KfK-PDV 186
August 1980
(1. Ex.)

PDV-Berichte

Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage Teil 2

G. Hommel
Kernforschungszentrum
Karlsruhe GmbH

Kernforschungszentrum Karlsruhe

PDV-Berichte

Die Kernforschungszentrum Karlsruhe GmbH koordiniert und betreut im Auftrag des Bundesministers für Forschung und Technologie das im Rahmen der Datenverarbeitungsprogramme der Bundesregierung geförderte Projekt Prozeßlenkung mit Datenverarbeitungsanlagen (PDV). Hierbei arbeitet sie eng mit Unternehmen der gewerblichen Wirtschaft und Einrichtungen der öffentlichen Hand zusammen. Als Projektträger gibt sie die Schriftenreihe PDV-Berichte heraus. Darin werden Entwicklungsunterlagen zur Verfügung gestellt, die einer raschen und breiteren Anwendung der Datenverarbeitung in der Prozeßlenkung dienen sollen.

Der vorliegende Bericht dokumentiert Kenntnisse und Ergebnisse, die im Projekt PDV gewonnen wurden.

Verantwortlich für den Inhalt sind die Autoren. Die Kernforschungszentrum Karlsruhe GmbH übernimmt keine Gewähr insbesondere für die Richtigkeit, Genauigkeit und Vollständigkeit der Angaben, sowie die Beachtung privater Rechte Dritter.

Druck und Verbreitung:

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640 7500 Karlsruhe 1

Bundesrepublik Deutschland

PDV

KfK-PDV 186

PROJEKT PROZESSLENKUNG MIT DV-ANLAGEN

FORSCHUNGSBERICHT KFK-PDV 186

VERGLEICH VERSCHIEDENER SPEZIFIKATIONSVERFAHREN AM BEISPIEL EINER PAKETVERTEILANLAGE

TEIL 2

VON

Euentert

G. HOMMEL (HRSG.)

KERNFORSCHUNGSZENTRUM KARLSRUHE GMBH

335 SEITEN

AUGUST 1980

Inhaltsverzeichnis

Teil 2

Die mit o gekennzeichneten Beiträge enthalten Ergebnisse aus Forschungs- und Entwicklungsvorhaben, die im Rahmen des 3. DV-Programms vom Bundesminister für Forschung und Technologie gefördert wurden.

Petri-Netze als Hilfsmittel zur methodischen Software - Entwicklung

H. Bischeltsrieder

Juni 1980

Inhalt

Zusammenfassung:

Derzeit weitverbreitet eingesetzte Software-Entwicklungs-Methoden und -Hilfsmittel sind für konventionelle und in ihrer Logik sequentielle Systeme und Programmabläufe geeignet. Sie bieten nicht ausreichend Unterstützung bei der Entwicklung von Prozeßsteuerungssoftware, verteilten Datenbanksystemen, Rechnernetzen etc., da sie Nebenläufigkeiten und Prozeßinteraktionen nicht berücksichtigen.

Petri-Netze sind gerade für die Darstellung von parallelen Abläufen sowie Prozeß-Synchronisation und -Kommunikation geeignet und können daher als Grundlage oder als zusätzliches Hilfsmittel beim Entwurf paralleler Prozesse und verteilter Systeme herangezogen werden. Anhand eines Beispiels aus der Prozeßsteuerung wird in diesem Vortrag eine Einsatzmöglichkeit von Petri-Netzen als Hilfsmittel zur Software-Entwicklung vorgestellt.

Einleitung:

Dieser Bericht versteht sich als Ergebnis eines Experiments. Im Rahmen des Arbeitskreises "Systematische Entwicklung von PDV-Systemen" war die Aufgabe gestellt, anhand eines Beispiels aus der realen Anwendungswelt, nämlich einer zu automatisierenden Paketverteilanlage, die Software-Erstellung mittels verschiedener Methoden und Tools, darunter auch Petri-Netze, zu demonstrieren.

