KfK 3086
November 1980

Modelle zur rechnergestützten Simulation von Kommunikationsflüssen in Rechnernetzen

B. Wolfinger
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 3086

Modelle zur rechnergestützten Simulation von Kommunikationsflüssen in Rechnernetzen

Bernd Wolfinger

von der Fakultät für Informatik der Universität Karlsruhe genehmigte Dissertation

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Kurzfassung

Die Realisierung verteilter DV-Systeme verlangt nach leistungsfähigen Hilfsmitteln zur Unterstützung des Entwurfs und der Analyse von Rechnernetzsoftware bei gleichzeitiger Berücksichtigung von Leistungskenngrößen der verwendeten Hardware-Komponenten. Die Untersuchung der Leistungsfähigkeit und Korrektheit der Kommunikationssoftware ist dabei von besonderer Bedeutung.

Die vorliegende Arbeit geht das Problem der Effizienzanalyse und -prognose für Rechnernetze durch Entwicklung generalisierter Modelle an, die eine detaillierte Nachbildung hierarchisch organisierter Kommunikationssoftware unter gleichzeitiger Miteinbeziehung der Betriebsmittelzuteilung in den Rechnernetzknoten erlauben. Ausgehend von einer Analyse der Struktur von Rechnernetzen werden zunächst diejenigen Teilkomponenten eines Rechnernetzes herausgearbeitet, die für die Modellbildung von Relevanz sind und der anschließenden Einführung der konzeptionellen Modelle als Grundlage dienen. Bei der Bereitstellung der konzeptionellen Modelle wird weitgehend versucht, erweiterbare Grundmodelle zu formulieren, die sich in einfacher Weise an unterschiedliche Randbedingungen anpassen lassen. Zentrale Modelle sind zum einen das Modell der "Protokolleinheit", das es erlaubt, Kommunikationsprotokolle in unterschiedlichen Schichten einer hierarchisch aufgebauten Kommunikationssoftware unter Beibehaltung einer identischen Grundstruktur zu beschreiben, zum anderen das Modell der "Betriebsmittelverwaltungsinstanz", das der Modellierung der Betriebsmittelvergabe in einem Rechnernetzknoten dient.

Ein Simulator (MOSAIC) wird eingeführt, der es ermöglicht, die vorgestellten konzeptionellen Modelle experimentell auszuwerten. Zur Strukturierung dieses Simulators wurde ein Konzept erarbeitet, das eine dynamische Konfigurierung des Simulationsprogramms aus vordefinierten Modellbausteinen unterstützt und ein breites Anwendungsspektrum abdeckt. Ein auf der Basis interaktiver Arbeitsmethoden realisiertes Konzept für die Durchführung von Simulationsexperimenten wird vorgestellt, das dem Experimentator zahlreiche Interaktionsmöglichkeiten während des Experimentablaufs gestattet und darüber hinaus durch den Einsatz graphischer Hilfsmittel einen hohen Grad an Benutzernähe besitzt.

Die Modelle konnten in ersten Anwendungen bereits erfolgreich eingesetzt werden; einige experimentelle Ergebnisse werden wiedergegeben.

Models for computer-aided simulation of communication flows in computer networks

Abstract

The realization of distributed DP systems requires efficient tools to support the design and analysis of computer network software, taking into account the performance characteristics of the hardware components used. Investigation of the efficiency and correctness of the communication software is of special importance.

This work considers the problem of performance evaluation and prediction for computer networks and introduces a set of generalized models which allow detailed description of hierarchically organized communication software and of resource administration within the computer network nodes. Starting from an analysis of computer network structures those model-relevant components of a computer network are first elaborated which form the basis for the introduction of the conceptional models. The conceptional models are mainly formulated as a set of generalized models which can be extended and adapted to different boundary conditions. The fundamental models are the model of a "protocol unit" which allows to describe communication protocols within different (protocol) layers of a hierarchically organized communication software using a common basic model structure, and the model of a "local resource administrator" describing resource management within a computer network node.

A simulator (MOSAIC) is introduced, which allows to evaluate experimentally the conceptional models presented. To structure this simulator a concept has been elaborated, which configurates the simulation program dynamically using a set of predefined submodels and being adaptable to cover a large set of different applications. A concept for the interactive execution of simulation experiments is developed, which supports numerous possibilities of interactions during the simulation run and which - by means of graphical features - offers a comfortable interface to the experimenter.

The models have been applied successfully to investigate existing and planned computer networks; some of the experimental results are summarized.

INHALTSVERZEICHNIS

Verwendete Abkürzungen

ABI:	Auftragsbearbeitungsinstanz (s. Modell für Übertragungskanal)
AR:	Arbeitsrechner
ATQ:	Warteschlange für Schnittstellenaufträge (s. Modell für Protokolleinheit)
BD:	Benutzerdaten (s. Modell für Dateneinheit)
BLOCKQ:	Warteschlange für Dateneinheiten (s. Modell für Empfangsprotokollmodul)
BVI:	Betriebsmittelverwaltungsinstanz
BT:	Benutzertask
COPYQ:	Warteschlange für Dateneinheiten (s. Modell für Sendeprotokollmodul)
D:	Rechnernetzdienst
DE:	Dateneinheit
DEQ:	Warteschlange für Dateneinheiten (s. Modell für Übertragungskanal)
DVS=DV-System:	Datenverarbeitungssystem
E/A:	Ein-/Ausgabe
EAS:	Ein-/Ausgabestation
EATQ:	Warteschlange für Schnittstellenaufträge (s. Modell für Empfangsprotokollmodul)
EPM:	Empfangsprotokollmodul
F(D):	Menge der in einem Rechnernetzdienst D beinhalteten Funktionen
f:	Element der Menge F(D)
i:	Index oder Zählvariable
ISO:	Internationale Standardisierungsorganisation
j:	Index oder Zählvariable
k:	Index oder Zählvariable
KI:	Kontrollinformation (s. Modell für Dateneinheit)
KORR(PE):	Menge der Korrespondenten einer Protokolleinheit PE
KP:	Kommunikationsprimitive
KSS:	Kommunikationssubsystem
M:	Zustandsübergangsfunktion (s. sequentielle Automaten)
m(i):	Anzahl der Rechnernetzdienste in der Protokollschicht S_i
N:	Ausgabefunktion (s. sequentielle Automaten)
n:	Anzahl der Protokollschichten in der Hierarchie der Kommunikationsprotokolle

NB(PE):	Menge der Nachbarn einer Protokolleinheit PE
NBHS(PE):	Teilmenge der Nachbarn einer Protokolleinheit PE
NBTS(PE):	Teilmenge der Nachbarn einer Protokolleinheit PE
PE:	Protokolleinheit
Q_{CPU}: Q_{PEND}: Q_{WCPU}: Q_{WHSP}:	Warteschlangen für Tasks (s. Modell für Betriebsmittelverwaltungsinstanz)
R:	Rechnernetz
RECQ:	Warteschlange für Dateneinheiten (s. Modell für Empfangsprotokollmodul)
REQQ:	Warteschlange für Schnittstellenaufträge (s. Modell für Übertragungskanal)
$S_1,\ldots,S_n$:	Protokollschichten in der Hierarchie der Kommunikationsprotokolle
SATQ:	Warteschlange für Schnittstellenaufträge (s. Modell für Sendeprotokollmodul)
SENDQ:	Warteschlange für Dateneinheiten (s. Modell für Sendeprotokollmodul)
SPM:	Sendeprotokollmodul
SSA:	Schnittstellenauftrag
t bzw. t_i:	Simulationszeit bzw. -zeitpunkt
TAB_E: $TAB_{E/S}$: TAB_S:	Tabellen mit Zustandsinformation (s. Modell für Protokolleinheit)
TIMEOUTQ:	Warteschlange für Timeouts (s. Modell für Protokolleinheit)
TKI:	Timeout-Kontrollinstanz (s. Modell für Protokolleinheit)
VR:	Vermittlungsrechner
X:	Menge der Eingabesymbole (s. sequentielle Automaten)
Y:	Menge der Ausgabesymbole (s. sequentielle Automaten)
Z:	Zustände bei der Formulierung von Modellen als sequentielle Automaten
ZD:	dynamische Taskzustände (s. Modell für Tasks)
ZI:	interne Taskzustände (s. Modell für Tasks)

1. EINLEITUNG UND PROBLEMSTELLUNG

Bei Projekten zur Konzipierung und Realisierung neuartiger DV-Systeme ist - im Gegensatz zu den erheblich sinkenden Hardwarekosten - ein signifikantes Wachstum der Kosten der Entwurfsphase zu beobachten (s. Myers /MYER 78/). Dieser Trend tritt bei verteilten DV-Systemen (Rechnernetze, Mehrrechnersysteme etc.), bedingt durch deren hohen Komplexitätsgrad, besonders deutlich zu Tage.

Die vorliegende Arbeit beschäftigt sich mit dieser Problematik, indem sie Hilfsmittel zur Leistungsprognose und Leistungsanalyse von Rechnernetzen bereitstellt und auf diese Weise die Projektierung und Optimierung verteilter DV-Systeme unterstützt.

Zum Zwecke der Untersuchung des Leistungsverhaltens von Rechnern /SCHR 77/ bzw. Rechnernetzen werden vorwiegend die nachfolgenden alternativen Vorgehensweisen herangezogen

(a) Messungen direkt am existierenden DV-System:

- Einsatz von Monitoren

  Bei dieser Methode werden Meßeinrichtungen in das zu untersuchende System zur Erfassung relevanter Leistungskenngrößen integriert; abhängig von der Art der Realisierung der Meßeinrichtungen werden Hardware- und Softwaremonitore unterschieden (s. /KLAR 71/).

- Vorgabe einer dedizierten Rechnerlast durch Benchmark - oder durch synthetische Programme

  Beide Techniken bestehen im wesentlichen darin, die Reaktionen eines zu untersuchenden Systems unter Einwirkung eines wohldefinierten, realitätsgetreuen Auftragsprofils zu studieren (s. /HIGH 77/).

(b) indirekte Untersuchung des DV-Systems durch Modellierung /INFO 77/:

Unter Modellierung versteht man den Vorgang, die wesentlichen Eigenschaften eines zu untersuchenden Systems A in einem vereinfachten Modellsystem A' nachzubilden. Durch Auswertung des Modellsystems A' können Rückschlüsse auf das System A gezogen werden, sofern zuvor eine hinreichende Übereinstimmung zwischen A und A' nachgewiesen werden konnte. Als Techniken zur Modellauswertung finden z.B. analytische oder - insbesondere bei kom-

plexeren Modellsystemen - Simulationsmethoden (s. /SHAN 75/, /ZEIG 76/) Anwendung.

Die Verfahrensweise bei der (rechnergestützten) Simulation ist dergestalt, daß das Modellsystem auf ein korrespondierendes (Rechner-) Programm, den Simulator, abgebildet wird. In Simulationsexperimenten wird das Verhalten des Simulators unter Einwirken einer vorgegebenen (Modell-) Belastung studiert. Hierzu wird der Simulator durch Ausführung des Programms an einer DV-Anlage ausgewertet.

Ein Zurückgreifen auf Methoden der direkten Systemvermessung ist für unsere Zwecke nicht sinnvoll, da eine wesentliche Forderung an das in dieser Arbeit bereitzustellende Instrument die Untersuchung von Entwurfsalternativen darstellt, die nicht notwendigerweise in einem existierenden DV-System bereits realisiert sind. Dementsprechend wollen wir uns in der Folge auf den Einsatz der Modellierung beschränken und die direkte Systemvermessung ausschließlich als ein Hilfsmittel zur Validation (s. /BEIL 73/) der Modellierung heranziehen.

Für die zu fordernde Leistungsfähigkeit eines Rechnernetzes ist der Umfang und die Qualität der angebotenen Rechnernetzdienste (Timesharing-Betrieb, File Transfer, Intertaskkommunikation etc.) ausschlaggebend. Die Abwicklung von Rechnernetzdiensten erfolgt unter Zuhilfenahme von Kommunikationsprotokollen. Unter einem (Kommunikations-) Protokoll versteht man dabei Vereinbarungen, die den Informationsaustausch zwischen zwei oder mehreren Kommunikationspartnern regeln. Die gesamte Kommunikationssoftware innerhalb eines Rechnernetzes ist zumeist hierarchisch, d.h. in mehreren übereinanderliegenden (Protokoll-) Schichten, organisiert, die sich gegenseitig beeinflussen können.

Aus der starken Abhängigkeitsbeziehung zwischen einem Rechnernetzdienst und dem zugrundeliegenden Kommunikationsprotokoll wird bereits ersichtlich, daß die Güte eines Rechnernetzdienstes maßgeblich beeinflußt wird durch die Effizienz des verwendeten Protokolls. Allerdings genügt zur globalen Optimierung eines Rechnernetzes eine unabhängige Optimierung der einzelnen Protokollschichten oft nicht, da die aus einer derartigen Betrachtungsweise resultierenden Optimierungsentscheidungen einander entgegenwirken können (s. /LEGO 78/).

Diese Tatsache motiviert die zentrale Anforderung an die hier zu entwerfenden Modelle zur Leistungsuntersuchung von Rechnernetzen: eine detaillierte Modellierung der Kommunikationsflüsse innerhalb der gesamten Protokollhierarchie soll unterstützt werden.

Was die Auswertungsmethode für die resultierenden Rechnernetzmodelle betrifft, so scheitert eine analytische Auswertung an dem angestrebten hohen Detaillierungsgrad und der sich daraus ergebenden Komplexität der Modelle. Einen Ausweg bietet die rechnergestützte Simulation, die daher auch als ausschließliche Methode zur Auswertung der in dieser Arbeit präsentierten Rechnernetzmodelle benutzt wird.

Dem Einsatz der Simulation bei der Modellierung von Kommunikationsflüssen in Rechnernetzen stehen primär die beiden nachfolgenden Restriktionen entgegen:

(R1) der hohe Aufwand, der üblicherweise mit der Formulierung eines adäquaten Rechnernetzmodells verbunden ist;

(R2) der beträchtliche Zeitbedarf, der für die Durchführung von Simulationsexperimenten erforderlich ist (hierunter fällt zum einen die CPU-Zeit, die für Experimentläufe, d.h. für die Modellauswertung durch eine DV-Anlage, benötigt wird und zum anderen die Zeit, die der Experimentator für die Interpretation der Simulationsresultate aufzuwenden hat).

Neben den aktuell stark sinkenden Kosten für Rechenleistung, welche den ersten Teil der Restriktion (R2) ohnehin abmildern, können die genannten Nachteile des weiteren durch Berücksichtigung folgender Randbedingungen entschärft werden:

(B1) Bereitstellung von Modellen zur Beschreibung von Rechnernetzen, die sich durch ein breites Anwendungsspektrum auszeichnen; diese Modelle sollen insbesondere eine flexible Anpassung an diejenigen Rechnernetzcharakteristika erlauben, die das interne Ablaufgeschehen maßgeblich beeinflussen (wie z.B. die zugrundeliegende Hardware- und Softwarestruktur oder das Auftragsprofil);

(B2) eine hohe Effizienz und Benutzerfreundlichkeit im Hinblick auf die Auswertung der Modelle.

Es existieren bereits zahlreiche Arbeiten, die sich mit der Simulation von Rechnernetzen unter Berücksichtigung von Kommunikationsprotokollen beschäftigen (einen Überblick gibt /SCHO 78/). So wurden u.a. für eine Vielzahl von Rechnernetzen - zumeist begleitend zur Planungs- und Implementierungsphase - in erster Linie als Hilfsmittel zur Unterstützung des Entwurfs und der Optimierung, Simulatoren erstellt. Beispiele von Rechnernetzen, deren Realisierung durch Simulationsstudien unterstützt wurde: CYCLADES-Netz /IRLA 76/,

EIN /HANS 77/, GMD-Netz /HAEN 76/, HMINET /WOLF 77/, NPL-Netz /PRIC 79/. Andere Simulatoren waren der gezielten Untersuchung eines speziellen Kommunikationsprotokolls gewidmet. Dennoch existiert ein erheblicher Mangel an Arbeiten, die wie z.B. Schneider /SCHN 78/, eine Anwendbarkeit der erarbeiteten Modelle für eine umfangreiche Klasse von Rechnernetzen anstreben.

Der Hauptnachteil der bekannten existierenden Simulatoren zur Modellierung von Rechnernetzen ist vornehmlich darin zu sehen, daß sie zumindest eine der nachfolgenden Einschränkungen besitzen und aus diesem Grunde nicht der obigen Randbedingung (B1) genügen:

(1) der Detaillierungsgrad des Simulators ist zu gering, um das Ablaufgeschehen in einem Rechnernetz realitätsnah zu beschreiben (z.B. können Protokolleigenschaften oder die Betriebsmittelzuteilung in den einzelnen Rechnern oder Auftragsprofile nur unzureichend im Modell nachgebildet werden);

(2) der Simulator ist zugeschnitten auf die Betrachtung von Teilaspekten bei der Rechnernetzanalyse (z.B. Optimierung von Kommunikationsprotokollen) und eine integrierte Untersuchung des Ablaufgeschehens in einem Rechnernetz ist daher nicht möglich;

(3) der Simulator erlaubt zwar die detaillierte Modellierung eines speziellen Rechnernetzes, jedoch eine flexible Modellanpassung an geänderte Rechnernetzcharakteristika - insbesondere die Nachbildung verschiedenartig strukturierter Protokollhierarchien zum Vergleich alternativer Realisierungen der Kommunikationssoftware - wird nicht unterstützt.

Darüber hinaus verfügen die meisten dieser Simulatoren nicht über geeignete Hilfsmittel zur Reduktion des Zeitbedarfs bei der Experimentdurchführung, wie sie durch die Randbedingung (B2) gefordert werden.

Die genannten Mängel existierender Modelle zur Simulation von Kommunikationsflüssen in Rechnernetzen, stellten für den Autor dieser Arbeit die Motivation dar - ausgehend von seinen bei der Modellierung der Nachrichtenflüsse des HMINET gewonnenen Erfahrungen - ein neuartiges Modellierungsinstrumentarium zur Leistungsprognose und -analyse von Rechnernetzen zu entwickeln, das die geforderten Randbedingungen einer allgemeinen Anwendbarkeit (B1) und einer hohen Effizienz und Benutzerfreundlichkeit (B2) weitgehend berücksichtigt.

Als Resultate dieser Bemühungen liefert die vorliegende Arbeit

- konzeptionelle Modelle zur Beschreibung sämtlicher relevanter Rechnernetzkomponenten, die das rechnernetzinterne Ablaufgeschehen primär beeinflussen;
- eine Implementierung dieser Modelle in einer höheren Programmiersprache und die Bereitstellung eines Programmsystems (Modellierungssystem MOSAIC) zur Modellauswertung durch rechnergestützte Simulation;
- Effizienzuntersuchungen für die Realisierung einer Hierarchie standardisierter Kommunikationsprotokolle in einem existierenden Rechnernetz als konkrete Anwendung der vorgestellten Modelle und des Modellierungssystems MOSAIC.

Die geforderten Randbedingungen (B1) und (B2) schlagen sich in erster Linie darin nieder, daß

- bei der Erstellung der konzeptionellen Modelle darauf geachtet wird, daß die Struktur der Modelle eine einfache und dennoch präzise Anpassung an verschiedenartige Rechnernetzcharakteristika erleichtert;
- zur Modellimplementierung eine in hohem Maße portable Programmiersprache herangezogen wird;
- das Modellierungssystem MOSAIC über eine erweiterbare Grundmenge vordefinierter Modellbausteine (in Form von Programm-Moduln) verfügt, die auf flexible Weise zu unterschiedlichen Rechnernetzmodellen zusammengefügt werden können;
- eine interaktive Durchführung der Simulationsexperimente zur Modellauswertung ermöglicht wird;
- graphische Hilfsmittel zum Aufbau des Rechnernetzmodells, zur Darstellung der Kommunikationsflüsse im (modellierten) Rechnernetz und zur Ausgabe der gemessenen Ergebnisvariablen bereitgestellt werden.

Was den Aufbau der Arbeit betrifft, so sollen zunächst die allgemeinen Strukturierungsmerkmale von Rechnernetzen herausgearbeitet und die modellrelevanten Rechnernetzkomponenten definiert werden (s. Kapitel 2), um den Entwurf adäquater Modelle zur Beschreibung von Kommunikationsflüssen in Rechnernetzen vorzubereiten.

Basierend auf den durch die Warteschlangen- und Automatentheorie zur Verfügung gestellten Beschreibungsmethoden werden in Kapitel 3 Teilmodelle für

diese Rechnernetzkomponenten formuliert.

In Kapitel 4 wird die Umsetzung der Teilmodelle in die Programmiersprache SIMULA sowie ihre Einbettung in das Modellierungssystem MOSAIC aufgezeigt, das der Modellauswertung mit Hilfe von Methoden der rechnergestützten Simulation dient.

Die Forderung nach einer effizienten und benutzernahen Modell- und Experimentdatenauswertung findet in Kapitel 5 ihre Berücksichtigung, zum einen durch die Erarbeitung eines Konzepts zur interaktiven Modellauswertung und zum anderen durch die Bereitstellung graphischer Komponenten zur Unterstützung der Interpretierbarkeit des Modells und der Experimentresultate durch den Experimentator.

Die Experimentserien des Kapitels 6 dienen neben der Leistungsanalyse und Optimierung der modellierten Hierarchie standardisierter Kommunikationsprotokolle dazu, Erfahrungen aufzuzeigen, die beim Einsatz des Modellierungssystems MOSAIC zur Untersuchung eines existierenden Rechnernetzes gewonnen werden konnten.

2. ANALYSE DER STRUKTUR VON RECHNERNETZEN

Unter einem Rechnernetz verstehen wir in dieser Arbeit eine Menge untereinander verbundener, autonomer DV-Systeme, die auf eine automatisierte Weise miteinander kommunizieren und Betriebsmittel (wie z.B. Programme, Daten, periphere Geräte) teilen (vgl. /DOLL 74/). Für autonome DV-Systeme setzen wir voraus, daß in ihnen jede Komponente einer "von Neumann"-Maschine (Hauptspeicher, CPU und Kanal) vorhanden ist, und daß sie aufgrund der vorhandenen Betriebssoftware in der Lage sind, ein vorgegebenes Spektrum an Aufträgen selbständig zu bearbeiten. Hieraus geht u.a. hervor, daß ein Multiprozessorsystem zwar als Teilmenge eines Rechnernetzes fungieren kann, jedoch nicht bereits selbst ein Rechnernetz darstellt.

Rechnernetze werden mit den unterschiedlichsten Zielsetzungen realisiert. So lassen sich /nach SCHU 77/ zur Klassifizierung existierender Rechnernetze vier grundsätzliche Kategorien herausarbeiten:

(a) der Lastverbund, der den Lastausgleich zwischen den im Rechnernetz beteiligten DV-Systemen als Hauptziel hat (Realisierungsbeispiel TYMNET /TYME 71/);

(b) der Funktionsverbund, der einem Rechnernetzbenutzer die durch einzelne DV-Systeme des Netzes angebotenen Funktionen (z.B. Timesharing-Betrieb) verfügbar macht (Realisierungsbeispiel ARPANET /HEAR 75/);

(c) der Datenverbund als Realisierung einer geographisch auf mehrere DV-Systeme verteilten Datenbank (bisherige Implementierungen meist nur experimentell);

(d) der Nachrichtenverbund zur Übermittlung von Nachrichten zwischen verschiedenen Rechnernetzbenutzern (Realisierungsbeispiel SITA-Netz /CHRE 75/).

Charakteristisch für sämtliche der genannten Rechnernetztypen ist, daß sie ihre verschiedenartigen Ziele - die Rechnernetzdienste - mit ein und demselben Hilfsmittel (Datenaustausch zwischen den am Rechnernetz beteiligten DV-Systemen) erreichen. Bei existierenden Rechnernetzen handelt es sich zumeist um Mischformen der angeführten Typen. Als Rechnernetzbenutzer, der die angebotenen (Rechnernetz-) Dienste beansprucht, fungiert der Mensch bzw. ein technischer Prozeß.

Im folgenden soll eine Analyse von Rechnernetzen durchgeführt werden, welche als Basis für die Modelle dient, die in Kapitel 3 zu definieren sein werden. Der Wunsch nach einer breiten Anwendbarkeit jener Modelle schlägt sich in der nachfolgenden Voraussetzung für die Analyse nieder:

Eine vereinheitlichende Sicht - sowohl den physikalischen Aufbau von Rechnernetzen als auch die Struktur von Rechnernetzsoftware betreffend - ist anzustreben, die unter Umständen für einige spezielle Rechnernetzrealisierungen nicht zutreffend ist, die jedoch mit einer Vielzahl existierender Rechnernetze und in erster Linie auch mit den jüngsten Standardisierungsbestrebungen (vgl. Arbeiten zur Definition eines Architekturmodells für "Open Systems" bei der ISO /ISO 78/) in Einklang steht.

Naturgemäß hat sich die nachfolgende Systemanalyse auf sämtliche relevanten Einflußfaktoren zu erstrecken, die das Ablaufgeschehen und insbesondere die Kommunikationsflüsse in einem Rechnernetz bestimmen: So ist neben den Hard- und Softwarecharakteristika von Rechnernetzen auch die Struktur der Aufträge zu analysieren, die durch ein Rechnernetz zu bearbeiten sind.

2.1. Physikalischer Rechnernetzaufbau

Aus topologischer Sicht legen wir folgende Definition eines Rechnernetzes R zugrunde:

$$R = DVS \cup KSS$$

wobei

DVS = Menge autonomer DV-Systeme (auch Arbeitsrechner genannt) zur Bearbeitung der Aufträge der Rechnernetzbenutzer; dabei soll verallgemeinernd zugelassen werden, daß ein Element aus DVS selbst wiederum ein Rechnernetz R' darstellt;

KSS = Kommunikationssubsystem, das den gesamten Datenaustausch zwischen Rechnernetzbenutzern und/oder Elementen der Menge DVS zu gewährleisten hat.

Eine Menge von Ein-/Ausgabestationen EAS bildet die Schnittstelle zwischen dem Rechnernetz und seinen Benutzern. Eingabestationen (z.B. Terminals, Kartenleser) dienen der Entgegennahme von Aufträgen, Ausgabestationen (z.B. Terminals, Zeilendrucker) der Abgabe der Bearbeitungsresultate an die Rechnernetzbenutzer. Ein-/Ausgabestationen können sowohl an Elemente der Menge DVS als auch direkt an das Kommunikationssubsystem angeschlossen sein (s. Abb. 2.1.).

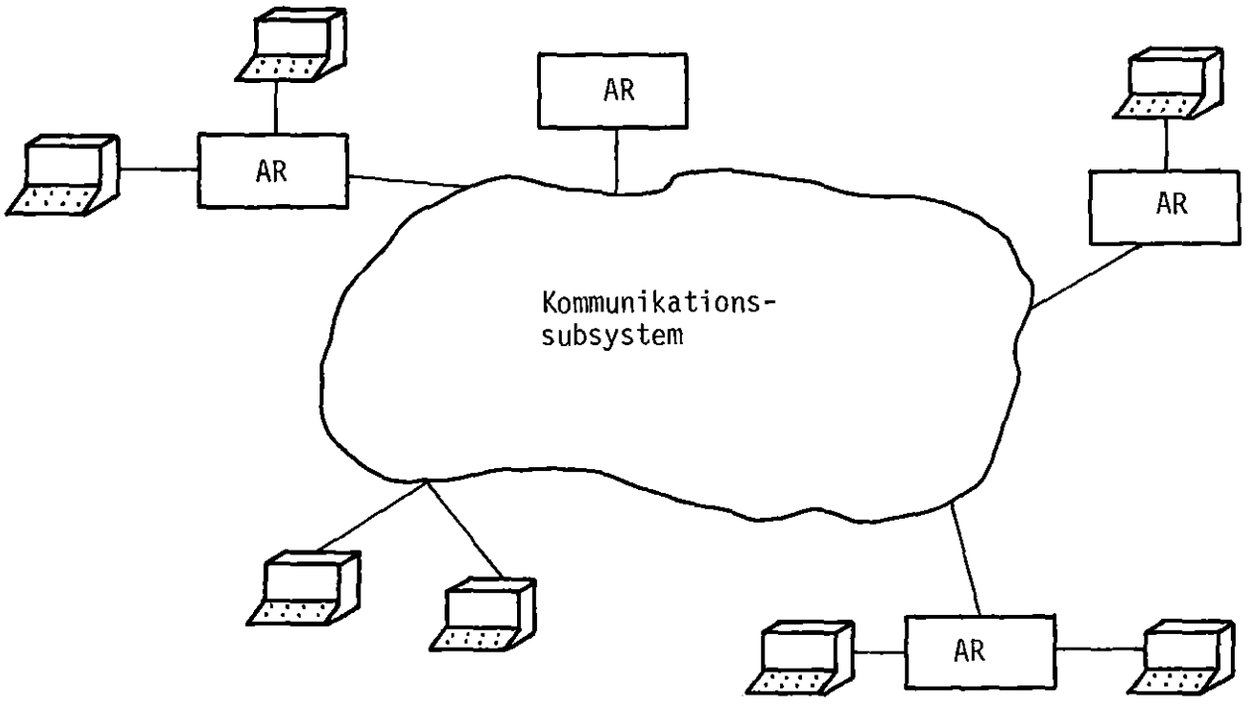

Abb. 2.1.: Idealisierte Struktur eines Rechnernetzes

AR = Element der Menge DVS

= Element der Menge EAS mit Rechnernetzbenutzer(n)

——— = Übertragungsleitung

Grundsätzlich gibt es folgende Möglichkeiten, ein Kommunikationssubsystem auszulegen (vgl. /STER 78/), die in Abb. 2.2. am Beispiel dreier zu verbindender Rechner AR_1, AR_2 und AR_3 schematisch dargestellt sind:

- direkte Verbindungen mittels Übertragungsleitungen [Punkt-zu-Punkt-Verbindungen, s. Fall (a)];
- Bus-Systeme, die sämtlichen zu verbindenden DV-Systemen bzw. Ein-/Ausgabestationen die koordinierte Benutzung eines Übertragungsmediums in bitserieller bzw. bit-paralleler Weise ermöglichen [Mehrpunktverbindungen, s. Fall (b)];
- ein Netz verbundener Vermittlungsrechner (VR), deren Aufgabe es ist, - durch Techniken wie Leitungs-, Paket- oder Nachrichtenvermittlung / FALK 77/, /ROBE 77/ - Übertragungswege zwischen den entsprechenden Kommunikationspartnern bereitzustellen [s. Fall (c)].

Kombinationen der oben genannten Möglichkeiten zum Aufbau eines Kommunikationssubsystems sind in existierenden Rechnernetzen üblich.

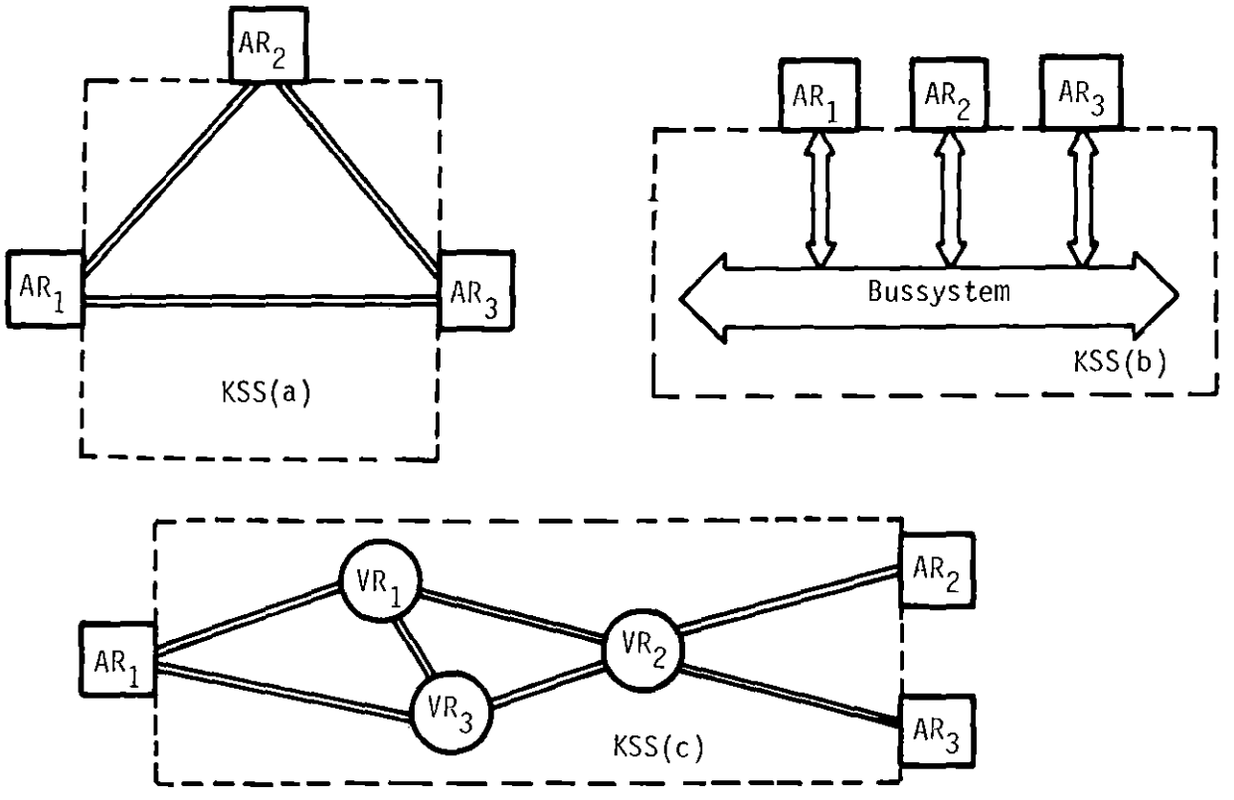

Abb. 2.2.: Mögliche Strukturen von Kommunikationssubsystemen

Die Vereinigungsmenge aus Vermittlungsrechnern und der Menge DVS soll im folgenden als Menge der Rechnernetzknoten bezeichnet werden.

Wesentliche physikalische Kenngrößen, die die Leistungsfähigkeit eines Rechnernetzes beeinflussen und die daher bei der Modellierung zu berücksichtigen sein werden, können bereits an dieser Stelle genannt werden:

- Leistungsfähigkeit und Zuverlässigkeit der CPU's, des Hauptspeichers oder der angeschlossenen peripheren Geräte (für Rechnernetzknoten);
- Fehlerraten, Kapazitäten, Zuverlässigkeit der Übertragungsleitungen und Vermittlungsrechner sowie die gewählte Topologie (für ein Kommunikationssubsystem).

Als Resultat der obigen Analyse ergibt sich die Notwendigkeit, in Kapitel 3 Modelle für die nachstehenden physikalischen Rechnernetzkomponenten bereitzustellen:

(P1) die Elemente der Menge DVS,
(P2) die Vermittlungsrechner,
(P3) die Ein-/Ausgabestationen,
(P4) die Verbindungen zwischen benachbarten Rechnernetzknoten oder zwischen Rechnernetzknoten und Ein-/Ausgabestationen.

2.2. Architektur der Rechnernetzsoftware

Die Hauptaufgabe eines Rechnernetzes besteht im Anbieten einer Reihe von Rechnernetzdiensten für die Rechnernetzbenutzer. Hierzu gehören zum einen anwendungsorientierte (Rechnernetz-) Dienste, wie z.B. die Übertragung von Dateien, das Starten eines Programms auf einem entfernten Rechner, der abgesetzte Zugriff auf Software, die koordinierte Bearbeitung von Aufträgen durch mehrere Rechnernetzknoten, etc. Daneben ist es notwendig, daß in einem Rechnernetz auch kommunikationsorientierte Dienste existieren, wie z.B. der Datenaustausch zwischen zwei direkt verbundenen Rechnernetzknoten, mit deren Hilfe die anwendungsorientierten Dienste in der Regel erst realisiert werden können. Die Möglichkeit des Zugriffs auf einen Rechnernetzdienst muß weder notwendigerweise für sämtliche Rechnernetzbenutzer gewährleistet sein, noch hat ein solcher Dienst auf jedem der Rechnernetzknoten zu existieren. Für die Bereitstellung eines Rechnernetzdienstes ist zumeist der koordinierte Einsatz mehrerer Rechnernetzknoten erforderlich. Da in diesen Fällen zur Abwicklung der Dienste Kommunikation (z.B. zwischen den Rechnernetzknoten) erforderlich ist, soll die Software, die Rechnernetzdienste realisiert, als Kommunikationssoftware bezeichnet werden.

Die Feststellung, daß anwendungsorientierte Rechnernetzdienste basierend auf kommunikationsorientierten Diensten abgewickelt werden, deutet bereits an, daß beim Entwurf von Kommunikationssoftware - analog zur Strukturierung von Betriebssystemsoftware - das Prinzip einer schichtenförmigen Softwarearchitektur angestrebt wird (siehe u.a. /ISO 78/).

Um dieses Strukturierungskonzept zu detaillieren, sei R ein Rechnernetz, dessen Kommunikationssoftware in n Schichten $S_1,\ldots,S_n$ ($n \geq 1$) organisiert ist. Eine Schicht S_i, $i \in \{1,\ldots,n\}$ realisiert dann einen oder mehrere un-

abhängige Dienste $D^i_1,\ldots,D^i_{m(i)}$, $m(i)\geq 1$. Jeder dieser Dienste D^i_k, $k \in \{1,\ldots,m(i)\}$ wird durch Bereitstellung einer Menge $KP(D^i_k)$ von Kommunikationsprimitiven über eine wohldefinierte Schnittstelle zur Benutzung durch höhere Schichten angeboten. Hierbei wird der Dienst D^i_k zum einen erbracht durch Realisierung einer Menge von Funktionen $F(D^i_k)$ innerhalb der Schicht S_i und gegebenenfalls durch zusätzliche Benutzung eines Dienstes, der durch eine Schicht S_j $(j<i)$ angeboten wird. Ein wesentliches Merkmal eines Dienstes ist, daß seine innere Struktur sämtlichen anderen Diensten verborgen bleibt.

Der Inhalt der Schicht S_i der Kommunikationssoftware besteht somit aus der Gesamtmenge von Funktionen $F(D^i_1) \cup F(D^i_2) \cup \ldots \cup F(D^i_{m(i)})$, die sie zu realisieren hat. Im Normalfall werden derartige Funktionen im Rechnernetz dezentral realisiert, d.h. in ihre Abwicklung sind Kommunikationspartner auf verschiedenen Rechnernetzknoten involviert. Eine solche dezentrale Organisation setzt allerdings die Existenz von Regeln voraus, die eine Absprache zwischen den Kommunikationspartnern darstellen, um ein koordiniertes Arbeiten zu ermöglichen. Diese Regeln werden zu einem Kommunikationsprotokoll /POUZ 75/ zusammengefaßt. Ohne Beschränkung der Allgemeinheit wollen wir voraussetzen, daß einem Dienst D^i_k genau ein Kommunikationsprotokoll zugeordnet ist, unter dessen Berücksichtigung die Funktionen $F(D^i_k)$ realisiert werden.

Als spezielles Beispiel zu den obigen Ausführungen zeigt Abb. 2.3. das Architekturmodell aus /ISO 78/, dessen Kommunikationssoftware in 7 Schichten organisiert ist. Das Beispiel beschränkt sich auf eine Darstellung zweier direkt verbundener Rechnernetzknoten DVS_1 und DVS_2, wobei jede Schicht der Kommunikationssoftware sowohl in DVS_1 als auch in DVS_2 verfügbar ist.

Aus operationaler Sicht läßt sich somit ein Rechnernetz beschreiben als System interagierender, organisatorischer Einheiten, die für die Ausführung gewisser Funktionen verantwortlich sind. Solche Funktionen werden entweder durch die Rechnernetzbenutzer definiert - die organisatorischen Einheiten nennen wir in diesem Falle Benutzerprogramme -, oder sie dienen der Durchführung von vordefinierten Rechnernetzdiensten. Jede organisatorische Einheit ist genau einem Rechnernetzknoten zugeordnet. Es lassen sich Klassen funktional äquivalenter organisatorischer Einheiten bilden, insbesondere bei dezentraler Durchführung eines Rechnernetzdienstes. Wir können zwei Arten von Interaktionen zwischen organisatorischen Einheiten E_1 und E_2 unterscheiden: implizite Interaktionen, falls sich E_1 und E_2 um dieselben benötigten Betriebsmittel bewerben oder explizite Interaktionen, falls eine konkrete Kommunikation, z.B. ein realer Austausch von Daten oder Kommunikationsprimitiven zwischen E_1 und E_2 stattfindet.

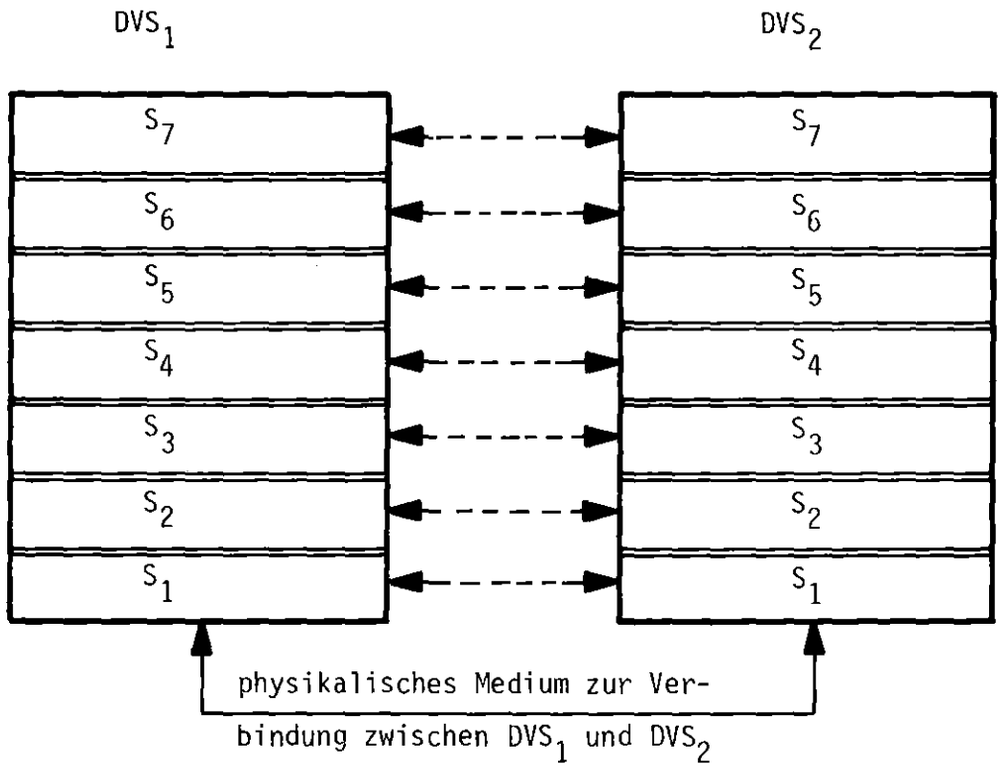

Abb. 2.3.: Das (aktuelle) Architekturmodell für "Open Systems" bei ISO/TC97/SC16

[Notation: ←- - -→ Kommunikationsprotokoll zur dezentralen Realisierung eines Dienstes in verschiedenen Rechnernetzknoten;

═══ Schnittstelle zwischen benachbarten Schichten;

S_7: Anwender-Schicht
S_6: Darstellungs-Schicht
S_5: Sitzungs-Schicht
S_4: Transport-Schicht
S_3: Netz-Schicht
S_2: Leitungs-Schicht
S_1: Physikalische Schicht]

Wie die Ausführungen über die Schichtenstruktur von Kommunikationssoftware zeigen, gibt es zwei Arten von expliziten Interaktionen zwischen den organisatorischen Einheiten, die für die Abwicklung eines Dienstes verantwortlich sind:

(a) "vertikale" Kommunikation:
diese Kommunikation tritt dann auf, wenn ein (höherer) Rechnernetzdienst über eine Schnittstelle den Dienst einer niedrigeren Schicht verwendet oder wenn eine Rückmeldung einer (niederen) Schicht an eine höhere erfolgt; die interagierenden organisatorischen Einheiten seien in diesem Falle als Nachbarn bezeichnet; Nachbarn befinden sich in demselben Rechnernetzknoten, die Art und Weise ihrer Kommunikation ist definiert durch die Eigenschaften der Schnittstelle; eine Schnittstelle zwischen benachbarten Schichten kann auf unterschiedliche Arten implementiert sein, wie z.B. durch expliziten Austausch von Kommunikationsprimitiven, Rückmeldungen, Signalen oder durch Prozeduraufrufe u.a.; zur Vereinheitlichung wollen wir für sämtliche Informationen, die zur Kommunikation zwischen Nachbarn ausgetauscht werden, den Oberbegriff Schnittstellenauftrag einführen;

(b) "horizontale" Kommunikation:
diese Kommunikationsart kennzeichnet den expliziten Datenaustausch zwischen organisatorischen Einheiten desselben Dienstes, in verschiedenen Rechnernetzknoten, der benötigt wird, um Funktionen in dezentraler Weise durchzuführen; die in dieser Weise interagierenden organisatorischen Einheiten sollen Korrespondenten genannt werden; die Kommunikation zwischen Korrespondenten wird durch ein Kommunikationsprotokoll geregelt, sie wird erreicht durch einen Austausch von Dateneinheiten, deren Syntax und Semantik durch das Kommunikationsprotokoll festgelegt ist; da keine direkte Verbindung zwischen Korrespondenten existiert, können Dateneinheiten zwischen Korrespondenten ausschließlich unter Zuhilfenahme eines Rechnernetzdienstes einer tieferen Schicht ausgetauscht werden.

