte Datenbank
VDBS	Verteiltes Datenbank-Verwaltungssystem
W(T)	Schreibmenge einer Transaktion T
WL(T)	Lokale Schreibmenge einer Transaktion T, WL(T) $\subseteq$ W(T)
WR	Schreib-Lesefaktor: mittl. Anteil |W(T)| / (|W(T)| + |R(T))|)
Z	Zustandsraum des Ausfallmodells (Makromodell)
z_i	i-ter Zustand im Zustandsraum Z
ZEN	Zentralisiertes Subsystem (bei Zuordnung Komponenten zu Rechnern)
2PC	2-Phasen-Commitprotokoll (2-Phase-Commit Protocol)
2PL	2-Phasen-Sperrprotokoll (2-Phase-Lock Protocol)
$\mathbb{R}^+$	Menge der nichtnegativen reellen Zahlen
$\mathbb{N}$	Menge der natürlichen Zahlen
ω	Kardinalität der Menge $\mathbb{N}$
$\lfloor x \rfloor$	kleinste ganze Zahl $\geq$ x, x aus reellen Zahlen
$\lceil x \rceil$	größte ganze Zahl $\leq$ x, x aus reellen Zahlen
x « y	x ist viel kleiner als y
x » y	x ist viel größer als y
mod(n,m)	modulo-Operation, n $\in \mathbb{N} \cup \{0\}$, m $\in \mathbb{N}$
a $\wedge$ b	log. AND
a $\vee$ b	log. OR
$\neg$a	log. NOT
a => b	log. Folgerung (aus der Aussage a folgt b)
a <=> b	log. Äquivalenz (a => b $\wedge$ b => a)

$\forall$, $\exists$	Allquantor, Existenzquantor
e ε Y	e ist Element der Menge Y
X $\cup$ Y	Vereinigung der Mengen X und Y
X $\cap$ Y	Durchschnitt der Mengen X und Y
X $\subseteq$ Y	X ist Teilmenge von Y
X = Y	Gleichheit der Mengen X und Y
|X|	Kardinalität der Menge X
ϕ	leere Menge
2^X	Potenzmenge über der Menge X
X1 × X2 × ...	Cartesisches Produkt über den Mengen X1, X2, ... Menge der geordneten n-tupel <x1,x2, ... > , xi ε Xi
f:X --> Y	Abbildung von der Menge X auf die Menge Y
domain(f)	Definitionsbereich X von f: X --> Y
range(f)	Zielbereich Y von f: X --> Y
Bag(X)	Multimenge über der Menge X (vergl. /PETE81/)
@(x,B)	Anzahl Vorkommen von x ε X in B = Bag(X) (für @(x,B) ε [0,1] ist B eine 'einfache' Menge)
x ε B	x ist Element von B = Bag(x) <=> @(x,B) > 0
|B|	Kardinalität von B = Bag(X),

$$|B| := \sum_{x \in X} @(x,B)$$

LITERATURVERZEICHNIS

AKBO81 Akyildiz, I.F.; Bolch, G.: Analytical Solution Techniques for Queueing Network Models of Computer Systems. Working Reports of the Institute for Mathematical Machines and Informatics, Universität Erlangen-Nürnberg, 4, 14(1981)

ALM82 Allen, F.W.; M.E.Loomis; M.V.Mannino: The Integrated Dictionary / Directory System. ACM COMP. SURVEYS 14 (Juni 1982), S. 245-286

ANSI78 ANSI/X3/SPARC/Study Group-DBMS: The ANSI/X3/SPARC Framework. INFORMATION SYSTEMS, 3, 3(1978), S. 173-191

BAGA81 Baer, J.L.; G.Gardarin et al: The two-step Commitment Protocol: Modeling, Specification and Proof Methodology. Proc. 5th Int. Conf. on Software Engineering, San Diego (1981)

BAIA84 Baiardi, F. et al: Structuring Processes for a Cooperative Approach to fault-tolerant Distributed Software. Proc. 4th Int. Symp. on Reliability in Distributed Software and Database Systems, Maryland (September 1984)

BAUE84 L.-Bauerfeld, W.: Zur leistungsberücksichtigenden Spezifikation und Verifikation von Kommunikationsdiensten. Diss. TU Berlin (September 1984)