Auch wenn von erfolgreichen Anwendungen der Petri-Netz-Theorie auf dem Gebiet der Software-Erstellung, insbesondere der Erstellung von Prozeßrechner-Software berichtet wird /1/, war dem Verfasser zum damaligen Zeitpunkt diesbezüglich nichts näheres darüber bekannt. Der allgemein bekannte Petri-Netz-Begriff selbst, Gerüchte (die sich inzwischen für den Verfasser bewahrheitet haben) über Möglichkeiten und erfolgreiche Versuche, Petri-Netze mit formaler Logik in Zusammenhang zu bringen, Erkenntnisse durch mehr aus Zufall gewonnenen Einblick in GMD-Intern-Berichte und eine gute Portion negativer Erfahrungen beim Versuch Petri-Netze in der Praxis (Darstellung von Betriebssystem-Abläufen) einzusetzen, waren Anfang 1979 die Basis für den vorliegenden und wohl mißlungenen Versuch zu zeigen, daß Petri-Netze für Software-Engineering bei parallelen Prozeßabläufen nicht geeignet seien.

Mittlerweile hat sich der Informationsaustausch unter den "Petri-Jüngern" wesentlich verbessert (Advanced Course in Hamburg, GI-Fachgruppe, diverse Workshops, mehr Veröffentlichungen etc.) und die dadurch gewonnenen neuen Aspekte und Erkenntnisse sind bei einer Überarbeitung dieses Berichts weitgehend eingeflossen.

Folgende Ziele wurden angestrebt:

- Ausnutzung der grafischen Darstellungsmöglichkeiten von Petri-Netzen zur Veranschaulichung des Systems und seines Ablaufs,

- Darstellung von technischem Prozeß (zu steuerndem Prozeß), Rechenprozeß (steuernden Prozeß) und der Schnittstelle zwischen steuerndem und zu steuernden Prozeß,

- Beschränkung auf eine möglichst einfache Petri-Netz-Form ("binäre Petri-Netze") um möglichst viele Aussagen über die Netzdarstellung formal gewinnen zu können,

- Verwendung der Relation zwischen "binären" Petri-Netzen und formaler Logik,

- Simulierbarkeit des Gesamtsystems und einzelner Komponenten,

- Entwicklung und Herausarbeitung einer systematischen Vorgehensweise nach der "top-down"-Methode.

Die letzte Zielsetzung zeigt, daß auf nichts in dieser Art Bestehendes zurückgegriffen wurde. Der Ansatz war zunächst rein pragmatisch. Erst nach den ersten Teilerfolgen wurde versucht, die Vorgehensweise theoretisch zu untermauern, was rückkoppelnd wieder Auswirkungen auf die zu entwickelnde Vorgehensweise hatte. Dieser Prozeß ist wohl noch nicht abgeschlossen, weshalb das vorliegende Papier mehr als Zwischenbericht zu betrachten ist, insbesondere da bisher nur anhand eines Beispiels (der Prozeßsteuerungsanlage) die hier beschriebene Vorgehensweise erprobt wurde.

Schließlich sei darauf hingewiesen, daß es sich bei dieser Arbeit um ein "Hobby-Projekt" handelt, das von der IABG wohlwollend unterstützt, aber nicht getragen wird. Deshalb ist es mit der Tool-Unterstützung nicht so weit her, wie vielleicht durch die folgenden Ausführungen der Eindruck entstehen mag. Was bisher zur Verfügung stand, ist ein Beweisprogramm für aussagenlogische Formeln, ein Lineal mit verschieden großen runden Löchern und Papier und Bleistift.

Teil I: Theoretische Grundlagen

0. DER BEGRIFF "BINÄRES PETRI-NETZ"

Ein "binäres Petri-Netz" kann als zweigeteilter, gerichteter Graph betrachtet werden, mit zwei Arten von Knoten, Stellen und Transitionen. Stellen werden graphisch als Kreise dargestellt, Transitionen als Balken oder Striche. Die Menge der Stellen, von denen ein Pfeil zu einer Transition t führt, nennt man Stellenvorbereich von t; die Menge der Stellen zu denen von einer Transition t aus Pfeile führen, bezeichnet man analog als Stellennachbereich von t.