Wichtige Ziele einer Schnittstelle sind u.a. die Durchführung einer Flußkontrolle sowie der Fehlerermittlung bzw. -korrektur für die zwischen den jeweiligen Nachbarn ausgetauschten Schnittstellenaufträge. (Die Flußkontrolle dient in diesem Zusammenhang dazu, die Geschwindigkeit des Senders von Schnittstellenaufträgen durch den Empfänger zu steuern).

Neben der Lösung ähnlicher Probleme (d.h. Fluß- und Fehlerkontrolle für auszutauschende Dateneinheiten) stellen sich Kommunikationsprotokollen zusätzlich folgende Aufgaben:

- Synchronisation der Aktivitäten von Korrespondenten;
- Fragmentierung von Daten beim Absenden und Reassemblierung durch den Ziel-Korrespondenten, z.B. wenn das Kommunikationssubsystem nur Dateneinheiten einer limitierten Länge transportieren kann;
- temporäres Zwischenpuffern von zu übertragenden Dateneinheiten;
- Effizienzerhöhung des Austauschs von Dateneinheiten (z.B. durch Datenkomprimierung, Codeumwandlung); etc.

Die hierarchische Strukturierung von Kommunikationssoftware spiegelt sich im Aufbau der Dateneinheiten wider. Grundsätzlich besteht eine Dateneinheit beim Eintreffen (bzw. Austreten) in die Schicht S_i aus einer Menge von Kontrollinformation $KI(S_i)$ und einem ggf. leeren Datenfeld, das Informationen beinhaltet, die ausschließlich für höhere Schichten von Relevanz sind und deren Semantik auch nur dort interpretiert wird. Die Kontrollinformation $KI(S_i)$ wird durch die für die Durchführung des betreffenden Rechnernetzdienstes verantwortliche organisatorische Einheit in der Schicht S_i generiert und durch den Ziel-Korrespondenten interpretiert und eliminiert. Der Aufbau der Kontrollinformation wird spezifiziert durch das Kommunikationsprotokoll. Die enthaltene Information bezieht sich z.B. auf die Angabe von Sende- bzw. Ziel-Korrespondenten, den Typ der Kontrollinformation (z.B. Quittung), die Länge des mitgeführten Datenfeldes und evtl. Angaben bzgl. der internen Struktur des Datenfeldes, wenn z.B. mehrere Dateneinheiten höherer Schichten zusammengefaßt werden.

Die Art der Strukturierung der Kommunikationssoftware und insbesondere die Leistungsfähigkeit der zugrundeliegenden Kommunikationsprotokolle impliziert starke Auswirkungen im Hinblick auf die Effizienz der Rechnernetzdienste, die den Rechnernetzbenutzern angeboten werden. Da jedoch besonders die Zuordnung der einzelnen Dienste bzw. der sie realisierenden Funktionen auf die verschiedenen Schichten der Kommunikationssoftware von elementarer Bedeutung für das Rechnernetzverhalten ist, genügt eine unabhängige Untersuchung der einzelnen Schichten i.a. nicht. Die Unterstützung einer integrierten Betrachtung der gesamten (Protokoll-) Hierarchie wird aus diesem Grunde ein vorrangiges Ziel bei der Modellbildung im nächsten Kapitel darstellen.

Neben der Kommunikationssoftware sind die (lokalen) Betriebssysteme in den Rechnernetzknoten ein entscheidender Einflußfaktor für das Ablaufgeschehen in

einem Rechnernetz, in erster Linie deshalb, weil sie durch die Art ihrer Betriebsmittelzuteilung die impliziten Interaktionen zwischen den organisatorischen Einheiten innerhalb eines Rechners festlegen (zu einem Überblick über wichtige Betriebsmittelzuteilungsstrategien in Betriebssystemen sei auf Brinch Hansen /BRIN 73/ verwiesen).

Die Schlußfolgerung, die sich aus der obigen Analyse von Rechnernetzsoftware ziehen läßt, ist, daß - zusätzlich zu den in 2.1. aufgeführten physikalischen Komponenten - Modelle für folgende Rechnernetzkomponenten benötigt werden:

(S1) die organisatorischen Einheiten, die Rechnernetzdienste realisieren,
(S2) die Benutzerprogramme,
(S3) die Dateneinheiten und Schnittstellenaufträge,
(S4) die Betriebssystemkomponenten, die die Betriebsmittelvergabe in einem Rechner organisieren.

2.3. Aufträge in einem Rechnernetz

Grundsätzlich lassen sich zwei Kategorien von Aufträgen in einem Rechnernetz unterscheiden, die in der Menge der bisher eingeführten Rechnernetzkomponenten bereits enthalten sind:

(1) die Benutzerprogramme, die die Aufträge der Rechnernetzbenutzer an das Rechnernetz repräsentieren, und
(2) die Schnittstellenaufträge, die die Inanspruchnahme oder die erfolgte Abwicklung einer Funktion eines Rechnernetzdienstes durch eine organisatorische Einheit einer benachbarten Schicht anzeigen.

Benutzerprogramme werden entweder direkt durch Rechnernetzbenutzer oder indirekt auf Initiative eines anderen Benutzerprogramms generiert bzw. gestartet. Wesentliche Charakteristika eines Benutzerprogrammes beziehen sich auf den Umfang der beanspruchten Betriebsmittel des Rechnernetzes (z.B. Rechenzeit, Speicherplatz, periphere Geräte und die ggf. benötigten Rechnernetzdienste). Benutzerprogramme können lokalen Charakter besitzen, wenn sämtliche Ressourcen am lokalen Rechnernetzknoten verfügbar sind, oder sie können Netzcharakter aufweisen, falls sie Rechnernetzdienste beanspruchen und so mit einem Korrespondenten kommunizieren. Es sei vorausgesetzt, daß ein Benutzerprogramm genau einem Rechnernetzknoten zugeordnet werden kann.

Schnittstellenaufträge werden erzeugt und interpretiert durch Benutzerprogramme oder durch die organisatorischen Einheiten, die Rechnernetzdienste realisieren. Schnittstellenaufträge müssen mit Angaben über die Art des Rechnernetzdienstes versehen sein, dessen Bereitstellung sie anfordern oder dessen Ausführungsende sie anzeigen. Auch Schnittstellenaufträge werden genau einem Rechnernetzknoten zugeordnet.

Zur Erzeugung realistischer Auftragsprofile, eine für die erfolgreiche Modellierung relevante Voraussetzung, muß - wie die obigen Ausführungen zeigen - bei der Modellbildung dafür Sorge getragen werden, daß unterschiedliche Arten von Aufträgen in einem Rechnernetz generiert werden können. Neben einer wirklichkeitsgetreuen Festlegung der Auftragsstruktur ist es darüber hinaus wesentlich, daß für die Zeitpunkte der Auftragserzeugung verschiedenartige Wahrscheinlichkeitsverteilungen zugrundegelegt werden können.

3. KONZEPTIONELLE MODELLE ZUR BESCHREIBUNG VON KOMMUNIKATIONSFLÜSSEN IN RECHNERNETZEN

Die in Kapitel 2 durchgeführte Analyse hatte in erster Linie zum Ziel, diejenigen Rechnernetzkomponenten herauszuarbeiten, die für eine Simulation der Kommunikationsflüsse in einem Rechnernetz von Relevanz sind. Im nächsten Schritt geht es nun darum, konzeptionelle Modelle für diese Rechnernetzkomponenten zu formulieren. Dabei sollen zunächst Modelle eingeführt werden, die eine Beschreibung schichtenförmig organisierter Kommunikationssoftware erlauben, wozu Modelle zur Nachbildung von Kommunikationsprotokollen benötigt werden. Das Hauptaugenmerk bei der Formulierung der konzeptionellen Modelle wird einerseits auf eine detaillierte Modellierung der Kommunikationsprotokolle gelegt, und andererseits ist die Möglichkeit des Nachbildens der vollständigen Protokollhierarchie im Rechnernetzmodell zu gewährleisten, wobei aus Effizienzgründen ein unterschiedlicher Detaillierungsgrad der verschiedenen Protokollschichten erreichbar sein sollte (über den Mangel an derartigen Modellen siehe u.a. /BROW 76/, /LEGO 76b/, /GOUD 76/).

Eine Modellierung von Kommunikationsprotokollen wird hauptsächlich aus zwei Gründen durchgeführt:

(a) zur Protokollverifikation (logische Protokollanalyse s. /GOUD 76/), d.h. zum Nachweis der Verklemmungsfreiheit, der Widerspruchsfreiheit und der Vollständigkeit von Kommunikationsprotokollen, s. /BOCH 77/, /SUNS 78/;

(b) zur Untersuchung der Leistungsfähigkeit von Protokollen bzw. Protokollschichten (quantitative Protokollanalyse s. /GOUD 76/); interessante Leistungskenngrößen betreffen z.B. die Dateneinheitenverzögerung innerhalb einer Protokollschicht in Abhängigkeit von Protokolleigenschaften, die Auslastung von Rechnernetzkomponenten als Funktion der Rechnernetzbelastung und der organisatorischen Struktur der Rechnernetzdienste, die Zuverlässigkeit eines Kommunikationssubsystems, etc. /SUNS 76/.

Da es in der vorliegenden Arbeit vor allem um Aussagen hinsichtlich des Einflusses von Kommunikationsprotokolleigenschaften auf die Leistungsfähigkeit eines Rechnernetzes geht, steht die Behandlung der Problematik (b) hier im Vordergrund.

Ein Rechnernetz - als Menge interagierender organisatorischer Einheiten (s. Kapitel 2) - kann gesehen werden als diskretes System, da die Änderung des

Rechnernetzzustandes zu diskreten Zeitpunkten erfolgt. Zur Formulierung der Modelle sind somit an dieser Stelle formale Beschreibungsmittel für diskrete Systeme erforderlich.

Falls Korrektheitsbeweise im Vordergrund stehen, gehen wichtige Beschreibungsmethoden für diskrete Systeme zurück auf die Theorie der Petri-Netze /PETE 77/ oder der sequentiellen Automaten: Die Automatentheorie definiert einen sequentiellen Automaten als 5-Tupel (Z,X,Y,M,N), wobei Z = eine Menge von Zuständen, X = eine Menge von Eingabesymbolen, Y = eine Menge von Ausgabesymbolen, M: $Z \times X \rightarrow Z$ eine Zustandsübergangsfunktion und N: $Z \times X \rightarrow Y$ eine Ausgabefunktion bezeichnet.

Falls die Untersuchung der Leistungsfähigkeit eines diskreten Systems das vorrangige Ziel darstellt, ist der Einsatz der Warteschlangentheorie erfolgversprechend /KLEI 76/: In der Warteschlangentheorie werden die organisatorischen Einheiten eines diskreten Systems als Bedienungsstation mit einer oder mehreren Warteschlangen beschrieben. Warteschlangen dienen der Aufnahme und Speicherung von Elementen, die einen Bearbeitungsauftrag für die Bedienungsstation darstellen. Die Voraussetzungen, die üblicherweise an Warteschlangenmodelle gestellt werden, erstrecken sich auf die Zugrundelegung spezieller Verteilungsfunktionen für das Eintreffen von Bearbeitungsaufträgen und ihre zugeordnete Bearbeitungsdauer, Strategien zur Warteschlangenabarbeitung, Speicherkapazität (z.B. maximale Länge) von Warteschlangen. Die Resultate der Warteschlangentheorie erlauben Aussagen bzgl. der Verweilzeiten von Elementen in einer Warteschlange, durchschnittliche Warteschlangenlänge, Auslastung der Bedienungsstation etc.

Die bisherigen Arbeiten auf dem Gebiet Modellierung von Kommunikationsprotokollen (/DANT 78/, /GOUD 77/, /LEGO 76a/, /LEMO 73/ u.a.) sind insbesondere gekennzeichnet durch eine oder mehrere der nachfolgenden Einschränkungen:

(a) die Modelle mußten stark vereinfacht werden, um bei der Modellauswertung spezielle Verfahren (z.B. analytische Methoden) einsetzen zu können; hierunter fallen die meisten der existierenden Warteschlangenmodelle;

(b) die Modelle sind auf die Behandlung der Problematik der Protokollverifikation zugeschnitten und klammern Fragen wie Leistungsfähigkeit von Kommunikationsprotokollen weitgehend aus; zur Protokollbeschreibung werden hier meist endliche sequentielle Automaten (d.h. sequentielle Automaten mit endlicher Zustandsmenge Z) verwendet /SUNS 78/;

(c) die Modelle sind auf die Untersuchung spezieller Kommunikationsprotokolle bzw. spezieller Protokolleigenschaften zugeschnitten und bieten u.a. keine Möglichkeit einer flexiblen Anpassung an geänderte Randbedingungen;

(d) die Modelle erlauben nicht die integrierte Untersuchung vollständiger Protokollhierarchien.

Für unsere Zwecke lassen sich die aktuell zur Verfügung stehenden Modelle zur Beschreibung von Kommunikationsprotokollen daher nicht zufriedenstellend verwenden, und hieraus resultiert die Notwendigkeit, grundlegend neue Modelle zu entwerfen.

Bei der hier gewählten Beschreibungsmethode zur Formulierung der konzeptionellen Modelle ist die Vorgehensweise derart, daß die organisatorischen Einheiten als Warteschlangenmodelle dargestellt werden. Da jedoch Warteschlangenmodelle nicht in der Lage sind, die interne Ablaufstruktur einer Bedienungsstation vollständig darzustellen, wird die Wirkungsweise der organisatorischen Einheiten durch Verwendung sequentieller Automaten präzisiert, wobei den Zustandsübergängen der Automaten zur weiteren Detaillierung Algorithmen zugeordnet werden können. Diese Beschreibungsmethode zeichnet sich nicht zuletzt dadurch aus, daß sie die Umsetzung der Modelle in eine Simulationssprache, z.B. in SIMULA /DAHL 68/, erleichtert und auf diese Weise einer anschließenden Modellauswertung durch rechnergestützte Simulation entgegenkommt.

3.1. Konzept zur Beschreibung von Kommunikationsprotokollen und Schichtenschnittstellen

Der Erstellung konzeptioneller Modelle für Kommunikationsprotokolle - als Basis einer quantitativen Protokollanalyse - sei die Einführung einiger Begriffe vorangestellt:

Definition:

Eine Protokolleinheit bezeichne denjenigen Teil einer organisatorischen Einheit in einem Rechnernetzknoten, der - für einen festen Rechnernetzdienst - der Durchführung der "vertikalen" sowie der "horizontalen" Kommunikation dient (zur Definition der Bezeichnungen "vertikale" und "horizontale" Kommunikation s. Abschnitt 2.2.).

Für eine Protokolleinheit PE lassen sich die beiden Klassen NBHS(PE) und NBTS(PE) von Nachbarn und die Klasse KORR(PE) von Korrespondenten definieren:

- NBHS(PE) bezeichnet die Menge aller Nachbarn höherer Schichten, die als Benutzer des durch die Protokolleinheit PE angebotenen Dienstes fungieren; NBHS(PE) kann Protokolleinheiten und/oder Benutzerprogramme enthalten.
- NBTS(PE) bezeichnet die Menge aller Nachbarn tieferer Schichten, deren angebotene Dienste durch die Protokolleinheit PE benutzt werden; bei den Elementen in NBTS(PE) kann es sich um Protokolleinheiten und/oder Übertragungskanäle handeln.
- KORR(PE) enthält sämtliche Korrespondenten, die in derselben Schicht wie die Protokolleinheit PE anzusiedeln sind und zu denen eine virtuelle oder direkte Verbindung grundsätzlich möglich ist; naturgemäß benutzen sämtliche Elemente aus KORR(PE) dasselbe Kommunikationsprotokoll.

Die Menge NB(PE): = NBHS(PE) $\cup$ NBTS(PE) beinhaltet genau die Nachbarn der Protokolleinheit PE. Falls der Bezug zur entsprechenden Protokolleinheit offensichtlich ist, werden die oben eingeführten Mengen in der Folge abkürzend als NBHS, NBTS, NB und KORR bezeichnet.

Als Beispiel für die Verwendung der oben eingeführten Begriffe und zur gleichzeitigen Erläuterung einer graphischen Darstellungsmethode für Hierarchien von Kommunikationsprotokollen - die im weiteren Verlauf dieser Arbeit benutzt werden wird - soll das Architekturmodell der Abb. 2.3. nochmals herangezogen werden. Dabei wollen wir voraussetzen, daß jede der Protokollschichten S_5, S_6 und S_7 genau einen Rechnernetzdienst beinhalte und sowohl im Rechnernetzknoten DVS_1 als auch in DVS_2 jeweils durch eine Protokolleinheit repräsentiert werde. Falls des weiteren $PE_{i,j}$ die Protokolleinheit bezeichnet, die den Dienst der Schicht S_i im Rechnernetzknoten DVS_j bereitstellt, dann ergibt sich die durch Abb. 3.1. illustrierte Darstellung.

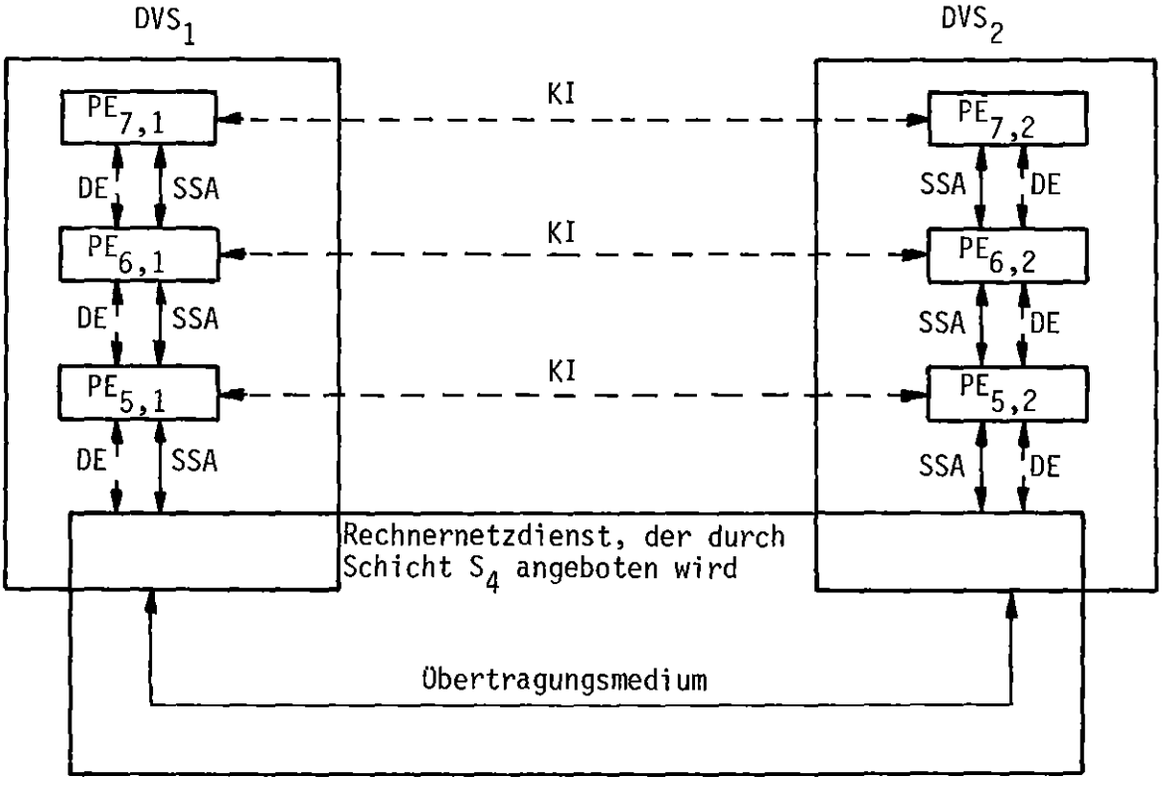

Abb. 3.1.: Darstellung einer Hierarchie von Kommunikationsprotokollen ("Protokolleinheiten-Sicht")

[Notation: ←SSA→ Austausch von Schnittstellenaufträgen
←DE→ Austausch von Dateneinheiten
←KI→ Austausch von Kontrollinformation, die in Dateneinheiten enthalten ist].

Für die Protokolleinheiten der Abb. 3.1. ergeben sich offensichtlich folgende Relationen:

- $KORR(PE_{i,1}) = \{PE_{i,2}\}$ und $KORR(PE_{i,2}) = \{PE_{i,1}\}$ für $i \in \{5,6,7\}$;
- $NBHS(PE_{i,j}) = \{PE_{i+1,j}\}$ für $i \in \{5,6\}$ und $j \in \{1,2\}$;
- $NBTS(PE_{i,j}) = \{PE_{i-1,j}\}$ für $i \in \{6,7\}$ und $j \in \{1,2\}$.

Um die Möglichkeit zu bieten, aus Teilmodellen das Modell einer vollständigen Protokollhierarchie aufzubauen, muß an dieser Stelle zum einen das allgemeine Modell einer Protokolleinheit bereitgestellt werden, und darüber hinaus sind die Interaktionen zwischen Protokolleinheiten zu modellieren.

3.1.1. Modell für Protokolleinheiten

Eine Protokolleinheit hat im wesentlichen die Aufgabe, den durch sie realisierten Rechnernetzdienst D für Elemente der Menge NBHS verfügbar zu machen. Da der Dienst D, für dessen Abwicklung die Protokolleinheit verantwortlich ist, in eine (endliche) Menge von Funktionen F(D) zerfällt, bietet sich ein Modell an, das der Durchführung jeder Funktion $f_i \in F(D)$ eineindeutig einen (Schnittstellen-) Auftrag SSA_i zuordnet. Ein Dienst wird somit durch Vorgabe sämtlicher verwendbarer Schnittstellenaufträge und deren Bedeutung vollständig beschrieben. Der Protokolleinheit selbst kommt aus dieser Sicht die Aufgabe zu, die Schnittstellenaufträge eines Nachbarn entgegenzunehmen, zu interpretieren und in der durch das Kommunikationsprotokoll vorgegebenen Weise darauf zu reagieren. Die Reaktion auf Schnittstellenaufträge kann, z.B. bei einem Sendevorgang, u.a. im Aufbereiten einer Dateneinheit und in der Übergabe eines Schnittstellenauftrags an einen Nachbarn bestehen. Schnittstellenaufträge (z.B. Kommunikationsprimitive und Rückmeldungen) können durch Elemente der Menge NBHS übergeben werden oder sie können von Elementen der Menge NBTS erhalten werden. Die Eigenschaften der Schnittstelle zwischen benachbarten Protokollschichten gehen in das Modell ein durch den Modus des Austauschs von Schnittstellenaufträgen, wohingegen das Kommunikationsprotokoll zwischen Korrespondenten seine Berücksichtigung findet in der Art der Bearbeitung von Schnittstellenaufträgen durch die Protokolleinheiten.

Ein erstes, stark vereinfachtes, Warteschlangenmodell einer Protokolleinheit (Abb. 3.2.) umfaßt eine Bedienungsstation (zur Bearbeitung von Schnittstellenaufträgen) und eine Warteschlange (zur Speicherung der Aufträge). Die Rückführung von Schnittstellenaufträgen aus der Bedienungsstation in die Warteschlange ATQ resultiert u.a. aus der Tatsache, daß die Bearbeitung von Aufträgen in bestimmten Fällen zurückgestellt werden kann.

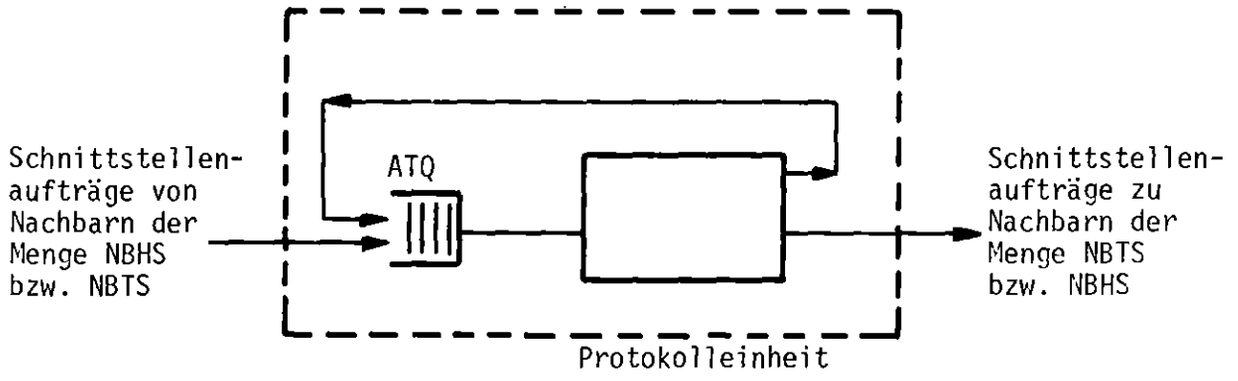

Abb. 3.2.: Warteschlangenmodell einer Protokolleinheit (Grobstruktur)

Es soll hier vorgeschlagen werden, die interne Struktur einer Protokolleinheit PE als sequentiellen Automaten darzustellen, wobei

X = Menge aller möglichen Schnittstellenaufträge, die durch Nachbarn übergeben werden können;
Y = Menge aller möglichen Schnittstellenaufträge, die an Nachbarn übergeben werden können;
Z = Menge der Zustände der Protokolleinheit, die geprägt wird durch
(a) den aktuellen Zustand des Kommunikationsprotokolls für jeden der Korrespondenten,
(b) den aktuellen Zustand der Schnittstelle zu jedem der Nachbarn,
(c) den Grad der Bearbeitung der momentan vorliegenden Schnittstellenaufträge;
M: $Z \times X \rightarrow Z$ = die möglichen Zustandsübergänge der Protokolleinheit;
N: $Z \times X \rightarrow Y$ = die Aktionen der Protokolleinheit.

Die Zustandsmenge Z sowie die Funktionen M und N sind nach Fixierung von Kommunikationsprotokoll und Schnittstelle zu konkretisieren. Die Vorgehensweise ist derart, daß sukzessive

(1) das der Protokolleinheit zugrundeliegende Protokoll als sequentieller Automat formuliert wird (ein Beispiel hierzu s. /WOLF 78a/);

(2) die Menge der möglichen Schnittstellenaufträge definiert wird, die zu bearbeiten sind (d.h. Detaillierung der Menge X);

(3) die Menge der möglichen Schnittstellenaufträge definiert wird, die durch die Protokolleinheit erzeugt werden können (d.h. Detaillierung der Menge Y);

(4) unter Berücksichtigung von Kommunikationsprotokoll und Schnittstelleneigenschaften, die Menge Z sowie die Funktionen M und N detailliert werden.

Zu Problemkreis (4) ist zu bemerken, daß die Abbildung einer Protokolleinheit auf die nachfolgende Grobstruktur nahezu immer möglich sein dürfte und im vorliegenden Modell angestrebt wird.

Dabei gehen wir von vier Grundzuständen einer Protokolleinheit aus:

Z1: Bereitschaft zur Auswahl und Analyse von Schnittstellenaufträgen;
Z2: Beendigung der Analyse eines Auftrags;
Z3: Beendigung der Bearbeitung eines Auftrags;
Z4: Blockierungszustand der Protokolleinheit (wir nennen eine Protokolleinheit blockiert, wenn für sämtliche Elemente der Warteschlange ATQ eine aktuelle Bearbeitung ausgeschlossen ist).

Ein Automatengraph für eine Protokolleinheit, der auf genau diesen Grundzuständen basiert, wird durch Abb. 3.3. gegeben. Dabei bezeichnen die Knoten des Graphen die Zustände und die gerichteten Pfeile die Zustandsübergänge. Die Markierung der Pfeile in der Form α/β besagt, daß die Gültigkeit der Bedingung α die Durchführung der Aktivität β (z.B. gegeben durch einen Algorithmus) nach sich zieht. Die Einführung des Zustands Z4 dient der Vermeidung unnötiger Aktivitäten der Protokolleinheit. Die Verdrängung eines in Bearbeitung befindlichen Auftrags kann z.B. in Zustand Z2 erlaubt sein; in diesem Fall wird der aktuell bearbeitete Auftrag zurückgestellt, und es erfolgt ein Übergang nach Z1.

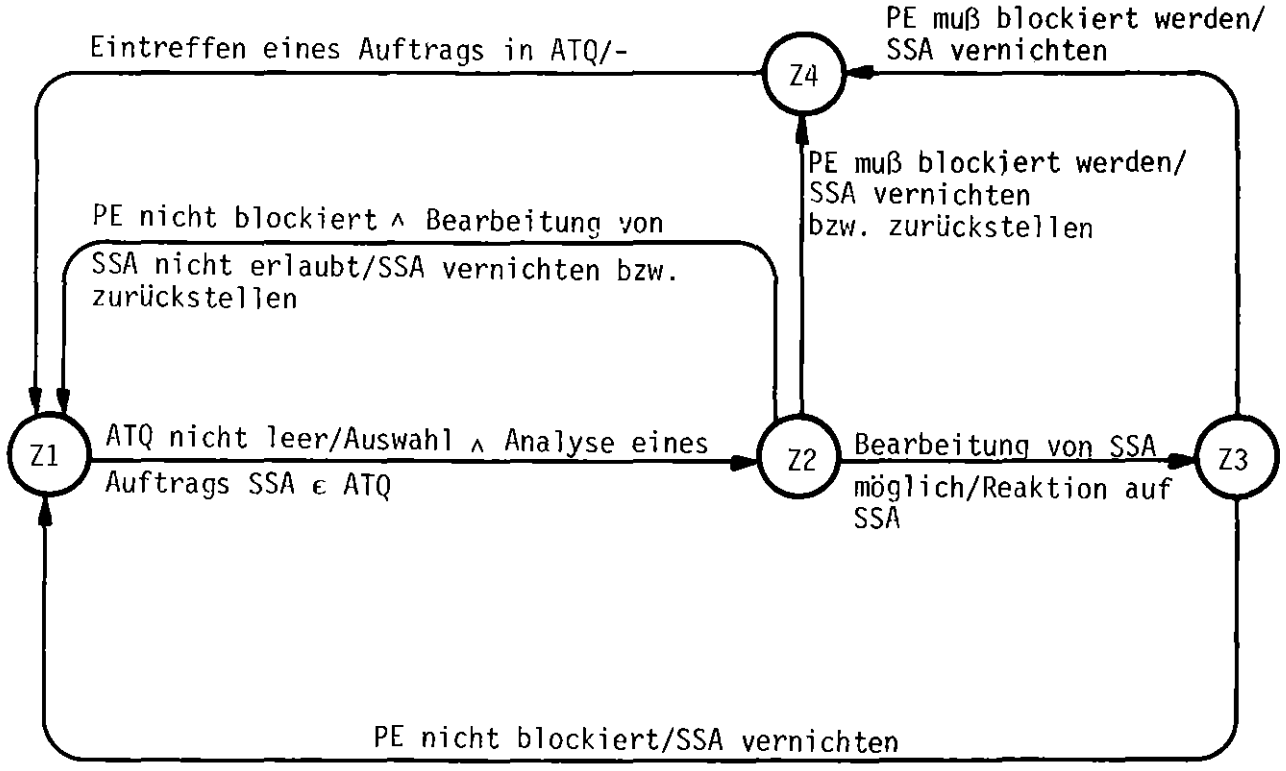

Abb. 3.3.: Zustandsübergangsdiagramm einer Protokolleinheit PE (Grobstruktur)

3.1.2. Modell für Protokollmoduln

Nach dieser eher makroskopischen Betrachtungsweise einer Protokolleinheit soll eine mögliche Verfeinerung des Modells dargelegt werden. Diese Verfeinerung trägt der Tatsache Rechnung, daß in existierenden Rechnernetzimplementierungen eine Protokolleinheit oft in verschiedene Protokollmoduln aufgetrennt wird, z.B. um den Moduln getrennte Speicherbereiche zuzuordnen oder um ihnen verschiedene Bearbeitungsprioritäten für die Betriebsmittelvergabe innerhalb des Rechners zuzuweisen. Zur Modellierung einer Protokolleinheit als Basis einer quantitativen Protokollanalyse erscheint uns das Warteschlangenmodell der Abb. 3.4. adäquat. Dieses verfeinerte Warteschlangenmodell beinhaltet drei Bedienungsstationen:

(1) den Sendeprotokollmodul (SPM), der für die Abwicklung der ausgabeseitigen Funktionen einer Protokolleinheit verantwortlich ist und die Reaktion auf Schnittstellenaufträge durchführt, die von Nachbarn erhalten und in der Warteschlange SATQ gespeichert werden. Da der SPM u.a. das Absenden von Dateneinheiten an Korrespondenten unterstützt, führt er zwei Warte-

schlangen für abzusendende Dateneinheiten: die Warteschlange SENDQ für Dateneinheiten, die auf eine Weitergabe an Nachbarn der Menge NBTS warten und die Warteschlange COPYQ, die Kopien von Dateneinheiten speichert, welche bereits an einen Korrespondenten abgesandt wurden (die Kopien werden für eine evtl. Übertragungswiederholung aufbewahrt);

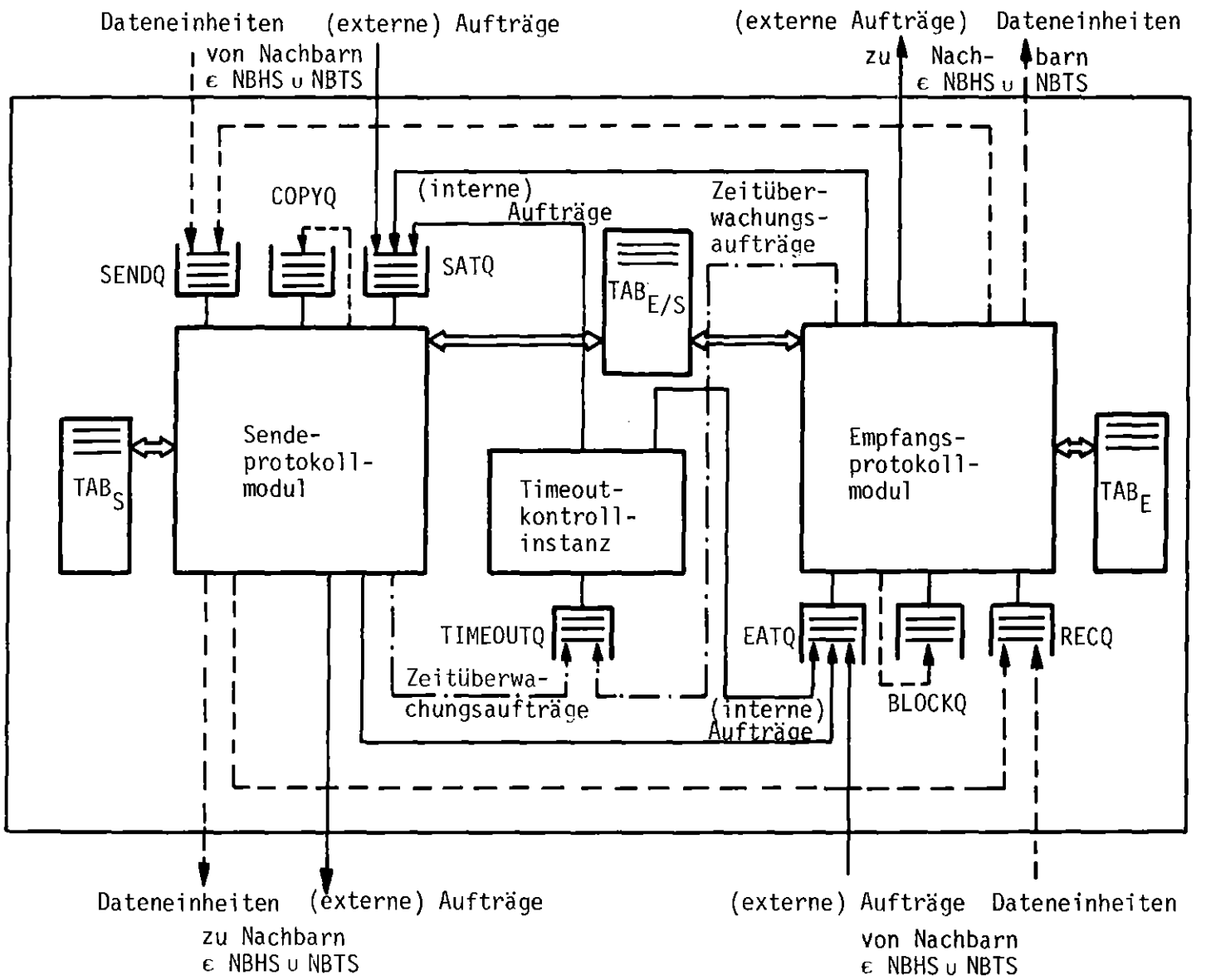

Abb. 3.4.: Detailliertes Warteschlangenmodell einer Protokolleinheit
[⟶ : Austausch von (Schnittstellen-) Aufträgen;
⇢ : Austausch von Dateneinheiten]

(2) den Empfangsprotokollmodul (EPM), welcher der Abwicklung der eingabeseitigen Funktionen einer Protokolleinheit dient. Dem EPM obliegt die Reaktion auf Schnittstellenaufträge, die von Nachbarn in die Warteschlange EATQ eingefügt werden. Zur Speicherung der zu empfangenden Dateneinheiten steht die Warteschlange RECQ zur Verfügung und Dateneinheiten, deren sofortige Weitergabe an einen Nachbarn nicht erlaubt ist (z.B. bei der Reassemblierung von Datenfragmenten), warten in der Warteschlange BLOCKQ;

(3) die Timeout-Kontrollinstanz (TKI), die von SPM oder EPM vorgegebene Zeitschranken zur Überwachung kritischer Aktivitäten als Anforderungen in der Warteschlange TIMEOUTQ speichert und den Ablauf solcher Zeitschranken an SPM bzw. EPM durch Übergabe eines (internen) Auftrags meldet.

Der durch Abb. 3.3. gegebene sequentielle Automat kann nach entsprechender Änderung sowohl zur Grobbeschreibung der internen Struktur des Sende- als auch des Empfangsprotokollmoduls herangezogen werden. Die Änderung beschränkt sich auf eine Ersetzung der Warteschlange ATQ im Zustandsübergangsdiagramm der Abb. 3.3. durch SATQ (im Falle des SPM) bzw. EATQ (im Falle des EPM). Die Abarbeitung der Aufträge durch die Protokollmoduln SPM bzw. EPM erfolgt im übrigen unter Berücksichtigung der aktuellen Information in den Tabellen TAB_S, TAB_E, bzw. $TAB_{E/S}$, die u.a. den aktuellen Zustand des Kommunikationsprotokolls und der Schnittstellen beinhalten. Die Interaktionen zwischen TKI und SPM bzw. EPM sowie zwischen den Protokollmoduln SPM und EPM untereinander, werden durch expliziten Austausch von (internen) Aufträgen im Modell berücksichtigt; auf diese Weise werden im detaillierten Modell die durch Nachbarn erzeugten (externen) Schnittstellenaufträge durch "Protokolleinheiten-lokale" Aufträge ergänzt.

Die interne Ablaufstruktur einer Protokolleinheit ist - wie bereits erwähnt - gekennzeichnet durch eine starke Abhängigkeit von den Charakteristika des zugrundeliegenden Kommunikationsprotokolls. Die Erstellung gegenseitig unabhängiger Modelle für die Protokolleinheiten innerhalb der Rechnernetzknoten erlaubt zwar eine optimale Anpassung der Protokolleinheitenstruktur an Protokollbesonderheiten, andererseits ist die dadurch entstehende Vielzahl von diversen Protokolleinheitenmodellen mit erheblichem Aufwand (z.B. für Modellimplementierung und Modellverifikation) verbunden. Bei näherer Betrachtung lassen sich jedoch eine Reihe von Aufgaben herausarbeiten, die Protokolleinheiten unterschiedlicher Protokollschichten gemeinsam sein können, wie z.B.

(a) die Fragmentierung bzw. Reassemblierung von Dateneinheiten,
(b) die Generierung von Zeitüberwachungsaufträgen zur Überwachung kritischer Aktivitäten,
(c) die Erstellung von Kontrollinformationen für den/die Korrespondenten,
(d) die Interpretation von und Reaktion auf Kontrollinformation,
(e) das Multiplexen/Demultiplexen von (logischen) Verbindungen zwischen Korrespondenten.

Solche Gemeinsamkeiten von Protokolleinheiten legen folgende Vorgehensweise nahe: In einem Grundmodell werden solche Funktionen berücksichtigt, die mehreren oder gar sämtlichen Protokolleinheiten eigen sind. Je nach Umfang der rechnerdienstspezifischen Eigenschaften einer Protokolleinheit ist dieses Grundmodell im konkreten Anwendungsfall auszuweiten. Das Grundmodell einer Protokolleinheit erstreckt sich

(1) auf die Definition typischer Schnittstellenaufträge, die für eine Vielzahl von Kommunikationsprotokollen Verwendung finden können;

(2) auf die Vorgabe bestimmter Algorithmen zur Bearbeitung typischer Schnittstellenaufträge durch eine Protokolleinheit.

3.1.3. Klasse von Basisprotokollen

Für das Grundmodell einer Protokolleinheit wurde - neben dem allgemein anwendbaren Warteschlangenmodell der Abb. 3.4. - bisher eine Menge von elementaren Aufgaben definiert, die bei der Modellierung einer Vielzahl von Kommunikationsprotokollen wiederkehren, und die deshalb in das Grundmodell eingearbeitet wurden. An dieser Stelle soll das Grundmodell einer Protokolleinheit nun präzisiert werden, wobei insbesondere auszuführen sein wird, auf welche Weise die elementaren Aufgaben im Grundmodell realisiert werden.

Es sei hierzu PE eine Protokolleinheit der Protokollschicht S_i. Um das Grundmodell für PE detaillieren zu können, sind primär Angaben bzgl. des Rechnernetzdienstes D erforderlich, den die Protokolleinheit PE im Zusammenwirken mit ihren Korrespondenten durchführen soll, oder - was gleichbedeutend ist - die Menge der Funktionen F(D) ist anzugeben, die die Protokolleinheit PE ihren Nachbarn der Menge NBHS(PE) anbietet. Darüber hinaus muß ein Kommunikationsprotokoll vorgegeben werden, das die Kommunikation zwischen der Protokolleinheit PE und ihren Korrespondenten regelt.

Der Rechnernetzdienst D, den die Protokolleinheit PE zusammen mit ihren Korrespondenten zu realisieren habe, wird hier beschrieben durch die Kommunikationsprimitive, mit denen er in Anspruch genommen werden kann. Zu diesem Zweck setzen wir voraus, daß eine Protokolleinheit PE_0 den Rechnernetzdienst D beanspruchen will und daß PE_0 den Dienst D genau über PE ansprechen kann (s. Abb. 3.5.).

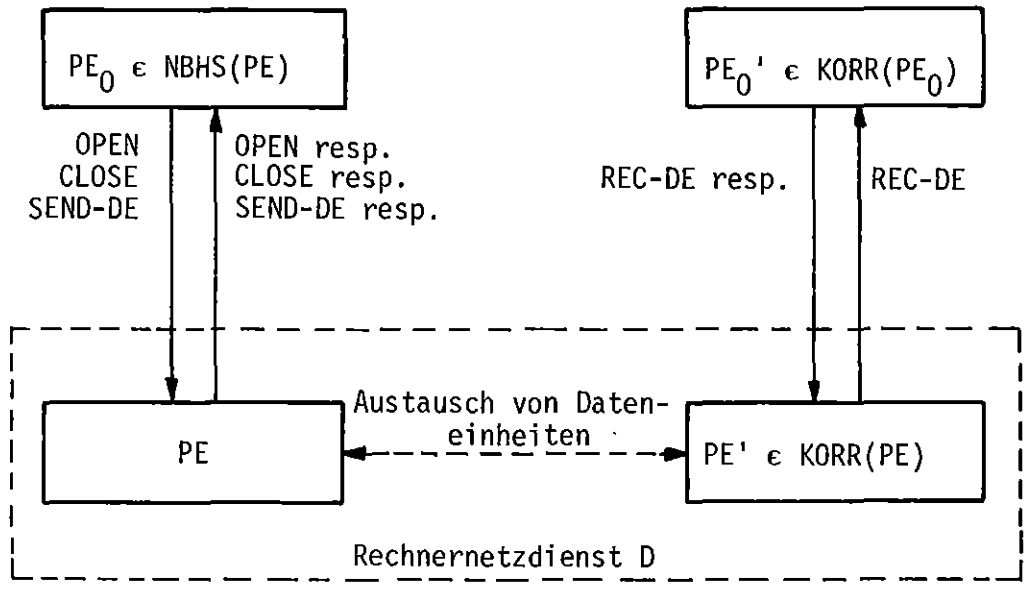

Abb. 3.5.: Durch das Grundmodell einer Protokolleinheit realisierter Rechnernetzdienst

Unter den obigen Voraussetzungen akzeptiere PE folgende Arten von Kommunikationsprimitiven von PE_0:

(1) Verbindungsaufbau (durch die Protokolleinheit PE_0 initiiert):
"OPEN CONNECTION (PE_0, PE_0')" - Schnittstellenaufträge [Kurzform: OPEN], mit deren Hilfe PE_0 veranlaßt, daß PE eine (logische) Verbindung innerhalb der Protokollschicht S_i aufbaut, über welche PE_0 mit ihrem Korrespondenten PE_0' anschließend kommunizieren kann;

(2) Transfer einer Dateneinheit (von der Protokolleinheit PE_0 zu PE_0'):
"SEND DATAUNIT (DE, PE_0, PE_0')" - Schnittstellenaufträge [Kurzform: SEND-DE], mit deren Hilfe PE_0 veranlaßt, daß PE eine Dateneinheit DE an die Protokolleinheit PE_0' weiterleitet;

(3) Verbindungsabbau (durch die Protokolleinheit PE_0 initiiert):
"CLOSE CONNECTION (PE_0, PE_0')" - Schnittstellenaufträge [Kurzform: CLOSE], mit deren Hilfe PE_0 veranlaßt, daß PE eine existierende (logische) Verbindung innerhalb der Protokollschicht S_i abbaut, über welche PE_0 mit ihrem Korrespondenten PE_0' bislang kommunizierte.