BAYE84 Bayer, R. et al: Verteilte Datenbanksysteme. Eine Übersicht über den heutigen Entwicklungsstand. INFORMATIK-SPEKTRUM 1, 7(1984), S. 1-19

BDL83 Bender, K.; Drobnik, O.; Lockemann, P.: Datenhaltungssysteme in der Prozeßdatenverarbeitung: Ein Anwendungsprofil. Interner Bericht 16/83, Fakultät für Informatik, TH Karlsruhe (1983)

BEFE86 Best, E.; Fernandez, C.: Notations and Terminology on Petri Net Theory - revised Version. Petri Net Newsletter 23 (April 1986), S. 21-46

BEGI84 Behr, P.M.; Giloi, W.K.: Mechanismen zur Realisierung von Fehlertoleranz im UPPER-System. GMD-SPIEGEL, Sankt Augustin (Juni 1984), S. 21-30

BEGR82 Beilner, H.; Grillo, D.: A Note on COPE - A Software Tool for Modeling and Evaluating the Performance of Computing Systems. Internal Report, Universität Dortmund (1982)

BEIL81 Beilner, H.: Hybride (heterogene) Modellierung. INFORMATIK-SPEKTRUM 1, 4(1981), S. 52-53

BEME83 Berthomieu, B.; Menasche, M.: An Enumerative Approach for Analyzing Timed Petri Nets. Proc. IFIP, Paris (September 1983)

BERN81 Bernstein, P.A. et al: Concurrency Control in Distributed Database Systems. ACM COMP. SURVEYS 2, 13(1981), S.185-223

BERN83a Bernstein, P.A. et al: An ADA-Compatible Distributed Data Manager. Proc. ACM-SIGMOD Conf., San Jose (Mai 1983)

BERN83b Bernstein, P.A. et al: The Failure and Recovery Problem for Replicated DBs. Proc. 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, Montreal (August 1983)

BERN83c Bernstein, P.A. et al: Recovery Algorithms for Database Systems. Proc. IFIP 83, Paris (September 1983)

BFS83 Bratley, P.; Fox, B.L.; Schrage, L.E.: A Guide to Simulation. Springer-Verlag, New York (1983)

BIFL79 Bieber, J.; Florek, S.: A Performance Tool for Design and Installation Support of DDBS. Proc. 1st IEEE Int. Conf. on Distributed Computing Systems, Huntsville (1979)

BORD81 Bordewisch, P.; W.Eickholz: Hybrides Simulationsmodell für die Systemfamilie Nixdorf 8860. Interner Bericht, Nixdorf, Paderborn (1981)

BORO82 Boudenant, J.; P.Roulin: A Reliable Distributed Data Management Algorithm in SIRIUS-DELTA. Proc. 12th Fault-tolerant Computing Symp., Santa Monica (Juni 1982)

BORR81 Borr, A.: Transaction Monitoring in ENCOMPASS. Proc. 7th VLDB, Cannes (Juli 1981)

BRLE82a Breitwieser, H.; M.Leszak: Improving Availability of partially redundant DBs by Majority Consensus Protocols. In /SCHN82/

BRLE82b Breitwieser, H.; M.Leszak: A Distributed Transaction Processing Protocol based on Majority Consensus. Proc. 1st ACM SIGACT/SIGOPS Symp. on Principles of Distributed Computing, Ottawa (August 1982)

BRMA86 Bruno, G.; Marchetto, G.: Process-translatable Petri Nets for Rapid Prototyping of Process-Control Systems. IEEE Trans. on Software Engineering SE-12, 2(1986), S. 346-357

BUEC82 Burkhardt, H.J.; Eckert, H. et al: Beschreibung von Diensten und Protokollen offener Kommunikationssysteme. GMD Informatik-Kolleg, Darmstadt (September 1982)

CHAN83 Chan, A.: On the Development of Distributed Database Management Technology. Proc. European Teleinformatics Conf., Elsevier Sci. Publ., Amsterdam (1983)

CHEN81 Cheng, W.K.: Performance Analysis of Update Synchronisation Algorithms for DDBs. Ph.D. Thesis, Univ. of Illinois (1981)

CHNG84 Chang, J.M.: Simplifying DDBS Design by using a Broadcast Network. Proc. ACM-SIGMOD Conf., Boston (Juni 1984)

CMP82 Ceri, S.; G.Martella; G.Pelagatti: Optimal File Allocation in a Computer Network. COMPUTER NETWORKS 6(1982), S. 345-357

CODD70 Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. COMM. of the ACM 6, 13(1970)

CW83 Computerwoche: Bericht über Leistungsmessung und Tuning. (Juli 1983), S. 19-27

DAGO83 D'Agostini, V.: Specification and Description Language (SDL) as a formal Description Technique of Communication Protocols. Proc. SETSS, Lund/Schweden (1983)

DANT80 Dantas, J.: Performance Analysis of DDBs. Ph.D. Thesis, UCLA (1980)

DAVS79 Davies, D.W. et al: Computer Networks and their Protocols. John Wiley Publ. Co. (1979)

DEBU78 Denning, P.J.; J.P.Buzen: The Operational Analysis of Queuing Network Models. ACM COMP. SURVEYS 3, 10(1978), S.225-261

DECA82 Devor, C.; Carlson, C.R.: Structural Locking Mechanisms and their Effect on DBMS Performance. INFORMATION SYSTEMS 4, 7(1982), S. 345-358

DIAZ82 Diaz, M.: Modeling and Analysis of Communication and Cooperation Protocols using Petrinets. COMPUTER NETWORKS 6(1982), S. 419-441

DILG83 Dilger, E.: Zur Begriffsbildung auf dem Gebiet fehlertolerierender Rechensysteme. Mitteilungen der GI-FG "Fehlertolerierende Rechensysteme", 1 (Mai 1983), S. 17-20

DUBO82 Dubourdieu, D.J.: Implementation of Distributed Transactions. Proc. 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Pacific Grove (Februar 1982)

ECMA85 ECMA: OSI Distributed Interactive Processing Environment. ECMA TC23/TG2, Genf (Januar 1985)

EGLT76 Eswaran, K.R.; J.N.Gray; R.A.Lorie; I.L.Traiger: The Notions of Consistency and Predicate Locks in Database Systems. COMM. of the ACM 11, 19(1976), S.624-633

EICK84 Eick, C.: Methoden und rechnergestützte Werkzeuge für den logischen Datenbankentwurf. Diss. TH Karlsruhe (1984)

ELFL83 Ellis, C.A.; R.A.Floyd: The ROE File System. Proc. 3rd Symp. on Reliability in Distributed Software and DBS, Clearwater Beach (Oktober 1983)

ESPO82 Esculier, C.; R.Popescu-Zeletin: A Description of Distributed

DBMSs. Unveröffentlichtes Manuskript (1982)

FERR78 Ferrari, D.: Computer Systems Performance Evaluation. Prentice-Hall, Englewood Cliffs (1978)

FRAN82 Franta, W.R. et al: Issues and Approaches to Distributed Testbed Instrumentation. IEEE COMPUTER 10, 15(1982), S. 71-81

GAJO78 Garey, M.; Johnson, D.: Computers and Intractability - A guide to the Theory of NP-Completeness. Freeman and Co., San Francisco (1978)

GARC79 Garcia-Molina, H.: Performance of Update Algorithms for Replicated Data. Ph.D. Thesis, Stanford (1979)

GATS84 Galatianos, G.A.; Tsai, W.: Performance Evaluation of Database Update Synchronization on Ethernet Environments. Proc. 4th Int. Conf. on Distributed Computing Systems, San Francisco (1984)

GELA81 Genrich, H.J.; Lautenbach, K.: System Modeling with High-Level Petri Nets. THEOR. COMP. SCIENCE 13(1981), S.109-136

GGK83 Garcia-Molina, H.; Germano, F.; Kohler, W.H.: Architectural Overview of a Distributed Software Testbed. Proc. 16th Ann. Hawaii Int. Conf. on System Sciences (Januar 1983)

GLKE84 Gligor, V.D.; G.L. Kent: Interconnecting Heterogeneous Database Management Systems. IEEE COMPUTER 1, 17(1984), S.33-43

GMSS81 Godbersen, H.P.; Munz, R.; Schiele, F.; Schneider, H.-J.; Steyer, F.: VDN-Abschlußbericht. BMFT, FKZ 081-5011 A (1981)

GODB81 Godbersen, H.P.: Mengengerüst und Verteilungsaspekte in Informationssystemen. Interner CIS-Bericht, TU Berlin (2/1982)

GODB83 Godbersen, H.P.: Funktionsnetze - eine Modellierungskonzeption zur Entwurfs- und Entscheidungsunterstützung. Ladewig-Verlag, Berlin / München (1983)