Nach erfolgter Bearbeitung eines jeden dieser Schnittstellenaufträge übergebe die Protokolleinheit PE eine entsprechende Rückmeldung an PE_0 (s. Abb. 3.5.); bei diesen Rückmeldungen handle es sich um die Schnittstellenaufträge

(1) "OPEN CONNECTION RESPONSE" [Kurzform: OPEN resp.],
(2) "SEND DATAUNIT RESPONSE" [Kurzform: SEND-DE resp.],
(3) "CLOSE CONNECTION RESPONSE" [Kurzform: CLOSE resp.].

Damit sind die Schnittstellenaufträge vollständig, die bei der Inanspruchnahme von Funktionen des Rechnernetzdienstes D durch PE_0 an der Schnittstelle zwischen PE und PE_0 benötigt werden. Für die Funktion "Transfer einer Dateneinheit (von PE_0 zu PE_0')" wird allerdings zusätzlich ein Schnittstellenauftrag für die Schnittstelle auf der Empfangsseite, d.h. zwischen PE' und PE_0', benötigt (s. Abb. 3.5.). Es handelt sich hierbei um den Schnittstellenauftrag

- "RECEIVE DATAUNIT (DE, PE_0, PE_0')" [Kurzform: REC-DE], der dazu dient, der Protokolleinheit PE_0' die Ankunft einer Dateneinheit DE anzuzeigen, die die Protokolleinheit $PE_0 \in KORR(PE_0')$ über PE und PE' an PE_0' sandte.

Ein "RECEIVE DATAUNIT (DE, PE_0, PE_0')" - Schnittstellenauftrag wird bei Ankunft einer Dateneinheit DE von PE' an PE_0' übergeben, und die erfolgte Bearbeitung durch PE_0' wird mit einer "RECEIVE DATAUNIT RESPONSE" - Rückmeldung [Kurzform: REC-DE resp.] der Protokolleinheit PE' angezeigt.

Bislang wurde angenommen, daß bei der Kommunikation zwischen PE_0 und PE_0' die Initiative durch die Protokolleinheit PE_0 ergriffen wird. Umgekehrt können wir nun jedoch voraussetzen, daß PE_0' die Funktionen des Dienstes D in Anspruch nimmt und mit PE_0 über ihren Nachbarn PE' kommuniziere. Auf diese Weise erhält man die Gesamtmenge der Schnittstellenaufträge, die an der Schnittstelle zwischen PE und PE_0 ausgetauscht werden können:

(1) PE_0 übergibt an PE
- (a) die Kommunikationsprimitiven: OPEN, CLOSE und SEND-DE;
- (b) die Rückmeldung: REC-DE resp.

(2) PE übergibt an PE_0
- (a) die Kommunikationsprimitive: REC-DE;
- (b) die Rückmeldungen: OPEN resp., CLOSE resp. und SEND-DE resp.

Das für die Schnittstelle zwischen PE und PE_0 Gesagte gilt natürlich für jede der Schnittstellen zwischen PE und irgendeinem seiner Nachbarn aus der Menge NBHS(PE).

Um die Schnittstelle zwischen PE und seinen Nachbarn der Menge NBTS(PE) zu detaillieren, setzen wir iterativ voraus, daß die Protokolleinheit PE zur Kommunikation mit PE' einen Rechnernetzdienst D_U der Protokollschicht S_{i-1} verwende und daß der Dienst D_U sämtliche Funktionen des Dienstes D enthalte. Diese - in Abb. 3.6. dargestellte - Erweiterung zeigt einerseits auf, welche Arten von Schnittstellenaufträgen die Protokolleinheit PE von ihren Nachbarn entgegenzunehmen und zu bearbeiten hat und gibt andererseits die Menge der Schnittstellenaufträge an, die durch die Protokolleinheit PE erzeugt und an Nachbarn übergeben werden.

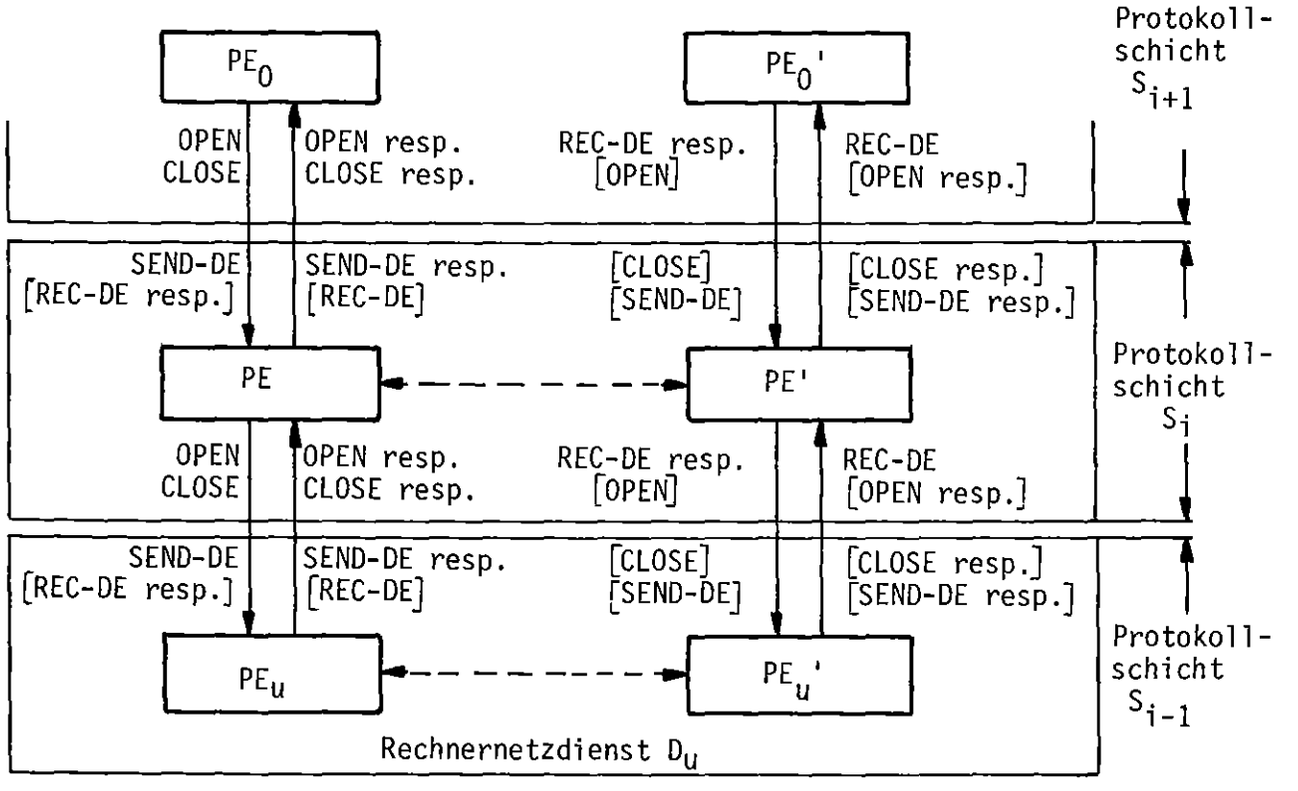

Abb. 3.6.: Mit dem Grundmodell einer Protokolleinheit in Zusammenhang stehende Schnittstellenaufträge
[Schnittstellenaufträge ohne Klammern zur Bereitstellung des Dienstes D für PE_0; Schnittstellenaufträge in Klammern zur Bereitstellung von D für PE_0']

Somit sind für das Grundmodell einer Protokolleinheit PE die Menge X der Eingabesymbole sowie die Menge Y der Ausgabesymbole bekannt, falls PE als sequentieller Automat dargestellt wird (s. Abschnitt 3.1.1.); und zwar gilt:

X = Y = {OPEN, OPEN resp., SEND-DE, SEND-DE resp., REC-DE, REC-DE resp., CLOSE, CLOSE resp.}.

Ehe wir nun die prinzipielle Vorgehensweise bei der Bearbeitung der Schnittstellenaufträge durch die Protokolleinheit PE aufzeigen können, ist zu untersuchen, in welcher Weise die Kommunikationsprotokolle strukturiert sein müssen, die für eine Abwicklung des Rechnernetzdienstes D geeignet sind. Hierzu wollen wir eine Klasse von Basisprotokollen einführen, die nicht nur für die Realisierung des Dienstes D, sondern auch für sonstige Rechnernetzdienste als Grundlage dienen können.

Die nachstehenden Anforderungen an die Basisprotokolle resultieren direkt aus den Charakteristika des Rechnernetzdienstes D:

(1) Existenz verschiedener Kommunikationsbeziehungen:
Da wir zugelassen haben, daß die Protokolleinheit PE die Funktionen F(D) gezielt durchführen kann für jeden der ihr bekannten Benutzer des Dienstes D (einer höheren Protokollschicht) und zu jedem ihrer Korrespondenten, ist es notwendig, daß auch der Zustand des Kommunikationsprotokolls in PE geführt wird für jeden bekannten Benutzer und für jeden Korrespondenten. (Die Menge der Benutzer, die der Protokolleinheit PE bekannt sind, kann z.B. mit der Menge NBHS(PE) zusammenfallen, was wir im folgenden ohne Beschränkung der Allgemeinheit voraussetzen wollen). Bezeichnen wir eine Kommunikationsbeziehung, die durch die Kombination des aktiven Benutzers und des angesprochenen Korrespondenten eindeutig festgelegt wird, als logische Verbindung, dann unterscheidet die Protokolleinheit PE im allgemeinen Fall somit |NBHS(PE)| · |KORR(PE)| logische Verbindungen, wobei |NBHS(PE)| bzw. |KORR(PE)| für die Anzahl der Elemente der Menge NBHS(PE) bzw. KORR(PE) steht. (In Sonderfällen ist |NBHS(PE)| = 1 und/oder |KORR(PE)| = 1 möglich).

(2) Protokollzustände und Protokollzustandsübergänge:
Basisprotokolle sollten die vier nachfolgenden elementaren Protokollzustände beinhalten, die einer Protokolleinheit PE für jede logische Verbindung - z.B. für die Protokolleinheit PE_0 als Benutzer des Dienstes D und für PE' (bzw. PE_0') als angesprochenen Korrespondenten der Menge KORR(PE) (bzw. KORR(PE_0)) - bekannt sind:

- Ruhezustand: keine logische Verbindung innerhalb der Protokollschicht S_i existiert zu dem Korrespondenten PE' für den Nachbarn PE_0;
- Initialisierungsphase (I-Phase): der Aufbau einer logischen Verbindung innerhalb der Protokollschicht S_i zu dem Korrespondenten PE' für den Nachbarn PE_0 ist im Gange;
- Datenaustauschphase (D-Phase): eine logische Verbindung innerhalb der Protokollschicht S_i zu dem Korrespondenten PE' für den Nachbarn PE_0 existiert, über die die Protokolleinheit PE_0 Dateneinheiten an PE_0' senden und/oder von PE_0' empfangen kann;
- Terminierungsphase (T-Phase): der Abbau der existierenden logischen Verbindung innerhalb der Protokollschicht S_i zu dem Korrespondenten PE' für den Nachbarn PE_0 ist im Gange.

Diese Protokollzustände werden sukzessive durchlaufen, so daß sich das in Abb. 3.7. dargestellte Zustandsübergangsdiagramm ergibt. Dabei ist zu berücksichtigen, daß für spezielle Kommunikationsprotokolle ggf. die Hinzunahme weiterer Protokollzustände erforderlich ist (z.B. Einbeziehung von Zuständen zur Fehlerbehandlung) und daß andererseits die elementaren Protokollzustände (insbesondere die Datenaustauschphase) in eine Reihe von Teilzuständen zerfallen können. Darüber hinaus lassen wir zu, daß die Initialisierungs- und die Terminierungsphase entfallen (z.B. bei Kommunikationsprotokollen, die nicht verbindungsorientiert sind).

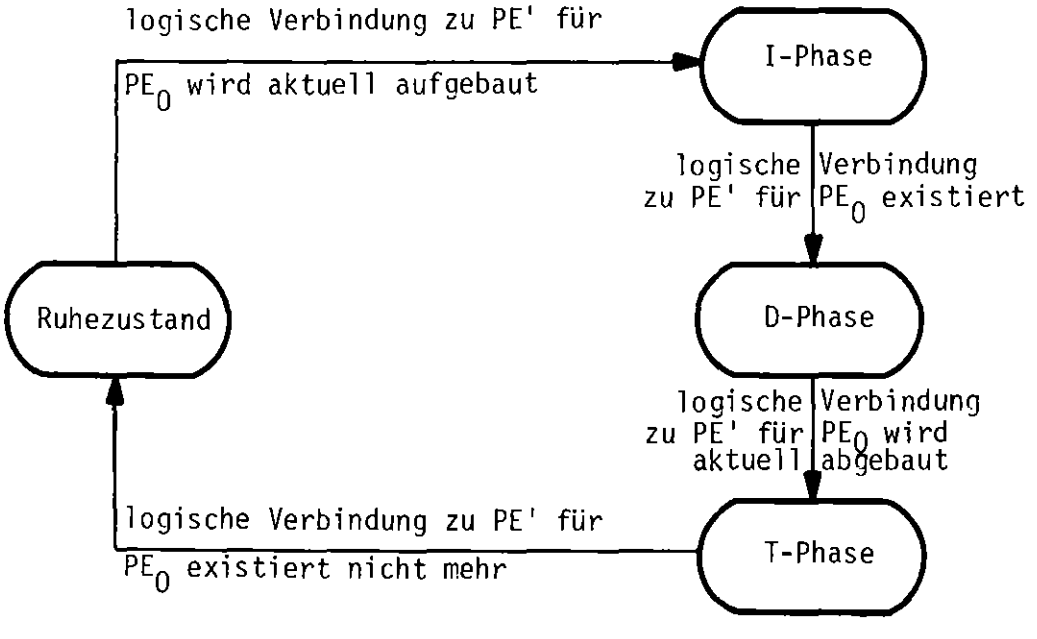

Abb. 3.7.: Zustandsübergangsdiagramm für Basisprotokolle (Grobstruktur)

Weitere Details hinsichtlich der Strukturierung der Basisprotokolle sind ersichtlich aus der Art der Abarbeitung der Schnittstellenaufträge durch die Protokolleinheit PE, auf die im folgenden eingegangen werden soll. Eine Reaktion der beiden Protokollmoduln innerhalb der Protokolleinheit PE hat zu erfolgen auf sämtliche Schnittstellenaufträge der Menge X (s.o.). Es sei vorausgesetzt, daß die Zuteilung der Schnittstellenaufträge der Menge X auf die Protokollmoduln dergestalt sei, daß

(a) durch den Sendeprotokollmodul die Schnittstellenaufträge:
OPEN, OPEN resp., SEND-DE, SEND-DE resp., CLOSE, CLOSE resp.
zu bearbeiten sind;

(b) durch den Empfangsprotokollmodul die Schnittstellenaufträge:
REC-DE, REC-DE resp.
zu bearbeiten sind.

Durch jeden Schnittstellenauftrag wird genau eine logische Verbindung angesprochen. Tab. 3.8. zeigt die grundsätzlichen Reaktionen des Sendeprotokollmoduls bei Erhalt der Schnittstellenaufträge OPEN, OPEN resp., CLOSE und CLOSE resp. in Abhängigkeit des Protokollzustands, in welchem sich die angesprochene logische Verbindung befindet.

Was die prinzipiellen Reaktionen der Protokollmoduln auf Schnittstellenaufträge anbelangt, die den Austausch von Dateneinheiten betreffen (d.h. Bearbeitung von SEND-DE und SEND-DE resp. im Sendeprotokollmodul sowie von REC-DE und REC-DE resp. im Empfangsprotokollmodul), wollen wir uns auf eine Zusammenfassung der wichtigsten elementaren Aufgaben einer Protokolleinheit während der Datenaustauschphase beschränken, die im Grundmodell auf eine generelle Art und Weise bewältigt werden:

(1) Korrespondentenauswahl:
Erhält die Protokolleinheit PE einen Schnittstellenauftrag "SEND DATAUNIT (DE, PE_0, PE_0')", so ist primär festzustellen, über welchen Korrespondenten PE' ϵ KORR(PE) die Protokolleinheit PE_0' erreicht werden kann. Die Bestimmung des Korrespondenten PE' erfolgt im vorliegenden Grundmodell einer Protokolleinheit PE unter Berücksichtigung

- der Adresse der Protokolleinheit PE_0';
- der topologischen Struktur des Rechnernetzes;
- sog. "Routing-Algorithmen", die bei Existenz von Alternativwegen eine Wegeauswahl erlauben.

Zustand der logischen Verbindung / Typ des zu bearbeitenden Auftrages	Ruhezustand	I-Phase	D-Phase	T-Phase
OPEN (von PE_0)	- OPEN an PE_u - Transition in I-Phase - Warten auf OPEN resp. von PE_u	wait	- OPEN resp. an PE_0	err
OPEN resp. (von PE_u)	err	- Initialisierung mit PE' (Austausch von Kontrollinformation) - Warten auf Reaktion von PE'	err	err
CLOSE (von PE_0)	- CLOSE resp. an PE_0	err	- Terminierung mit PE' (Austausch von Kontrollinformation) - Transition in T-Phase - Warten auf Reaktion von PE'	wait
CLOSE resp. (von PE_u)	err	err	err	- Transition in Ruhezustand

Tab. 3.8.: Prinzipielle Reaktion eines Sendeprotokollmoduls (Grundmodell) auf Schnittstellenaufträge in Abhängigkeit des aktuellen Protokollzustands [err: Fehlermeldung; wait: sofortige Bearbeitung des betreffenden Schnittstellenauftrags nicht möglich]

(2) Aufbereitung von Dateneinheiten:
Die durch den Nachbarn PE_0 übergebene Dateneinheit DE ist in der Protokolleinheit PE für die Übertragung aufzubereiten (z.B. mit Kontrollinformation KI für PE' zu versehen). Neben der Erstellung von Kontrollinformation kann bei der Aufbereitung von DE eine der beiden zusätzlichen Aufgaben anfallen:

- Fragmentierung/Reassemblierung:
  Falls der angesprochene Nachbar in der Protokollschicht S_{i-1} nur Dateneinheiten einer limitierten Länge akzeptiert und die Dateneinheit DE $\cup$ KI diese Länge übersteigt, so ist eine Fragmentierung (Zerstückelung) der Dateneinheit DE durchzuführen. Jedes der entstehenden Fragmente (Teilstücke) der Dateneinheit DE ist mit Kontrollinformation zu versehen und separat zu übertragen. Beim Empfänger PE' sind die einzelnen Fragmente wieder zu der vollständigen Dateneinheit DE zu reassemblieren.
  Das vorliegende Grundmodell einer Protokolleinheit enthält die Fragmentierung optional; zum Zwecke der Reassemblierung führen die Fragmente einer Dateneinheit eine laufende Nummer mit sich, wobei erstes und letztes Fragment einer Dateneinheit besonders gekennzeichnet sind.

- Blockbildung:
  Zur Steigerung der Auslastung eines Übertragungsmediums sehen existierende Rechnernetzdienste zum Teil die Möglichkeit vor, mehrere Dateneinheiten (der Protokollschicht S_{i+1}), die in einer Protokolleinheit PE auf den Transport zu demselben Korrespondenten PE' ϵ KORR(PE) warten, in ein und derselben Dateneinheit (der Protokollschicht S_i) zu befördern. Trivialerweise obliegt der Protokolleinheit PE' die Aufgabe, die ursprünglichen Dateneinheiten (der Protokollschicht S_{i+1}) wiederherzustellen. Das vorliegende Grundmodell enthält die Blockbildung optional.

(3) Nachbarauswahl:
Falls die Protokollschicht S_i nicht die niedrigste Schicht der Protokollhierarchie darstellt, ist im Anschluß an die Korrespondenten - eine Nachbarauswahl zu treffen, die feststellt, an welchen Nachbarn die Protokolleinheit PE die aufbereitete(n) Dateneinheit(en) zu übergeben hat. Diese Auswahl erfolgt im vorliegenden Grundmodell ausschließlich unter Berücksichtigung der Adresse der Protokolleinheit PE'.

(4) Flußkontrolle/Quittungsverkehr:

Beim Datenaustausch zwischen zwei korrespondierenden Protokolleinheiten obliegt es der Flußkontrolle, dafür zu sorgen, daß ein Sender PE seinem Korrespondenten PE' (Empfänger) nicht in zu rascher Folge Dateneinheiten übergibt; denn es ist u.a. aus Effizienzgründen zu vermeiden, daß der Empfänger Dateneinheiten erhält, für die eine sofortige Bearbeitung oder eine Zwischenspeicherung zwecks späterer Bearbeitung nicht möglich sind.
Bei einer Vielzahl existierender Kommunikationsprotokolle basiert die Flußkontrolle auf einem sog. Fenstermechanismus /s. CERF 76/, welcher dergestalt arbeitet, daß der Sender zu jedem Zeitpunkt einen maximalen Sendekredit - die Fenstergröße W_S - besitzt. Eine Fenstergröße von W_S besagt, daß der Sender maximal W_S Dateneinheiten an den Empfänger absenden darf, ehe er auf eine "Quittung" (s.u.) zu warten hat.
Im vorliegenden Grundmodell einer Protokolleinheit PE ist ein Sendekredit vorgesehen für jeden der Korrespondenten PE' $\in$ KORR(PE) und zwar für jeden der zu unterscheidenden Benutzer (d.h. für jede der bekannten logischen Verbindungen).

Bemerkung: Die im Grundmodell vorgesehene Regelung der Flußkontrolle ist optional; sie entfällt z.B. falls $W_S = \infty$.

Wie die bisherigen Ausführungen über die Flußkontrolle bereits aufzeigen, ergibt sich die Notwendigkeit, daß der Empfänger den Erhalt von Dateneinheiten dem Sender durch Übergabe einer "Quittung" (=Kontrollinformation eines speziellen Typs) bestätigt. Wir setzen voraus, daß das Absenden von Quittungen ebenfalls nach einem Fenstermechanismus erfolgt, welcher dergestalt arbeitet, daß der Empfänger zu jedem Zeitpunkt einen maximalen Empfangskredit - die Fenstergröße W_E - besitzt. Die Fenstergröße W_E besagt, daß der Empfänger maximal W_E Dateneinheiten von dem Sender empfangen darf, ehe er eine Quittung zu generieren hat.
Im vorliegenden Grundmodell einer Protokolleinheit PE ist ein Empfangskredit vorgesehen pro bekannter logischer Verbindung.

Bemerkungen:
- Der im Grundmodell vorgesehene Ablauf des Quittungsverkehrs ist optional; die Erzeugung von Quittungen entfällt z.B. falls $W_E = \infty$.
- Zur Vereinfachung setzt das Grundmodell ein fehlerfreies Übertragungsmedium zwischen Sender und Empfänger voraus.
- Zur Vermeidung von Verklemmungszuständen (Deadlocks) ist pro logischer Verbindung vorauszusetzen, daß $W_S \geq W_E$.

(5) Speicherverwaltung:
Zum Zwischenspeichern von zu übertragenden und/oder entgegenzunehmenden Dateneinheiten kann einer Protokolleinheit PE ein fester Speicherbereich konstanter Größe zugeordnet sein. Dies impliziert, daß innerhalb von PE eine Speicherverwaltung durchzuführen ist.
Das vorliegende Grundmodell sieht eine getrennte Speicherverwaltung für Sende- und Empfangsprotokollmoduln vor, die allerdings annulliert werden kann und sich ggf. durch eine geänderte Art der Speicherverwaltung (z.B. globale Speicherverwaltung in einem Rechnernetzknoten) ersetzen läßt.

Die obigen Ausführungen zeigen auf, daß das Grundmodell einer Protokolleinheit PE u.a. Parameter zu enthalten hat zur

- Kennzeichnung der Mengen NBTS(PE), NBHS(PE) und KORR(PE);
- Vorgabe der Protokollcharakteristika:
  - o Charakterisierung der durch PE zu erzeugenden Kontrollinformation (z.B. hinsichtlich Länge);
  - o maximale Länge der durch PE akzeptierten Dateneinheiten;
  - o Fenstergrößen W_S bzw. W_E (pro logischer Verbindung) zur Steuerung der Flußkontrolle bzw. des Quittungsverkehrs;
- Definition der Eigenschaften der Protokollmoduln:
  - o Puffergrößen (für das Zwischenspeichern abzusendender bzw. zu empfangender Dateneinheiten);
  - o maximale Längen und Abarbeitungsstrategien für die Warteschlangen SATQ und EATQ;
  - o Eigenschaften der Schnittstellen zu den Nachbarn;
  - o zeitliche Dauer für die Bearbeitung eines Schnittstellenauftrags (in Abhängigkeit des Auftragstyps und der Länge der ggf. in die Bearbeitung involvierten Dateneinheit).

3.1.4. Verbindung zwischen Protokolleinheiten

Um aus dem vorgestellten Modell für Protokolleinheiten eine vollständige Protokollhierarchie aufbauen zu können, sind die expliziten Interaktionen zwischen Protokollmoduln, Timeoutkontrollinstanzen sowie Benutzerprogrammen zu modellieren, die insbesondere aus dem gegenseitigen Austausch von Schnittstellenaufträgen resultieren.

Im vorliegenden Rechnernetzmodell sind zwei Gruppen von expliziten Interaktionen zwischen organisatorischen Einheiten vorgesehen:

(1) Direkte Verbindungen zwischen organisatorischen Einheiten in demselben Rechnernetzknoten:

In diese Kategorie fallen die Interaktionen:

(a) zwischen den Sendeprotokollmoduln (oder den Empfangsprotokollmoduln) zweier hierarchisch benachbarter Protokolleinheiten, die aus der Übergabe externer Schnittstellenaufträge und evtl. der Weitergabe von Dateneinheiten resultieren;
(b) zwischen dem Sende- und Empfangsprotokollmodul innerhalb einer Protokolleinheit, die aus der Übergabe interner Schnittstellenaufträge resultieren;
(c) zwischen der Timeoutkontrollinstanz und dem Sendeprotokollmodul (oder dem Empfangsprotokollmodul) innerhalb einer Protokolleinheit, die aus der Übergabe interner Schnittstellenaufträge resultieren;
(d) zwischen einem Benutzerprogramm und einem Sendeprotokollmodul, z.B. bei der Inanspruchnahme eines Rechnernetzdienstes durch das Benutzerprogramm mittels Übergabe eines Schnittstellenauftrags und evtl. einer (abzusendenden) Dateneinheit;
(e) zwischen einem Empfangsprotokollmodul und einem Benutzerprogramm, z.B. bei der Bereitstellung der Funktion eines Rechnernetzdienstes durch den Empfangsprotokollmodul mittels Übergabe eines Schnittstellenauftrags und evtl. einer (zu empfangenden) Dateneinheit.

Sämtliche expliziten Interaktionen (Verbindungen) dieser Gruppe sollen auf identische Art und Weise modelliert werden; und zwar geht das Modell davon aus, daß die zwischen den interagierenden organisatorischen Einheiten auszutauschenden Schnittstellenaufträge (und Dateneinheiten) direkt, d.h. ohne Verzögerung, in die entsprechende Warteschlange des Empfängers übergeben werden. Für die Entscheidung, ob die Übergabe eines Schnittstellenauftrags aktuell erfolgen kann, ist der Momentanzustand der Schnittstelle ausschlaggebend.

(2) Verbindung zwischen Korrespondenten der niedrigsten modellierten Protokollschicht über einen Rechnernetzdienst:

Für diesen Verbindungstyp genügt das für die direkten Verbindungen der ersten Gruppe zugrundegelegte Modell i.a. nicht. Um dies zu erläutern, sei PE_1 eine Protokolleinheit in dem Rechnernetzknoten DVS_1 und $PE_2 \in KORR(PE_1)$ eine Pro-

tokolleinheit im Rechnernetzknoten DVS_2; ferner seien PE_1 und PE_2 Elemente der Protokollschicht S_1, d.h. der niedrigsten modellierten Protokollschicht, und D bezeichne den Rechnernetzdienst, unter dessen Benutzung PE_1 und PE_2 kommunizieren (Abb. 3.9.).

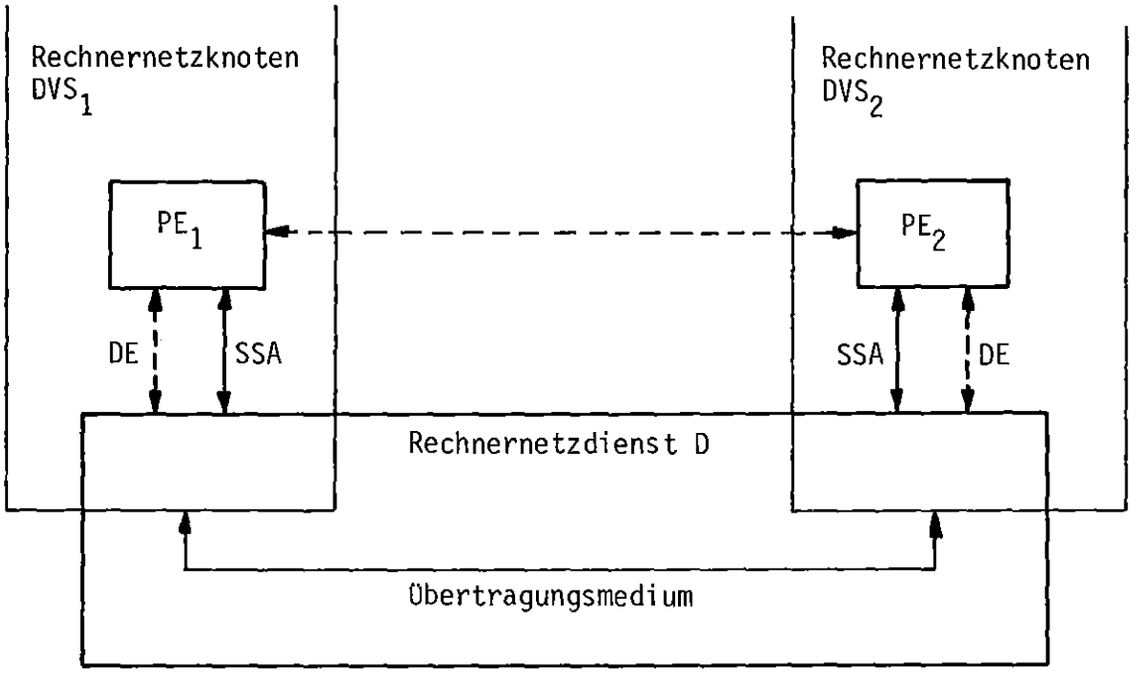

Abb. 3.9.: Kommunikation zwischen Korrespondenten der Protokollschicht S_1 (Notation s. Abb. 3.1.)

Für die Kommunikation zwischen den Protokolleinheiten PE_1 und PE_2 fungiert der Rechnernetzdienst D als (logischer oder physikalischer) Übertragungskanal. Die Suche nach einem Modell für die Verbindung zwischen PE_1 und PE_2 ist somit äquivalent zur Suche nach einem Modell für den Übertragungskanal, den der Dienst D aus der Sicht von PE_1 und PE_2 darstellt.

Der Rechnernetzdienst D beinhaltet indes einerseits die gesamte Kommunikationssoftware derjenigen Protokollschichten, die im Modell der Protokollhierarchie nicht detailliert wurden, sowie andererseits die Verbindung zwischen DVS_1 und DVS_2 (Übertragungsmedium). Es kann nun jedoch nicht Aufgabe des Modells für den Übertragungskanal sein, Details der Kommunikationssoftware zu berücksichtigen, auf welcher der Dienst D basiert; dies ließe sich exakter durch Hinzunahme weiterer Protokollschichten in das Modell der Protokollhierarchie erreichen. Aus diesem Grunde wollen wir es als vorrangiges Ziel ansehen, daß das Modell für den Übertragungskanal in der Lage sein soll, die Eigenschaften

des Übertragungsmediums widerzuspiegeln.

Als Modell für den Übertragungskanal zwischen den Korrespondenten PE_1 und PE_2 wählen wir demzufolge das in Abb. 3.10. dargestellte Warteschlangenmodell. Es umfaßt zwei Bedienungsstationen - die Auftragsbearbeitungsinstanzen ABI_j, $j \in \{1,2\}$ - deren Aufgabe es ist, auf Schnittstellenaufträge zu reagieren, die durch die Protokolleinheiten PE_j in die Auftragswarteschlange $REQQ_j$ übergeben wurden. Die Reaktion der Auftragsbearbeitungsinstanz ABI_j bei der Bearbeitung eines Elementes SSA der Warteschlange $REQQ_j$ besteht u.a. darin, daß

(a) falls SSA den Transfer einer Dateneinheit von PE_j nach PE_k [k = 1 (bzw. 2) falls j = 2 (bzw. 1)] bewirkt:

- der zeitliche Overhead berücksichtigt wird, der im Rechnernetzknoten DVS_j durch den Rechnernetzdienst D entsteht;
- die durch den Auftrag SSA referenzierte und in DEQ_j gespeicherte Dateneinheit DE verzögert wird (Modellierung der Verweildauer von DE im Übertragungsmedium);
- der zeitliche Overhead berücksichtigt wird, der im Zielrechner DVS_k durch den Rechnernetzdienst D entsteht;
- ein Schnittstellenauftrag vom Typ "RECEIVE DATAUNIT (DE, PE_j, PE_k)" erzeugt wird und zusammen mit DE an den Empfangsprotokollmodul der Protokolleinheit PE_k übergeben wird;
- eine Rückmeldung für SSA generiert wird und an den Sendeprotokollmodul der Protokolleinheit PE_j übergeben wird;

(b) falls SSA einen sonstigen Schnittstellenauftrag darstellt:

- der zeitliche Overhead berücksichtigt wird, der im Rechnernetzknoten DVS_j durch den Rechnernetzdienst D entsteht;
- eine Rückmeldung für SSA generiert wird und an den Sendeprotokollmodul der Protokolleinheit PE_j übergeben wird.

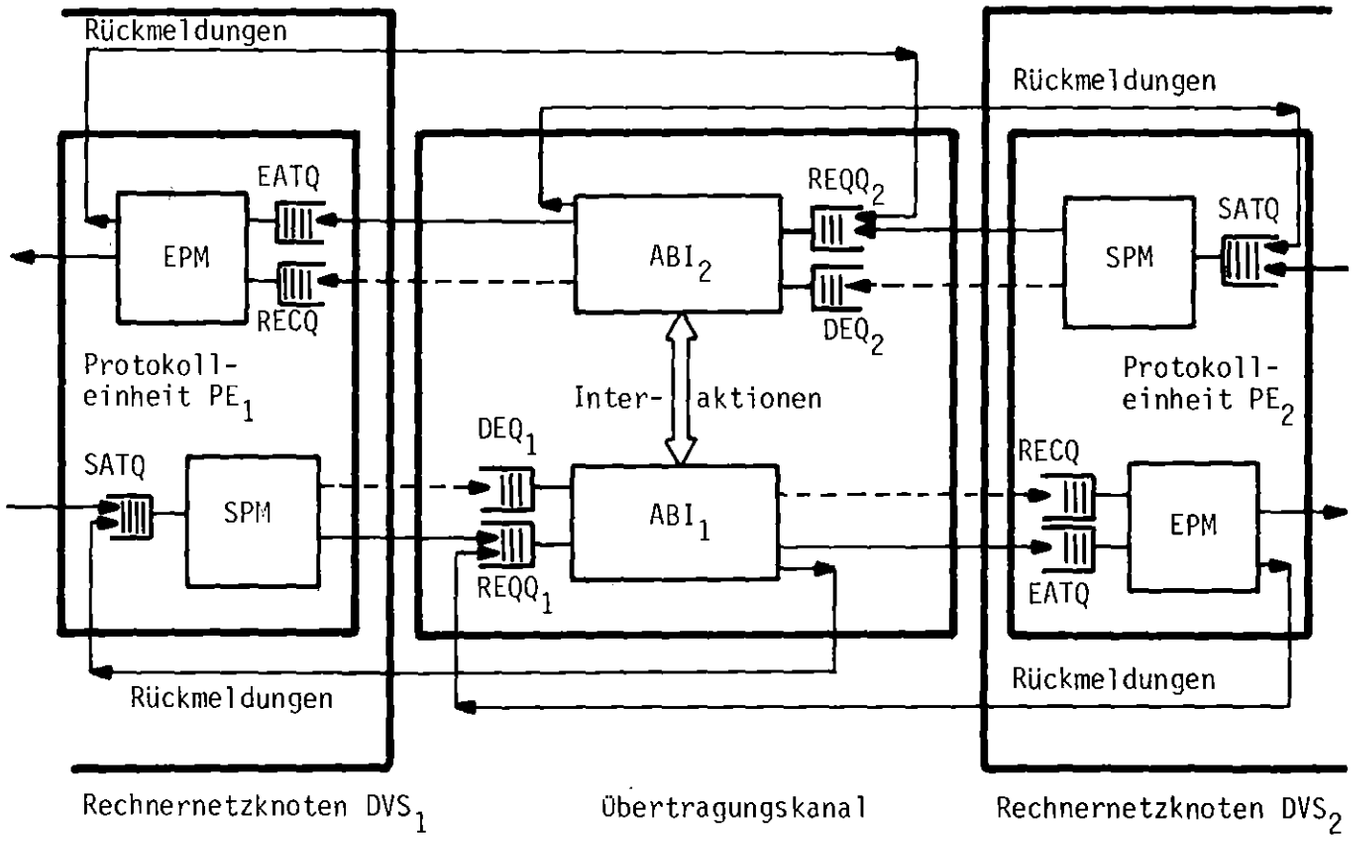

Abb. 3.10.: Warteschlangenmodell eines Übertragungskanals zwischen zwei Korrespondenten PE_1 und PE_2
[Notation: ——→ Austausch von Schnittstellenaufträgen;
– – → Austausch von Dateneinheiten]

Eine Eigenschaft, die die Leistungsfähigkeit eines Übertragungsmediums offenkundig stark beeinflußt, ist der Übertragungsmodus, d.h. die Betriebsart, in welcher das Übertragungsmedium arbeitet; man unterscheidet

(a) simplex-Übertragung [Richtungsbetrieb: Übertragung permanent nur in eine Richtung; kein Richtungswechsel]

(b) halbduplex-Übertragung [Wechselbetrieb: Übertragung temporär nur in eine Richtung; Richtungswechsel nach jeder Übertragung möglich]

(c) vollduplex-Übertragung [Gegenbetrieb: parallele Übertragung in beiden Richtungen].

In dem Modell eines Übertragungskanals wollen wir der Betriebsart des Übertragungsmediums Rechnung tragen, wodurch sich direkte Auswirkungen auf das Warteschlangenmodell der Abb. 3.10. ergeben. Der Modus des Übertragungskanals bestimmt in erster Linie die im Warteschlangenmodell angedeuteten Interaktionen zwischen ABI_1 und ABI_2: Ist der Übertragungsmodus vollduplex, so arbeiten ABI_1 und ABI_2 unabhängig voneinander; im Modus halbduplex haben sich ABI_1 und ABI_2 zu synchronisieren, da beide Bedienungsstationen nicht gleichzeitig aktiv sein können; bei simplex-Übertragungsmodus entfällt entweder ABI_1 oder ABI_2 mit den beiden zugeordneten Warteschlangen.

Um die interne Struktur des Modells eines Übertragungskanals als sequentiellen Automaten darzustellen, führen wir eine Menge Z = {Z0,Z1,Z2,Z3} von Zuständen eines Übertragungskanals ein, wobei

Z0: Ruhezustand;
Z1: ein Schnittstellenauftrag aus Warteschlange $REQQ_1$ befindet sich derzeit in Bearbeitung;
Z2: ein Schnittstellenauftrag aus Warteschlange $REQQ_2$ befindet sich derzeit in Bearbeitung;
Z3: Schnittstellenaufträge aus jeder der Warteschlangen $REQQ_1$ und $REQQ_2$ befinden sich derzeit in Bearbeitung.

Für die Beschreibung eines Übertragungskanals als sequentiellen Automaten reicht i.a. eine Teilmenge der eingeführten Menge von Zuständen aus. Die Menge Z der tatsächlich benötigten Zustände ist abhängig von dem Modus des zu modellierenden Übertragungskanals, und es ergibt sich insbesondere

(a1) für simplex-Übertragung (mit permanenter Sendeberechtigung im Rechnernetzknoten DVS_1):
Z = {Z0, Z1};

(a2) für simplex-Übertragung (mit permanenter Sendeberechtigung im Rechnernetzknoten DVS_2):
Z = {Z0, Z2};

(b) für halbduplex-Übertragung:
Z = {Z0, Z1, Z2};

(c) für vollduplex-Übertragung:
Z = {Z0, Z1, Z2, Z3}.

Das Zustandsübergangsdiagramm für einen Übertragungskanal, das der Abb. 3.11. entnommen werden kann, vereinfacht sich für Übertragungskanäle des Modus simplex oder halbduplex ebenfalls durch den Wegfall von Zuständen.

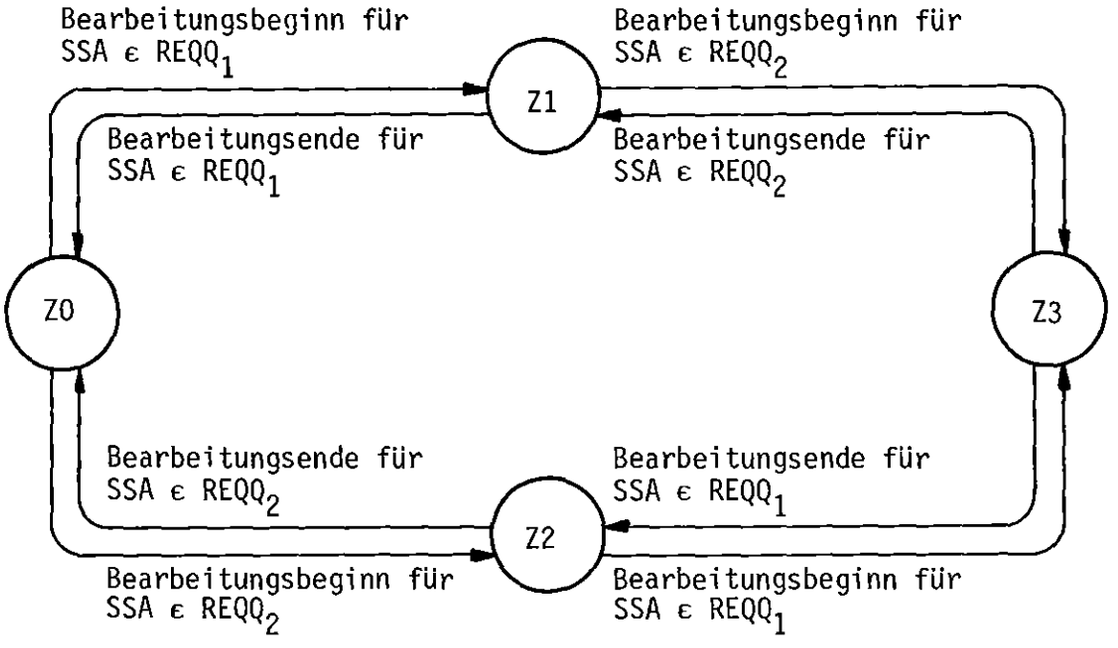

Abb. 3.11.: Zustandsübergangsdiagramm eines Übertragungskanals

Was den Zeitpunkt für den Beginn der Bearbeitung eines Schnittstellenauftrags anbelangt, so ergibt sich in Abhängigkeit des Modus des Übertragungskanals, daß bei Simplex- und Halbduplex-Modus nur im Zustand Z0 mit der Bearbeitung des nächsten Schnittstellenauftrags begonnen werden kann, wohingegen bei Vollduplex-Modus die Bearbeitung der Warteschlange $REQQ_1$ (bzw. $REQQ_2$) darüber hinaus auch im Zustand Z2 (bzw. Z1) begonnen werden kann.