GRAY78 Gray, J.: Notes on Database Operating Systems. In: Operating Systems - An Advanced Course, Springer Lecture Notes (1978)

GRAY81 Gray, J.: The Transaction Concept: Virtues and Limitations. Proc. 7th VLDB, Cannes (Juli 1981)

GTDS83 Geist, R.; Trivedi, K.; Dugan, J.B.; Smotherman, M.: Design of the Hybrid Automated Reliability Predictor (HARP). Proc. IEEE/AIAA Digital Avionics Systems Conf. (November 1983)

HOAR78 Hoare, C.A.R.: Communicating Sequential Processes. COMM. of the ACM 8, 21(1978), S. 666-677

HUSL81 Huslende, R.: A combined Evaluation of Performance and Reliability for Degradable Systems. Proc. ACM-SIGMETRICS Conf. on Measurement and Modeling of Computing Systems,

Las Vegas (September 1981)

IBM84 IBM: Graphical Data Display Manager - Application Programming Guide (Release 4). Order No. SC33-0148-2 (1984)

INTL84 INTEL: INA960 - The first ISO-Standard Implementation of the Transport Layer (1984)

ISO83 Int. Org. for Standardization: Temporal Ordering Specification Language, Draft Tutorial, ISO/TC97/SC16/WG1 N157 (August 1983)

ISO84a Int. Org. for Standardization: Information Processing Systems-Open Systems Interconnection - Basic Reference Model. International Standard, ISO 7498 (1984)

ISO84b Int. Org. for Standardization: Specification of Protocols for Common Application Service Elements (CASE). Part 1: Commitment, Concurrency and Recovery. DP 8650/3 (1984)

ISO85 Int. Org. for Standardization: ESTELLE - A formal Description Technique based on an Extended State Transition Model. ISO/TC97/SC21, DP 9074 (Juni 1985)

JOBM82 Jobmann, M.: ILMAOS - Eine Sprache zur Formulierung von Rechensystemmodellen. Bericht Nr. 92, Univ. Hamburg (1982)

JOHN77 Johnson: Packet Switching Services and the Data Communication User. Ovum Ltd., London (1977)

JOWE84 Jochum, F.; Weissmann, V.E.: A Formal Framework for Experimental Evaluation of Information Systems. LIVE-Bericht 6/84, TU Berlin (1984)

JOWI84 Jochum, F.; Winter, D.: ISAC - Eine Analyse- und Entwurfs-Methode für komplexe Softwaresysteme. 11. GI-Jahrestagung, München, Springer-Verlag (Oktober 1981)

KFK83 Kernforschungszentrum, Hauptabteilung Datenverarbeitung und Instrumentierung: Verfügbarkeitsstatistik der IBM3081. In: DV-Planungsbrief (unveröffentlichter Bericht), Karlsruhe (Januar 1983)

KIM84 Kim, W.: Highly Available Systems for DB Applications. ACM COMPUTING SURVEYS 1, 16(1984), S. 71-98

KLEI74 Kleijnen, J.P.: Statistical Techniques in Simulation. Marcel Dekker Inc., New York (1974)

KLEI82 Kleijnen, J.P.: Experimentation with Models: Statistical Design and Analysis Techniques. In: F.E. Cellier (Ed.): Progress in Modeling and Simulation, Academic Press, London (1982)

KLCK76 Kleinrock, L.: Queueing Systems, Vol. II: Computer Applications. John Wiley, New York (1976)

KOPE82 Kopetz, H.: The Failure-Fault Model. Proc. 12th Fault-tolerant Computing Symp., Santa Monica (Juni 1982)

KOST82 Kostro, K.: Angewandte EDV-Leistungsvorhersage mit Warteschlangenmodellen. Diss. TU Wien (Februar 1982)

LARO82 Landers, T.; Rosenberg, R.L.: An Overview of MULTIBASE. In /SCHN82/

LEBR82 Leszak, M.; Breitwieser, H.: A fault-tolerant Scheme for Distributed Transaction Commitment. Proc. 3rd IEEE Int. Conf. on Distributed Computing Systems, Miami (Oktober 1982)

LEFA85 Lee, K.; Favrel, J.: Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets. IEEE Trans. on SYSTEMS, MAN, and CYBERNETICS 2, 15(April 1985), S. 272-280

LEGO85 Leszak, M.; Godbersen, H.P.: DEAMON: A Tool for Performance-Availability Evaluation of Distributed Systems based on Function Nets. In /TPN85/

LESZ83 Leszak, M.: Erfahrungen bei der Portierung des Software-Werkzeuges "FUN" vom VM- auf das MVS-Betriebssystem. Unveröffentlichter Bericht (November 1983)

LIND83 Lindsay, B.G. et al: Computation and Communication in R*: A Distributed Database Manager. IBM Research Report RJ 3740, San Jose (1983)

LISI84 Li, V.O.K; Silvester, J.A.: Performance Analysis of Networks with unreliable Components.
IEEE Trans. on COMMUNICATIONS 10, 32(1984), S. 1105-10

LITW82 Litwin, W. et al: SIRIUS Systems for Distributed Data Management. In /SCHN82/

MCB84 Marsan, M.A.; Conte, G.; Balbo, G.: A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems.
ACM Trans. on COMPUTER SYSTEMS 2, 2(1984), S. 93-122

MEFA76 Merlin, P.M.; Farber, D.J.: Recoverability of Communication Protocols.
IEEE Trans. on COMMUNICATIONS 9, COM-24(1976), S. 1036-43

MEIS82 Meisner, N.M.: Methodology for Assessing the Robustness of a Local Network-based Computer System.
Proc. Symp. on Local Computer Networks, Florenz (1982)

MEYE80 Meyer, J.F.: On Evaluating the Performability of Degradable Computer Systems.
IEEE Trans. on COMPUTERS 8, C-29(1980), S. 720-31

MEYE85 Meyer, J.F. et al: Stochastic Activity Networks: Structure, Behavior, and Applications. In /TPN85/

MILN80 Milner, R.: A Calculus of Communicating Systems. LNCS 92, Springer-Verlag, Berlin / Heidelberg (1980)

MILN82 Milner, R.: Four Combinators for Concurrency. Proc. 1st ACM SIGACT/SIGOPS Symp. on Principles of Distributed Computing, Ottawa (August 1982)

MOLI83 Mohan, C.; Lindsay, B.: Efficient Commit Protocols for the Tree Processes Model of Distributed Transactions. Proc. 2nd ACM SIGACT/SIGOPS Symp. on Principles of Distributed Computing, Montreal (August 1983)

MOPO82 Mohan, C.; R. Popescu-Zelletin: Impact of DDB Management on the ISO-OSI and ANSI/X3/SPARC Frameworks. Proc. Hawaii Int. Conf. on Computer Systems (1982)

MSS83 Meyer, B.E.; Schneider, H.-J.; Stübel, G.: Computergestützte Unternehmensplanung. de Gruyter (1983)

MUEL82 Müller-Ettrich, G.: Leistungsmessung und Leistungsverbesserung von Datenbanksystemen - Datenbank Tuning. Verlagsges. R. Müller, Köln (1982)

MUNZ79 Munz, R.: Gross Architecture of the DDB System VDN. Proc. IFIP TC-2 Working Conf. on Database Architecture, North-Holland Publ. (1979)

MUNZ80 Munz, R.: Transaction Management in the DDB System VDN. Proc. IFIP, Melbourne (Oktober 1980)

NCC83 Norwegian Computing Center: SIMULA Programmer's Guide for IBM System 360/370. Oslo (Januar 1983)

NG83 Ng, F.K.: On Concurrency and Reliability of a Real-Time DBMS for 5ESS Switching System. Proc. 3rd Symp. on Reliability in Distributed Software and DB Systems, Clearwater (Okt. 1983)

NIGA81 Nigam, A.: A Specification and Proof Technique for Message-based Systems and its Application to DDB Algorithms. Ph.D. Thesis, Univ. of Rochester (1981)

NOYE84 Nounou, N.; Yemini, Y.: Algebraic Specification-Based Performance Analysis of Communication Protocols. Proc. 4th Int. Workshop on Protocol Specification, Testing, and Verification, Sky Top (Juni 1984)

NTG82 NTG-Empfehlung 3004: Zuverlässigkeitsbegriffe im Hinblick auf komplexe Hardware und Software. NACHRICHTENTECHNISCHE ZEITSCHRIFT 5, 35(1982), S. 327-333