Die Parameter des Modells eines Übertragungskanals zwischen DVS_1 und DVS_2 umfassen:

- den Übertragungsmodus,
- die Wahrscheinlichkeit für den vollständigen Ausfall des Übertragungsmediums,
- die Wahrscheinlichkeit für das Auftreten von Übertragungsfehlern (gegeben durch die Bitfehlerrate),

- die maximale Anzahl von erlaubten Übertragungswiederholungen bei Auftreten von Übertragungsfehlern,
- die maximale Länge der durch den Übertragungskanal akzeptierten Dateneinheiten,
- den zeitlichen Overhead (im Sende- und im Zielrechner) in Abhängigkeit des Typs der zu bearbeitenden Schnittstellenaufträge,
- die Übertragungskapazität des Übertragungsmediums.

Das Modell eines Übertragungskanals wurde zwar für Punkt-zu-Punkt-Verbindungen eingeführt, dennoch läßt es sich - durch Überlagerung mehrerer Übertragungskanäle - auch zur Modellierung von Mehrpunktverbindungen heranziehen.

3.2. Modelle für Dateneinheiten und Kommunikationsprimitive

Nachdem nun die grundlegenden Modelle bereitgestellt wurden, um schichtenförmig strukturierte Protokollhierarchien nachzubilden, wollen wir uns der Frage zuwenden, wie die zwischen Protokolleinheiten ausgetauschten Informationselemente (Dateneinheiten und Schnittstellenaufträge) modelliert werden können.

(1) Dateneinheiten:

Zur Definition des Modells einer Dateneinheit gehen wir wiederum aus von einer Protokollhierarchie mit n Protokollschichten $S_1,\ldots,S_n$. Des weiteren seien sämtliche kommunizierenden Benutzerprogramme zu einer Schicht S_{n+1} zusammengefaßt. Obwohl wir den Begriff "Dateneinheit" als Oberbegriff der zwischen beliebigen Korrespondenten im Rechnernetz ausgetauschten Informationselemente führen, darf dies nicht darüber hinwegtäuschen, daß eine Protokollabhängigkeit der Dateneinheitenstrukturen vorliegt, die wir im Dateneinheitenmodell zu berücksichtigen haben. Da Dateneinheiten innerhalb jeder Protokolleinheit umgeformt (z.B. fragmentiert, reassembliert) werden, ist eine Beschreibung der generellen Struktur einer Dateneinheit nur sinnvoll zum Zeitpunkt des Passierens einer Schnittstelle zwischen benachbarten Protokollschichten.

Die Grobstruktur einer Dateneinheit beim Eintreffen (bzw. Austreten) in die Protokollschicht S_i wurde bereits in Kap. 2 eingeführt als Menge von Kontrollinformation $KI = KI(S_i)$ zur Interpretation innerhalb von S_i und einem ggf. leeren Datenfeld für höhere Schichten. Auch für das Modell von Dateneinheiten wird diese Struktur übernommen, wobei allerdings eine etwas detailliertere Darstellung angestrebt wird.

Für $j \in \{1,\ldots,n+1\}$ beliebig, jedoch fest, bezeichne DE_j das Modell einer Dateneinheit, die zwischen zwei Korrespondenten PE_S - dem Sender - und PE_E - dem Empfänger - der Protokollschicht S_j ausgetauscht wird und die gerade die Protokolleinheit PE_S in Richtung eines Nachbarn $\in NBTS(PE_S)$ verläßt oder die Protokolleinheit PE_E über einen Nachbarn $\in NBTS(PE_E)$ erreicht.

Wir definieren nun rekursiv

(a) <u>für j = n+1</u>:

DE_{n+1}: = BD, wobei BD <u>Benutzerdaten</u> bezeichnet, d.h. Daten, die zwischen Benutzerprogrammen ausgetauscht werden.

Bei der Modellierung ist zumeist die interne Struktur (z.B. Bitmuster, Codierung) von Benutzerdaten irrelevant, so daß eine Kennzeichnung der Daten in bezug auf ihre Länge und ihren Empfänger oft genügt.

(b) <u>für $j \leq n$</u>:

Es gibt drei mögliche Strukturen für die Dateneinheit DE_j:

(b1) Dateneinheit mit leerem Datenfeld:

DE_j: = $KI(S_j)$, wobei $KI(S_j)$ Kontrollinformation bezeichnet, d.h. Information, die durch PE_S generiert wurde und die in PE_E zu interpretieren ist.

Kontrollinformation steuert die Abwicklung des Kommunikationsprotokolls zwischen PE_S und PE_E. In existierenden Protokollimplementierungen ist die Kontrollinformation oft in einem Nachrichtenkopf, dem sog. "Header", enthalten.

<u>Modellparameter für Kontrollinformation</u> umfassen:

- die Charakterisierung der Sende-, Empfangsprotokolleinheit, der Protokollschicht und des Kommunikationsprotokolls;
- die Länge der Kontrollinformation;
- den Typ der Kontrollinformation (z.B. Quittung, Sendeaufforderung); etc.

(b2) Dateneinheit bei Fragmentierung:

DE_j: = $KI(S_j) \cup F(DE_{j1})$, $j < j1 \leq n+1$, wobei $F(DE_{j1})$ ein Fragment (Teilstück) einer Dateneinheit DE_{j1} bezeichnet, die von einem Nachbarn $\in NBHS(PE_S)$ an die Protokolleinheit PE_S übergeben wurde.

Fragmentierung durch die Protokolleinheit PE_S ist genau dann notwendig, wenn die von einem Nachbarn ϵ NBHS(PE_S) erhaltene Dateneinheit DE_{j1} vereinigt mit der zu erstellenden Kontrollinformation $KI(S_j)$ wegen zu großer Länge von dem Nachbarn ϵ NBTS(PE_S) nicht akzeptiert werden würde.

(b3) Dateneinheit bei Blockbildung:

DE_j: = $KI(S_j) \cup DE_{j1} \cup DE_{j2} \cup \ldots \cup DE_{jm}$, $j < j1,\ldots,jm \leq n+1$, wobei $DE_{j1},\ldots,DE_{jm}$ Dateneinheiten bezeichnen, die von einem oder mehreren Nachbarn ϵ NBHS(PE_S) an die Protokolleinheit PE_S übergeben wurden und deren Weg über denselben Korrespondenten $PE_E \epsilon$ KORR(PE_S) führt.

Blockbildung tritt u.a. auf, wenn Dateneinheiten verschiedener Nachbarn zu einer neuen Dateneinheit zusammengefügt werden.

Bemerkung: Der Normalfall $DE_j = KI(S_j) \cup DE_{j1}$, $j < j1 \leq n+1$, ist als Spezialfall sowohl in (b2) als auch in (b3) enthalten.

Folgende Parameter einer Dateneinheit vervollständigen obiges Modell:

- Verweise auf die Sende- und Empfangsprotokolleinheit sowie auf den Zielrechner;
- Angabe der Gesamtlänge, der Länge des Datenfeldes, einer Priorität, etc.;
- Kennzeichnung der mitgeführten Information (Kontrollinformation, Benutzerdaten, Dateneinheitenfragmente, etc.).

(2) Schnittstellenaufträge:

(Schnittstellen-) Aufträge, die durch Sende-, Empfangsprotokollmoduln, Timeout-Kontrollinstanzen, Benutzerprogramme und Übertragungskanäle generiert und durch Sende-, Empfangsprotokollmoduln, Benutzerprogramme oder Übertragungskanäle bearbeitet werden, lassen sich als Datenstruktur mit den nachfolgenden Auftragsparametern modellieren:

- Kennzeichnung des Auftragstyps: es werden interne und externe Aufträge unterschieden (s. Modell für Protokolleinheiten), wobei für externe Schnittstellenaufträge zusätzlich zwischen Kommunikationsprimitiven und Rückmeldungen differenziert wird;

(a) Beispiele für Typen externer Schnittstellenaufträge (s. Basisprotokolle): die Kommunikationsprimitiven OPEN, SEND-DE, REC-DE, CLOSE und die entsprechenden Rückmeldungen;

(b) Beispiele für Typen interner Schnittstellenaufträge:
"SEND-CTRLINF(DE,PE')" - Schnittstellenauftrag, der besagt, daß die durch DE referenzierte Kontrollinformation an den Korrespondenten PE' zu übertragen ist;

"SEND-COPY(DE,PE')" - Schnittstellenauftrag, der besagt, daß die in der Warteschlange COPYQ wartende Dateneinheit DE an den Korrespondenten PE' zu übertragen ist (Übertragungswiederholung);

- Priorität zur Steuerung der Auswahl auf Bearbeitung wartender Aufträge;
- Verweis auf die auftragsgenerierende organisatorische Einheit;
- Verweis auf die organisatorische Einheit, der die Abarbeitung des Auftrags obliegt;
- Parameter, die nur für spezielle Auftragstypen von Relevanz sind:
  - o Verweis auf die in die Bearbeitung eines Auftrags involvierte Dateneinheit oder den zu adressierenden Korrespondenten (z.B. bei SEND-DE, SEND-CTRLINF oder SEND-COPY Schnittstellenaufträgen),
  - o Angabe über den Erfolg bei der Durchführung der Funktion eines Rechnernetzdienstes (z.B. bei Rückmeldungen).

3.3. Betriebsmittelzuteilung innerhalb eines Rechners

In unseren bisherigen Betrachtungen setzten wir uns u.a. auch mit der Modellierung der Interaktionen zwischen organisatorischen Einheiten innerhalb eines Rechnernetzknotens auseinander. Allerdings beschränkten wir uns bislang auf die Modellierung der expliziten Interaktionen, d.h. auf die konkrete Kommunikation zwischen organisatorischen Einheiten, die durch Austausch von Schnittstellenaufträgen und Dateneinheiten realisiert wird.

Neben den expliziten Interaktionen, deren Modellierung vollständig aufgezeigt wurde, ist für eine detaillierte Untersuchung der Effizienz von Rechnernetzen auch die Modellierung der impliziten Interaktionen zwischen organisatorischen Einheiten von Bedeutung. Diese impliziten Interaktionen resultieren aus der Konkurrenz von organisatorischen Einheiten um gemeinsam benötigte

(Hardware-) Rechnerbetriebsmittel (Prozessoren, Hauptspeicher, periphere Geräte). Die Zuweisung solcher Betriebsmittel erfolgt in existierenden Rechnern durch Betriebsmittelverwaltungsinstanzen. Die Strategien, die zur Optimierung der Betriebsmittelzuteilung an organisatorische Einheiten zugrundegelegt werden, dienen z.B. der Minimierung der Systemverweilzeiten bzw. der Maximierung des Durchsatzes von Benutzerprogrammen. Implizite Interaktionen existieren zwischen organisatorischen Einheiten desselben Rechnernetzknotens.

Zur Modellierung der Betriebsmittelvergabe, die im folgenden durchgeführt werden soll, benutzen wir nachstehende

Definition: Die organisatorischen Einheiten innerhalb eines Rechnernetzknotens, die die Nutzarbeit des Rechners leisten und an die Rechnerbetriebsmittel dynamisch zugewiesen werden, nennen wir Tasks. Als Nutzarbeit wird hierbei die Durchführung der Funktionen aufgefaßt, die entweder durch Rechnernetzbenutzer mittels eines Benutzerprogramms definiert wurden oder die der Realisierung von Rechnernetzdiensten dienen. Es sei vorausgesetzt, daß Tasks eine sequentielle Struktur besitzen.

Der hier eingeführte Taskbegriff entspricht dem in der Literatur üblichen Begriff des sequentiellen Prozesses /s. BRIN 73/. [Die Benutzung des Ausdrucks "Prozeß" in diesem Zusammenhang wurde vermieden, um nicht mit dem Prozeßbegriff der Programmiersprache SIMULA (s. Kap. 4) zu kollidieren.]

Im vorliegenden Rechnernetzmodell unterscheiden wir zwei Gruppen von Tasks:

(1) Benutzertasks zur Modellierung von Benutzerprogrammen,
(2) Protokolltasks, die der Ausführung der Funktionen eines Rechnernetzdienstes dienen.

Die Motivation für die unterschiedliche Modellierung für beide Typen von Tasks ist darin zu sehen, daß der für Protokolltasks benötigte Detaillierungsgrad bedeutend höher ist als für Benutzertasks: Zur Modellierung von Benutzertasks genügt im wesentlichen eine Charakterisierung der Sequenz und des Umfangs der angeforderten Betriebsmittel, wohingegen für Protokolltasks die interne Struktur, insbesondere die Art und Weise der Abwicklung des Kommunikationsprotokolls, eine wichtige Rolle spielt.

Diejenigen organisatorischen Einheiten, die in einem Rechnernetzknoten die Funktionen des Betriebssystems durchführen, werden im Rechnermodell nicht durch eine spezielle Gruppe von Tasks repräsentiert, da wir das Modell an dieser Stelle nicht unnötigerweise zu stark detaillieren wollen. Das Modell enthält dennoch zwei Möglichkeiten, dem Betriebsmittelbedarf für den Ablauf des Betriebssystems Rechnung zu tragen, und zwar einerseits indem die entsprechenden Betriebsmittel (insbesondere die CPU) den Benutzer- bzw. Protokolltasks als "Overhead" anteilsmäßig zugeschlagen werden oder andererseits - falls der Betriebsmittelbedarf des Betriebssystems präziser berücksichtigt werden soll - die organisatorischen Einheiten für die Realisierung von Betriebssystemfunktionen als permanente Benutzertasks modelliert werden.

3.3.1. Statische Beschreibung für Tasks

Ehe wir uns der Modellierung der Betriebsmittelvergabe an Tasks zuwenden, benötigen wir Modelle zur Beschreibung der internen Struktur von Tasks. Aus diesen Modellen muß sich insbesondere eine Aussage über Art und Umfang der sukzessive benötigten Rechnerbetriebsmittel ableiten lassen. Da die Struktur einer Task als sequentiell vorausgesetzt wird, ist die Formulierung einer Task als sequentieller Automat möglich.

(1) Benutzertasks

Die interne Struktur einer Benutzertask BT kann ausgehend von folgenden modellrelevanten, internen Zuständen dargestellt werden, die die verschiedenen Benutzertaskaktivitäten repräsentieren:

ZI1': Rechenanweisungen mit unterbrechbarer CPU-Belegung;
ZI1": Rechenanweisungen mit ununterbrechbarer CPU-Belegung;
ZI2 : Ein-/Ausgabe-Anweisungen → Belegung (lokaler) peripherer Geräte;
ZI3 : Inanspruchnahme eines Rechnernetzdienstes durch Übergabe eines Schnittstellenauftrags an eine (benachbarte) Protokolleinheit;
ZI4 : Warteaufruf, d.h. Erwarten eines speziellen Ereignisses zur Auflösung eines temporären Blockierungszustands (z.B. Erwarten von Benutzerdaten einer mit BT korrespondierenden Benutzertask).

Jedem der Zustände ist eine Menge von Rechnerbetriebsmitteln zugeordnet, die zur Durchführung der entsprechenden Aktivität benötigt werden. Die Auftrennung der Rechenanweisungen in unterbrechbare und nicht unterbrechbare Rechen-

anweisungen kann z.B. in Realzeitbetriebssystemen von Wichtigkeit sein. Die Zustandsübergänge zwischen Benutzertaskzuständen können Abb. 3.12. entnommen werden.

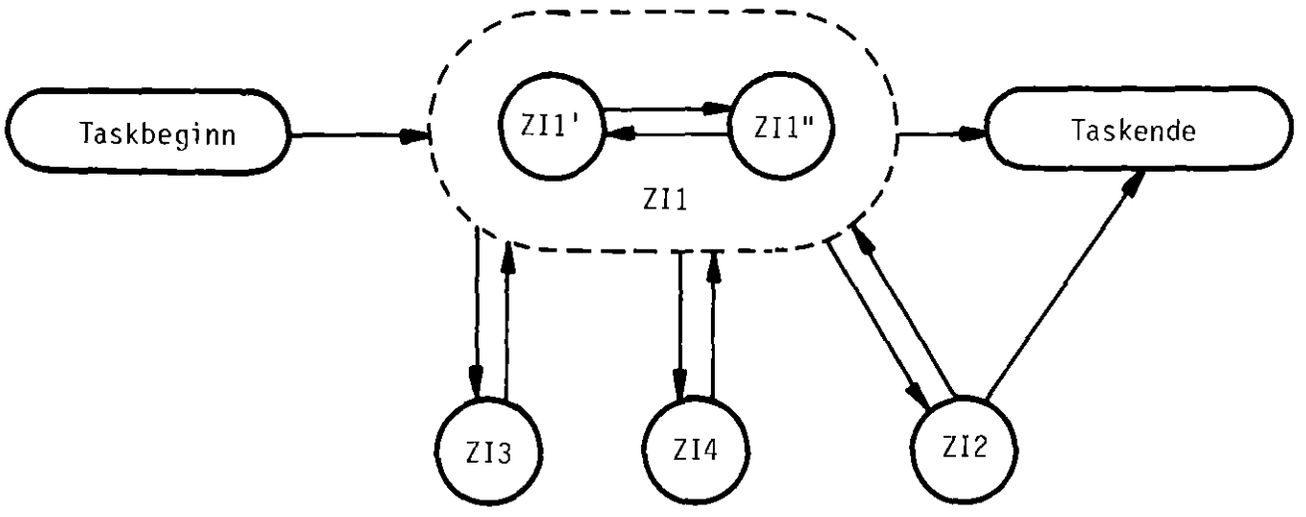

Abb. 3.12.: Zustandsübergangsdiagramm einer Benutzertask (statisches Modell)

Bei Verlassen eines Zustands wird die evtl. notwendig werdende Auswahl des Nachfolgezustands durch vorgegebene Verteilungen gesteuert. Zustandsübergänge können grundsätzlich nur dann ausgeführt werden, wenn die im Nachfolgezustand benötigten Rechnerbetriebsmittel zur Verfügung stehen.

Wichtige Parameter, die die statische Beschreibung einer Benutzertask vervollständigen, sind neben der Spezifikation der in den einzelnen Zuständen benötigten Betriebsmittel u.a. die Taskpriorität sowie Aussagen bzgl. der Unterbrechbarkeit in den internen Zuständen.

(2) Protokolltasks

Zur Abbildung von Protokollfunktionen auf Tasks können Protokolltasks mit Protokollmoduln gleichgesetzt werden. Dies impliziert, daß die interne Struktur einer Protokolltask durch die interne Struktur des entsprechenden Sende- bzw. Empfangsprotokollmoduls vorgegeben ist. Allerdings ist auch hier jeder der internen Zustände bzw. der Zustandsübergänge eines Protokollmoduls mit Angaben bzgl. Art und Umfang der benötigten Rechnerbetriebsmittel und bzgl. der Unterbrechbarkeit zu versehen.

Beispielsweise soll keine Unterbrechbarkeit für Protokollmoduln zugelassen werden während der Reaktion auf einen Schnittstellenauftrag (d.h. während der Transition Z2 → Z3 im Zustandsübergangsdiagramm für Protokollmoduln, s. Abb. 3.3.). Durch das Zurückstellen der Unterbrechungswünsche, die während der Reaktion auf einen Schnittstellenauftrag auftreten, kann erreicht werden, daß bei sämtlichen Protokollmodul-Unterbrechungen ein wohldefinierter Zustand der Protokolleinheit (und insbesondere auch des Kommunikationsprotokolls) vorliegt. Die obige Ununterbrechbarkeitsvoraussetzung ist in den meisten Fällen dadurch gerechtfertigt, daß

(a) Protokollmoduln i.a. höhere Priorität besitzen als Benutzertasks und daher ohnehin nur durch andere Protokollmoduln in ihrer CPU-Belegung verdrängt werden können, und
(b) der CPU-Zeitbedarf für die Reaktion auf einen Schnittstellenauftrag gering ist und daher Unterbrechungswünsche - wenngleich nicht sofort - so doch nur mit geringfügiger Verspätung im Modell berücksichtigt werden.

Die Begründung dafür, daß die Timeout-Kontrollinstanz als dritte Komponente innerhalb einer Protokolleinheit nicht als Protokolltask interpretiert wird, ist darin zu sehen, daß der gesamte Betriebsmittelbedarf für die Erzeugung und Auswertung von Timeouts den Protokollmoduln zugeordnet wird.

3.3.2. Dynamische Beschreibung für Tasks

Aus den Modellen zur Beschreibung des internen Ablaufs von Tasks lassen sich direkt Angaben bzgl. der Sequenz und des Umfangs der durch die Task sukzessive benötigten Betriebsmittel entnehmen. Durch die sequentielle Natur der Betriebsmittelvergabe ist es angebracht, auch das dynamische Verhalten einer Task während ihrer Ausführung durch einen sequentiellen Automaten zu modellieren /s. WOLF 78a/. Die dynamischen Zustände von Tasks können charakterisiert werden durch die aktuell allokierten und benötigten Betriebsmittel sowie durch den erreichten Status im internen Taskablauf (z.B. aktuell erreichter interner Zustand des statischen Taskmodells).

(1) Benutzertasks

Bei den für Benutzertasks relevanten Betriebsmitteln handelt es sich insbesondere um Hauptspeicher, CPU und periphere Geräte. Ein dynamisches Modell einer Benutzertask kann daher ausgehend von folgenden dynamischen Zuständen

erstellt werden (die auftretenden internen Benutzertaskzustände beziehen sich auf Abb. 3.12.):

ZD0: keine Betriebsmittel allokiert; warten auf Hauptspeicher; interner Benutzertaskzustand: Taskbeginn;

ZD1: Hauptspeicher allokiert; warten auf CPU; interner Taskzustand: gewünschte Transition nach ZI1, d.h. ZI1' bzw. ZI1";

ZD2: Hauptspeicher und CPU allokiert; aktiv bis zum Entzug der CPU (nur möglich, falls Taskzustand = ZI1'!) bzw. bis zur Beendigung der Rechenanweisungen; interner Taskzustand: ZI1' oder ZI1";

ZD3: Hauptspeicher allokiert; warten auf Eintritt eines speziellen Ereignisses; interner Taskzustand: ZI3 oder ZI4;

ZD4: Hauptspeicher allokiert; warten auf peripheres Gerät bzw. bereits Ein-/Ausgabe aktiv; interner Taskzustand: ZI2;

ZD5: keine Betriebsmittel allokiert; Benutzertask beendet; interner Zustand: Taskende.

Eine Vervollständigung der dynamischen Beschreibung einer Benutzertask BT wird durch das Zustandsübergangsdiagramm in Abb. 3.13. gegeben.

(Bemerkungen zur verwendeten Notation:

- ZI repräsentiert den internen Zustand der Task BT;
- Z_{next} bezeichnet den internen Zustand von BT, der als nächster erreicht werden soll;
- $ZI_j \rightarrow ZI_k$ kennzeichnet einen Übergang aus dem internen Taskzustand ZI_j in den Zustand ZI_k;
- ZI1 steht sowohl für den internen Zustand ZI1' als auch für ZI1";
- die Markierung der Pfeile in der Form α/β besagt, daß das Eintreffen des Ereignisses α die Änderung des dynamischen Zustands der Task BT auslöst und die durch β bezeichnete Änderung des internen Zustands von BT nach sich zieht.)

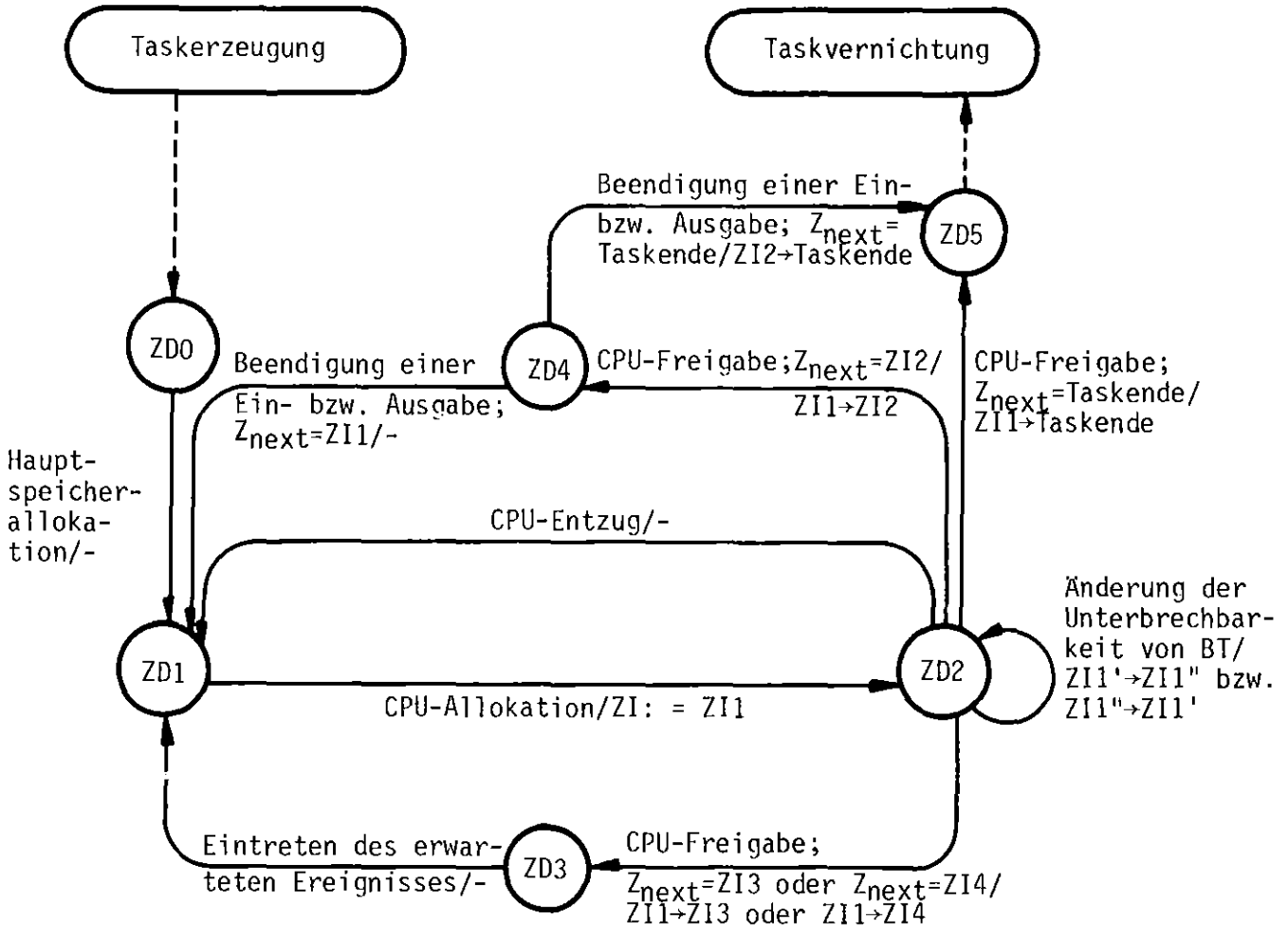

Abb. 3.13.: Zustandsübergangsdiagramm einer Benutzertask (dynamisches Modell)

(2) Protokolltasks

Im Gegensatz zu Benutzertasks nehmen wir - ohne Beschränkung der Allgemeinheit - Protokolltasks im folgenden als hauptspeicherresident an. Diese Voraussetzung wird durch eine Vielzahl existierender Implementierungen nahegelegt; sie impliziert, daß als einziges Betriebsmittel, um welches sich Protokolltasks bewerben, die CPU zu modellieren ist. Da Protokollmoduln üblicherweise im Rechner als permanente Tasks vorhanden sind, entfallen überdies sämtliche dynamischen Zustände, die sich auf Taskerzeugung bzw. -vernichtung beziehen. Somit vereinfacht sich das dynamische Modell von Protokolltasks gegenüber dem Benutzertaskmodell.

Unter Verwendung der internen Zustände von Protokollmoduln wie sie sich, nach entsprechender Modifikation aus Abb. 3.3. ergeben, lassen sich folgende dynamische Zustände für hauptspeicherresidente Protokolltasks definieren:

ZD1: Hauptspeicher allokiert; warten auf CPU; interner Protokollmodulzustand ≠ Z4 und Auftragswarteschlange nicht leer;

ZD2: Hauptspeicher und CPU allokiert; aktiv bis zu einem evtl. CPU-Entzug (CPU-Verdrängung kann nur bei Unterbrechbarkeit des aktuell erreichten internen Taskzustandes erfolgen) bzw. CPU-Freigabe bei Beendigung der eigenen Aktivität (CPU-Freigabe erfolgt bei Übergang in den internen Zustand Z4 sowie bei Transition nach Z1 und leerer Auftragswarteschlange); interner Protokollmodulzustand: Z1, Z2 oder Z3;

ZD3: Hauptspeicher allokiert; warten auf Eintreffen eines Auftrags; interner Protokollmodulzustand: Z4 oder Zustand Z1 (bei leerer Auftragswarteschlange).

Die Übergänge der dynamischen Zustände von Protokolltasks (s. Abb. 3.14.) entsprechen im wesentlichen der durch die Zustände ZD1, ZD2, ZD3 definierten Teilmenge des dynamischen Modells für Benutzertasks.

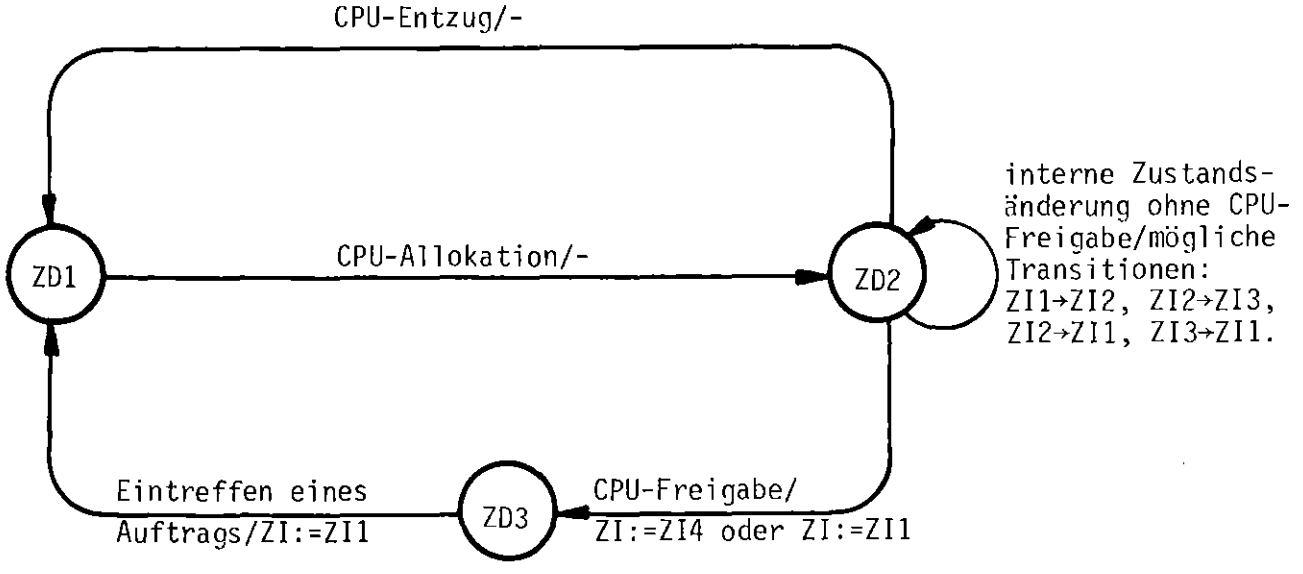

Abb. 3.14.: Zustandsübergangsdiagramm einer Protokolltask (dynamisches Modell) (Notation s. Abb. 3.13.)

3.3.3. Betriebsmittelverwaltung in einem Rechner

Mit den Modellen zur Beschreibung des dynamischen Zustands einer Task haben wir die Grundlage dazu geschaffen, das Problem der Betriebsmittelvergabe an Tasks in Angriff zu nehmen. Eine erste Möglichkeit, Betriebsmittelzuteilung zu modellieren, wäre ein gegenseitiges Weiterreichen von Betriebsmitteln durch Tasks. Diese Lösung steht jedoch in Konflikt zu einer Reihe von Anforderungen an die Modellierung der Betriebsmittelvergabe /s. WOLF 78a/, wie z.B.:

- Berücksichtigung unterschiedlicher Rechnerarchitekturen (Ein- bzw. Mehrprozessorsysteme) oder Rechnertypen (Realzeit- bzw. Batchsysteme) und ihrer Auswirkungen im Betriebssystem;
- Beachtung der Prioritäten und Unterbrechbarkeitseigenschaften von Tasks;
- Bereitstellung einer Instanz, die - zur Unterstützung der Modellauswertung - über den aktuellen dynamischen Zustand sämtlicher Tasks und über die Belegung der Rechnerbetriebsmittel Kenntnis besitzt.

Um den obigen Anforderungen Rechnung zu tragen, wollen wir eine dedizierte Kontrollinstanz einführen, die der Überwachung der Vergabe von Rechnerbetriebsmitteln an Tasks dient und die innerhalb eines jeden Rechnernetzknotens genau einmal auftritt. Dabei soll die Modellierung der Betriebsmittelvergabe in einem Rechnernetzknoten hier primär die Konkurrenz der Protokolltasks um Rechnerbetriebsmittel berücksichtigen. Die Betriebsmittelbelegung durch Benutzertasks ist nur insofern von Relevanz als dadurch Protokolltasks verzögert werden. Dies impliziert, daß der Detaillierungsgrad für die Vergabe derjenigen Betriebsmittel, die durch Protokolltasks in Anspruch genommen werden, höher zu sein hat als für die Menge der restlichen Rechnerbetriebsmittel. Unter der Voraussetzung hauptspeicherresidenter Protokolltasks genügt daher eine recht oberflächliche Berücksichtigung des Betriebsmittels "Hauptspeicher", wohingegen die Verwaltung des Betriebsmittels "CPU" detaillierter in das Modell einzugehen hat. Für die in unserem Modell eines Rechnernetzknotens auftretende Kontrollinstanz wollen wir aus diesem Grunde voraussetzen, daß sie für das Betriebsmittel "CPU" Allokation und Entzug durchführt, während für das Betriebsmittel "Hauptspeicher" kein Entzug zugelassen wird und es im wesentlichen nur dazu dient, die Menge der Benutzertasks pro Rechnernetzknoten zu limitieren.

Für derartige Kontrollinstanzen geben wir die folgende

Definition: Die funktionelle Einheit, die innerhalb eines Rechnernetzknotens die Vergabe und den Entzug von Rechnerbetriebsmitteln (CPU, Hauptspeicher) an Tasks durchführt, werde als Betriebsmittelverwaltungsinstanz (BVI) bezeichnet.

Eine Betriebsmittelverwaltungsinstanz stellt ein Analogon zur Realisierung der Betriebsmittelzuteilung in existierenden Betriebssystemen dar. (Zur Einbettung einer Betriebsmittelverwaltungsinstanz in einen kommunizierenden Rechnernetzknoten s. Abb. 4.2.).

Bei der Formulierung eines Warteschlangenmodells (s. Abb. 3.15.) kann eine Betriebsmittelverwaltungsinstanz als Bedienungsstation zur Bearbeitung von Tasks angesehen werden (die Taskbearbeitung besteht in der Allokation bzw. im Entzug von Betriebsmitteln). Die Warteschlangen einer BVI dienen der Speicherung zu bearbeitender Tasks. Zur Erleichterung der Taskauswahl bei der Betriebsmittelvergabe durch die BVI werden Tasks desselben dynamischen Zustands in jeweils einer Warteschlange gespeichert. Den Taskwarteschlangen kommt folgende Bedeutung zu:

- Q_{WHSP}: enthält Benutzertasks, die auf Hauptspeicherzuteilung warten, d.h. dynamischer Taskzustand = ZD0;
- Q_{WCPU}: enthält Benutzer- oder Protokolltasks, die auf CPU-Zuteilung warten, d.h. dynamischer Taskzustand = ZD1;
- Q_{CPU}: enthält CPU-aktive Benutzer- oder Protokolltask(s), d.h. dynamischer Taskzustand = ZD2;
- Q_{PEND}: enthält temporär blockierte Benutzer- bzw. Protokolltasks, d.h. dynamischer Taskzustand = ZD3.

Zur Erleichterung der Modellauswertung kann eine zusätzliche Auftrennung der Warteschlangen Q_{WCPU}, Q_{CPU} und Q_{PEND} in jeweils zwei verschiedene Warteschlangen (einerseits für Benutzertasks und andererseits für Protokolltasks) vorteilhaft sein.

Die Komponente E/A dient der Modellierung von Verzögerungen Ein-/Ausgabe aktiver Benutzertasks und enthält genau die Benutzertasks, die sich im dynamischen Zustand ZD4 befinden. Der Detaillierungsgrad der Komponente E/A kann variieren von paralleler Abfertigung sämtlicher auf periphere Geräte wartender Benutzertasks (einfachster Fall) bis zur Einführung einer Bedienungsstation für jedes der an den Rechner angeschlossenen peripheren Geräte.

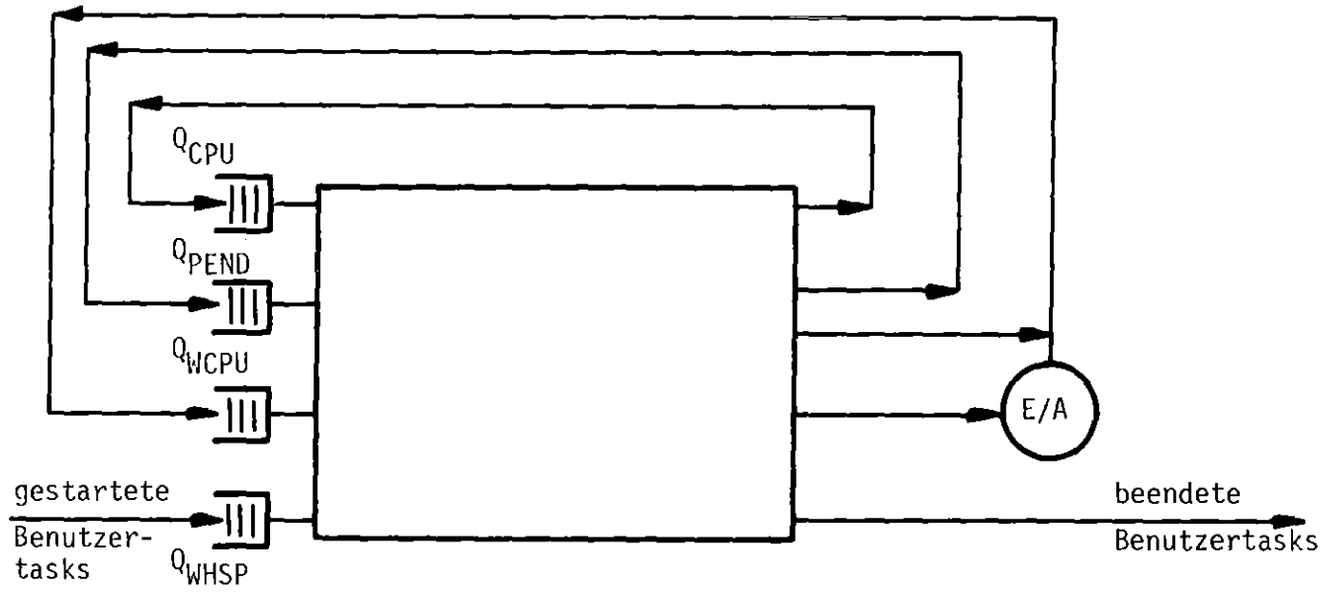

Abb. 3.15.: Warteschlangenmodell für die Betriebsmittelvergabe in einem Rechner

Bei der Bearbeitung der Warteschlangen Q_{WHSP}, Q_{WCPU}, Q_{CPU} und Q_{PEND} reagiert die Betriebsmittelverwaltungsinstanz auf "Interrupts", die eine Änderung der aktuellen Verteilung der Rechnerbetriebsmittel implizieren können. Solche Interrupts sind

- Änderung des dynamischen Zustands einer oder mehrerer Tasks;
- Ablauf von Zeitscheiben;
- Erzeugung einer Benutzertask;
- Erzeugung eines Schnittstellenauftrags zur Bearbeitung durch eine Protokolltask oder durch eine Benutzertask.

Bei Eintreffen eines Interrupts entscheidet die Betriebsmittelverwaltungsinstanz anhand der Art des Interrupts und unter Berücksichtigung des aktuellen Zustands der Betriebsmittelvergabe, ob Betriebsmittel einer Task zuzuteilen oder zu entziehen sind. Eine Modifikation der Betriebsmittelallokation einer Task impliziert eine Änderung des dynamischen Zustands der Task und damit auch einen Wechsel der belegten BVI-Warteschlange.

Die Bearbeitung der diversen Taskwarteschlangen durch die Betriebsmittelverwaltungsinstanz wird derart abgewickelt, daß

- Benutzertasks aus Q_{WHSP} in Q_{WCPU} umgehängt werden, sobald der benötigte Hauptspeicher vollständig zur Verfügung steht;
- Tasks aus Q_{WCPU} in Q_{CPU} umgehängt werden, sobald ihnen ein freier Prozessor zugeteilt werden kann;
- prozessorbelegende Tasks die Warteschlange Q_{CPU} selbständig verlassen, falls sie ihren Rechenzustand beendet haben oder diesen Tasks bei einer Verdrängung durch eine höherpriore Task die CPU entzogen wird; die Verdrängung einer CPU-aktiven Task wird derart realisiert, daß der Task ein "Flag" gesetzt wird (ein gesetzter Flag bewirkt, daß die "geflagte" Task die CPU sofort freigibt, wenn sie einen unterbrechbaren Rechenzustand, d.h. den internen Zustand ZI1', erreicht);
- suspendierte Tasks in Q_{PEND} bei Eintreffen des jeweils erwarteten Ereignisses aus dem dynamischen Zustand ZD3 in den Zustand ZD1 überzuführen und in die BVI-Warteschlange Q_{WCPU} umzuhängen sind.

Zur weiteren Präzisierung ihrer Wirkungsweise soll eine Betriebsmittelverwaltungsinstanz als sequentieller Automat dargestellt werden. Dazu seien die folgenden Grundzustände einer BVI definiert:

- "Bereit": In diesem Zustand ist die BVI ruhend, jedoch bereit zur Reaktion auf sämtliche Typen von Interrupts;
- "Aktiv": Dieser Zustand bezeichnet einen nicht unterbrechbaren Entscheidungszyklus, der die Notwendigkeit einer evtl. Neuverteilung der Betriebsmittel überprüft und folgendermaßen abläuft: Interruptanalyse, Analyse des Zustands der aktuellen Betriebsmittelvergabe, Bestimmung von Tasks denen Betriebsmittel zuzuteilen oder zu entziehen sind, Änderung des dynamischen Zustands von Tasks, Erneuerung der Zustandsinformation der aktuellen Betriebsmittelvergabe, Vernichtung oder Zurückstellen des behandelten Interrupts;
- "Blockiert": In diesem Zustand ist nur die Bearbeitung spezieller Typen von Interrupts (z.B. Freigabe eines Prozessors) sinnvoll, und daher ist die BVI hier - im Gegensatz zum Zustand "Bereit" - nur für eine reduzierte Menge von Interrupts reaktionsbereit; die Einführung dieses zusätzlichen Zustands unterstützt eine effizientere Modellauswertung.

Die Zustandsübergänge im Automatengraphen, der sich zur Modellierung einer Betriebsmittelverwaltungsinstanz erstellen läßt, umfassen

(a) "Bereit" → "Aktiv": Bei Vorliegen von Interrupts während des BVI-Zustands "Bereit" erfolgt eine Auswahl des höchstprioren Interrupts und eine Reaktion hinsichtlich dieses Interrupts in einem Entscheidungszyklus.

(b) "Aktiv" → "Bereit": Nach Beendigung eines Entscheidungszyklus erfolgt eine Transition in den Grundzustand "Bereit" sofern weiterhin eine Reaktion auf sämtliche Typen von Interrupts sinnvoll ist.

(c) "Aktiv" → "Blockiert": Der Übergang in den Zustand "Blockiert" erfolgt nach abgeschlossenem Entscheidungszyklus, sofern (mindestens) ein Typ von Interrupts existiert, deren aktuelle Bearbeitung nicht zulässig ist.

(d) "Blockiert" → "Aktiv": Diese Transition erfolgt genau dann, wenn ein Interrupt vorliegt, dessen Typ eine momentane Interruptreaktion als sinnvoll erscheinen läßt.

Die Parameter einer Betriebsmittelverwaltungsinstanz betreffen u.a. die Charakteristika der zu bearbeitenden Taskwarteschlangen. So ist z.B. die Länge der Warteschlange Q_{CPU} beschränkt durch die Anzahl der CPUs im Rechner, und die Warteschlangenauswahl- und Unterbrechungsstrategien für Tasks sind abhängig von Betriebssystemeigenschaften.