NUTT72 Nutt, G.J.: Evaluation Nets for Computer Systems Performance Evaluation. Proc. AFIPS Fall Joint Comp. Conf., Montvale (1972)

OZSU83 Ozsu, T.: Modeling and Analysis of DDB Concurrency Control Mechanisms. Ph.D. Thesis, Ohio State Univ. (März 1983)

PAGE83 Page, B.: Der Gültigkeitsnachweis von komplexen Simulationsmodellen. ANGEWANDTE INFORMATIK 4(1983), S. 149-157

PESI83 Peterson, J.L.; Silberschatz, A.: Operating System Concepts. Addison Wesley, New York (1983)

PETE81 Peterson, J.L.: Petri Nets Theory and the Modeling of Systems. Prentice Hall, Englewood Cliffs (1981)

RAZO84 Razouk, R.R.: The Derivation of Performance Expressions for Communication Protocols from Timed Petri Net Models. Proc. ACM-SIGCOMM Symp. on Communications Architecture and Protocols, Montreal (Juni 1984)

RIES79 Ries, D.: The Effects of Concurrency Control on DBMS Performance. Ph.D. Thesis, Univ. of California at Berkeley (April 1979)

ROBI82 Robinson, J.: Separating Policy from Correctness in Concurrency Control Design. IBM-Research Report RC9308 (März 1982)

ROTH80 Rothnie, J.B. et al: Introduction to a System for DDBs (SDD-1). ACM TODS 1, 5(1980), S. 1-17

SCHE81 Scheuernstuhl, G.: Methodologische Untersuchungen für ein System zur halb-automatischen Reorganisation großer Datenbanken. Diss. TU Berlin (Juli 1981)

SCHI84 Schiffner, G.: A Specification and Performance Evaluation Model for Multicomputer Database Machines. Diss. Universität Bremen (1984)

SCHN81 Schneider, H.-J.: Techniques and formal Tools for Design, Realization, and Evaluation of Evolutionary Information Systems. Proc. IFIP Working Conf. on Evolutionary Information Systems, Budapest (September 1981)

SCHN82 Schneider, H.-J. (ed.): Proc. 2nd Int. Conf. on Distributed Databases, Berlin (September 1982)

SCHN83 Schneider, H.-J.: Lexikon der Informatik und Datenverarbeitung. Oldenbourg Verlag, München/Wien (1983)

SCHN84a Schneider, H.-J.: Software der fünften Generation - die Lösung der Softwarekrise? Wiss.-Magazin der TU Berlin 6(1984), S. 80-84

SCHN84b Schneider, H.-J.: Reliability in Distributed Hardware, Communication, Database Management, and Application Software Systems Development and Maintenance. Unveröffentlichtes Manuskript (September 1984)

SCHN84c Schneider, H.-J.: Von Personal bis Mainframe - dedizierte Systeme in einem Verbund. Fachbericht IAO-Tagung: Integrierte Bürosysteme, Stuttgart (1984)

SCHW77 Schweppe, H.F.: On different Classes of Predicates for Distributed Data.
Proc. GI-Fachtagung Datenbanken in Rechnernetzen mit Kleinrechnern, Karlsruhe (April 1977)

SCHW85 Schwärmer, B.: Einsatz von rechnergestützten Werkzeugen zur modelltechnischen Systemanalyse.
ANGEWANDTE INFORMATIK 6 (1985), S. 225-231

SCNK84 Scheschonk, G.: Ein auf Petrinetzen basierende Konstruktions-, Analyse und (Teil-)Verifikationsmethode zur Modellierungsunterstützung bei der Entwicklung von Informationssystemen.
Diss. TU Berlin (Juni 1984)

SCWS80 Schneeweiß, W.G.: Zuverlässigkeits-Systemtheorie.
Datakontext-Verlag, Köln 1980

SEVC83 Sevcik, K.C.: Comparison of Concurrency Control Methods using Analytical Models. Proc. IFIP, Paris (September 1983)

SHSA83 Shantikumar, J.G.; Sargent, R.G.: A unifying view of Hybrid Simulation/Analytic Models and Modeling.
OPERATIONS RESEARCH 4 31(1983), S. 1030-52

SINH85 Sinha, M.K. et al: Timestamp based Certification Schemes for Transactions in DDBS. Proc. ACM-SIGMOD Conf., Austin (Mai 1985)