Auf zwei der bekannten Probleme, die sich bei der Betriebsmittelzuteilung an eine Menge konkurrierender Tasks ergeben, soll hier noch etwas näher eingegangen werden:

- Deadlockprobleme: Aus der Vergabe der Betriebsmittel CPU und Hauptspeicher resultierende Verklemmungen werden dadurch vermieden, daß die Voraussetzungen einer hierarchischen Betriebsmittelvergabe (s. /BRIN 73/) für die Ressourcen CPU und Hauptspeicher in unserem Modell erfüllt sind.
- Synchronisationsprobleme: Zugriffe von Tasks auf gemeinsame Datenbereiche treten im Modell jeweils für die beiden Protokollmoduln innerhalb einer Protokolleinheit auf, die u.a. auf gemeinsamer Zustandsinformation (z.B. hinsichtlich des Kommunikationsprotokolls) arbeiten. Zur Lösung des hieraus resultierenden "Mutual exclusion"-Problems reicht für Einprozessorsysteme die bereits vorausgesetzte Ununterbrechbarkeit für Protokollmoduln während der Bearbeitung eines Schnittstellenauftrags aus; für Mehrprozessorsysteme wird darüber hinaus ein Instruktionspaar LOCK/UNLOCK benötigt /SALT 66/.

Das vorgestellte Modell für die Betriebsmittelzuteilung kann für solche Rechner erheblich vereinfacht werden, die keine Benutzertasks sondern ausschließlich hauptspeicherresidente Protokolltasks enthalten, wie es beispielsweise bei dedizierten Vermittlungsrechnern der Fall sein kann. Die Vereinfachungen gegenüber dem allgemeinen Modell erstrecken sich auf die Eliminierung der Warteschlange Q_{WHSP} sowie der Komponente E/A.

Andererseits muß das BVI-Modell z.B. dann zusätzlich präzisiert werden, wenn Protokolltasks nicht hauptspeicherresident sind. In diesem Falle kann es notwendig werden, die Verwaltung des Betriebsmittels "Hauptspeicher" exakter nachzubilden.

Mit der Modellierung der Vergabe der Betriebsmittel in einem Rechnernetzknoten sind nun die konzeptionellen Modelle bereitgestellt für sämtliche der in Kapitel 2 definierten Rechnernetzkomponenten; und zwar bietet sich für die in den Abschnitten 2.1. bzw. 2.2. eingeführten Komponenten (P1)-(P4) bzw. (S1)-(S4) folgende Abbildung auf die konzeptionellen Modelle an:

- (P1),(P2): ein Rechnernetzknoten DVS wird modelliert durch eine Betriebsmittelverwaltungsinstanz, in deren Warteschlangen sich sämtliche Benutzertasks und Protokollmoduln befinden, die DVS beinhaltet;
- (P3): eine an einen Rechnernetzknoten DVS angeschlossene Ein-/Ausgabestation EAS wird i.a. nur durch die Aufträge repräsentiert, die EAS für den Rechnernetzknoten DVS erzeugt;
- (P4): die physikalischen Eigenschaften einer Verbindung zwischen Rechnernetzknoten finden ihre Berücksichtigung im Modell des ihr entsprechenden Übertragungskanals;
- (S1): die organisatorischen Einheiten, die Rechnernetzdienste realisieren, entsprechen den Protokolleinheiten (bzw. Protokollmoduln);
- (S2),(S3): die Dateneinheiten und Schnittstellenaufträge finden sich direkt unter den konzeptionellen Modellen wieder;
- (S4): die Betriebssystemkomponenten, die die Betriebsmittelvergabe organisieren, werden abgebildet auf eine Betriebsmittelverwaltungsinstanz.

4. MOSAIC - EIN PROGRAMMSYSTEM ZUR AUSWERTUNG DER RECHNERNETZMODELLE DURCH SIMULATION

Nach der Einführung von konzeptionellen Teilmodellen, aus denen sich das Gesamtmodell eines Rechnernetzes aufbauen läßt, stehen wir nun vor zwei zentralen Aufgaben:

(1) dem Gültigkeitsnachweis für die in Kapitel 3 vorgestellten Teilmodelle,
(2) der Erarbeitung eines Instruments, das es - bei vorgegebener Problemstellung (z.B. Leistungsanalyse eines existierenden Rechnernetzes oder eines geplanten Kommunikationsprotokolls) - erlaubt, die Modelle an das zu untersuchende System anzupassen und unter verschiedenartigen Randbedingungen auszuwerten.

Die aufgeworfene Frage nach dem Gültigkeitsnachweis für die Modelle beinhaltet u.a. die beiden, grundlegend verschiedenen, Aspekte:

(a) logische Korrektheit der konzeptionellen Modelle (z.B. verklemmungsfreies Arbeiten der Modelle):
Generelle Korrektheitsstudien auf der Stufe der konzeptionellen Modelle erscheinen wegen der starken Anwendungsabhängigkeit und wegen des variablen Detaillierungsgrads der vorgestellten Modelle nicht praktikabel; sofern es sich als sinnvoll erwies, wurden indes notwendige Bedingungen für die Korrektheit der Modelle bereits in Kapitel 3 formuliert.

(b) Grad der Übereinstimmung der Modelle mit der Realität, d.h. Validation der Modelle (s. hierzu /BEIL 73/, /EFRO 75/, /TEOR 75/):
Die Diskrepanz zwischen einem Modellsystem A' und einem zu untersuchenden System A (Realität) hängt nicht selten sehr stark vom gewählten Detaillierungsgrad der Modelle ab. Da die hier vorgestellten Modelle einen variablen Detaillierungsgrad besitzen, ist somit offensichtlich, daß sich die Frage nach der Modellgenauigkeit nur in folgender, modifizierter Form beantworten läßt:

Voraussetzung: Gegeben seien ein zu untersuchendes System A (z.B. ein Rechnernetz oder ein Kommunikationsprotokoll) und Genauigkeitsanforderungen, denen ein Modellsystem A' genügen soll.

Problem: Ist es möglich, aus der Menge der in dieser Arbeit vorgestellten konzeptionellen Modelle ein Rechnernetzmodell (d.h. ein Modellsystem A') zu erstellen, das die gewünschten Genauigkeitsanforderungen erfüllt?

Die Behandlung dieses Problems setzt allerdings zunächst die Existenz eines Instruments zur Modellauswertung voraus, und muß daher bis Kapitel 6 zurückgestellt werden.

In Kapitel 4 und 5 wollen wir uns der Aufgabe zuwenden, ein Instrument zur Auswertung der eingeführten konzeptionellen Modelle bereitzustellen. Der Versuch, die erarbeiteten Rechnernetzmodelle durch analytische Lösungsverfahren (s. /BASK 75/, /GORD 67/, /KOBA 77/, /SCHW 77/, /WONG 78/) auszuwerten, scheitert an der Komplexität der Modelle und ihrer Wechselbeziehungen. Hingegen zeigten Erfahrungen, die der Autor bei der Modellierung der Kommunikationsflüsse des HMINET[+)] sammeln konnte (s. u.a. /WOLF 77/), daß sich die Methode der rechnergestützten Simulation für die Auswertung derartiger Rechnernetzmodelle erfolgreich einsetzen läßt. Die Simulation soll deshalb auch in der vorliegenden Arbeit als ausschließliches Auswertungshilfsmittel verwendet werden.

Im folgenden wird der Simulator MOSAIC (Modellierungssystem zur Simulation allgemeiner Informationsflüsse in Computernetzen) vorgestellt, der durch den Autor am Institut für Datenverarbeitung in der Technik des Kernforschungszentrums Karlsruhe entwickelt wurde und zur Auswertung der in Kapitel 3 dargestellten konzeptionellen Teilmodelle eines Rechnernetzes dient.

Für die Erstellung des Programmsystems MOSAIC wurde eine Reihe wesentlicher Anforderungen berücksichtigt:

(1) Anwendbarkeit des Simulators bei verschiedenartigen Randbedingungen (Rechnernetzcharakteristika, Auftragsprofile etc.);
(2) Einsatzmöglichkeit des Simulators für Untersuchungen mit unterschiedlichem Detaillierungsgrad;
(3) einfache Erweiterbarkeit des Simulators;
(4) benutzernahe Resultatdarstellung durch Einsatz graphischer Hilfsmittel;
(5) effiziente Experimentdurchführung durch Unterstützung interaktiver Simulationsexperimente;
(6) hohe Portabilität der zur Implementierung verwendeten Programmiersprache(n).

+) Diese Simulationsstudie wurde 1975/76 im Institut für Datenverarbeitung in der Technik/Kernforschungszentrum Karlsruhe im Rahmen eines Kooperationsvertrages mit dem Hahn-Meitner-Institut für Kernforschung Berlin durchgeführt.

Besonders im Hinblick auf die Anforderungen (1) - (4) wurde für MOSAIC das Konzept eines Modellierungssystems erarbeitet. Die grundlegende Idee hierbei ist, die zur Simulation von Kommunikationsflüssen in Rechnernetzen benötigten Modellbausteine in Form eines Baukastensatzes bereitzustellen und aus diesen Bausteinen während der Programmausführung ein ablauffähiges Simulationsprogramm zu konfigurieren. Diese Methode erlaubt u.a. eine gegenseitig unabhängige Definition sowie den problemlosen Austausch von Modellbausteinen. Folgende Hauptkomponenten eines Modellierungssystems können unterschieden werden (vgl. Abb. 4.1.):

- Die Modellbausteine, die eine Umsetzung der in Kapitel 3 präsentierten Teilmodelle in die verwendete Programmiersprache darstellen;
- ein Konfigurator, der zu Beginn eines Simulationsexperimentes aus den Modellbausteinen unter Berücksichtigung der Eingabedaten das Rechnernetzmodell als ablauffähiges Simulationsprogramm erstellt;
- ein Auftragsgenerator, der zur Erzeugung der Aufträge der Rechnernetzumgebung fungiert und während eines Simulationsexperiments auf das Rechnernetzmodell einwirkt;
- die Meßeinrichtungen, die dazu dienen, im Laufe eines Simulationsexperiments Modellvariablen (z.B. Leistungskenngrößen) zu vermessen und ggf. die erfaßten Meßdaten für eine spätere Auswertung abzuspeichern;
- die Einrichtungen zur Datenauswertung, die dedizierte Komponenten sowohl zur statistischen Auswertung der gesamten Meßdaten als auch zur graphischen Aufbereitung der ermittelten Experimentresultate umfassen.

Als Implementierungssprache für die Erstellung des Modellierungssystems MOSAIC wurde - abgesehen von der Hauptkomponente "Graphische (Resultat-) Aufbereitung", deren Implementierung in FORTRAN erfolgte, - die höhere Programmiersprache SIMULA, s. /DAHL 68/, gewählt, zum einen um der Forderung nach hoher Portabilität zu genügen und zum anderen auch deshalb, weil das "CLASS"-Konzept in SIMULA die Definition in sich abgeschlossener gegenseitig unabhängiger Modellkomponenten erlaubt, die in beliebiger Anzahl reproduzierbar sind. (Eine "CLASS" in SIMULA bezeichnet eine syntaktische Einheit, die durch Definition eines strukturierten Datentyps und einer Menge darauf auszuführender Operationen festgelegt wird). In diesem Sinne unterstützt die Sprache SIMULA den Aufbau von Modellierungssystemen, da sich die benötigten Modellbausteine als "CLASS" darstellen lassen. Wie Vergleiche zwischen verschiedenen Simulations-

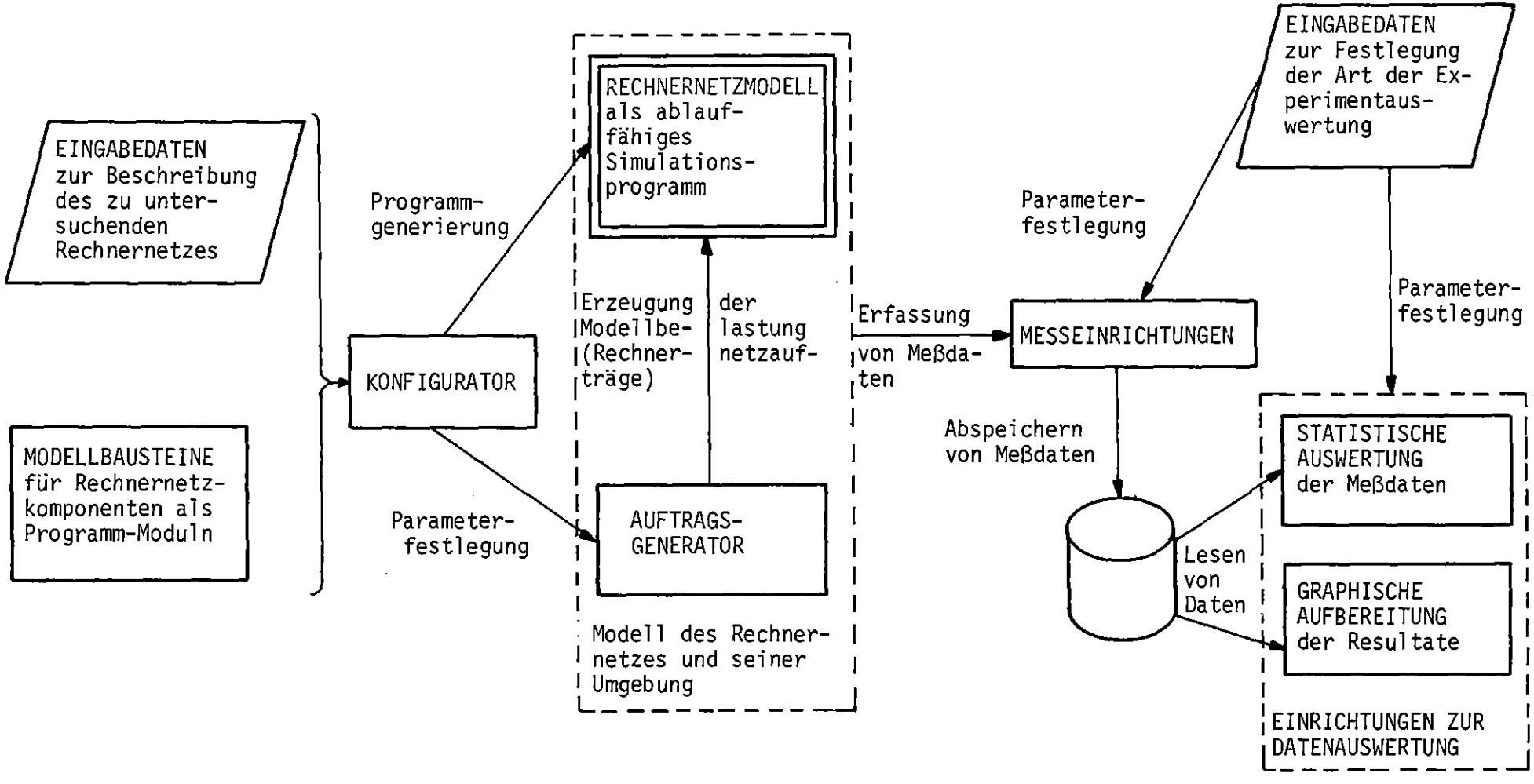

Abb. 4.1: Zusammenwirken der Hauptkomponenten des Modellierungssystems MOSAIC während eines Simulationsexperiments

sprachen zeigen, besitzt SIMULA darüber hinaus eine wesentlich höhere Effizienz als Sprachen wie z.B. GPSS, wenn als Vergleichskriterium der CPU-Zeitbedarf für die Ausführung einander äquivalenter Programme gewählt wird (s. Vergleich in /NIED 79/).

Das Modellierungssystem MOSAIC enthält die meisten der für eine Rechnernetzsimulation relevanten Komponenten, und kann als Nukleus (Grundmodell) angesehen werden, der als Basis detaillierterer Modellierungssysteme dienen kann. Die Erweiterbarkeit sowie die Austauschbarkeit von Modellbausteinen, die MOSAIC dem zugrundeliegenden Modellierungssystemkonzept verdankt, erleichtern derartige Anpassungen von MOSAIC an unterschiedliche Rechnernetze.

Ehe die Hauptkomponenten des Modellierungssystems MOSAIC präziser dargestellt werden, soll in einer SIMULA-ähnlichen Notation ein Überblick über die Grobstruktur der aktuellen Implementierung von MOSAIC gegeben werden. (Eingefügte Kommentare sind dabei zwischen die Zeichen /* und */ eingeschlossen):

```
begin /* Modellierungssystem MOSAIC
         Initialisierung (1. Teil):
         Definition und Vorbesetzung von Parametern zur globalen Festlegung
         der Rechnernetzstruktur [z.B. Anzahl der Rechnernetzknoten (NUMCOMP),
         Anzahl der Arbeitsrechner (NUMHOST), Anzahl der Protokollschichten
         pro Rechnernetzknoten ...] */

      /* Abschnitt A: MOSAIC-Hauptkomponente "GRAPHISCHE AUFBEREITUNG" */
         external FORTRAN procedure PLOTNET, PLOTQUEUES, PLOTDIAGRAM;
      /* A: "GRAPHISCHE AUFBEREITUNG" (Ende) */

      /* Abschnitt B: MOSAIC-Hauptkomponente "STATISTISCHE AUSWERTUNG" */
         external procedure MEAN_VALUE, VARIANCE,...;
      /* B: "STATISTISCHE AUSWERTUNG" (Ende) */

      SIMULATION begin

         ref (LOAD_GENERATOR) array LOADGEN(1:NUMCOMP);
             /* Verweis auf die Auftragsgeneratoren in den Rechnernetzknoten */
         ref (RECHNERNETZKNOTEN) array DVS(1:NUMCOMP);
             /* Verweis auf die Rechnernetzknoten */
         ref (OBERTRAGUNGSKANAL) array KANAL(1:NUMCOMP,NUMHOST+1:NUMCOMP);
             /* Verweis auf die Übertragungskanäle */
```

```
   ...
/* Abschnitt C: MOSAIC-Hauptkomponente "MODELLBAUSTEINE" */
   LINK class DATENEINHEIT...;
   LINK class KONTROLLINFORMATION...;
   LINK class SCHNITTSTELLENAUFTRAG...;
   PROCESS class ÜBERTRAGUNGSKANAL...;
   PROCESS class RECHNERNETZKNOTEN...;
   PROCESS class TASK...;
   TASK class BENUTZERTASK...;
   class SCHICHT...;
   class PROTOKOLLEINHEIT...;
   TASK class SENDE_PROTOKOLLMODUL...;
   TASK class EMPFANGS_PROTOKOLLMODUL...;
   PROCESS class TIMEOUT...;
/* C: "MODELLBAUSTEINE" (Ende) */

/* Abschnitt D: MOSAIC-Hauptkomponente "AUFTRAGSGENERATOR" */
   PROCESS class LOAD_GENERATOR...;
      /* Komponente zur Generierung der Aufträge (Workload)
         innerhalb eines Rechnernetzknotens */
/* D: "AUFTRAGSGENERATOR" (Ende) */

/* Abschnitt E: Definition von Prozeduren */
   procedure GENTASK...; /* Prozedur zur Generierung von Benutzer-
                            tasks */
   procedure GENPMREQ...; /* Prozedur zur Generierung von Schnitt-
                             stellenaufträgen */

...
/* E: Prozeduren (Ende) */

/* Abschnitt F: MOSAIC-Hauptkomponente "KONFIGURATOR" */
   procedure INITIALIZE...;
/* F: "KONFIGURATOR" (Ende) */
```

```
      /* Abschnitt G: MOSAIC-Hauptkomponente "MESSEINRICHTUNGEN" */
         PROCESS class SNAPSHOT...; /* Komponente zur Erfassung von Meß-
                                       daten */
         PROCESS class PRINTDISC...; /* Komponente zum Abspeichern von
                                        Meßdaten */
         procedure TEST...; /* Trace-Prozeduren, z.B. zum Zwecke der Pro-
                               grammverifikation */
      /* G: "MESSEINRICHTUNGEN" (Ende) */

      /* Hauptprogramm (Beginn) */

      ...
      activate new PRINTDISC; /* Aktivierung der Meßeinrichtungen */
      INITIALIZE; /* Aufruf des Konfigurators und Initialisierung
                     (2. Teil) */
      activate new SNAPSHOT...; /* Aktivierung der Datenerfassung */
      for I:=1 step 1 until NUMCOMP do
      begin LOADGEN(I): - new LOAD_GENERATOR...;
          activate LOADGEN(I); /* Aktivierung des Auftragsgenerators im
                                  Rechnernetzknoten DVS(I) */
      end;
      HOLD(SIMTIME); /* Durchführung des Simulationsexperiments für das
                        Zeitintervall [0, SIMTIME] */
      /* Hauptprogramm (Ende) */
   end of SIMULATION;
end of MOSAIC.
```

4.1. Implementierung der konzeptionellen Teilmodelle eines Rechnernetzes

Die Formulierung der konzeptionellen Teilmodelle eines Rechnernetzes als Warteschlangensysteme und als sequentielle Automaten diente als direkte Grundlage für die Implementierung der Modellbausteine in MOSAIC. Um das Zeitverhalten eines zu untersuchenden Systems A nachzubilden, können dabei sowohl den Zuständen als auch den Transitionen der sequentiellen Automaten Zeiten zugeordnet werden oder - was äquivalent ist - den Warteschlangensystemen können Bedienungszeiten für die Abfertigung von Warteschlangenelementen zugewiesen werden.

Diese Zeiten sind im konkreten Anwendungsfall so zu wählen, daß sie der Ausführungsdauer der modellierten Hard- oder Software des zu untersuchenden Systems A entsprechen.

Als Überblick über die Interaktionen zwischen den implementierten Teilmodellen soll am Beispiel zweier kommunizierender Rechnernetzknoten das Zusammenspiel der wichtigsten Modellbausteine in MOSAIC beschrieben werden (vgl. Abb. 4.2.): Jeder Rechnernetzknoten enthält eine Anzahl von Protokollschichten $S_1,\ldots,S_n$; jede Schicht enthält eine Menge von Protokolleinheiten (PE), die ihrerseits wiederum jeweils einen Sendeprotokollmodul (SPM) und einen Empfangsprotokollmodul (EPM) beinhalten; parallel hierzu existiert eine Menge von (evtl. kommunizierenden) Benutzertasks und genau eine Betriebsmittelverwaltungsinstanz pro Rechnernetzknoten, die die Zuteilung und den Entzug von Betriebsmitteln für Benutzertasks, Sende- und Empfangsprotokollmoduln organisiert. Als Verbindung zwischen Rechnernetzknoten fungieren Übertragungskanäle, die pro Übertragungsrichtung eine Auftragsbearbeitungsinstanz (ABI) beinhalten.

Für die Modellbausteine Sende-, Empfangsprotokollmodul, Benutzertask und Betriebsmittelverwaltungsinstanz, denen in MOSAIC eine zentrale Bedeutung zukommt, wird anhand von Struktogrammen eine Beschreibung gegeben, deren primäres Ziel es ist, aufzuzeigen, welcher Teil der Implementierung Allgemeingültigkeit besitzt und in welcher Weise die Anwendungsabhängigkeit in den Modellen berücksichtigt ist.

(a) Protokollmoduln:

Die Protokollmoduln innerhalb einer Protokolleinheit sind durch eine starke Abhängigkeit von dem zugrundeliegenden Kommunikationsprotokoll gekennzeichnet. Die aktuelle Implementierung der Protokollmoduln umfaßt die in Abschnitt 3.1.3. eingeführte Klasse von Basisprotokollen und darüber hinaus eine Reihe von standardisierten Protokollen, z.B. eine Version des ISO-Standards HDLC (balanced class) /ISO 77/, der CCITT-Empfehlung X.25 /CCITT 77/, des durch den Arbeitskreis PIX vorgeschlagenen Message Link Protokolls /HERT 78/, etc.

In den Struktogrammen für Sende- und Empfangsprotokollmodul (vgl. Abb. 4.3. und 4.4.) geht die Protokollabhängigkeit zum einen ein in die zu bearbeitenden Schnittstellenaufträge (s. Verteiler: Auftragsbearbeitung) und zum anderen in die Arbeitsweise der verwendeten Prozeduren (z.B. CREATECINF, FRAGMENTATE, GENDATABLOCK, REACTTOHEADER, DEBLOCK).

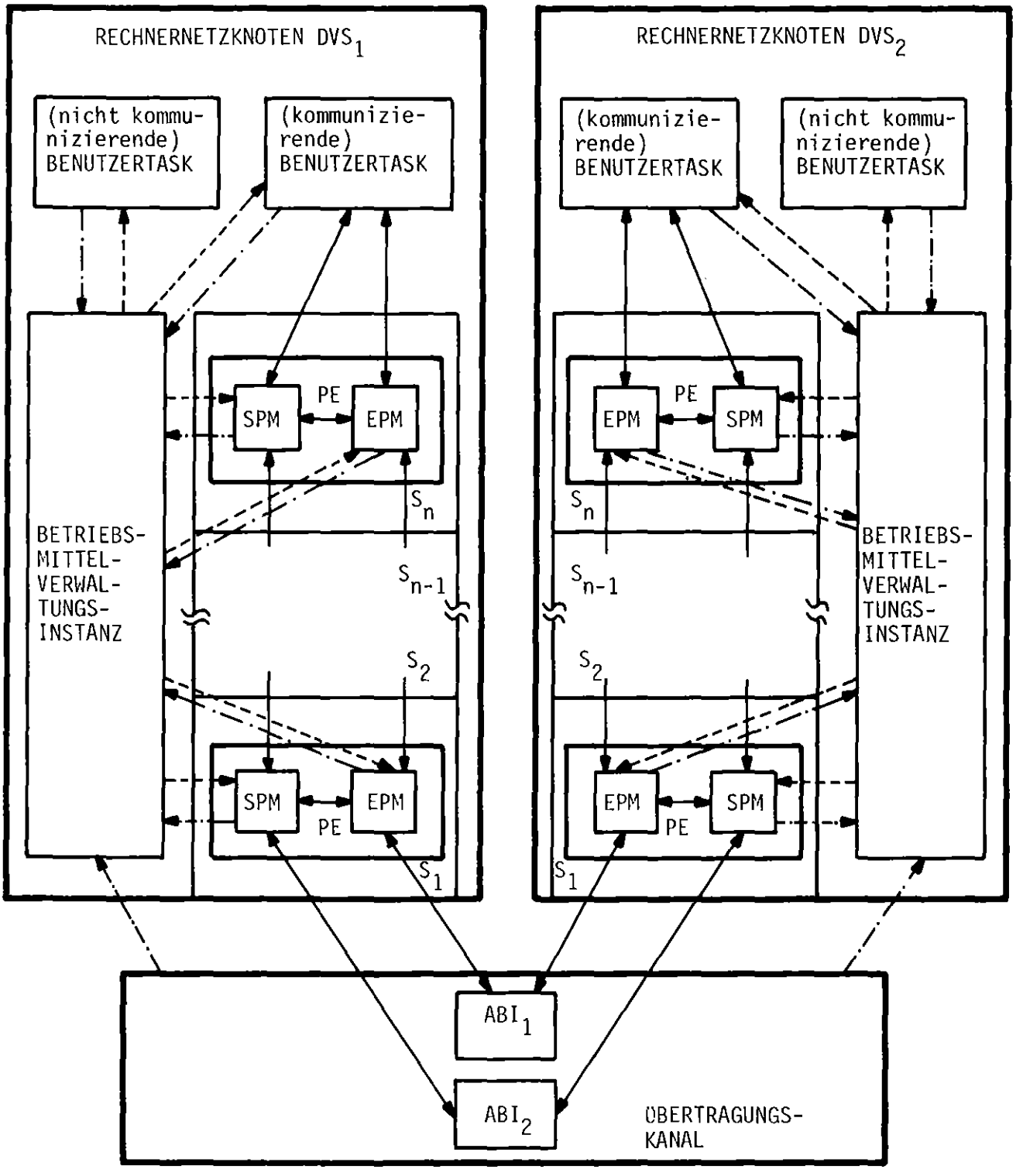

Abb. 4.2.: Zusammenspiel der wichtigsten Modellkomponenten am Beispiel zweier kommunizierender Rechnernetzknoten DVS_1 und DVS_2

[Notation: ⟷ Austausch von Schnittstellenaufträgen und Dateneinheiten
– – → Zuteilung/Entzug von Rechnerbetriebsmitteln
–·–· → Aktivierung durch Setzen eines Interrupts]

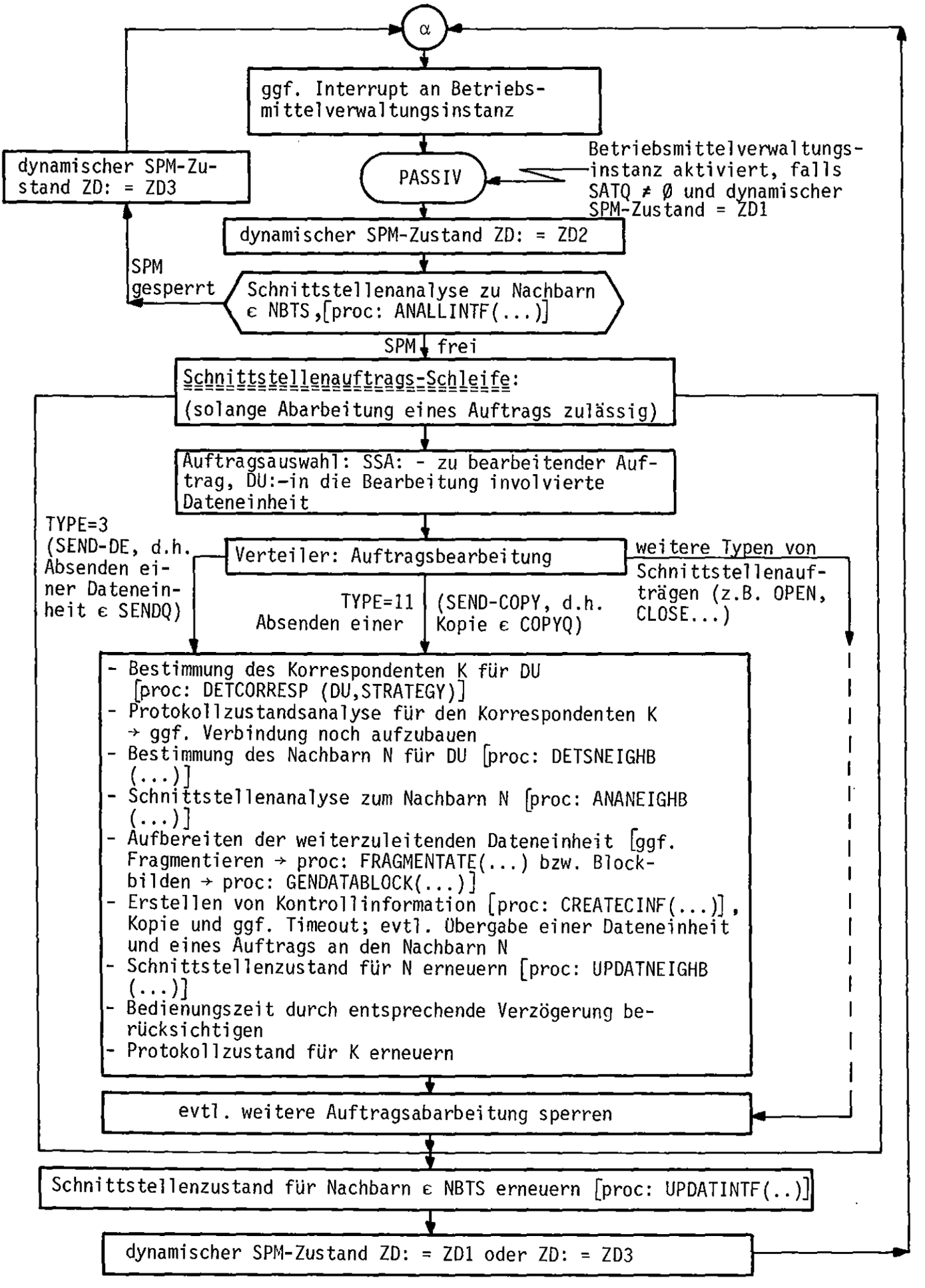

Abb. 4.3: Grobstruktur eines Sendeprotokollmoduls SPM (proc: P kennzeichnet den Aufruf einer Prozedur P; zur Bedeutung der Prozeduren s. Programmbeschreibung in Anhang A)

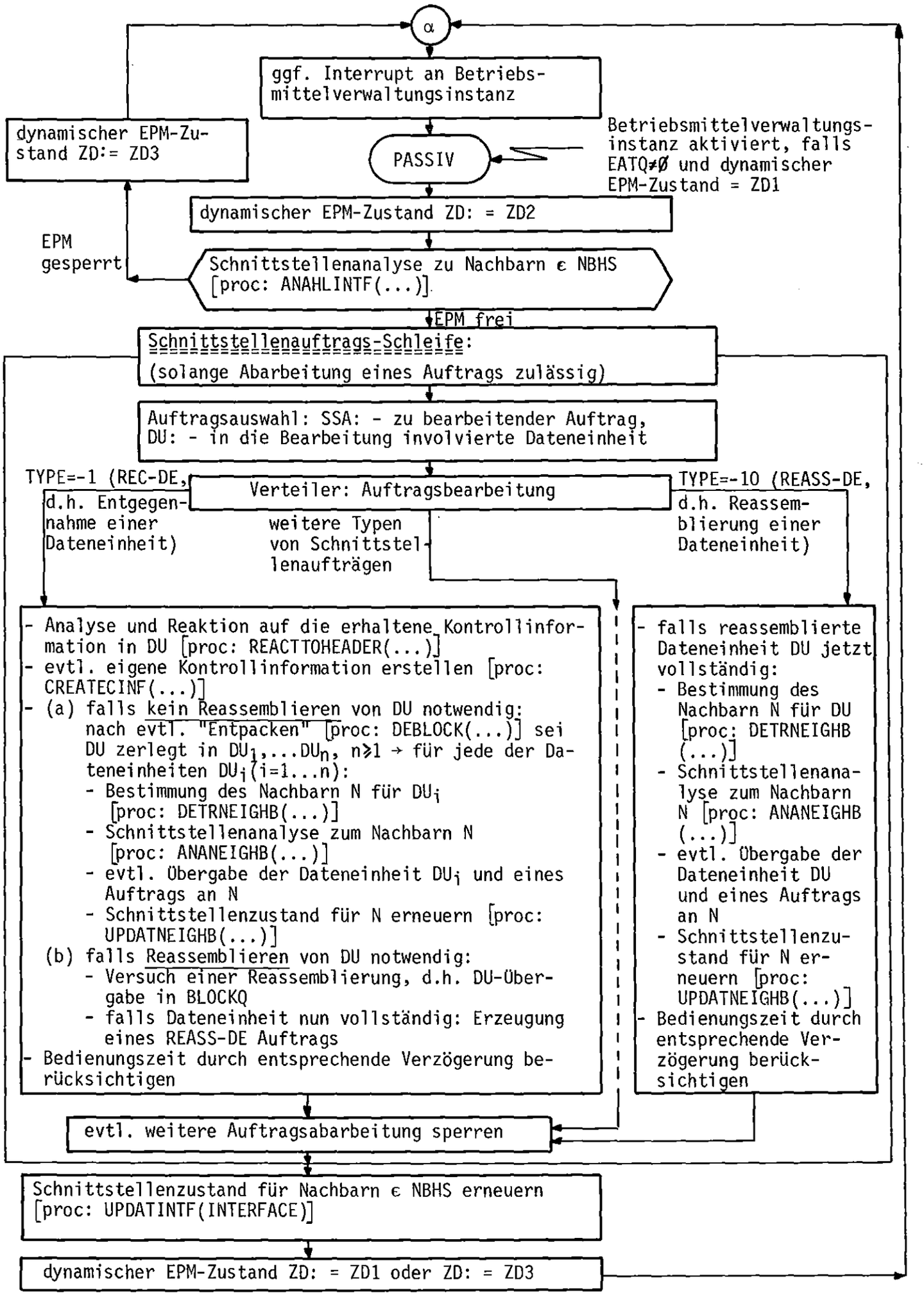

Abb. 4.4: Grobstruktur eines Empfangsprotokollmoduls EPM (Notation s. Abb. 4.3 und Bedeutung der Prozeduren s. Anhang A)

(b) Benutzertasks:

Die Implementierung einer Benutzertask basiert auf der dynamischen Taskbeschreibung (vgl. Abb. 3.13.). Um verschiedene Arten von Benutzertasks zu modellieren, wird zugelassen, daß eine Task nur eine Teilmenge der möglichen internen Zustände durchläuft. Auf diese Weise erhält man z.B. nicht kommunizierende Benutzertasks, falls der interne Taskzustand ZI3 eliminiert wird; andererseits kann bei kommunizierenden Benutzertasks vorausgesetzt werden, daß nur eine gewisse Art von Aufträgen an das Rechnernetz generiert wird (z.B. nur Schnittstellenaufträge zum Absenden von Dateneinheiten) und so können gezielt spezielle Typen kommunizierender Benutzertasks definiert werden. Bei kommunizierenden Benutzertasks nehmen wir im übrigen an, daß jeweils genau zwei Tasks über einen Rechnernetzdienst miteinander kommunizieren, d.h. Dateneinheiten austauschen. (Durch eine geeignete Zerlegung kommunizierender Benutzertasks läßt sich eine paarweise Kommunikation immer erreichen).

Im Struktogramm einer Benutzertask (vgl. Abb. 4.5.) geht die Anwendungsabhängigkeit ein in die Art der Auswahl des internen Benutzertaskzustands, die u.a. durch vorgegebene Wahrscheinlichkeiten gesteuert wird.

(c) Betriebsmittelverwaltungsinstanz:

Die Betriebsmittelverwaltungsinstanz ist in hohem Maße betriebssystemabhängig; insbesondere ist die angewandte Strategie der Betriebsmittelvergabe von dominierendem Einfluß. CPU-Zuteilung nach einem Zeitscheibenverfahren sowie die Modellierung von Realzeitanwendungen durch ununterbrechbare CPU-Belegung für Tasks sind in der aktuellen Implementierung optional erhalten; hingegen ist eine Implementierung von Mehrprozessorsystemen bislang noch nicht erfolgt. Das Warteschlangenmodell einer Betriebsmittelverwaltungsinstanz (vgl. Abb. 3.15.) wurde dergestalt implementiert, daß die Warteschlangen Q_{WHSP}, Q_{WCPU}, Q_{CPU} und Q_{PEND} ausschließlich Benutzertasks und keine Protokollmoduln enthalten; aus implementierungstechnischen Gründen erschien es als vorteilhaft, sämtliche Protokollmoduln - unabhängig von ihrem dynamischen Taskzustand - in einer dedizierten Warteschlange Q_{PM} zu speichern. Die Komponente E/A zur Modellierung der E/A-Aktivitäten von Benutzertasks wurde als Bedienungsstation implementiert, die sämtliche Elemente ihrer Warteschlange parallel bearbeitet.

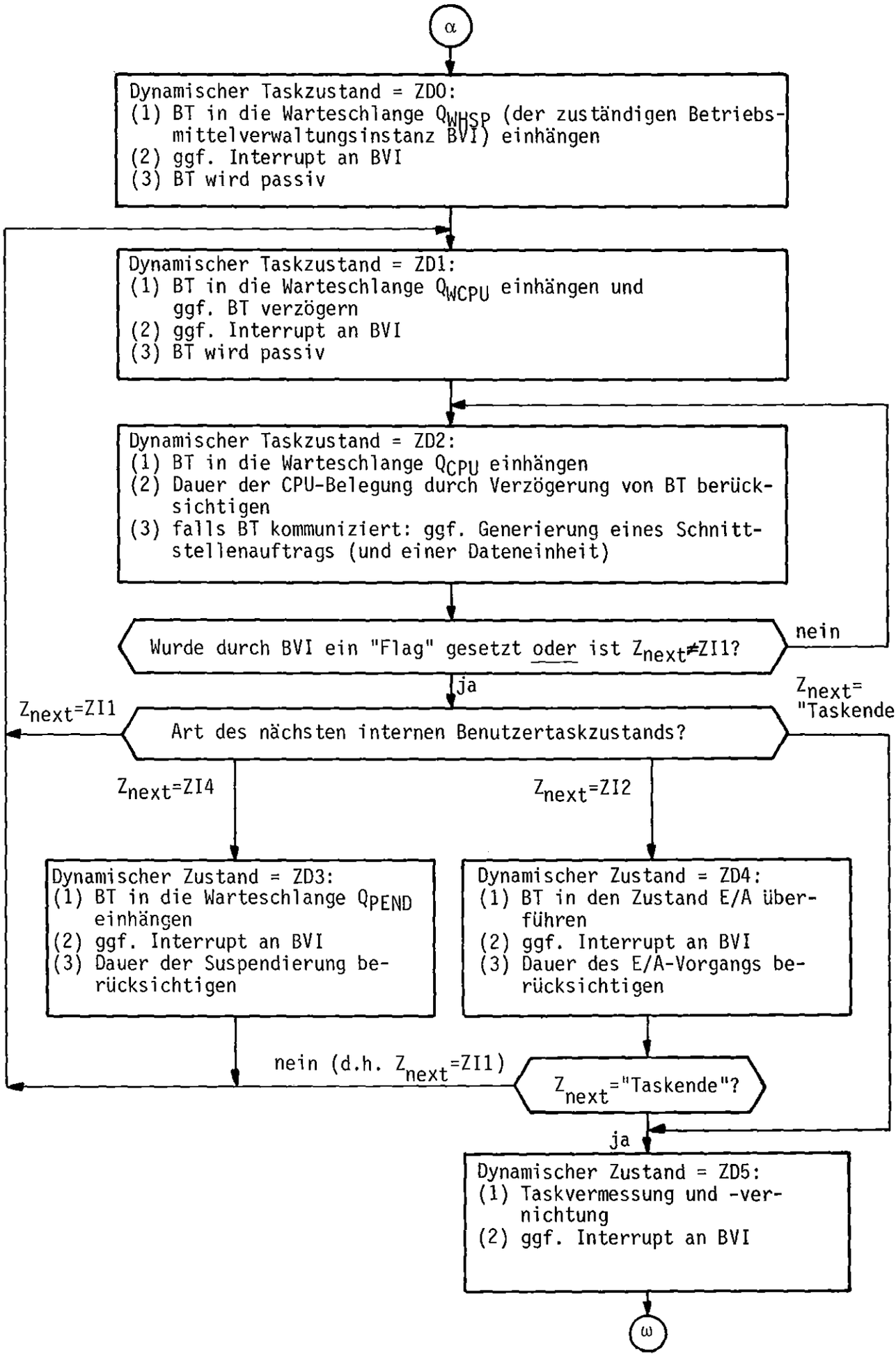

Abb. 4.5: Struktogramm einer Benutzertask BT (Notation s. Abschnitt 3.3.)

Im Struktogramm einer Betriebsmittelverwaltungsinstanz (vgl. Abb. 4.6.) findet die Strategie der Betriebsmittelvergabe Eingang in die Prozedur QSCHEDULER zur Bestimmung der zu bearbeitenden Warteschlange WS sowie in die Art der Auswahl der höchstprioren Task innerhalb der Warteschlange WS. Die Variable SQ dient zur Kennzeichnung der Warteschlange WS und zwar ist SQ = 0 für Q_{WHSP}, SQ = 1 für Q_{WCPU}, SQ = 3 für Q_{PM} und SQ < 0 falls keine Warteschlange zur Bearbeitung ermittelt werden konnte. Die Variable J fungiert als Hilfsvariable für den Aufruf der Prozedur QSCHEDULER. Die Menge der Interrupts, auf welche die Betriebsmittelverwaltungsinstanz im Zustand "Blockiert" reagiert, ist identisch mit sämtlichen Interrupts, die CPU-Freigabe anzeigen.

Details der Implementierung der weiteren Teilmodelle sollen an dieser Stelle nicht dargestellt werden. Für eine Grobbeschreibung sämtlicher Modelle sei auf Anhang A verwiesen und weitere Information hinsichtlich der Implementierung der Modellbausteine in MOSAIC - insbesondere auch ein Programmlisting - kann /WOLF 79/ entnommen werden.

4.2. Erstellen eines ablauffähigen Simulationsprogramms

Die angestrebte allgemeine Anwendbarkeit des Modellierungssystems MOSAIC zieht Auswirkungen hinsichtlich der Strukturierung der zu verwendenden Rechnernetzmodelle nach sich und findet bei der Konfigurierung eines ablauffähigen Simulationsprogramms Berücksichtigung; wir begnügen uns dabei nicht mit den bei der Auswertung von Rechnernetzmodellen üblichen Parameterstudien an einem "statischen" Simulationsmodell, sondern versuchen, die Anpassung von MOSAIC an verschiedenartige Randbedingungen (wie z.B. Rechnernetztopologie, Struktur der Protokollhierarchie, Auftragsprofil etc.) zu unterstützen.