SISW82 Siewiorek, D.P.; Swarz, R.S.: The Theory and Practice of Reliable System Design. Digital Press (1982)

SKEE81 Skeen, D.: Crash Recovery in a Distributed Database System.
Ph.D. Diss., Univ. of California at Berkeley (1981)

SMK81 Sauer, C.H.; MacNair, E.A.; Kurose, J.F: Computer / Communication System Modeling with the Research Queueing Package Version 2.
IBM Research Report RA128, Yorktown Heights (November 1981)

SPAN82 Spaniol, O.: Konzepte und Bewertungsmethoden für lokale Rechnernetze. INFORMATIK-SPEKTRUM 5(Sept. 1982), S.152-170

STON79 Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES. IEEE Trans. on SOFTWARE ENGINEERING 5, 3(Mai 1979), S. 185-194

STUE75 Stübel, G.: Methodologische und Softwareengineering-orientierte Untersuchungen für ein Unternehmensmodell verschiedener Strukturiertheitsgrade. Diss., Stuttgart (1975)

TEFR82 Teorey, T.J.; J.P.Fry: Design of Database Structures.
Prentice-Hall, Englewood Cliffs (1982)

TOMA83 Tokuda, H.; E.G.Manning: An IPC Model for a Distributed Software Testbed. Proc. ACM-SIGCOMM Symp. on Communication Architectures and Protocols, Austin (März 1983)

TPN85 Proc. ACM-SIGMETRICS/IEEE Int. Workshop on Timed Petri Nets, Turin (Juli 1985)

TRYU83 Trivedi, K.S.; P. Yu: Reliability and Performance Analysis of a Ringnet. IBM Research Report RC9792, Yorktown Heights (September 1983)

VITT84 Vitter, J.S.: Faster Methods for Random Sampling. COMM. of the ACM 7, 27(1984), S. 703-718

VOSS84 Voss, K.: A Net Model of a Local Area Network. In: LNCS 188, Advances in Petri Nets 1984, Springer-Verlag (1985)

VSE83 Vernon, M.; de Souza e Silva, E.; Estrin, G.: Performance Evaluation of Asynchronous Concurrent Systems: The UCLA Graph Model of Behavior. Proc. Performance 83, Maryland (Mai 1983)

WALT82 Walter, B.: Transaktionsorientierte Recovery-Konzepte für verteilte Datenbanksysteme. Diss., Stuttgart (1982)

WANE84 Walter, B.; Neuhold, E.J.: POREL: A Distributed Database System. In: C.Mohan (ed.): Recent Advances in Distributed Database Management, IEEE Press (1984)

WILA84 Wilkinson, W.K.; Lai, M.: Managing replicated Data in JASMIN. Proc. 4th Int. Symp. on Reliability in Distributed Software and Database Systems, Maryland (September 1984)

WILD79 Wild, J.K.: Einbettung eines DB-Maschinen-Konzeptes in einen heterogenen Informationsverbund. Diss. TU Berlin (1979)

WIPR78 Wilson, J. R.; Pritsker, A.B.: A Survey of Research on the Simulation Startup Problem. SIMULATION, 1, 31 (1978), S. 55-58

WLCH83 Welch, P.D.: The Statistical Analysis of Simulation Results. In: S.S. Lavenberg (ed.): Computer Performance Modeling Handbook, Academic Press (1983)

WLS83 Wilms, P.F.; B.G.Lindsay; P.Selinger: Distributed Execution Protocols for Data Definition in R*. Proc. ACM-SIGMOD Conf., San Jose (Mai 1983)

WONG83 Wong, E.: Dynamic Rematerialization: Processing Distributed Queries using Redundant Data. IEEE Trans. on SOFTWARE ENGINEERING 3, 9(1983), S. 228-232

WST82 Weber, H.; G.Schiffner; H.Trümner: Datenbanken im Vielteilnehmer-Breitband-Dialogsystem. Abschlußbericht, Uni Bremen (Juli 1982)

ZEIG84 Zeigler, B.P.: Theory and Application of Modeling and Simulation - A Software Engineering Perspective. In: Handbook on Software Eng., Van Nostrand Reinhold, New York (1984)

ZUBE85 Zuberek, W.M.: Enhanced M-Timed Petri Nets, Modeling and Performance Evaluation. Technical Report #8514, Dptm. of Comp. Science, Memorial Univ. of Newfoundland (Juli 1985)