Das Konzept zur Erreichung dieses Ziels besteht, wie bereits erwähnt, darin /s. WOLF 78a, HOLL 75/, daß die zur Verfügung gestellte Menge von Modellbausteinen - d.h. die in Abschnitt 4.1. eingeführten Rechnernetzkomponenten - erst bei der Durchführung eines Experiments zu einem ablauffähigen Simulationsprogramm verknüpft wird. Zur Konfigurierung eines solchen Gesamtmodells aus den vorgegebenen Teilmodellen dient in dem Modellierungssystem MOSAIC die Prozedur INITIALIZE, die

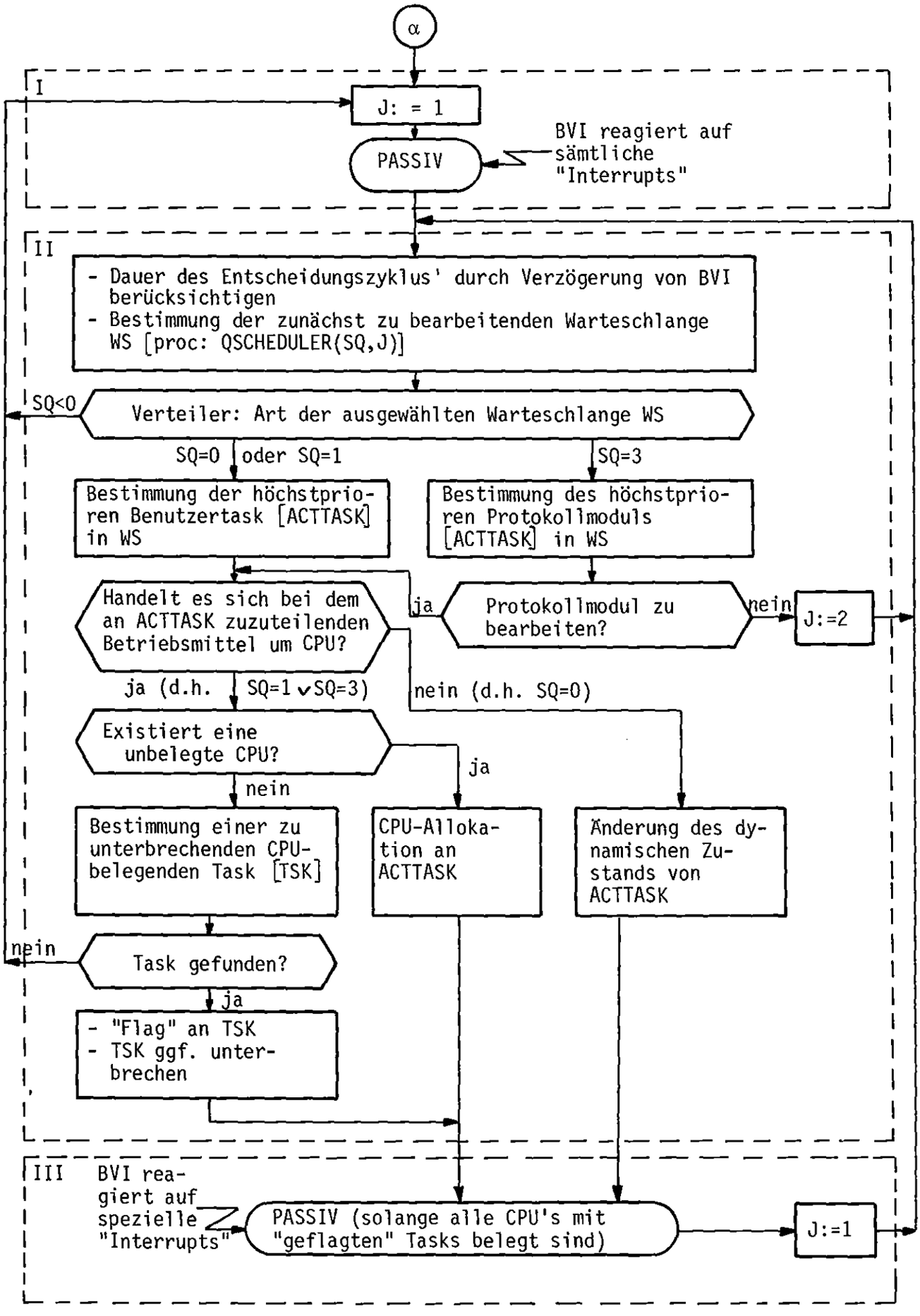

Abb. 4.6: Grobstruktur einer Betriebsmittelverwaltungsinstanz BVI
[BVI-Zustände: "Bereit" (I), "Aktiv" (II) und "Blockiert" (III)]

(a) die Eingabedaten des Experimentators entgegennimmt;
(b) von jedem der vordefinierten Bausteintypen die für das Simulationsexperiment benötigte Anzahl von Exemplaren erzeugt, nachdem die jeweiligen Parameter der Teilmodelle in der durch die Eingabedaten spezifizierten Weise vorbesetzt wurden;
(c) die Interaktionen zwischen den Teilmodellen definiert und auf diese Weise die operationale und funktionale Struktur des Gesamtmodells vervollständigt;
(d) eine komplette Beschreibung des aufgebauten Modells liefert und somit eine Kontrolle der Eingabe durch den Experimentator erlaubt.

Die Reihenfolge, in welcher MOSAIC die Rechnernetzkomponenten erzeugt, ist dergestalt, daß

(1) sämtliche Rechnernetzknoten DVS_i (i=1,...,NUMCOMP) definiert werden (dies schließt sowohl die Generierung einer Modellkomponente, die die Aufträge der Rechnernetzbenutzer an DVS_i erzeugt, s. Abschnitt 4.3., als auch die Festlegung sämtlicher Protokollschichten ein);

(2) innerhalb jeder Protokollschicht sämtliche Protokolleinheiten und insbesondere die Sende- und Empfangsprotokollmoduln erzeugt werden;

(3) die Übertragungskanäle definiert werden;

(4) die Rechnernetztopologie festgelegt wird (z.B. durch Vorgabe von Routing-Tabellen).

Eine detaillierte Beschreibung der E/A-Schnittstelle, die MOSAIC dem Experimentator anbietet, findet sich in /WOLF 79/.

4.3. Modellierung der Rechnernetzaufträge

Bei der Durchführung eines Simulationsexperiments wird das Modellverhalten unter Einwirkung einer vorgegebenen Belastung untersucht. Um die Berücksichtigung unterschiedlicher Arten von realen Auftragsprofilen zu erleichtern, ist es angebracht, eine (oder gar mehrere) dedizierte Modellkomponente(n) zur Erzeugung von Aufträgen bereitzustellen. Das Modellierungssystem MOSAIC beinhaltet für jeden Rechnernetzknoten eine derartige Komponente, den Auftragsgenerator, der genau die durch die Umgebung des Rechnernetzknotens spezifizierten Aufträge modelliert und durch die PROCESS class LOAD_GENERATOR realisiert wird. Naturgemäß existiert eine starke Ab-

hängigkeit des Auftragsgenerators von der Art der Simulationsstudie:

Beschränkt sich die Simulation auf eine Untersuchung der Kommunikationssoftware ohne Miteinbeziehung der in den Rechnernetzknoten ablaufenden Benutzertasks, so obliegt dem Auftragsgenerator die Erzeugung von Schnittstellenaufträgen für die Protokolleinheiten der höchsten modellierten Protokollschicht; hingegen handelt es sich in Simulationsstudien, die die Benutzerschicht mit einschließen, bei den zu erzeugenden Aufträgen um Benutzertasks, welche ihrerseits Schnittstellenaufträge für die Protokolleinheiten generieren.

Ein Auftragsgenerator läßt sich charakterisieren durch die Frequenz der Erzeugung von Anforderungen sowie durch die Eigenschaften (Betriebsmittelbedarf, Prioritäten, etc.) der generierten Aufträge. Die folgende Sequenz von Aktionen wird durch einen Auftragsgenerator in einem "Generierungszyklus" durchlaufen:

(1) Berücksichtigung der Verzögerung, die bis zur Erzeugung des nächsten Auftrags modelliert werden soll;

(2) Entscheidung, ob eine Benutzertask oder ein Schnittstellenauftrag zu generieren ist;

(3) Bestimmung der Charakteristika der zu erzeugenden Benutzertask oder des Schnittstellenauftrags (z.B. Typ des Auftrags);

(4) Erzeugung des Auftrags unter Benutzung der Prozedur GENTASK (für Benutzertasks) bzw. GENPMREQ (für Schnittstellenaufträge);

(5) falls eine kommunizierende Benutzertask erzeugt wurde: Erzeugung des Kommunikationspartners unter Benutzung der Prozedur GENTASK;

(6) weiter bei (1).

Bemerkung: Die beiden durch die Komponente LOAD_GENERATOR benutzten (in MOSAIC global definierten) Prozeduren GENTASK und GENPMREQ besitzen folgende Bedeutung:

(a) procedure GENTASK (COMPID, TYPE, NUM, TSK);
/* Eingabeparameter: COMPID, TYPE, NUM.
Ausgabeparameter: TSK.
Aufgabe: Erzeugung einer Benutzertask.
Effekt: Eine Benutzertask (mit Nummer NUM und Typ TYPE) wird im Rechnernetzknoten mit

der globalen Nummer COMPID generiert, mit den benötigten Taskparametern versehen und aktiviert. TSK verweist auf die generierte Benutzertask. */

(b) procedure GENPMREQ (COMPID, TYPE, TSK, SPM, RECCOMP, SENDUSER);

/* Eingabeparameter: COMPID, TYPE, TSK, SPM, RECCOMP, SENDUSER.

Aufgabe: Erzeugung eines Schnittstellenauftrags.

Effekt: Ein Schnittstellenauftrag (Typ TYPE) wird im Rechnernetzknoten mit der globalen Nummer COMPID durch die Task TSK generiert, mit den benötigten Parametern versehen und evtl. gemeinsam mit einer Dateneinheit an den Sendeprotokollmodul SPM übergeben, wobei ggf. der Betriebsmittelverwaltungsinstanz ein "Interrupt" gesetzt wird. RECCOMP bzw. SENDUSER erlauben eine Identifikation des Korrespondenten bzw. des Benutzers des Protokollmoduls SPM und nehmen damit Bezug auf die angesprochene logische Verbindung. */

4.4. Einrichtungen zur Erfassung von Experimentdaten

Als Grundlage für die Auswertung von Simulationsexperimenten sind Meßeinrichtungen vorzusehen, die - in Analogie zu den Komponenten für die Erzeugung der Rechnernetzaufträge - während eines Experiments ebenfalls auf das Simulationsmodell einwirken. Welche Art von Information für den Experimentator von Interesse und daher zu erfassen ist, hängt stark vom konkreten Anwendungsfall ab. Für Simulationsstudien auf der Basis des Modellierungssystems MOSAIC ist primär die Vermessung solcher Variablen durchzuführen, die die Leistungsfähigkeit eines Rechnernetzes charakterisieren; deshalb erfassen die Meßeinrichtungen von MOSAIC u.a.:

- Bedienungs-, Warte- und Verweilzeiten von Benutzertasks, Schnittstellenaufträgen und Dateneinheiten;
- Warteschlangenlängen und Aktivzeiten bzw. Auslastungen von Bedienungsstationen (CPU's, Protokollmoduln, Übertragungskanälen u.a.);
- Speicherbelegungen (z.B. im Falle der Zwischenspeicherung von Dateneinheiten in Protokollmoduln).

Die Struktur der Meßeinrichtungen hat der Tatsache Rechnung zu tragen, daß zwei unterschiedliche Arten der Vermessung zu unterstützen sind: einerseits die periodische Vermessung, die nach Ablauf von vorgegebenen Zeitintervallen (evtl. unterschiedlicher Länge) durchgeführt wird und zum andern die ereignisgesteuerte Vermessung, die an das Eintreten bestimmter Ereignisse gekoppelt ist (z.B. das Einfügen eines Elements in eine zu analysierende Warteschlange).

Für die periodische Vermessung sind in MOSAIC die Modellkomponenten SNAPSHOT und PRINTDISC zuständig, die periodisch die aktuellen Werte interessierender Variablen erfassen bzw. berechnen und an einem Sichtgerät zur direkten Analyse durch den Experimentator (im Falle der PROCESS class SNAPSHOT) oder auf einen Externspeicher zur anschließenden statistischen Auswertung bzw. zur graphischen Aufbereitung (im Falle der PROCESS class PRINTDISC) ausgeben.

Die ereignisgesteuerte Vermessung ist derart realisiert, daß zu analysierende Benutzertasks und Dateneinheiten mit ihrem "Lebenslauf" versehen werden (z.B. mit Angaben über die Zeitpunkte des Eintreffens bei den bzw. des Verlassens der durchlaufenen Rechnernetzkomponenten) und die Elemente bei ihrer Vernichtung ausgewertet werden. Zu analysierende Bedienungsstationen werden durch dedizierte Vermessungsinstanzen ergänzt, die bei jeder Änderung einer zu vermessenden Variablen aktiv werden und den aktuellen Stand der Vermessungsinformation erneuern.

Durch Einführung der speziellen Hilfsmittel zum Aufbau eines ablauffähigen Simulationsmodells, zur Erzeugung der Modellbelastung und zur Modellvermessung stellt das Modellierungssystem MOSAIC nun sämtliche Komponenten bereit, die grundsätzlich für eine Modellauswertung durch rechnergestützte Simulation erforderlich sind. Die Einrichtungen zur Datenauswertung als weitere Hauptkomponente eines Modellierungssystems werden erst im nachfolgenden Kapitel detailliert, da sie einen Teil des dort beschriebenen Gesamtkonzepts zur Optimierung der Experimentdurchführung darstellen.

5. KONZEPTE ZUR EFFIZIENZERHÖHUNG BEI DER DURCHFÜHRUNG VON SIMULATIONS-EXPERIMENTEN UND IHRE REALISIERUNG IN MOSAIC

Die Notwendigkeit eines umfangreichen Experimentierens mit dem Simulator kennzeichnet den Verlauf eines Simulationsprojekts im Anschluß an die Modellimplementierung. Insbesondere lassen sich drei Phasen eines Simulationsprojekts definieren, die maßgeblich auf der Durchführung und Auswertung geeigneter Simulationsexperimente basieren:

(a) die Verifikationsphase,
(b) die Validationsphase,
(c) die Phase des konkreten Einsatzes des Simulators.

In der Verifikationsphase dienen Simulationsexperimente dem Nachweis der Korrektheit der Modellimplementierung in bezug auf die erfolgte Spezifikation.

In der Validationsphase erlauben die Experimente, nachzuprüfen, inwieweit - bei der Untersuchung eines (realen) Systems A - signifikante Verhaltensunterschiede zwischen dem Modell und dem zu untersuchenden System A in modellrelevanten Bereichen vorliegen. Manche Autoren /s. BEIL 73/ gehen von einer Zweiteilung der Validationsphase aus: eine Kalibrierungsphase, die der Reduktion der festgestellten Verhaltensunterschiede durch Modelländerung dient und eine sich anschließende Phase der Gültigkeitsbestätigung zur quantitativen Abschätzung der Abweichungen zwischen Modell- und realem Systemverhalten.

Die Phase der konkreten Anwendung von Experimentresultaten (Einsatz des Simulators) kann erst nach erfolgreicher Verifikation und Validation begonnen werden (s. Abb. 5.1.). Der Einsatz eines validierten Modells zur Simulation von Kommunikationsflüssen in Rechnernetzen kann z.B. darin bestehen, durch Simulationsexperimente unter verschiedenartigen, realistischen Randbedingungen iterativ zu einer Optimierung von Protokollparametern zu gelangen.

Aus der Vielzahl der im Laufe eines Simulationsprojekts anfallenden Experimente resultiert naturgemäß ein dringender Bedarf nach einer effizienten Methode für die Experimentdurchführung und -auswertung. Eine beträchtliche Aufwandsreduktion beim Experimentieren kann u.a. durch eine interaktive Arbeitsweise erzielt werden, die (nach Posch, Swik /POSC 79/) drei Grundforderungen an den Simulator impliziert:

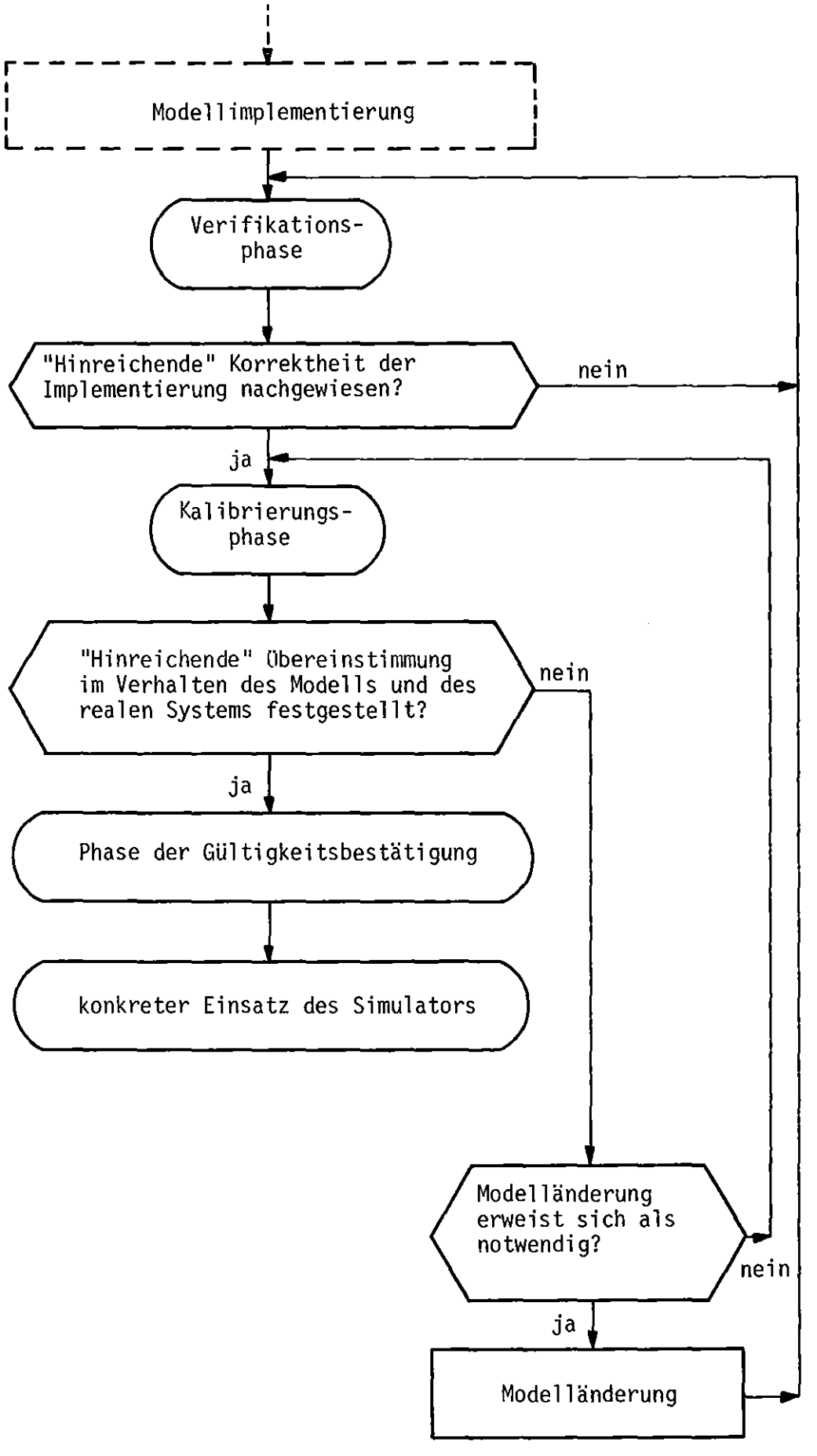

Abb. 5.1: Phasen eines Simulationsprojektes im Anschluß an die Modellimplementierung

(1) Kontrollierbarkeit von Simulationsablauf und Experimentparametern durch den Experimentator im Dialog (zur Vermeidung einer Vielzahl unnützer Simulationsexperimente);

(2) Präsentation der Experimentresultate in graphischer Form (mit Zeitgewinn bei der Resultatinterpretation durch den Experimentator);

(3) Änderung des Simulators im Dialog.

Diese Grundforderungen, die in existierenden Simulatoren zur Leistungsanalyse von Rechnernetzen selten und zumeist nur in Ansätzen berücksichtigt sind, werden durch MOSAIC weitgehend befriedigt [s. bereits Anforderungen (4), (5) an MOSAIC in Kapitel 4].

Zur Realisierung eines Experimentierkonzepts, das obigen Forderungen Rechnung trägt, wurden die graphischen Komponenten PLOTNET, PLOTQUEUES und PLOTDIAGRAM in MOSAIC integriert. Ferner wurde ein Vorschlag für die Organisation der Experimentdurchführung an einem Rechner mit Timesharing-Betrieb erarbeitet.

5.1. Graphische Komponenten zur Unterstützung einer interaktiven Experimentdurchführung mit MOSAIC

Die graphischen Komponenten, die MOSAIC enthält, dienen

(a) der Vorgabe der Struktur des zu untersuchenden Rechnernetzmodells durch den Experimentator,

(b) der Ausgabe der aktuellen Warteschlangenbelegungen im Rechnernetzmodell zu einem beliebigen Zeitpunkt während des Simulationsexperiments,

(c) der Ausgabe der gemessenen Ergebnisvariablen.

Die Implementierung der Graphikkomponenten erfolgte bislang für TEKTRONIX-Bildschirmgeräte aus der Serie 4010, die insbesondere wegen ihres weit verbreiteten Einsatzes ausgewählt wurden. Als Implementierungssprache wurde FORTRAN verwendet, das nahezu als Standardsprache für die Erstellung von Graphiksoftware angesehen werden kann und ferner mit dem Vorteil einer weitgehenden Rechnerunabhängigkeit ausgestattet ist. Als Anschlußmöglichkeit für die graphischen Komponenten ("Graphik-Teil") an den in SIMULA implementierten "Modell-Teil" von MOSAIC bot sich die Benutzung externer Prozeduren an (s. Grobstruktur der Implementierung von MOSAIC in Kapitel 4). Durch diese Art der Strukturierung wurde eine klare Abgrenzung zwischen dem "Gra-

phik-Teil" und dem "Modell-Teil" in MOSAIC erzielt.

(a) Interaktive, graphische Eingabe der Topologie des Rechnernetzmodells

Diese Aufgabe wird realisiert durch die FORTRAN-Subroutine PLOTNET, die aus der Eingabe des Experimentators am TEKTRONIX-Display ein graphisches Rechnernetzmodell aufbaut.

Wirkungsweise der

procedure PLOTNET

/* Eingabeparameter: - Anzahl der Arbeits- und Vermittlungsrechner,
- Anzahl der Protokollschichten,
- Identifikation der Rechnernetzknoten und ihre Positionierung auf dem Bildschirm,
- Verbindungen zwischen den Rechnernetzknoten.

Aufgabe: Darstellung der Rechnernetztopologie mit Hilfe der Eingabeparameter, die der Experimentator im Dialog mit den aktuellen Werten versorgt.

Effekt: Eine idealisierte Struktur des zu untersuchenden Rechnernetzes wird am Bildschirm aufgebaut (Beispiel für die Darstellung eines existierenden Rechnernetzes s. Abb. 6.1. in Kapitel 6), und die erhaltenen Eingabedaten werden auf einer Externdatei (DISC) zwischengespeichert und können so durch den Konfigurator sowie durch die Prozedur PLOTQUEUES weiterverwendet werden. */

(b) Ausgabe der Warteschlangenbelegungen im Rechnernetzmodell

Für die Darstellung des Dateneinheitenflusses steht die FORTRAN-Subroutine PLOTQUEUES zur Verfügung, die - in Analogie zu PLOTNET - das Rechnernetzmodell in graphischer Form wiedergibt und darüber hinaus die Belegung der Protokollmodulwarteschlangen durch Dateneinheiten angibt. Die Information zur Beschreibung der Struktur des Rechnernetzmodells erhält PLOTQUEUES aus der Externdatei DISC und die Angaben bzgl. der Belegung der Protokollmodulwarteschlangen über die Meßeinrichtungen von MOSAIC.

Wirkungsweise der

procedure PLOTQUEUES

/* Eingabeparameter: - sämtliche Eingabeparameter der Prozedur PLOTNET,
- rechnerspezifische Anzahl von Dateneinheiten, die innerhalb derselben Protokollschicht auf Be-

arbeitung durch Sendeprotokollmoduln (d.h. auf Ausgabe) bzw. durch Empfangsprotokollmoduln (d.h. auf Eingabe) warten.

Aufgabe: Darstellung der Rechnernetztopologie und sämtlicher Dateneinheiten, die sich zum aktuellen Simulationszeitpunkt in den verschiedenen Protokollschichten des Rechnernetzes befinden.

Effekt: Das zu Beginn des Simulationsexperiments durch PLOTNET aufgebaute graphische Rechnernetzmodell wird am Bildschirm dargestellt unter Berücksichtigung der aktuellen Belegung der Protokollschichten durch Dateneinheiten. (Die Darstellung der Protokollhierarchie erfolgt in Analogie zu Abb. 6.1. in Kapitel 6). */

(c) Ausgabe des zeitlichen Verlaufs von Ergebnisvariablen während eines beliebigen Zeitintervalls des Simulationsexperiments

Zur Charakterisierung des Verlaufs eines Simulationsexperiments ist es notwendig, die zeitliche Entwicklung der experimentrelevanten Variablen aufzuzeichnen. Zu diesem Zweck enthält das Modellierungssystem MOSAIC die bereits in Kapitel 4 eingeführten Meßeinrichtungen.

Eine Weiterverarbeitung der durch die Meßeinrichtungen erfaßten Meßdaten (s. Abb. 4.1.) hat aus mehreren Gründen zu erfolgen: Zum einen sind die erhaltenen Werte in geeigneter Weise aufzubereiten, um dem Experimentator eine vereinfachte Interpretation der Daten zu ermöglichen; zum andern ist eine statistische Aufbereitung der Daten unerläßlich (z.B. um die durch die Experimentdurchführung korrelierten Werte in eine unabhängige Stichprobe umzuwandeln).

Für die graphische Aufbereitung der Meßdaten beinhaltet MOSAIC die FORTRAN-Subroutine PLOTDIAGRAM. Neben direkten Meßdaten (beispielsweise die dynamische Entwicklung der Speicherbelegung für einen bestimmten Speicherbereich) können überdies bereits statistisch aufbereitete Werte (z.B. durchschnittliche Belegung eines Speicherbereichs) in Form von Schaubildern dargestellt werden.

Wirkungsweise der

procedure PLOTDIAGRAM

/* Eingabeparameter: Text zur Beschriftung der Graphik; Länge der Zeitachse; Anzahl der im selben Schaubild darzustellenden Variablen; für jede dieser Variablen: Zeitintervall zwischen zwei aufeinanderfolgenden Meßpunkten und die Werte für sämtliche Meßpunkte.

Aufgabe: Darstellung einer (oder mehrerer überlagerter) Variablen in Abhängigkeit von der Simulationszeit unter Verwendung linearer Interpolation zwischen den, für diskrete Zeitpunkte gegebenen, Variablenwerten.

Effekt: Ein Achsenkreuz mit Beschriftung wird erstellt und der Verlauf der darzustellenden Variablen wird über der Simulationszeit aufgetragen (PLOTDIAGRAM wird aktuell z.B. benutzt für die Darstellung von Rechnerauslastungen, Dateneinheitenverweilzeiten, -durchsatz, etc.). */

Sämtliche der durch die Meßeinrichtungen erfaßten Variablen, insbesondere die Leistungskenngrößen, können mit Hilfe von PLOTDIAGRAM dargestellt werden.

5.2. Gesamtkonzept für die interaktive Experimentdurchführung mit MOSAIC

Die durch ein interaktives Experimentieren ermöglichte gezielte Vorgehensweise führt i.a. zu Verkürzungen bzw. zur Reduktion der benötigten Anzahl von Experimentläufen und demzufolge zu beträchtlichen Einsparungen an Rechenzeit.

Als konkrete Vorteile interaktiver Experimentdurchführung sind u.a. zu nennen:

- Verwendbarkeit aktueller Resultate als Entscheidungsgrundlage für die Steuerung des weiteren Experimentablaufs,
- Möglichkeit der Beobachtung des dynamischen Verlaufs von experimentrelevanten Modellvariablen,
- vereinfachte Lokalisierung von Fehlern während der Phase der Modellverifikation durch detaillierten Einblick in den Ablauf eines Experiments.

Unter Verwendung der bislang eingeführten Komponenten des Modellierungssystems MOSAIC soll nun ein Gesamtkonzept zur Organisation interaktiver Simulationsexperimente vorgestellt werden. Diese Methode basiert darauf, daß

der gesamte Simulationszeitraum durch den Experimentator on-line in i_0 verschiedene, ggf. unterschiedlich lange Zeitintervalle $[t_{i-1}, t_i]$ $i=1,\ldots,i_0$ zerlegt wird. Nach Ablauf eines jeden Intervalls, d.h. zu den Simulationszeitpunkten t_i $(i=1,\ldots,i_0)$, erhält der Experimentator die volle Kontrolle über das Simulationsexperiment. Die Unterbrechungszeitpunkte werden derart definiert, daß t_0 mit dem Experimentbeginn gleichgesetzt wird und zu jedem der Zeitpunkte t_i $(i=0,\ldots,i_0-1)$ der nächste Unterbrechungszeitpunkt evtl. unter Berücksichtigung der aktuellen Experimentresultate festgelegt wird.

Auf diese Weise ergibt sich der folgende dynamische Ablauf eines Experiments:

(a) Simulationszeitpunkt $t = t_0 = 0$ (Experimentbeginn):

- unter Verwendung eines vordefinierten Eingabedatensatzes baut die MOSAIC-Komponente "Konfigurator" aus den "Modellbausteinen" das ablauffähige Simulationsprogramm auf; der Experimentator hat dabei die Möglichkeit, den gewählten Eingabedatensatz durch zusätzliche komprimierte oder ausführliche Eingabe (am TEKTRONIX-Display) zu ergänzen und - falls erwünscht - eine graphische Definition der Rechnernetztopologie mit Hilfe der Prozedur PLOTNET durchzuführen;
- zur Kontrolle der Eingabe durch den Experimentator können die wesentlichen Charakteristika des erstellten Rechnernetzmodells auf Wunsch am Bildschirm ausgegeben werden;
- bei korrekten Eingabedaten erfolgt die Definition des ersten Unterbrechungszeitpunktes t_1 und eine Steuerung des Experimentablaufs im Zeitraum $[0, t_1]$.

(b) während des Zeitintervalls $[t_{i-1}, t_i]$ $(i = 1,\ldots,i_0)$:

- der "Auftragsgenerator" wirkt durch das Erzeugen der (Rechnernetz-) Aufträge auf das Modell ein;
- die "Meßeinrichtungen" erfassen Experimentdaten zur Beschreibung des Modellverhaltens;
- durch den Experimentator angeforderte Informationen über den Ablauf des Simulationsexperiments (z.B. Trace) werden dynamisch am Bildschirm ausgegeben;
- das Simulationsexperiment kann zu jedem Zeitpunkt beendet werden.

(c) zu den Simulationszeitpunkten $t = t_i$ $(i = 1,\ldots,i_0)$:

- durch den Experimentator angeforderte aktuelle Experimentresultate werden in graphischer Form ausgegeben (z.B. der Momentanzustand des Rechnernetzmodells durch die Prozedur PLOTQUEUES oder der zeitliche Verlauf von Experimentvariablen durch die Prozedur PLOTDIAGRAM);
- das Simulationsexperiment kann beendet werden;
- die Möglichkeit der Änderung von Simulationsparametern (z.B. die Modellbelastung betreffend) besteht;
- der nächste Unterbrechungszeitpunkt t_{i+1} ist zu definieren (falls das Experimentende noch nicht erreicht wurde), und eine Steuerung des Experimentablaufs im Zeitraum $[t_i, t_{i+1}]$ wird vorgenommen.

Es erscheint sinnvoll, die graphische Aufbereitung der Resultate während des Experiments durchzuführen, wohingegen die statistische Auswertung der Meßdaten am vorteilhaftesten nach Beendigung des Simulationsexperiments erfolgt. Zur statistischen Auswertung von Meßdaten wurden die in /DROB 74/ erläuterten - in SIMULA implementierten - Prozeduren in MOSAIC eingebettet. Diese Prozeduren zur Stichprobenauswertung lassen sich nach Bedarf zur Berechnung von Mittelwerten, Varianzen, Vertrauensintervallen u.ä. einsetzen.

6. EINSATZ VON MOSAIC ZUR LEISTUNGSANALYSE UND -OPTIMIERUNG EXISTIERENDER RECHNERNETZE

6.1. Erfahrungen bei der Entwicklung der Grundausbaustufe von MOSAIC

Das durch die Implementierung der konzeptionellen Modelle zur Gesamtbeschreibung eines Rechnernetzes entstandene Modellierungssystem MOSAIC liegt seit Ende 1978 in einer ausgetesteten und zuverlässig arbeitenden ersten Ausbaustufe - dem MOSAIC-Grundmodell - vor. Eine intensive Modellverifikation für das MOSAIC-Grundmodell konnte durch eine Vielzahl gezielter Simulationsexperimente unter verschiedenartigen Randbedingungen erreicht werden. Die Arbeiten zur Verifikation wurden durch das in Kapitel 5 vorgestellte Konzept der interaktiven Experimentdurchführung erleichtert und insbesondere durch Einsatz folgender Hilfsmittel unterstützt:

- Ausdruck der Eingabedaten zu Experimentbeginn zur Kontrolle der Daten durch den Experimentator,
- Ausgabe des Momentanzustands der wichtigsten Komponenten des Rechnernetzmodells zu beliebigen Simulationszeitpunkten,
- Protokollierung (Trace) der wichtigsten Ereignisse im Simulationsexperiment während eines beliebigen Zeitintervalls,
- Ausgabe eines vollständigen "Lebenslaufs" der Dateneinheiten und Benutzertasks zum Zeitpunkt ihrer Vernichtung,
- "Stillegung" des Auftragsgenerators, um zu erkennen, ob die restlichen Aufträge vollständig abgearbeitet werden und das Rechnernetzmodell in seinen Ruhezustand zurückkehrt,
- Verwendung von Prozeduren, um während des Simulationsexperiments die Konsistenz von Protokollzuständen für Korrespondenten zu überprüfen,
- Betrachtung stark vereinfachter Spezialfälle, für die eine analytische Lösung oder zumindest eine Approximation existiert, um die analytischen Resultate zur Kontrolle der Modellresultate heranzuziehen.

Seit der Fertigstellung des MOSAIC-Grundmodells wird es zu Effizienzstudien für geplante bzw. in Realisierung befindliche Rechnernetze eingesetzt. Diese Phase der Erprobung von MOSAIC soll dazu dienen,

- Entscheidungshilfen für die Planung bzw. Implementierung der zu untersuchenden Rechnernetze zu geben (z.B. durch Vergleich von Entwurfsalternativen mit Hilfe des Simulators),

- zusätzliche Modellbausteine (z.B. für standardisierte Kommunikationsprotokolle) in MOSAIC zu integrieren, um das Anwendungsspektrum von MOSAIC weiter auszudehnen,
- Erfahrungen auf dem bislang noch unzureichend erforschten Gebiet der Modellvalidation zu sammeln.

Erste Vorläufer der in dieser Arbeit präsentierten Modelle waren bereits für die Leistungsanalyse des HMINET eingesetzt worden und hatten zu einem validierten Simulator für das HMINET geführt (s. hierzu Resultate in /WOLF 77/); daneben hatten diese Modelle die Leistungsanalyse von Realisierungsvarianten für ein Protokoll zur dezentralisierten Kontrolle in Rechnernetzen erlaubt /s. WOLF 78b/.

Zum aktuellen Zeitpunkt erfolgt eine Erprobung für MOSAIC - neben dem Einsatz bei einem deutschen DV-Hersteller - vornehmlich innerhalb eines Auftrags zur Durchführung einer Simulationsstudie für das BERNET-Rechnernetz[+].

6.2. Erfahrungen beim Einsatz von MOSAIC zur Simulation der Kommunikationsflüsse des BERNET-Rechnernetzes

Das BERNET-Rechnernetz wird zur Zeit in Berlin installiert, um den Zugriff auf alle Dienstleistungen der Berliner wissenschaftlichen Rechenzentren zu ermöglichen. Im BERNET-Rechnernetz sind Großrechner (Arbeitsrechner) unterschiedlicher Hersteller über ein Paketvermittlungsnetz der Deutschen Bundespost untereinander verbunden. Bei dem Paketvermittlungsnetz handelt es sich um einen einzigen Rechnernetzknoten, der Paketvermittlung auf der Basis des Kommunikationsprotokolls X.25, s. /CUNN 77/, durchführt. Für den Anschluß der Großrechner an das Paketvermittlungsnetz werden Prozeßrechner vom Typ AEG 80-40 als Netzeingangsprozessoren (NEP) verwendet. (Abb. 6.1. zeigt die geplante Konfiguration der ersten Netzausbaustufe von BERNET; zur Erstellung der Abbildung wurde die MOSAIC-Prozedur PLOTNET, s. Abschnitt 5.1., herangezogen).

[+] Im Rahmen eines Kooperationsvertrages zwischen dem Institut für Datenverarbeitung in der Technik/Kernforschungszentrum Karlsruhe und dem Land Berlin wird das in dieser Arbeit vorgestellte Modellierungssystem MOSAIC aktuell dazu herangezogen, eine detaillierte Simulation der Kommunikationsflüsse im BERNET-Rechnernetz durchzuführen.

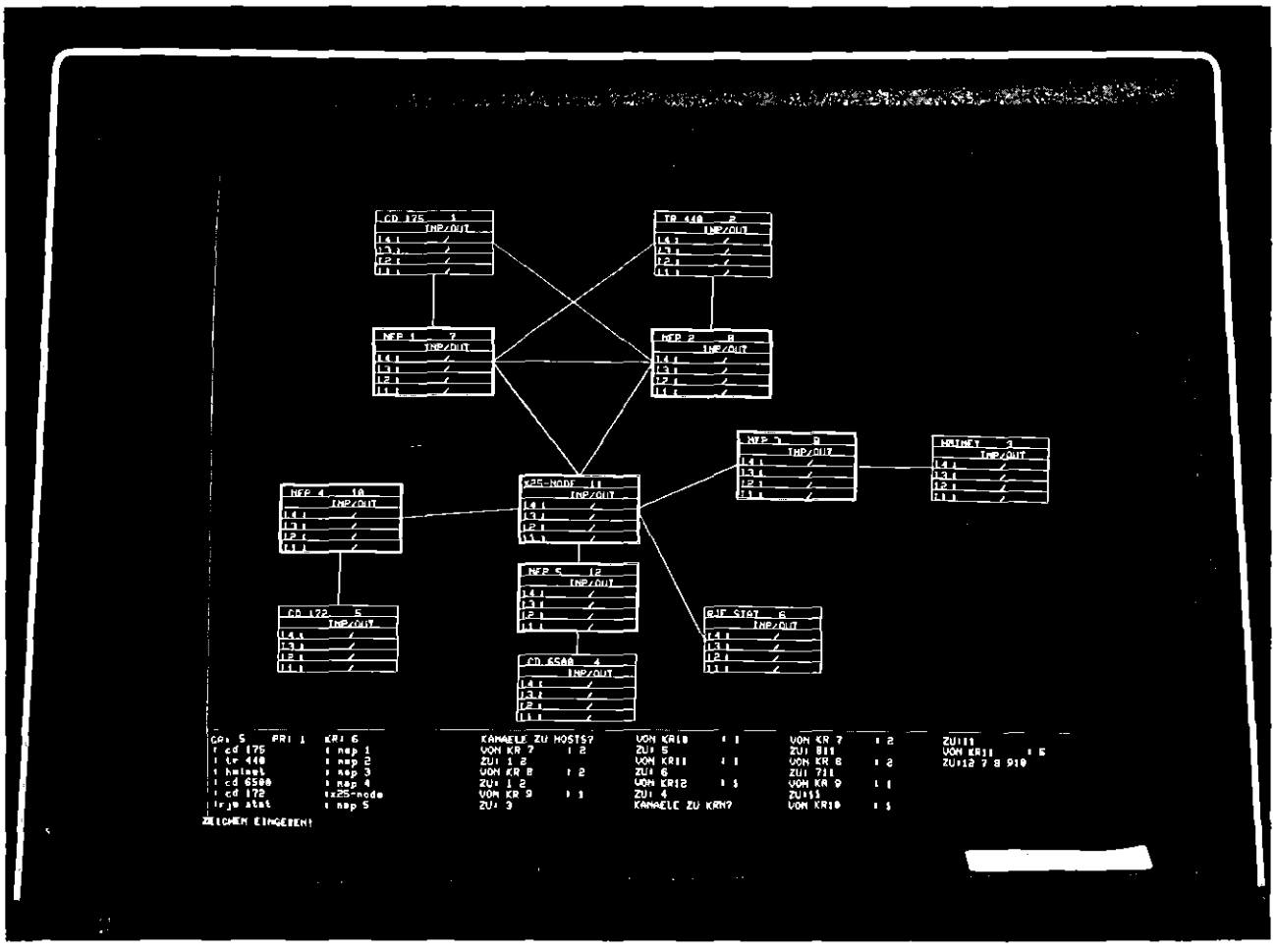

Abb. 6.1.: Graphische Darstellung der Konfiguration des BERNET-Rechnernetzes durch MOSAIC

[Notation: L1,...,L4 bezeichnet die Schichten der Protokollhierarchie; Rechnernetzknoten werden durch Rechtecke, Übertragungsleitungen durch direkte Verbindungslinien zwischen Rechnernetzknoten dargestellt]

Als Kommunikationsprotokolle werden in BERNET weitgehend standardisierte Protokolle eingesetzt, wodurch sich das Interesse an einer Untersuchung dieses Rechnernetzes natürlich erhöht. (Zu einer eingehenden Systemanalyse des BERNET-Rechnernetzes sei auf /STRA 78/ verwiesen).

Eines der wichtigsten Probleme bei der aktuellen Implementierung des BERNET-Rechnernetzes ist die Dimensionierung der Kommunikationsprotokolle und der Netzeingangsprozessoren (z.B. optimale Festlegung von Protokollparametern, von Speicherbereichen etc.). Um diesbezügliche Aussagen zu erhalten, wird ein Simulator benötigt, der eine integrierte Betrachtungsweise für die gesamte Hierarchie der Kommunikationsprotokolle erlaubt und überdies - zumindest für die Netzeingangsprozessoren - auch die Betriebsmittelzuteilung innerhalb der Rechnernetzknoten modelliert.

Ausgehend von dem MOSAIC-Grundmodell wurde durch entsprechende Festlegung der Eingabedaten und durch Modellerweiterungen (die Hauptarbeit bestand hier in der Anpassung der im Grundmodell implementierten Kommunikationsprotokolle an die in BERNET verwendeten Protokolle) eine MOSAIC-Version erzeugt, die auf die Leistungsanalyse des BERNET-Rechnernetzes zugeschnitten ist. Das resultierende Modellierungssystem soll als MOSAIC (BERNET) bezeichnet werden.

Wegen der Modellerweiterungen mußte das Modellierungssystem MOSAIC (BERNET) nochmals verifiziert werden. In erster Linie erstreckte sich diese Verifikation auf die in MOSAIC (BERNET) implementierten Kommunikationsprotokolle. Beispielsweise wurden zur Verifikation der HDLC-Implementierung die Randbedingungen der in /BUX 79/ veröffentlichten HDLC-Studie approximiert und das Modellverhalten (Durchsatz, Verweilzeiten) unter diesen Randbedingungen analysiert. Die relativen Resultat-Abweichungen zwischen der MOSAIC-Implementierung und der in /BUX 79/ angegebenen Implementierung des Kommunikationsprotokolls HDLC betrugen bei sämtlichen Vergleichen weniger als 5%.

Nach Abschluß der Verifikation konnte unter Benutzung erster Messungen am BERNET-Rechnernetz (z.B. Laufzeiten von Kommunikationssoftware in den Netzeingangsprozessoren) bereits mit der Phase der Modellkalibrierung begonnen und das zeitliche Verhalten des Modellierungssystems MOSAIC (BERNET) an das beobachtete Verhalten des BERNET angepaßt werden. Bei dieser ersten Kalibrierung wurden nochmals geringfügige Modelländerungen notwendig, die jedoch die obigen Resultate der Modellverifikation nicht betrafen, so daß diese Verifikation keiner Wiederholung bedurfte. Bei Gewinnung weiterer Meßdaten wird der iterative Vorgang der Modellvalidation fortgesetzt werden. Als Voraussetzung für die Erfassung weiterer Meßdaten sind allerdings die bislang in BERNET nur sehr spärlich vorhandenen Meßinstrumente (zur Meßdatenerfassung im realen Rechnernetz) noch erheblich auszubauen.

Als Anwendungsbeispiele für das Modellierungssystem MOSAIC wollen wir zwei verschiedenartige Serien von Pilotexperimenten EXP_1 und EXP_2 zu einer Effizienzanalyse für das Kommunikationssubsystem des BERNET-Rechnernetzes durchführen. Bei dem Kommunikationssubsystem handelt es sich in BERNET um diejenige Teilmenge des Rechnernetzes, die entsteht, wenn ausschließlich die Netzeingangsprozessoren und das Paketvermittlungsnetz (X.25-NODE) betrachtet werden. Die Untersuchung wurde auf das Kommunikationssubsystem begrenzt, da bislang Messungen bzgl. der Anschlüsse zu den Arbeitsrechnern nicht zugänglich

sind, die zur Festlegung der Modellparameter und insbesondere zur Modellvalidation benötigt werden.

Die Simulationsexperimente setzen ein Rechnernetz mit r ($r \geq 2$) Netzeingangsprozessoren $NEP_1,\ldots,NEP_r$ voraus, die allesamt mit einem zentralen Vermittlungsrechner X.25-NODE über fehlerfreie Vollduplex-Leitungen verbunden sind. Unter Benutzung der Beschreibungshilfsmittel, die in dieser Arbeit u.a. eingeführt wurden, zeigt Abb. 6.2. die Protokollhierarchie in den Rechnernetzknoten NEP_1, X.25-NODE und NEP_r.

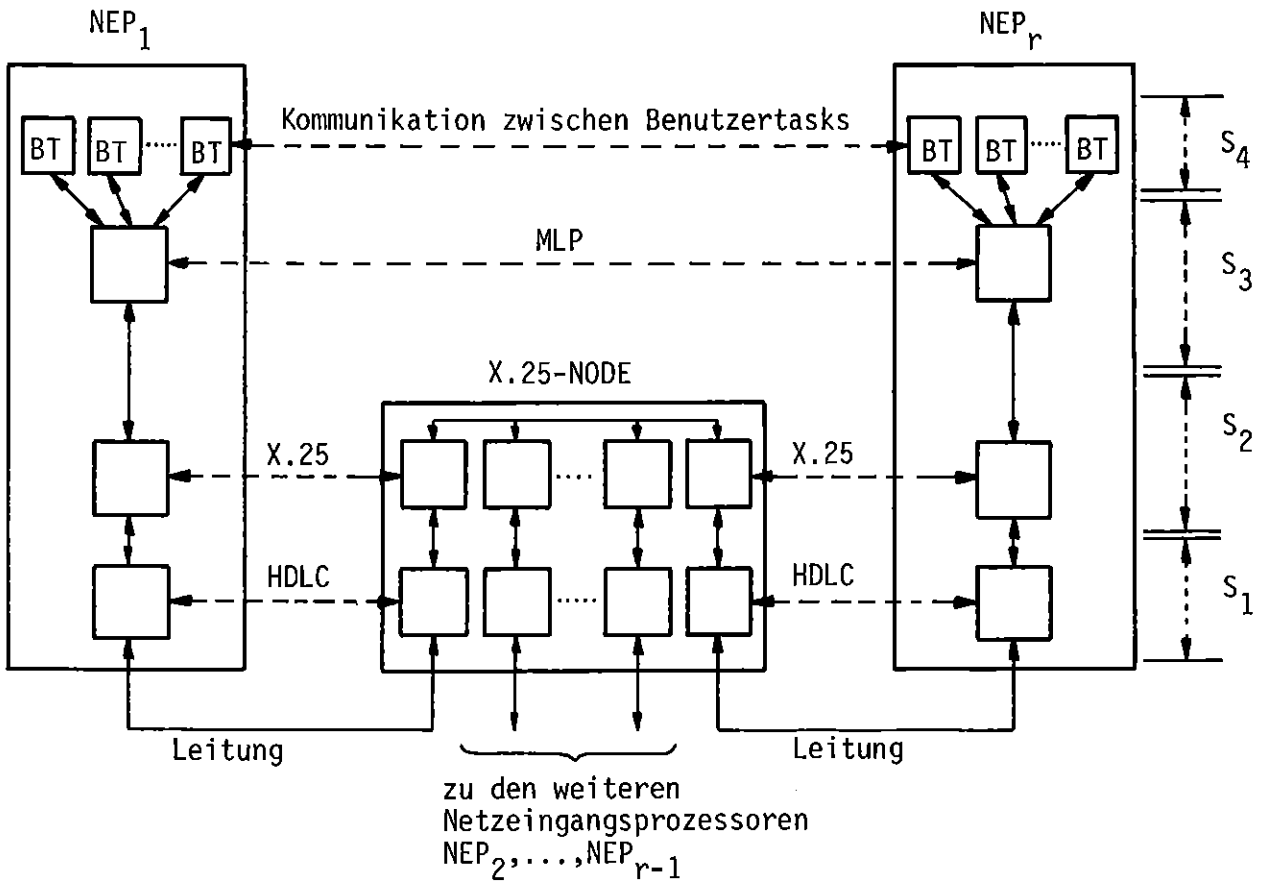

Abb. 6.2.: Struktur der Kommunikationssoftware für die BERNET-Experimentserien ("Protokolleinheiten-Sicht")

[Notation: BT = kommunizierende Benutzertask
MLP = Message Link Protokoll (s. /HERT 78/)
X.25 = X.25-Protokoll (Level 3) (s. /CCITT 77/)
HDLC = HDLC-Protokoll (s. /ISO 77/)
$S_1,\ldots,S_4$ = Schichten der Protokollhierarchie]

Die Kommunikationsprotokolle HDLC, X.25 und MLP beschränken sich in sämtlichen Experimenten auf die Datenaustauschphase. Für Sendevorgänge wird bei HDLC eine Fenstergröße von 7, bei X.25 eine Fenstergröße von 2 gewählt. Quittungen innerhalb der Schicht des Message Link Protokolls werden nicht generiert; hingegen quittieren die Schichten S_1 und S_2 jede erhaltene Dateneinheit. Die Zeiten für die Abwicklung der Kommunikationssoftware werden entsprechend den bisherigen Messungen im BERNET-Rechnernetz gewählt. Innerhalb der Schicht S_4 treten keine Verzögerungen für Dateneinheiten auf. Die zu übertragenden Dateneinheiten (Messages), die innerhalb der Schicht S_3 auszutauschen sind, haben eine konstante Länge von 1[KBit]. In jeder Protokollschicht S_j wird die Dateneinheitenlänge um die Länge der in S_j hinzugefügten Kontrollinformation erhöht. Für die prioritätsmäßige Einbettung der Protokolleinheiten in die Rechnernetzknoten wird vorausgesetzt, daß die niedrigsten Schichten die höchste Priorität besitzen. Das Betriebssystem der Netzeingangsprozessoren findet im Modell seine Berücksichtigung z.B. durch prioritätsgesteuerte Betriebsmittelzuteilung an Tasks und durch die näherungsweise Nachbildung des Paging-Verhaltens. Hingegen wird der Paketvermittlungsrechner X.25-NODE weniger detailliert modelliert; dennoch wird auch für diesen Rechnernetzknoten die gesamte Abwicklung der Kommunikationsprotokolle nachvollzogen.

(a) Experimentserie EXP_1

Ziel dieser Experimentserie war eine Untersuchung des maximalen (Netto-) Durchsatzes, der - unter BERNET-Randbedingungen - durch jede der Protokollschichten S_j ($j = 1,2,3$) für die Gesamtheit der Nachbarn auf Schicht S_{j+1} erzielt werden kann. Mit anderen Worten ging es somit um eine Quantifizierung des Effizienzverlustes pro Schicht (für jede der Protokollschichten S_1,S_2 und S_3).

Die Experimentserie EXP_1 wurde zunächst durchgeführt für unterschiedliche Leitungskapazitäten. Dabei wurden Leitungsgeschwindigkeiten gewählt, die durch die Deutsche Bundespost (aktuell oder in naher Zukunft) angeboten werden. Die Untersuchungen beschränken sich auf den Fall $r = 2$ (s. obige allgemeine Randbedingungen), d.h. auf zwei Netzeingangsprozessoren, die über den Rechnernetzknoten X.25-NODE kommunizieren. Es wurde generell Datentransfer in beiden Richtungen vorausgesetzt.

Der maximale Durchsatz durch die Übertragungsleitungen ist bereits durch die Leitungsgeschwindigkeit gegeben und entspricht der doppelten Leitungskapazität wegen der vorausgesetzten bidirektionalen Übertragung.

Der maximal erreichbare Durchsatz bis einschließlich Schicht S_1 wurde ermittelt durch die Analyse der Leitungsschicht bei der Kommunikation zwischen dem Netzeingangsprozessor und dem X.25-NODE unter Voraussetzung des Protokolls HDLC.

Der maximal erreichbare Durchsatz bis einschließlich Schicht S_2 wurde ermittelt durch Analyse des Protokolls X.25 unter Einbeziehung der Leitungsschicht, wobei die gesamte Kommunikation über genau eine logische Verbindung (d.h. einen permanenten "Virtual Circuit", s. /CCITT 77/) abgewickelt wurde.

Der maximal erreichbare Durchsatz bis einschließlich Schicht S_3 wurde ermittelt durch Analyse des Message Link Protokolls unter Miteinbeziehung der Schichten S_1 und S_2, wobei auch hier die gesamte Kommunikation über eine logische Verbindung (d.h. einen "Message Link", s. /HERT 78/) stattfindet.

Die Resultate dieser Experimente sind in den Abbildungen 6.3. und 6.4. dargestellt.

Die Simulationsresultate (maximal erreichbarer Durchsatz und prozentuale Auslastung der Übertragungskapazität) zeigen einen beträchtlichen Durchsatzrückgang bzw. Effizienzverlust in den Protokollschichten S_2 und S_3. Die Resultate unterstreichen ferner, daß unter den aktuellen Randbedingungen in BERNET eine Erhöhung der momentanen Leitungskapazität von 9.6 [KBit/sec] auf 19.2 oder gar 48.0 [KBit/sec] kaum bis über die Leitungs-Schicht durchschlägt.

Auf der Suche nach einer Erklärung dieser Resultate konnte der Hauptspeicher in den Netzeingangsprozessoren als maßgeblicher Engpaß identifiziert werden. Für die Experimente war - wie es den aktuellen Verhältnissen im BERNET-Rechnernetz entspricht - ein Hauptspeicherausbau von 256 [KByte] für die Netzeingangsprozessoren vorausgesetzt worden, woraus der beobachtete Speicherengpaß resultierte.

Aus diesem Grunde wurden die obigen Experimente für einen Hauptspeicherausbau von 512 [KByte] wiederholt. Die Folge war ein beachtlicher Anstieg des maximalen Durchsatzes, der durch jede der Protokollschichten erzielt werden kann (Resultate vgl. ebenfalls in Abb. 6.3. und Abb. 6.4.).

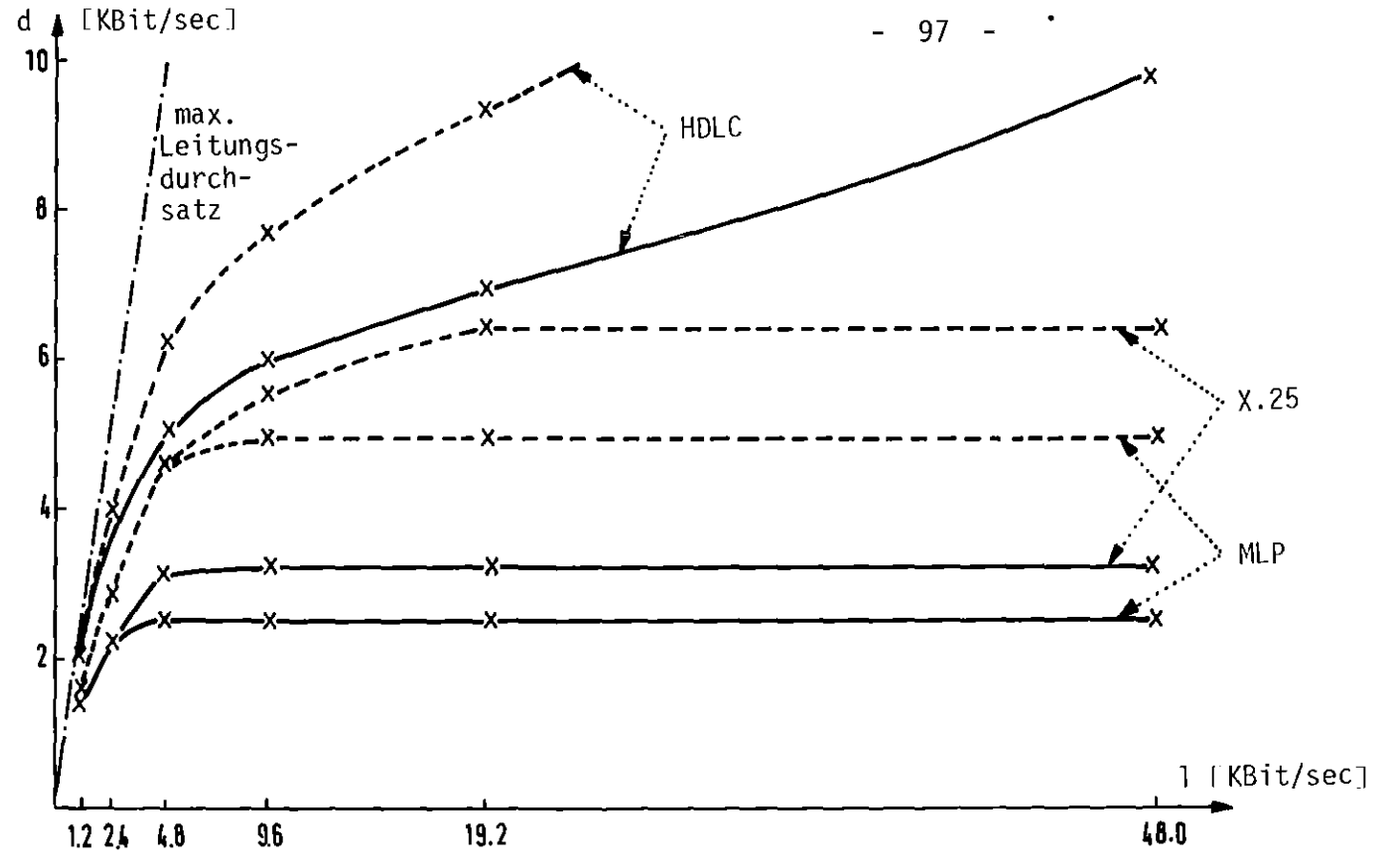

Abb. 6.3.: Abhängigkeit des maximalen Durchsatzes d [KBit/sec] von der Leitungsgeschwindigkeit l [KBit/sec] für verschiedene Schichten der Protokollhierarchie im BERNET-Rechnernetz
(—— Hauptspeicherausbau der Netzeingangsprozessoren: 256 [KByte]
---- Hauptspeicherausbau der Netzeingangsprozessoren: 512 [KByte])

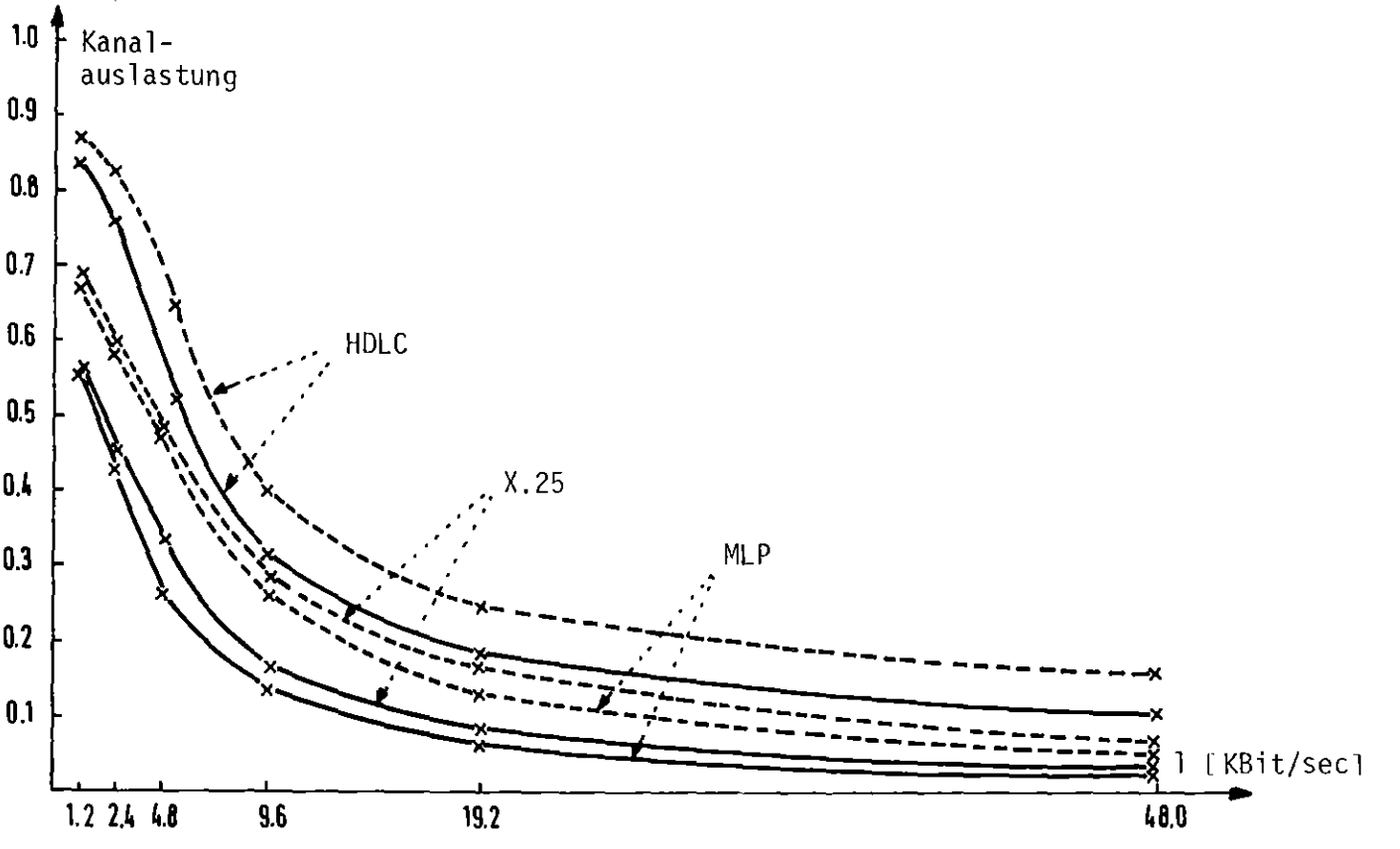

Abb. 6.4.: Abhängigkeit der prozentualen Auslastung des Übertragungskanals von der Leitungsgeschwindigkeit l [KBit/sec] bei maximalem Durchsatz für verschiedene Schichten der Protokollhierarchie im BERNET-Rechnernetz
(—— Hauptspeicherausbau der Netzeingangsprozessoren: 256 [KByte]
---- Hauptspeicherausbau der Netzeingangsprozessoren: 512 [KByte])

Im übrigen wurde auch versucht, durch Benutzung mehrerer paralleler logischer Verbindungen den Durchsatz zu erhöhen. Durch diesen Ansatz konnte allerdings nahezu keine Effizienzsteigerung erreicht werden (relativer Durchsatzgewinn < 4%).

Die Serie der Pilotexperimente EXP_1 legt somit den Schluß nahe, daß sich unter den aktuellen Randbedingungen eine Erhöhung des Durchsatzes durch eine Erhöhung der Leitungskapazität kaum erreichen läßt, wohingegen ein Hauptspeicherausbau signifikanten Durchsatzgewinn verspricht.

(b) Experimente EXP_2

Nachdem sich die Untersuchungen der Experimentserie EXP_1 in erster Linie auf eine Durchsatzanalyse bezogen, soll durch die Experimentserie EXP_2 auch die Leistungskenngröße "Dateneinheitenverweilzeit" analysiert werden. Generelles Ziel der Experimente in EXP_2 ist die Suche nach derjenigen Belastung des BERNET-Rechnernetzes, die zu "optimalem" Verhalten des aktuellen Kommunikationssubsystems in BERNET führt. Hierzu definieren wir als Funktion der Rechnernetzbelastung λ den Quotienten

$$y(\lambda) := \frac{\text{Durchsatz}}{\text{mittlere Dateneinheitenverweilzeit}} \quad [\text{KBit/sec}^2]$$

und versuchen, das Maximum der Funktion $y(\lambda)$ zu bestimmen. (Bei der Definition der Funktion $y(\lambda)$ werden zwei Optimierungskriterien eines Rechnernetzes, nämlich der Wunsch nach hohem Durchsatz als auch nach geringer Dateneinheitenverweilzeit, kombiniert).

Für sämtliche Experimente der Serie EXP_2 wurden - neben den allgemeinen Randbedingungen, die bereits für EXP_1 galten - die speziellen Voraussetzungen getroffen, daß $r = 5$ Netzeingangsprozessoren betrachtet werden, die Kapazität aller Leitungen 9.6 [KBit/sec] betrage, pro Netzeingangsprozessor eine logische Verbindung existiere und ein Netzeingangsprozessor jeden anderen mit derselben Wahrscheinlichkeit als Zielrechner ermittelt. Unter der zusätzlichen Voraussetzung, daß jeder Netzeingangsprozessor mit 256 [KByte] Hauptspeicher ausgestattet sei, wurden durch MOSAIC (BERNET) für unterschiedliche Rechnernetzbelastungen die in Abb. 6.5. dargestellten mittleren Verweilzeiten pro Dateneinheit innerhalb der Schichten S_1 bis S_3 gemessen. Für die Funktion $y(\lambda)$ ergab sich der in Abb. 6.6. dargestellte Verlauf mit einem Maximum bei $\lambda \approx 3$ [sec^{-1}]. (λ wird bezogen auf das gesamte Rechnernetz).

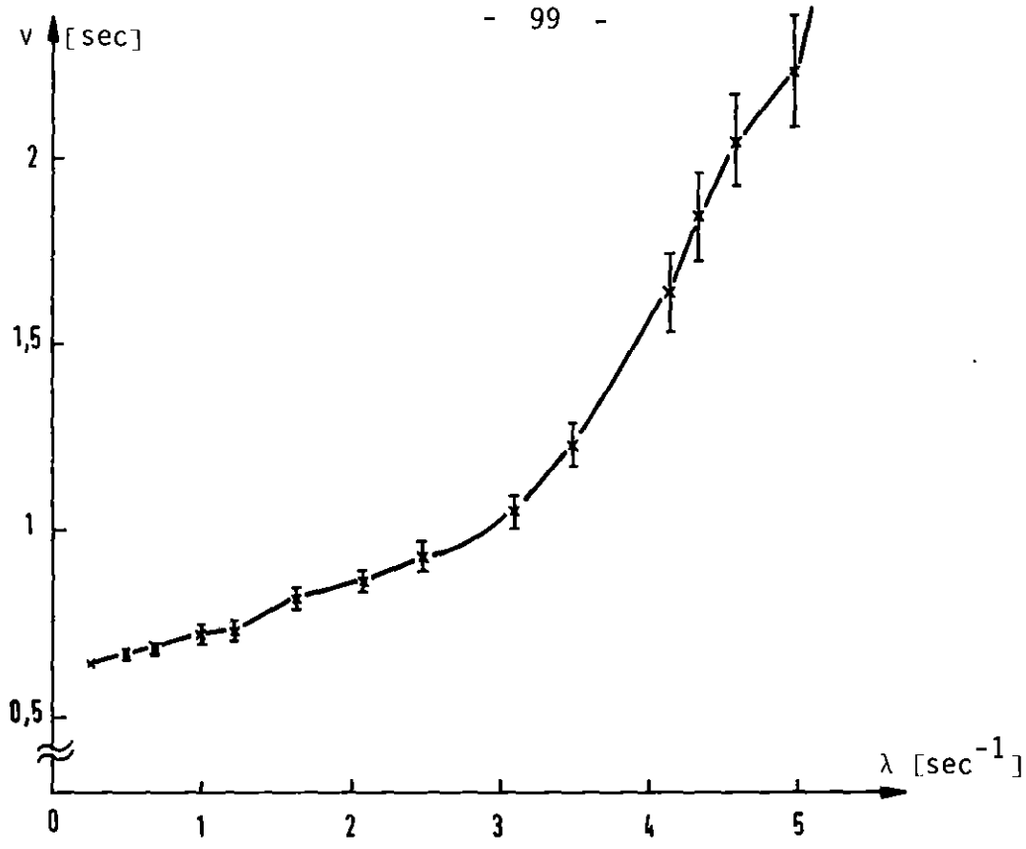

Abb. 6.5.: Abhängigkeit der mittleren Dateneinheitenverweilzeit v [sec] von der Ankunftsrate λ [1/sec] von Dateneinheiten für das Kommunikationssubsystem des BERNET-Rechnernetzes (incl. den 95%-Konfidenzintervallen)

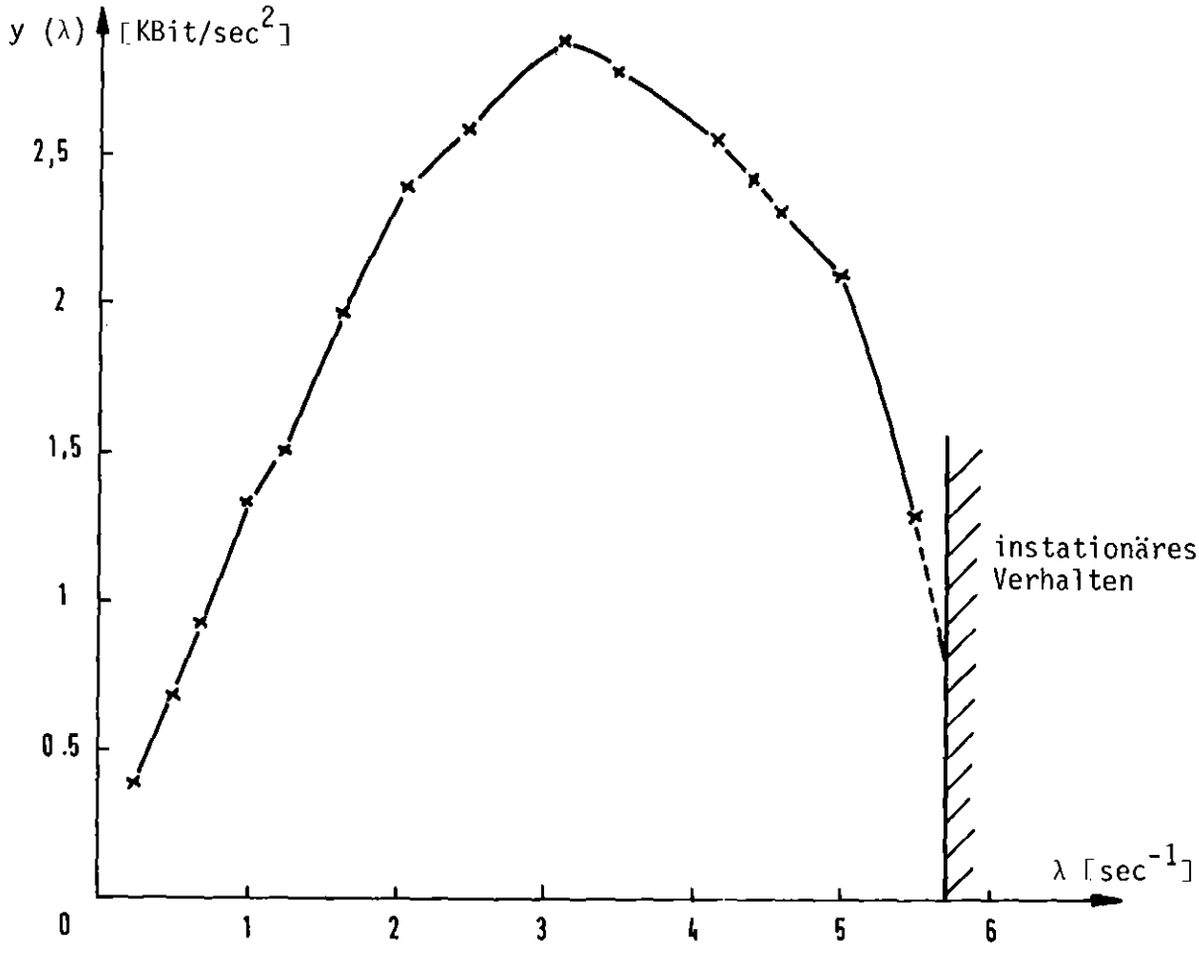

Abb. 6.6.: Abhängigkeit der Funktion y(λ) = Durchsatz/mittlere Dateneinheitenverweilzeit [KBit/sec^2] von der Ankunftsrate λ [1/sec] von Dateneinheiten für das Kommunikationssubsystem des BERNET-Rechnernetzes

Der in EXP_2 beobachtete, grundsätzliche Verlauf der Funktion $y(\lambda)$ ist charakteristisch. Bei beschränktem Durchsatz gilt insbesondere $\lim_{\lambda \to 0} y(\lambda) = \lim_{\lambda \to \infty} y(\lambda) = 0$. Auch der Übergang in eine instationäre Phase (für eine Ankunftsrate λ, die den maximal erreichbaren Durchsatz übersteigt) entspricht den Erwartungen.

Eine gleichzeitige Berücksichtigung der beiden Optimierungskriterien (hoher Durchsatz und geringe Dateneinheitenverweilzeit), die durch die Funktion $y(\lambda)$ verknüpft werden, ist im allgemeinen nicht möglich. Allerdings existieren Rechnernetze, deren angebotene Dienste die Entscheidung erleichtern, da sie die Betonung eines der beiden Optimierungskriterien verlangen: so ist für Rechnernetze, deren Hauptaufgabe im Anbieten von Diensten wie Transfer von Dateien u.ä. besteht, in erster Linie ein hoher Durchsatz zu erzielen, wobei die Dateneinheitenverweilzeiten nur eine untergeordnete Rolle spielen. Bei Rechnernetzen, deren Hauptanwendung in einem Timesharing-Betrieb liegt, dominiert die Forderung nach geringer Dateneinheitenverzögerung. Hingegen bietet es sich an, bei Rechnernetzen, die wie das BERNET grundsätzlich verschiedene Klassen von Rechnernetzdiensten anbieten, Optimierungskriterien wie die Funktion $y(\lambda)$ heranzuziehen.

In den zukünftigen Studien für das BERNET-Rechnernetz wird es, parallel zur Fortsetzung der Modellvalidation, darum gehen - u.a. durch Optimierung der Kommunikationsprotokolle - eine höhere Effizienz des BERNET-Rechnernetzes zu erzielen, als sie die obigen Pilotexperimente für die aktuelle Implementierung prognostizieren.

7. ZUSAMMENFASSUNG UND AUSBLICK

Die Realisierung verteilter DV-Systeme verlangt nach leistungsfähigen Hilfsmitteln zur Unterstützung des Entwurfs und der Analyse von Rechnernetzsoftware bei gleichzeitiger Berücksichtigung von Leistungskenngrößen der verwendeten Hardware-Komponenten. Eine wichtige Klasse von Fragestellungen betrifft dabei die Untersuchung der Leistungsfähigkeit und Korrektheit der Kommunikationssoftware, die die Dienste eines Rechnernetzes bereitstellt. Derartige Fragen lassen sich im allgemeinen nur durch ein Zurückgreifen auf Methoden der rechnergestützten Simulation beantworten. Die Beseitigung von Entwurfsfehlern nach der Installation eines Rechnernetzes verursacht hohe Kosten während des laufenden Rechnernetzbetriebs und rechtfertigt somit den Aufwand einer Simulationsstudie zum Vergleich von Entwurfsalternativen.

Die vorliegende Arbeit geht das Problem der Effizienzanalyse und -prognose für Rechnernetze durch Entwicklung generalisierter Modelle an, die eine detaillierte Nachbildung hierarchisch organisierter Kommunikationssoftware unter gleichzeitiger Miteinbeziehung der Betriebsmittelzuteilung in den Rechnernetzknoten erlauben.

Die eingeführten Modelle werden durch Methoden der rechnergestützten Simulation ausgewertet. Zu diesem Zweck wurden die Modelle in einer höheren Programmiersprache implementiert und in das Modellierungssystem MOSAIC eingebettet.

Ein hohes Maß an Effizienz und Benutzernähe werden durch eine spezielle, in das Modellierungssystem integrierte Konzeption zur Durchführung von Simulationsexperimenten gewährleistet.

Ausgehend von einer Analyse der Struktur von Rechnernetzen werden im ersten Teil der Arbeit zunächst diejenigen Teilkomponenten eines Rechnernetzes herausgearbeitet, die für die Modellbildung von Relevanz sind und der anschliessenden Einführung der konzeptionellen Modelle als Grundlage dienen. Bei der Bereitstellung der konzeptionellen Modelle wird weitgehend versucht, erweiterbare Grundmodelle zu formulieren, die sich in einfacher Weise an unterschiedliche Randbedingungen anpassen lassen. Zentrale Modelle sind zum einen das Modell der "Protokolleinheit", das es erlaubt, Kommunikationsprotokolle in unterschiedlichen Schichten einer hierarchisch aufgebauten Kommunikationssoftware unter Beibehaltung einer identischen Grundstruktur zu beschreiben, zum anderen das Modell der "Betriebsmittelverwaltungsinstanz", das der Modellierung der Betriebsmittelvergabe in einem Rechnernetzknoten dient.

Der zweite Teil der Arbeit führt den Simulator MOSAIC ein, der es ermöglicht, die vorgestellten konzeptionellen Modelle unter verschiedenartigen Randbedingungen auszuwerten und dadurch die grundsätzliche Einsatzfähigkeit der Modelle aufzuzeigen. Zur Strukturierung dieses Simulators wurde ein Konzept erarbeitet, das eine dynamische Konfigurierung des Simulationsprogramms aus vordefinierten Modellbausteinen unterstützt und daher in seiner Anwendungsbreite erheblich über die üblichen Parameterstudien hinausgeht, die die meisten existierenden Simulatoren ausschließlich zulassen. In diesem Sinne kann der vorgeschlagene Aufbau von Modellierungssystemen als Vorbild für Simulatoren mit einer abweichenden Zielsetzung dienen.

Im dritten Teil der Arbeit wird ein auf der Basis interaktiver Arbeitsmethoden realisiertes Konzept für die Durchführung von Simulationsexperimenten mit MOSAIC oder einem ähnlich strukturierten Simulator vorgestellt, das dem Experimentator zahlreiche Interaktionsmöglichkeiten während eines Simulationsablaufs gestattet und darüber hinaus durch den Einsatz graphischer Hilfsmittel einen hohen Grad an Benutzernähe besitzt.

Die in dieser Arbeit vorgestellten Modelle konnten in ersten Anwendungen bereits erfolgreich eingesetzt werden; einige signifikante experimentelle Ergebnisse werden wiedergegeben.

Durch die Möglichkeit, Randbedingungen wie z.B. die Konfiguration und den Aufbau der Kommunikationssoftware eines zu untersuchenden Rechnernetzes bereits durch Variation von Eingabedaten grundlegend zu ändern, lassen sich mit Hilfe von MOSAIC in einfacher Weise selbst Fragestellungen analysieren, die einen unterschiedlichen Detaillierungsgrad für das Rechnernetzmodell voraussetzen. Das Spektrum der Probleme, die sich durch die hier vorgestellten konzeptionellen Modelle und das Modellierungssystem MOSAIC untersuchen lassen, reicht von der gezielten Analyse eines speziellen Kommunikationsprotokolls über die integrierte Analyse einer vollständigen Protokollhierarchie bis hin zur Analyse der Kommunikationssoftware eines Rechnernetzes unter Berücksichtigung wesentlicher Charakteristika des Betriebssystems in den Rechnernetzknoten.

Durch die Möglichkeit einer mikroskopischen Modellierung von Kommunikationsprotokollen in MOSAIC, lassen sich neben der Leistungsfähigkeit von Protokollen bedingt auch Fragen im Hinblick auf die Protokollverifikation - allerdings nur unter einer eher heuristischen Vorgehensweise - beantworten. Zwar kann durch derartige Simulationsstudien nicht die Korrektheit eines Kommuni-

kationsprotokolls nachgewiesen werden, jedoch kann das Auffinden von Protokollfehlern (wie z.B. Verklemmungszuständen) entscheidend unterstützt werden.

Weiterführende Arbeiten werden Gelegenheit bieten, die hier eingeführten Modelle sowie den Simulator MOSAIC zu Effizienzstudien für eine Reihe existierender und geplanter Rechnernetze einzusetzen und zu erweitern. Insbesondere im Hinblick auf die aktuellen Standardisierungsbemühungen für Kommunikationssoftware stellt das Modellierungssystem MOSAIC ein erstes, universelles Instrument zur Leistungsbewertung von Kommunikationsprotokollen dar, das zur Beurteilung von Alternativvorschlägen herangezogen werden kann.

Schlußwort

An dieser Stelle sei Herrn Prof. Dr. G. Krüger besonders gedankt, der mir die Anfertigung der vorliegenden Arbeit ermöglichte und durch seine fruchtbaren Diskussionsbeiträge Wesentliches zum Inhalt dieser Arbeit beitrug.

Gleichfalls möchte ich Herrn Prof. Dr. A. Schreiner für seine kritischen Anmerkungen und Ratschläge zur Gestaltung der Arbeit danken.

Des weiteren danke ich einigen Mitarbeitern des IDT, insbesondere Herrn Dr. O. Drobnik, der mir oft ein wertvoller Gesprächspartner war, Herrn H. Marker für seine Mithilfe bei der Implementierung des Modellierungssystems MOSAIC und bei der Durchführung der Experimente, sowie den Herren Prof. Dr. J. Nehmer und Dr. E. Holler für ihre Anregungen, die zur Erhöhung der Übersichtlichkeit des Textes beitrugen.

Nicht zuletzt geht mein Dank auch an die Herren Dipl.-Phys. H.W. Strack-Zimmermann und Dipl.-Ing. M. Wilhelm, die - in ihrer Funktion als gemeinsame Leiter des BERNET-Projekts - durch bedeutende Hinweise die Anpassung von MOSAIC an das BERNET-Rechnernetz unterstützten.

LITERATURVERZEICHNIS

/BASK 75/ F. Baskett, K.M. Chandy, R.R. Muntz, F.G. Palacios
Open, closed, and mixed networks of queues with different classes of customers.
J. ACM 22/2 (1975), S. 248-260

/BEIL 73/ H. Beilner
Zur Gültigkeitsbestimmung diskreter Simulationsmodelle.
Proc. Fachtagung Methodik der rechnergestützten Simulation (Karlsruhe, 1973), S. 332-343

/BOCH 77/ G.V. Bochmann, J. Gecsei
A unified method for the specification and verification of protocols.
Proc. IFIP Congress (Toronto, 1977), S. 229-234

/BRIN 73/ P. Brinch Hansen
Operating system principles.
Prentice-Hall (Englewood Cliffs, 1973)

/BROW 76/ J.C. Browne
A critical overview of computer performance evaluation.
Proc. 2nd Intern. Conf. on Software Engineering (San Francisco, 1976), S. 138-145

/BUX 79/ W. Bux, H.L. Truong
A queueing model for HDLC-controlled data links.
Proc. Intern. Symp. Flow Control in Computer Networks (Versailles, 1979), S. 287-306

/CCITT 77/ CCITT
Recommendation X.25: Interface between data terminal equipment (DTE) and data circuit terminating equipment (DCE) for terminals operating in the packet mode on public data networks.
Orange Book, Vol. VIII.2, (Genf, 1977)

/CERF 76/ V.G. Cerf, A. McKenzie, R. Scantlebury, H. Zimmermann
Proposal for an international end to end protocol.
SIGCOMM 6/1 (1976), S. 63-89

/CHRE 75/ G.J. Chretien, W.M. Konig, J.H. Rech
The SITA network.
in: R.L. Grimsdale, F.F. Kuo (ed.) "Computer communication networks", Noordhoff (1975), S. 373-396

/CUNN 77/ I.M. Cunningham, W.J. Older, A.K. Trivedi
DATAPAC software architecture.
Proc. COMPSAC77 (Chicago, 1977), S. 752-758

/DAHL 68/ O.J. Dahl, B. Myhrhaug, K. Nygaard
SIMULA 67 - Common base language.
Norsk Regnesentral (Oslo, 1968)

/DANT 78/ A.A.S. Danthine, J. Bremer
Modelling and verification of end-to-end transport protocols.
Computer Network Protocols Symp. (Liège, 1978), F5.1-F5.12

/DOLL 74/ D.R. Doll
Telecommunications turbulence and the computer network evolution.
Computer 2 (1974), S. 34-43

/DROB 74/ O. Drobnik, F. Schumacher, R. Senger
Simulation von Warteschlangensystemen - Programmsystem zur Stichprobenerhebung und -auswertung.
Kernforschungszentrum Karlsruhe KfK 1979 (Karlsruhe, 1974)

/EFRO 75/ D.C. Efron
A methodology for tuning and verifying package simulation models.
Proc. 3rd Symp. on the Simulation of Computer Systems (Boulder, 1975), S. 186-195

/FALK 77/ G. Falk, J.M. Mc.Quillan
Alternatives for data network architectures.
Computer 11 (1977), S. 22-29

/GORD 67/ W.J. Gordon, G.F. Newell
Closed queueing systems with exponential servers.
Operations Research 15 (1967), S. 254-265

/GOUD 76/ M.G. Gouda, E.G. Manning
On the modelling, analysis and design of protocols - a special class of software structures.
Proc. 2nd Intern. Conf. on Software Engineering (San Francisco, 1976), S. 256-262

/GOUD 77/ M.G. Gouda
Protocol machines: Towards a logical theory of communication protocols.
Univ. of Waterloo, Dissertation, CCNG T-Report T-74 (Waterloo, 1977)

/HAEN 76/ J. Haenle
Simulation of the protocols of the GMD-Net.
Proc. Conf. on Computer Networks and Teleprocessing (Aachen, 1976), S. 119-130

/HANS 77/ A. Hansen
Simulation model of the EIN-subnetwork.
NTNF report (Oslo, 1977)

/HEAR 75/ F.E. Heart
The ARPA network.
in: R.L. Grimsdale, F.F. Kuo (ed.) "Computer communication networks", Noordhoff (1975), S. 19-33

/HERT 78/ F.R. Hertweck, E. Raubold, F. Vogt
The ML-protocol-description.
PIX/HLP/TAG/78/01/, GMD (Darmstadt, 1978)

/HIGH 77/ H.J. Highland
Role of simulation in computer performance measurement and evaluation.
in: Infotech International "Performance modelling and prediction", Infotech State of the Art Report (Maidenhead, 1977), S. 193-220

/HOLL 75/ E. Holler, O. Drobnik, R. Knöpker
Entwurf und Modellierung von Mehrrechnersystemen für Prozeßlenkungsaufgaben.
Kernforschungszentrum Karlsruhe KfK-PDV 57 (Karlsruhe, 1975)

/INFO 77/ Infotech International
Performance modelling and prediction.
Infotech State of the Art Report (Maidenhead, 1977)

/IRLA 76/ M.I. Irland, N.B. Cohen
Simulation of switch protocol (MV8) in Cigale.
Univ. of Waterloo, CCNG Report E-53 (Waterloo, 1976)

/ISO 77/ International Organization for Standardization
ISO/TC97/SC6
HDLC - Proposed balanced class of procedures.
Dokument Nr. ISO/TC97/SC6 N1444 (1977)

/ISO 78/ International Organization for Standardization
ISO/TC97/SC16
Reference model of open systems architecture.
Dokument Nr. ISO/TC97/SC16 N117 (1978)

/KLAR 71/ R. Klar
Messung von Rechneraktivitäten.
Univ. Erlangen/Nürnberg, Dissertation (Erlangen, 1971)

/KLEI 76/ L. Kleinrock
Queueing systems 2: Computer applications.
J. Wiley and Sons (New York/London/Sydney/Toronto, 1976)

/KOBA 77/ H. Kobayashi, A.G. Konheim
Queueing models for computer communications system analysis.
IEEE Trans. Commun., Vol. COM-25, No. 1 (1977), S. 2-29

/LEGO 76a/ H. Le Goff
Etude générale et évaluations de protocols de transport dans les réseaux informatiques.
Univ. de Rennes, Dissertation (Rennes, 1976)

/LEGO 76b/ H. Le Goff, G. Le Lann
Communication and synchronisation tools in a distributed environment.
ECI Conf., Springer Verlag "Lecture notes in computer science (No. 44)" (Amsterdam, 1976), S. 50-61

/LEGO 78/ H. Le Goff, G. Le Lann
Trade-off simulation.
in: S. Schoemaker (ed.) "Computer networks and simulation", North-Holland Publishing Company (Amsterdam/New York/Oxford, 1978), S. 155-167

/LEMO 73/ G. Le Moli
A theory of colloquies.
Proc. 1st European Workshop on Computer Networks (Arles, 1973), S. 153-173

/MYER 78/ W. Myers
The need for software engineering.
Computer 11 (1978), S. 12-26

/NIED 79/ J. Niedereichholz, F. Stockheim
Kostenvergleich der Simulationssprachen GPSS 1100 und SIMULA I.
Angew. Informatik 1 (1979), S. 1-8

/PARN 72/ D.L. Parnas
A technique for software module specification with examples.
Comm. ACM, Vol. 15 (1972), S. 330-336

/PETE 77/ J.L. Peterson
Petri nets.
ACM Computing Surveys, Vol. 9 (1977), S. 223-252

/POSC 79/ B. Posch, R. Swik
Interaktive Simulation kontinuierlicher dynamischer Systeme auf Mini-Prozeßrechnern.
Elektronische Rechenanlagen 21, Nr. 1 (1979), S. 13-22

/POUZ 75/ L. Pouzin
Network protocols.
in: Infotech International "Network systems and software", Infotech State of the Art Report (Maidenhead, 1975), S. 601-627

/PRIC 79/ W.L. Price
A review of the flow-control aspects of the network simulation studies at the National Physical Laboratory.
Proc. Intern. Symp. Flow Control in Computer Networks (Versailles, 1979), S. 17-32

/ROBE 77/ L.G. Roberts
Packet network design - the third generation.
Proc. IFIP Congress (Toronto, 1977), S. 541-546

/SALT 66/ J.H. Saltzer
Traffic control in a multiplexed computer system.
MIT, Dissertation (Cambridge, 1966)

/SCHN 78/ G.M. Schneider
VANS - - A resource - sharing computer network design tool.
in: S. Schoemaker (ed.) "Computer networks and simulation", North-Holland Publishing Company (Amsterdam/New York/Oxford, 1978), S. 227-248

/SCHO 78/ S. Schoemaker
Computer networks and simulation.
North-Holland Publishing Company (Amsterdam/New York/
Oxford, 1978)

/SCHR 77/ A. Schreiner, U. Schoder
Betrieb und Bewertung von EDV-Anlagen.
Vorlesungsskript, Universität Karlsruhe (Karlsruhe, 1977)

/SCHU 77/ H.R. Schuchmann
Was ist Rechnerverbund?
Elektronische Rechenanlagen 19, Nr. 3 (1977), S. 129-138

/SCHW 77/ M. Schwartz
Computer-communication network design and analysis.
Prentice-Hall (Englewood Cliffs, 1977)

/SHAN 75/ R.E. Shannon
Systems simulation: The art and science.
Prentice-Hall (Englewood Cliffs, 1975)

/SUNS 76/ C.A. Sunshine
Factors in interprocess communication protocol efficiency for computer networks.
Proc. AFIPS Conf., Vol. 45 (New York, 1976), S. 571-576

/SUNS 78/ C.A. Sunshine
Survey of protocol definition and verification techniques.
Computer Network Protocols Symp. (Liège, 1978), F1.1-F1.4

/STER 78/ J. Stern
Communication systems.
Proc. 3rd Jerusalem Conf. on Information Technology
(Jerusalem, 1978), S. 271-277

/STRA 78/ H.W. Strack-Zimmermann, M. Wilhelm
BERNET - The Berlin (West) computer network.
EUROCOMP 1978 (London, 1978)

/TEOR 75/ T.J. Teorey
Validation criteria for computer system simulations.
Proc. 3rd Symp. on the Simulation of Computer Systems
(Boulder, 1975), S. 160-173

/TYME 71/ L. Tymes
TYMNET - A terminal oriented communication network.
Proc. AFIPS Conf., Vol. 38 (1971), S. 211-216

/WOLF 77/ B. Wolfinger, O. Drobnik
Models for the simulation of communication systems in computer networks.
Proc. Intern. Symp. SIMULATION '77 (Montreux, 1977), S. 61-66

/WOLF 78a/ B. Wolfinger, O. Drobnik
Simulation of protocol layers of communication in computer networks.
in: S. Schoemaker (ed.) "Computer networks and simulation", North-Holland Publishing Company (Amsterdam/New York/Oxford, 1978), S. 119-140

/WOLF 78b/ B. Wolfinger, O. Drobnik, E. Holler
Design and simulation of decentralized control structures for reliable distributed systems.
in: M. Arató, E. Knuth (ed.) "Selected papers on operating systems", (Budapest, 1978), S. 323-336

/WOLF 79/ B. Wolfinger
MOSAIC - Ein Modellierungssystem zur rechnergestützten Simulation von Kommunikationsflüssen in Rechnernetzen.
KfK/IDT, Primärbericht Nr. 09.01.01P07A (Karlsruhe, 1979) *)

/WONG 78/ J.W. Wong
Queueing network modeling of computer communication networks.
ACM Computing Surveys, Special Issue: "Queueing network models for computer system performance", Vol. 10, No. 3 (1978), S. 343-351

/ZEIG 76/ B.P. Zeigler
Theory of modelling and simulation.
J. Wiley and Sons (New York/London/Sydney/Toronto, 1976)

*) Primärberichte enthalten unveröffentlichte Informationen von vorläufigem und betriebsinternem Charakter. Eine zur Verfügungstellung der Berichte ist nach entsprechender einzelvertraglicher Vereinbarung über die Nutzung des darin enthaltenen know how (know-how-Vertrag) möglich. Entsprechende Anfragen sind an die Stabsabteilung Patente und Lizenzen des KfK zu richten.

Anhang A: Grobbeschreibung der MOSAIC-Modellbausteine

Im folgenden werden, in einer SIMULA-ähnlichen Notation, Grobbeschreibungen der Rechnernetzmodelle gegeben, die als Bausteine für das Modellierungssystem MOSAIC dienen, wobei u.a. eine Definition der wichtigsten Parameter der Modelle erfolgt. Ausführlicher Gebrauch wird von dem SIMULA-Konzept der Oberklassen gemacht, das es erlaubt, im Anschluß an eine Klassenvereinbarung der Art

```
class A .....;
begin ..... end;
```

die Elemente einer Klasse B durch die Klassenvereinbarung

```
A class B .....;
  begin ..... end;
```

mit den durch Verbund A gegebenen Eigenschaften zu versehen.

Zur Kennzeichnung des Beginns bzw. des Endes der zur Erläuterung in die nachfolgenden Programmfragmente eingefügten Kommentare, finden die Zeichen /* bzw. */ Verwendung. Schlüsselworte (Key-words) in SIMULA sind durch Unterstreichung hervorgehoben. Die Art der Spezifikation von Prozeduren ähnelt einem Vorschlag von Parnas /s.PARN72/. Die Programmbeschreibungen beziehen sich auf die aktuelle Version des Modellierungssystems MOSAIC.

A.1. Komponenten zur Modellierung von Protokollhierarchien

A.1.1. Protokollschicht:

```
class SCHICHT (COMPID,LAYNUM);
integer LAYNUM /* Nummer der Protokollschicht */,
        COMPID /* Rechneridentifikation */;
begin   integer MAXDULENGTH /* max. Länge von Dateneinheiten in
                               der Schicht LAYNUM */,
                NUMOFPU     /* Anzahl der Protokolleinheiten in
                               der Schicht LAYNUM */;
        /* ... weitere Variablen ... */;
end of SCHICHT;
```

A.1.2. Protokolleinheit:

```
class PROTOKOLLEINHEIT (LAYNUM,LOCNUM);
integer LAYNUM /* Nummer der Protokollschicht */,
        LOCNUM /* lokale Nummer der Protokolleinheit innerhalb
                  der Schicht LAYNUM */;
begin   integer NUMPROTOCOL /* Kennzeichnung des zugrundeliegenden
                               Kommunikationsprotokolls */,
                MAXBLOCKL   /* max. Länge der Dateneinheiten inner-
                               halb der Protokolleinheit */,
                PUCLASS     /* Kennzeichnung der Korrespondenten ϵ
                               KORR durch identische PUCLASS */;
        boolean MASTER      /* zur Unterscheidung unsymmetrischer
                               Protokolle */;
        ref (SENDE_PROTOKOLLMODUL) OWNSPM;
        ref (EMPFANGS_PROTOKOLLMODUL) OWNRPM;
            /* Verweise auf die Protokollmoduln (innerhalb der
               Protokolleinheit) */;
        /* ... weitere Variablen ... */;
end of PROTOKOLLEINHEIT;
```

A.1.3. Sende-Protokollmodul:

```
TASK class SENDE_PROTOKOLLMODUL (COMPID,NUMCORR,MAXUSERS,LAYNUM,PUNUM,TYPE,
                                 SENDQ,COPYQ,SATQ,TIMEOUTQ);
    integer COMPID   /* Rechneridentifikation */,
            NUMCORR  /* Anzahl der Korrespondenten ϵ KORR */,
            MAXUSERS /* Anzahl der Nachbarn ϵ NBHS */,
            LAYNUM   /* Nummer der Protokollschicht */,
            PUNUM    /* Nummer der zugeordneten Protokolleinheit */,
            TYPE     /* Typ des zugeordneten Rechners */;
    ref (HEAD) SENDQ,COPYQ,TIMEOUTQ,SATQ;
                     /* Definition sämtlicher Sende-Protokollmodul-
                        Warteschlangen */
```

```
begin /* ... weitere Variablenvereinbarungen:
         (1) die Variablen zur Kennzeichnung des Schnittstellenzu-
             stands existieren als Vektoren (Dimension = Anzahl
             der Nachbarn ϵ NBTS)

         (2) die Variablen zur Kennzeichnung des Protokollzustands
             existieren als Matrizen (Dimension = MAXUSERS x NUMCORR)
             ... */

      /* Verwendete Prozeduren: */
       procedure FRAGMENTATE (DU,DU1);
      /* Eingabeparameter: DU.
         Ausgabeparameter: DU1.
         Aufgabe: Fragmentierung einer Dateneinheit DU.
         Effekt: DU1 verweist auf die Dateneinheit, die bei der Frag-
                 mentierung entsteht, DU verweist auf die zu fragmen-
                 tierende und nach Ausführung der Prozedur um DU1
                 "gekürzte" Dateneinheit. */

      boolean procedure ANALLINTF (STRATEGY);
      /* Eingabeparameter: STRATEGY.
         Aufgabe: Analyse des Schnittstellenzustands zur nächsttiefe-
                  ren Schicht unter Berücksichtigung der Strategie
                  STRATEGY.
         Effekt: Ein Prozedurergebnis = TRUE besagt, daß eine aktuelle
                 Übergabe von Schnittstellenaufträgen zumindest an ei-
                 nen Nachbarn ϵ NBTS erlaubt ist. */

      integer procedure DETCORRESP (DU,STRATEGY);
      /* Eingabeparameter: DU,STRATEGY.
         Aufgabe: Bestimmung des Korrespondenten, an den die Datenein-
                  heit DU weiterzuleiten ist (Berücksichtigung der
                  Strategie STRATEGY).
         Effekt: Nach Ausführung der Prozedur beinhaltet DETCORRESP
                 die Nummer des Rechners, der den gesuchten Korres-
                 pondenten enthält. */
```

```
procedure GENDATABLOCK (DU,DU1,HEADER);
/* Eingabeparameter: DU.
   Ausgabeparameter: DU1,HEADER.
   Aufgabe: Generierung eines "Dateneinheitenblocks" unter
            Verwendung der Dateneinheit DU und evtl. weiterer
            Dateneinheiten der SENDQ, die für denselben
            Korrespondenten bestimmt sind.
   Effekt: Nach Ausführung der Prozedur verweist DU1 auf die
           generierte Dateneinheit (Block), die Variable
           HEADER beinhaltet Information, die ggf. zum Über-
           schreiben der bereits existierenden Kontrollin-
           formation benötigt wird. */

procedure DETSNEIGHB (LAYNUM,TCH,SPM,RPM,CORR,NBTYPE,MAXL);
/* Eingabeparameter: LAYNUM,CORR.
   Ausgabeparameter: TCH,SPM,RPM,NBTYPE,MAXL.
   Aufgabe: Bestimmung des Nachbarn ϵ NBTS, über welchen der
            Korrespondent CORR des Sende-Protokollmoduls er-
            reicht werden kann.
   Effekt: NBTYPE kennzeichnet den Typ des gefundenen Nach-
           barn (1 = Übertragungskanal, 2 = Sende-Protokoll-
           modul, 3 = Empfangs-Protokollmodul) und in Abhängig-
           keit des Typs verweist TCH, SPM oder RPM auf den er-
           mittelten Nachbarn. */

boolean procedure ANANEIGHB (INTERFACE,TCH,SPM,RPM,DU,NBTYPE);
/* Eingabeparameter: INTERFACE,TCH,SPM,RPM,DU,NBTYPE.
   Aufgabe: Analyse des Schnittstellenzustands für den Nach-
            barn TCH, SPM bzw. RPM (Typ der Schnittstelle ge-
            geben durch INTERFACE, Typ des Nachbarn gegeben
            durch NBTYPE).
   Effekt: Ein Prozedurresultat = TRUE besagt, daß der aktu-
           elle Zustand der Schnittstelle eine Weitergabe der
           Dateneinheit DU gestattet. */
```

```
      procedure UPDATNEIGHB (INTERFACE,TCH,SPM,RPM,DU,NBTYPE);
      /* Eingabeparameter: INTERFACE,TCH,SPM,RPM,DU,NBTYPE.
         Aufgabe: Erneuern der Zustandsinformation des Nachbarn
                  TCH, SPM bzw. RPM (Typ der Schnittstelle ge-
                  geben durch INTERFACE, Typ des Nachbarn gege-
                  ben durch NBTYPE).
         Effekt: Die Auswirkungen einer Übergabe der Dateneinheit
                 DU an den Nachbarn TCH, SPM bzw. RPM werden be-
                 handelt (z.B. Erhöhung des belegten Speichers
                 etc.). */

      procedure UPDATINTF (INTERFACE);
      /* Eingabeparameter: INTERFACE.
         Aufgabe: Erneuern der Zustandsinformation der Schnitt-
                  stelle zu benachbarten Protokollschichten (Typ
                  der Schnittstelle gegeben durch INTERFACE).
         Effekt: Für sämtliche Nachbarn ∈ NBHS wird der Zustand
                 der Schnittstelle aktualisiert. */

      procedure CREATECINF (DU,HTYPE,PROTOCOL);
      /* Eingabeparameter: DU,HTYPE,PROTOCOL.
         Aufgabe: Erstellen von Kontrollinformation des Typs HTYPE
                  für die Dateneinheit DU.
         Effekt: Gemäß des gegebenen Kommunikationsprotokolls
                 PROTOCOL wird für die Dateneinheit DU Kontrollin-
                 formation des Typs HTYPE generiert und in die
                 entsprechende Warteschlange von DU als erstes
                 Element eingefügt. */

      /* ... Variablen-Initialisierungen ... */
      while true do
      begin
         /* ... Sende-Protokollmodul-Schleife ... */
      end of loop;
end of SENDE-PROTOKOLLMODUL;
```

Bemerkungen: (1) Aus der obigen Klassenvereinbarung ist ersichtlich, daß ein Sende-Protokollmodul die Eigenschaften einer TASK beinhaltet; zur Definition der Oberklasse TASK s.u.

(2) Die Grobstruktur der Sende-Protokollmodul-Schleife kann dem Struktogramm der Abb. 4.3 in Kap. 4 entnommen werden.

A.1.4. Empfangs-Protokollmodul:

```
TASK class EMPFANGS_PROTOKOLLMODUL (COMPID,NUMCORR,MAXUSERS,LAYNUM,
      PUNUM,TYPE,RECQ,BLOCKQ,EATQ,TIMEOUTQ);
    integer COMPID /* Rechneridentifikation */,
            NUMCORR /* Anzahl der Korrespondenten ϵ KORR */,
            MAXUSERS /* Anzahl der Nachbarn ϵ NBHS */,
            LAYNUM /* Nummer der Protokollschicht */,
            PUNUM  /* Nummer der zugeordneten Protokolleinheit */,
            TYPE   /* Typ des zugeordneten Rechners */;
    ref (HEAD) RECQ,BLOCKQ,EATQ,TIMEOUTQ;
            /* Definition sämtlicher Empfangs-Protokollmodul-Warte-
               schlangen */
    begin /* ... weitere Variablenvereinbarungen:
             (1) die Variablen zur Kennzeichnung des Schnittstellenzu-
                 stands existieren als Vektoren (Dimension = Anzahl
                 der Nachbarn ϵ NBHS)
             (2) die Variablen zur Kennzeichnung des Protokollzustands
                 existieren als Matrizen (Dimension = MAXUSERS x NUMCORR)
                 ... */

          /* Verwendete Prozeduren: */
          procedure REACTTOHEADER (HEADER,DU,PROTOCOL);
          /* Eingabeparameter: HEADER,DU,PROTOCOL.
             Aufgabe: Reaktion auf die im HEADER enthaltene Kontroll-
                      information der Dateneinheit DU unter Berücksich-
                      tigung des Protokolls PROTOCOL.
             Effekt: Die Kontrollinformation im HEADER wird ausgewertet
                     und die durch das Protokoll vorgeschriebene Reak-
                     tion erfolgt; z.B.
```

(1) bei positiver Quittung: quittierte Dateneinheit(en) der COPYQ ist zu vernichten und evtl. TIMEOUT(s) zu eliminieren
(2) bei negativer Quittung: ggf. ist Übertragungswiederholung zu veranlassen
... */

```
boolean procedure ANAHLINTF (STRATEGY);
/* Eingabeparameter: STRATEGY.
   Aufgabe: Analyse des Schnittstellenzustands zur nächsthöheren Schicht unter Berücksichtigung der Strategie STRATEGY.
   Effekt: Ein Prozedurergebnis = TRUE besagt, daß eine aktuelle Übergabe von Schnittstellenaufträgen zumindest an einen Nachbarn ϵ NBHS erlaubt ist. */

procedure DETRNEIGHB (LAYNUM,USER,SPM,RPM,DU,NBTYPE);
/* Eingabeparameter: LAYNUM,DU.
   Ausgabeparameter: USER,SPM,RPM,NBTYPE.
   Aufgabe: Bestimmung des Nachbarn ϵ NBHS, an den die Dateneinheit DU beim Verlassen des Empfangs-Protokollmoduls weiterzuleiten ist.
   Effekt: NBTYPE kennzeichnet den Typ des gefundenen Nachbarn (1 = keine Weiterleitung der Dateneinheit erfolgt, da der aktuell erreichte Protokollmodul das Ziel der Dateneinheit darstellt, 2 = Sende-Protokollmodul, 3 = Empfangs-Protokollmodul, 4 = Benutzertask) und in Abhängigkeit des Typs verweist SPM, RPM oder USER auf den ermittelten Nachbarn. */

boolean procedure ANANEIGHB (INTERFACE,USER,SPM,RPM,DU,NBTYPE);
/* Eingabeparameter: INTERFACE,USER,SPM,RPM,DU,NBTYPE.
   Aufgabe: Analyse des Schnittstellenzustands für den Nachbarn USER, SPM bzw. RPM (Typ der Schnittstelle gegeben durch INTERFACE, Typ des Nachbarn gegeben durch NBTYPE).
   Effekt: Ein Prozedurresultat = TRUE besagt, daß der aktuelle Zustand der Schnittstelle eine Weitergabe der Dateneinheit DU gestattet. */
```

```
procedure UPDATNEIGHB (INTERFACE,USER,SPM,RPM,DU,NBTYPE);
/* Eingabeparameter: INTERFACE,USER,SPM,RPM,DU,NBTYPE.
   Aufgabe: Erneuern der Zustandsinformation des Nachbarn
            USER, SPM bzw. RPM (Typ der Schnittstelle gege-
            ben durch INTERFACE, Typ des Nachbarn gegeben
            durch NBTYPE).
   Effekt: Die Auswirkungen einer Übergabe der Dateneinheit
           DU an den Nachbarn USER, SPM bzw. RPM werden be-
           handelt (z.B. Erhöhung des belegten Speichers
           etc.). */

procedure UPDATINTF (INTERFACE);
/* Eingabeparameter: INTERFACE.
   Aufgabe: Erneuern der Zustandsinformation der Schnittstel-
            le zu benachbarten Protokollschichten (Typ der
            Schnittstelle gegeben durch INTERFACE).
   Effekt: Für sämtliche Nachbarn ∈ NBTS wird der Zustand der
           Schnittstelle aktualisiert. */

procedure CREATECINF (SPM,DU,CINF,PROTOCOL,HTYPE);
/* Eingabeparameter: SPM,DU,PROTOCOL,HTYPE.
   Ausgabeparameter: CINF.
   Aufgabe: Erstellen von Kontrollinformation des Typs HTYPE
            als Antwort auf die Dateneinheit DU.
   Effekt: Gemäß des gegebenen Kommunikationsprotokolls PROTO-
           COL wird für die empfangene Dateneinheit DU die
           Kontrollinformation CINF des Typs HTYPE generiert
           und in die SENDQ-Warteschlange des Sende-Protokoll-
           moduls SPM eingefügt, wobei gleichzeitig beglei-
           tend ein entsprechender Schnittstellenauftrag in
           die Warteschlange SATQ übergeben wird. */

procedure DEBLOCK (DU1,DU);
/* Eingabeparameter: DU.
   Ausgabeparameter: DU1.
   Aufgabe: Auftrennen eines "Dateneinheitenblocks" DU.
   Effekt: Die Prozedur trennt eine Dateneinheit DU1 von ei-
           nem Dateneinheitenblock DU ab. */
```

```
            /* ... Variablen-Initialisierungen ... */
            while true do
            begin
               /* ... Empfangs-Protokollmodul-Schleife (s. Abb. 4.4
                     in Kap. 4) ... */
            end of loop;
         end of EMPFANGS-PROTOKOLLMODUL;
```

A.1.5. Dateneinheit:

```
      LINK class DATENEINHEIT (QUEUE,NUM,ORIGLAY);
      ref (HEAD) QUEUE /* Warteschlange zur Speicherung von Kontrollinfor-
                          mation und (nur bei einem "Dateneinheitenblock"!)
                          von Dateneinheiten höherer Protokollschichten */;
      integer NUM /* rechnerinterne Dateneinheiten-Nummer */,
              ORIGLAY /* Nummer der Protokollschicht, in welcher die Datenein-
                         heit erzeugt wurde */;
      begin integer SENDCOMP /* Senderechner (globale Rechnernummer) */,
                    RECNODE  /* nächster zu durchlaufender Rechnernetzknoten
                                im Kommunikationssubsystem */,
                    RECCOMP  /* Empfangsrechner (globale Rechnernummer) */,
                    LENGTH   /* Gesamtlänge der Dateneinheit */,
                    DATALENGTH /* Länge der Dateneinheit abzüglich der mitge-
                                  führten Kontrollinformation */,
                    NUMFIRST /* rechnerinterne Nummer der ersten Dateneinheit,
                                die als Dateneinheitenfragment übertragen wur-
                                de */,
                    SUMFRAG /* Anzahl der Dateneinheitenfragmente, die beim
                               Zerteilen einer Dateneinheit entstehen */,
                    PRIORITY /* Priorität */;
            boolean FRAGWAIT /* true, falls die Dateneinheit bei der Reassemblie-
                                rung noch weitere Fragmente erwartet */;
            ref (BENUTZERTASK) RECTASK /* Verweis auf die Benutzertask, an wel-
                                          che die Dateneinheit zu übertragen
                                          ist */,
                               SENDTASK /* Verweis auf die Benutzertask, durch
                                           welche die Dateneinheit generiert
                                           wurde */;
               /* ... weitere Variablen ... */;
      end of DATENEINHEIT;
```

A.1.6. Kontrollinformation:

```
LINK class KONTROLLINFORMATION;
begin integer LAYNUM /* Gültigkeitsbereich (Schichtennummer) der Kon-
                        trollinformation */,
              NUMPU /* rechnerspezifische Nummer der Protokolleinheit,
                       durch die die Kontrollinformation generiert
                       wurde */,
              LOCPUNUM /* schichtenspezifische Nummer der Protokollein-
                          heit, durch die die Kontrollinformation gene-
                          riert wurde */,
              NSUSER /* dient - im Falle des Multiplexens - der Abbil-
                        dung der sendenden Protokolleinheit in Schicht
                        LAYNUM auf die der nächsttieferen Schicht bekann-
                        ten, parallel aktiven Benutzer */,
              SENDER /* globale Nummer des Rechners, durch den die Kon-
                        trollinformation generiert wurde */,
              RECCORR /* globale Nummer des Rechners, welcher den Korres-
                         pondenten beinhaltet, an den die Kontrollinfor-
                         mation adressiert ist */,
              PUCLASS /* kennzeichnet die Menge von Korrespondenten, der
                         die beiden Protokolleinheiten angehören, welche
                         die Kontrollinformation austauschen */,
              TYPE /* Typ der Kontrollinformation (z.B. Quittung, Sende-
                      aufforderung, etc.) */,
              LENGTH /* Länge der Kontrollinformation (wird bei der Gene-
                        rierung festgelegt und ist grundsätzlich unabhän-
                        gig von dem Umfang der tatsächlich mitgeführten
                        Information!) */,
              DUNUM /* rechnerinterne Nummer der Dateneinheit, die die Kon-
                       trollinformation in ihrer Warteschlange enthält */,
              ACKEDDU /* falls die Kontrollinformation eine Quittung dar-
                         stellt: rechnerinterne Nummer der quittierten
                         Dateneinheit */,
              PREDNUM /* bei Fragmentierung: rechnerinterne Nummer der er-
                         sten Dateneinheit, die als Dateneinheitenfragment
                         übertragen wurde */,
```

```
                NFRAG /* bei Fragmentierung: Anzahl der Dateneinheitenfrag-
                         mente, die beim Zerteilen einer Dateneinheit ent-
                         stehen */,
                NBLOCK /* bei Blockbildung: Anzahl der Dateneinheiten höhe-
                          rer Schichten, aus denen der Dateneinheitenblock
                          besteht */;
        boolean ACK  /* true, falls die Kontrollinformation eine Quittung
                        beinhaltet */,
                ACKOK /* true, falls eine positive Quittung vorliegt */,
                SENDLOCK /* true, falls die Kontrollinformation dem Emp-
                            fänger eine temporäre Sperrung der sendenden
                            Protokolleinheit mitteilt */,
                INITREQ /* true, falls die Kontrollinformation eine Auf-
                           forderung an den Empfänger zum Aufbau einer
                           (logischen) Verbindung darstellt */,
                TRANSEND /* true, falls eine existierende (logische) Ver-
                            bindung zwischen der sendenden und der empfan-
                            genden Protokolleinheit abgebaut werden soll */;
        /* ... und evtl. weitere Variablen ... */;
    end of KONTROLLINFORMATION;
```

Bemerkungen: (1) Die class KONTROLLINFORMATION modelliert die Gesamtheit der Kontrollinformation, die zwischen Korrespondenten zur Abwicklung eines Kommunikationsprotokolls ausgetauscht wird und umfaßt daher Headers, Flagbytes, Trailers, Paritybits etc.

(2) Die Parameter der Kontrollinformation sind zwar aus modellierungstechnischen Gründen identisch für sämtliche Protokollschichten; es müssen jedoch nicht sämtliche der Parameter tatsächlich belegt sein.

A.1.7. Schnittstellenauftrag:

```
    LINK class SCHNITTSTELLENAUFTRAG (COMPID,ORIGIN,REFDU,TYPE);
    integer COMPID /* globale Nummer des Rechners, der den Auftrag bein-
                      haltet */,
            ORIGIN /* rechnerspezifische Nummer der Protokolleinheit, durch
                      die der Auftrag generiert wurde */,
```

```
        TYPE /* Auftragstyp (s. Abschnitt 3.1.3):
                (a) TYPE > 0 für Aufträge an einen Sende-Protokollmodul:
                    externe Aufträge:
                    TYPE = 1: "OPEN", d.h. Eröffnen einer (logischen)
                              Verbindung,
                    TYPE = 2: "OPEN resp.", d.h. Rückmeldung für die Kom-
                              munikationsprimitive OPEN,
                    TYPE = 3: "SEND-DE", d.h. Transfer einer Dateneinheit
                              ∈ SENDQ,
                    TYPE = 4: "SEND-DE resp.", d.h. Rückmeldung für SEND-DE,
                    TYPE = 5: "SEND-INT", d.h. Transfer eines Interrupts
                              (im Sinne von X.25),
                    TYPE = 6: "SEND-INT resp.", d.h. Rückmeldung für
                              SEND-INT,
                    TYPE = 7: "CLOSE", d.h. Abbau einer (logischen) Ver-
                              bindung,
                    TYPE = 8: "CLOSE resp.", d.h. Rückmeldung für CLOSE,
                    ...
                    interne Aufträge:
                    TYPE = 10: "SEND-CTRLINF", d.h. Absenden von Kontroll-
                               information,
                    TYPE = 11: "SEND-COPY", d.h. Absenden einer Dateneinheit
                               ∈ COPYQ,
                    ...
                (b) TYPE < 0 für Aufträge an einen Empfangs-Protokollmodul:
                    externe Aufträge:
                    TYPE = -1: "REC-DE", d.h. Entgegennahme einer Daten-
                               einheit ∈ RECQ,
                    TYPE = -2: "REC-DE resp.", d.h. Rückmeldung für REC-DE,
                    ...
                    interne Aufträge:
                    TYPE = -10: "REASS-DE", d.h. Weiterleiten einer re-
                                assemblierten Dateneinheit ∈ BLOCKQ,
                    ... */;
    ref (DATENEINHEIT) REFDU /* Verweis auf die evtl. in die Auftragsbearbei-
                             tung involvierte Dateneinheit */;
    begin integer PRIORITY /* Priorität für die Auftragsbearbeitung */;
          /* ... und evtl. weitere Variablen ... */;
    end of SCHNITTSTELLENAUFTRAG;
```

A.1.8. Zeitüberwachungsauftrag:

```
PROCESS class TIMEOUT (TIMEINTERV,DU,SPM);
ref (DATENEINHEIT) DU /* Verweis auf die zu überwachende Dateneinheit
                         (die Überwachung dient der Vermeidung von Dead-
                         locks bei Sendevorgängen, s.u.) */;
real TIMEINTERV /* Dauer des Überwachungszeitraumes */;
ref (SENDE_PROTOKOLLMODUL) SPM /* Verweis auf den Sende-Protokollmodul,
                                  der für das Absenden der Dateneinheit
                                  DU verantwortlich ist */;
begin boolean ACTBYRECPM /* wird durch den Empfangs-Protokollmodul auf
                            true gesetzt, falls nach Erhalt einer positi-
                            ven Quittung eine weitere Überwachung des
                            Sendevorgangs überflüssig ist */;
      ref (SCHNITTSTELLENAUFTRAG) REQ /* Verweis auf einen Auftrag vom
                                         Typ SEND-COPY, der nach Ablauf
                                         des Zeitüberwachungsauftrags ei-
                                         ne Übertragungswiederholung von
                                         DU impliziert */;
      HOLD (TIMEINTERV);
      if ¬ ACTBYRECPM then
      begin /* Erzeugen eines SEND-COPY-Auftrags REQ für eine Übertra-
               gungswiederholung */;
            REQ.INTO (SPM.SATQ);
            /* ggf. Interrupt an die Betriebsmittelverwaltungsinstanz */;
      end;
      this TIMEOUT.OUT;
end of TIMEOUT;
```

Bemerkungen: (1) In der aktuellen Implementierung von MOSAIC dienen Zeitüberwachungsaufträge ausschließlich der Überwachung von Sendevorgängen, wobei das Ablaufen eines Zeitüberwachungsauftrags äquivalent ist zum Erhalt einer negativen Quittung.

(2) Da ein Zeitüberwachungsauftrag das Ablaufen des zu überwachenden Zeitintervalls selbst "bemerkt", erübrigt sich die Einführung einer speziellen Timeout-Kontrollinstanz wie im konzeptionellen Modell einer Protokolleinheit.

(3) Zeitüberwachungsaufträge werden durch Sende-Protokoll-moduln erzeugt und in deren TIMEOUTQ gespeichert; Zeitüberwachungsaufträge vernichten sich selbst.

A.1.9. Verbindung zwischen Korrespondenten auf niedrigster Protokollschicht:

```
PROCESS class ÜBERTRAGUNGSKANAL (NUM,COMP1,COMP2,MODE);
integer NUM /* globale Nummer des (Übertragungs-) Kanals */,
        COMP1,COMP2 /* globale Nummern der durch den Kanal verbundenen
                       Rechnernetzknoten (COMP1 < COMP2) */,
        MODE /* Übertragungsmodus (1 = simplex: COMP2 → COMP1,
                2 = simplex: COMP1 → COMP2, 3 = halbduplex, 4 = voll-
                duplex) */;
begin integer MAXRETRANS /* max. Anzahl von Übertragungswiederholungen
                            im Fehlerfall */,
              MAXPACKLENGTH /* max. Dateneinheitenlänge im Kanal */;
      real BITERRRATE /* Bitfehlerrate */,
           CAPACITY /* Übertragungskapazität (Nutzdatenrate) */;
      boolean array ACTOFQ (1:2) /* ACTOFQ(i) = true [für i = 1,2], falls
                                    eine Aktivierung der Warteschlange
                                    REQQ(i) zu versuchen ist */,
                    CHANNACT (1:2) /* CHANNACT(i) = true [für i = 1,2],
                                      falls sich momentan eine Datenein-
                                      heit der Warteschlange DEQ(i) in
                                      Übertragung befindet */;
      ref (HEAD) array REQQ (1:2) /* Auftragswarteschlangen (s. Abb. 3.10
                                     in Kap. 3) - REQQ(1) ist dem Rechner
                                     mit der Nummer COMP1 zugeordnet */,
                       DEQ (1:2) /* Dateneinheitenwarteschlangen (s. Abb.
                                    3.10 in Kap. 3) - Zuordnung s. Auf-
                                    tragswarteschlangen */;
      ref (ABI) array RABI (1:2) /* Verweis auf die beiden Auftragsbear-
                                    beitungsinstanzen. RABI(1) modelliert
                                    z.B. die Dateneinheitenverzögerung bei
                                    Übertragung von COMP1 → COMP2 */;
      /* ... Variablen-Initialisierungen ... */;
```

```
        while true do
        begin PASSIVATE; /* Jeder der beiden durch den Kanal verbundenen
                            Rechner kann eine Aktivierung herbeiführen */
              for i: = 1 step 1 until 2 do
              begin /* Versuch einer Dateneinheiten-Übertragung für jede
                       der beiden Kanalrichtungen (NUMSQ kennzeichne die
                       aktuell zu bearbeitende Richtung) */
                    /* Falls eine Abarbeitung der Auftragswarteschlange in
                       Richtung NUMSQ momentan sinnvoll und erlaubt ist:
                       (1) Auftragsauswahl,
                       (2) Verzögerung der zu übertragenden Dateneinheit
                           (durch die hierfür zuständige Auftragsbearbei-
                           tungsinstanz) */
              end;
        end of loop;
  end of ÜBERTRAGUNGSKANAL;
```

Bemerkungen: (1) Im Modellierungssystem MOSAIC sind die Auftragsbearbeitungsinstanzen als PROCESS class ABI implementiert.

(2) MOSAIC ermöglicht bislang keine dynamische Erweiterung der Menge existierender Übertragungskanäle während eines Simulationsexperiments, so daß bereits zu Experimentbeginn sämtliche benötigten Kanäle vorzusehen sind.

A.2. Komponenten zur Modellierung der Betriebsmittelverwaltung in Rechnern

A.2.1. Benutzertask:

```
TASK class BENUTZERTASK (GLONUM,NUM,COMPID);
integer GLONUM /* globale Tasknummer */,
        NUM    /* rechnerspezifische Nummer */,
        COMPID /* globale Nummer des Rechners, der die Benutzertask
                  beinhaltet */;
begin integer TYPE /* Tasktyp: Kommunizierend, lokal etc. */;
      real REQTIME /* Dauer des nächsten internen (Benutzertask-) Zu-
                      stands "Rechnen" */,
           SUSPENDTIME /* Dauer des nächsten internen Zustands ZI4 */,
           REQIOTIME /* Dauer des nächsten internen Zustands ZI2 */;
```

```
        ref (BENUTZERTASK) RECTASK /* nur bei kommunizierenden Benutzer-
                                      tasks: Verweis auf den Korrespon-
                                      denten */;
        ref (RECHNERNETZKNOTEN) REFC /* Verweis auf den Rechner, der die
                                        Benutzertask enthält */;
        /* ... Variablen-Initialisierungen ... */;
        /* Dynamische Ablaufstruktur einer Benutzertask (s. Abb. 4.5
           in Kap. 4) */
    end of BENUTZERTASK;
```

A.2.2. Task:

```
    PROCESS class TASK;
    begin integer TSTATE /* Variable zur Charakterisierung des dynamischen
                            Taskzustands */,
                  PRIORITY /* Taskpriorität (zur Steuerung der Betriebsmittel-
                              zuteilung) */;
        boolean PREEMPTABLE /* Indikator, der anzeigt, ob es sich bei ei-
                               nem internen Zustand "Rechnen" um den Zu-
                               stand ZI1' (PREEMPTABLE = true) oder um
                               ZI1" handelt */;
        /* ... weitere Variablen ... */;
    end of TASK;
```

A.2.3. Rechnernetzknoten:

```
    PROCESS class RECHNERNETZKNOTEN (COMPID,TYPE,NUMALLPU,NUMCPUS);
    integer COMPID /* globale Rechnernummer */,
            TYPE /* Rechnertyp */,
            NUMALLPU /* Anzahl sämtlicher Protokolleinheiten innerhalb des
                        Rechnernetzknotens */,
            NUMCPUS /* Anzahl der CPU's */;
    begin integer CSTATE /* Variable zur Charakterisierung des aktuellen
                            Grundzustands der Betriebsmittelverwaltungs-
                            instanz */,
                  STRATEGY /* Strategie zur Steuerung der Betriebsmittelzu-
                              teilung */;
        real TIMESLICE /* Dauer einer CPU-Zeitscheibe */;
```

```
ref (HEAD) QIO /* Warteschlange zur Speicherung der E/A-aktiven
                  Benutzertasks */,
           QCPU /* Warteschlange zur Speicherung der CPU-belegen-
                   den Benutzertasks (entspricht der Warteschlange
                   Q_CPU in Abb. 3.15 in Kap. 3 ohne Protokoll-
                   moduln) */,
           QPM /* Warteschlange für sämtliche Protokollmoduln des
                  Rechners */;
ref (HEAD) array QT (0:2) /* Warteschlangen zur Speicherung von
                             Benutzertasks:
                             QT(0): entspricht der Warteschlange
                                    Q_WHSP in Abb. 3.15,
                             QT(1): entspricht der Warteschlange
                                    Q_WCPU in Abb. 3.15 ohne Proto-
                                    kollmoduln,
                             QT(2): entspricht der Warteschlange
                                    Q_PEND in Abb. 3.15 ohne Proto-
                                    kollmoduln */;
/* ... weitere Variablenvereinbarungen ... */;

/* Verwendete Prozeduren: */
procedure QSCHEDULER (SERVQ,QSTRATEGY);
/* Eingabeparameter: QSTRATEGY.
   Ausgabeparameter: SERVQ.
   Aufgabe: Auswahl der Warteschlange, deren Bearbeitung (durch die
            Betriebsmittelverwaltungsinstanz) höchstprior ist.
   Effekt: SERVQ kennzeichnet die ausgewählte Taskwarteschlange:
           SERVQ = -1: keine Warteschlange zu bearbeiten,
           SERVQ = 0 : Bearbeitung von QT(0) höchstprior,
           SERVQ = 1 : Bearbeitung von QT(1) höchstprior,
           SERVQ = 3 : Bearbeitung von QPM höchstprior;
           (QSTRATEGY dient zur Steuerung der Warteschlangenaus-
           wahl). */;

procedure SELFREECPU (FCPU);
/* Ausgabeparameter: FCPU.
   Aufgabe: Auswahl einer freien CPU.
   Effekt: FCPU kennzeichnet eine CPU, deren aktuelle Belegung
           durch eine Task möglich ist. */;
```

```
        procedure SELLOWPTINS2 (TSK);
        /* Ausgabeparameter: TSK.
           Aufgabe: Auswahl der Task mit niedrigster Priorität aus der
                    Menge der Tasks, die aktuell eine CPU belegen und de-
                    nen noch kein "Flag" gesetzt wurde.
           Effekt: TSK verweist auf die gefundene Task. */;

        /* ... Variablen-Initialisierungen ... */
        while true do
        begin
             /* ... Betriebsmittelverwaltungsinstanz-Schleife (s. Abb. 4.6
                in Kap. 4) ... */
        end of loop;
    end of RECHNERNETZKNOTEN;
```

Bemerkungen: (1) Die Prozeduren SELFREECPU und SELLOWPTINS2 sind nur notwendig im Falle von Mehrprozessorsystemen.

(2) Die aktuelle Implementierung von MOSAIC ermöglicht nur die Modellierung von Einprozessorsystemen; das Konzept ist jedoch bereits für evtl. Erweiterungen (in bezug auf eine Simulation von Mehrprozessorsystemen) ausgelegt.

A.3. Prozeduren zur Ergänzung der Rechnernetzmodelle

Für eine Reihe von Aktivitäten, die verschiedenen Modellkomponenten gemeinsam sind, stehen in MOSAIC die folgenden global gültigen Prozeduren zur Verfügung:

```
procedure SCHEDNEXTACT (TSK);
/* Eingabeparameter: TSK.
   Aufgabe: Bestimmung des internen Zustands der Benutzertask TSK, welcher
            nach Verlassen des aktuellen Zustands angenommen wird; die Aus-
            wahl des Nachfolgezustands erfolgt in Abhängigkeit des Momentan-
            zustands unter Zugrundelegung einer vorgegebenen Wahrscheinlich-
            keitsverteilung.
   Effekt: Nach Ausführung der Prozedur verweist die Variable TSK.NEXTACT
           auf den ausgewählten Nachfolgezustand. */
```

```
procedure SELECTUT (QUEUE,STRATEGY,TSK);
/* Eingabeparameter: QUEUE,STRATEGY.
   Ausgabeparameter: TSK.
   Aufgabe: Auswahl einer Task TSK aus der Warteschlange QUEUE unter Be-
            rücksichtigung der Strategie STRATEGY.
   Effekt: TSK verweist auf die gefundene Task. */

procedure SELECTPM (QUEUE,COMPID,STRATEGY,ELEM,SEND);
/* Eingabeparameter: QUEUE,COMPID,STRATEGY.
   Ausgabeparameter: ELEM,SEND.
   Aufgabe: Auswahl eines Protokollmoduls aus der Warteschlange QUEUE unter
            Berücksichtigung der Strategie STRATEGY (COMPID dient der Identifi-
            kation des Rechners).
   Effekt: Nach Prozedurausführung enthält ELEM die rechnerinterne Nummer der
           Protokolleinheit, die den gefundenen Protokollmoduln beinhaltet,
           und SEND = true (false) besagt, daß es sich um einen Sende- (Emp-
           fangs-) Protokollmodul handelt. */

procedure SELECTCP (COMPID,RECCOMP,RECNODE,ROUTALG);
/* Eingabeparameter: COMPID,RECCOMP,ROUTALG.
   Ausgabeparameter: RECNODE.
   Aufgabe: Bestimmung des nächsten Rechnernetzknotens, an den eine Datenein-
            heit weiterzuleiten ist, die den Rechner COMPID mit dem Endziel
            RECCOMP verläßt (ROUTALG bezeichnet den zu verwendenden Routing-
            Algorithmus).
   Effekt: RECNODE gibt die globale Rechnernummer des gefundenen Rechnernetz-
           knotens. */

procedure SELECTREQUEST (COMPID,QUEUE,STRATEGY,REQ,LASTREQ);
/* Eingabeparameter: COMPID,QUEUE,STRATEGY,LASTREQ.
   Ausgabeparameter: REQ.
   Aufgabe: Auswahl eines Schnittstellenauftrags REQ aus der Warteschlange
            QUEUE unter Berücksichtigung der Strategie STRATEGY und des zu-
            letzt bearbeiteten Warteschlangenelements LASTREQ (COMPID dient
            der Identifikation des Rechners).
   Effekt: REQ verweist auf den durch die Prozedur ausgewählten Schnitt-
           stellenauftrag */
```

```
procedure DELETEDU (DU,COMPID,PUNUM);
/* Eingabeparameter: DU,COMPID,PUNUM.
   Aufgabe: Vernichtung einer Dateneinheit (COMPID und PUNUM dienen der Iden-
            tifikation des Rechners bzw. der Protokolleinheit, die die Da-
            teneinheit beinhalten).
   Effekt: Die Prozedur führt sämtliche bei der Vernichtung der Dateneinheit
           DU notwendigen Aktivitäten (z.B. Speicherbereinigung, Dateneinhei-
           tenvermessung) durch. */

procedure COPYDU (DUOLD,DUNEW);
/* Eingabeparameter: DUOLD.
   Ausgabeparameter: DUNEW.
   Aufgabe: Erstellen einer Kopie für beliebig strukturierte Dateneinheiten.
   Effekt: DUNEW verweist auf eine Kopie der Dateneinheit DUOLD. */
```

Anhang B: Programmstatistik der aktuellen Ausbaustufe von MOSAIC

Länge des Quellprogramms:

(a) "Modell-Teil" : ca. 6000 SIMULA-Anweisungen
(b) "Graphik-Teil" : ca. 500 FORTRAN-Anweisungen

Länge des Loadmoduls: ca. 220 [KByte]

Einige Laufzeiten [sec] auf einer IBM 3033 am Beispiel der Experimentserien EXP_1 und EXP_2 aus Kapitel 6:

Experiment	Anzahl Rechnernetzknoten	Anzahl Protokollschichten	zu übertragender Dateneinheiten	Speicherplatz [KByte]	CPU-Laufzeiten [sec]
EXP_1	3	1	401 - 2836	320	9 - 46
EXP_1	3	2	628 - 2588	320	18 - 56
EXP_1	3	3	667 - 2513	320	25 - 42
EXP_2	6	3	356	450	7
EXP_2	6	3	1256	450	23
EXP_2	6	3	2367	450	45
EXP_2	6	3	4258	450	91