In (oder auf) den Stellen können sich Marken befinden, bei einem binären Petri-Netz pro Stelle höchstens eine Marke. Man sagt, eine Stelle ist markiert oder eine Stelle ist unmarkiert, je nachdem. Die Marken können sich durch Schalten von Transitionen bewegen. Eine Transition t ist schaltbereit oder schaltfähig, wenn alle Stellen des Stellenvorbereichs von t markiert und alle Stellen des Stellennachbereichs von t unmarkiert sind. Eine schaltfähige Transition t schaltet, indem sie aus allen Stellen ihres Stellenvorbereichs die Marken entfernt und alle (unmarkierten) Stellen ihres Stellennachbereichs markiert.

Eine Transition nennt man blockiert, wenn sie nicht schaltfähig ist. Ein Petri-Netz ist total verklemmt wenn alle Transitionen blockiert sind.

Es sei darauf hingewiesen, daß Petri-Netze und binäre Petri-Netze in der Literatur manchmal auch anders definiert werden und sich die hier gegebene Definition und die folgenden Ausführungen und Betrachtungen nicht immer genau mit anderen Definitionen und Auffassungen decken. Der hier definierte Begriff stellt außerdem einen Spezialfall dar.

1. PETRI-NETZ-DARSTELLUNG UND AUSSAGENLOGISCHE AUSDRÜCKE

Grundlegender Begriff der formalen Logik ist die Folgerung (wenn ... dann ...), in der Regel bezeichnet durch "$\Rightarrow$".

Seien A und B Aussagen, dann gilt:

A	B	A $\Rightarrow$ B
w	w	w
w	f	f
f	w	w
f	f	w

mit w: die Aussage ist wahr,
f: die Aussage ist falsch

Die äquivalente Petri-Netz-Darstellung ist

wobei gilt:

A	B	T
m	m	b
m	u	s
u	m	b
u	u	b

mit m: die Stelle ist markiert
u: die Stelle ist unmarkiert
s: die Transition ist schaltbereit
b: die Transition ist blockiert (d. h. sie kann nicht schalten)

(Dabei ist wichtig, zu beachten, daß - in binären Petri-Netzen - eine Transition nur dann schaltbereit ist und schalten kann, wenn alle Eingabestellen markiert sind und alle Ausgabestellen unmarkiert sind.)

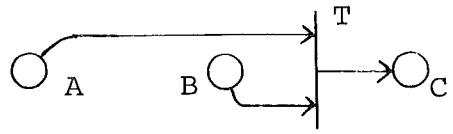
Die Gegenüberstellung der aussagenlogischen Formulierung und der Netz-Darstellung zeigt, daß eine wahre Aussage (z. B. A $\Rightarrow$ B mit A: wahr und B: wahr) einem Netz entspricht, in dem keine Transition schaltbereit ist (also einem total verklemmten Netz), eine falsche Aussage (z. B. A $\Rightarrow$ B mit A: wahr und B: falsch) einem Netz, in dem (mindestens) eine Transition schaltbereit ist.

Dazu weitere Beispiele:

Im folgenden bedeute

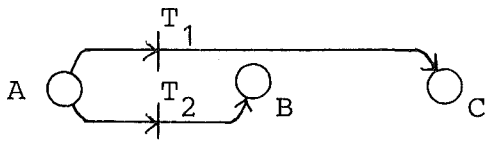
w: die Aussage ist wahr
f: die Aussage ist falsch
m: die Stelle ist markiert
u: die Stelle ist unmarkiert
s: mindestens eine Transition ist schaltbereit
b: alle Transitionen sind nicht schaltbereit/ alle Transitionen sind blockiert
A, B, C: (elementare) Aussagen

$(A \wedge B) \Rightarrow C$



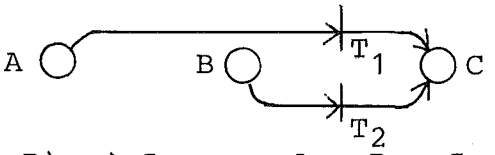
A	B	C	$A \wedge B \Rightarrow C$	A	B	C	T
w	w	w	w	m	m	m	b
w	w	f	f	m	m	u	s
w	f	w	w	m	u	m	b
w	f	f	w	m	u	u	b
f	w	w	w	u	m	m	b
f	w	f	w	u	m	u	b
f	f	w	w	u	u	m	b
f	f	f	w	u	u	u	b

$A \Longrightarrow (B \wedge C)$



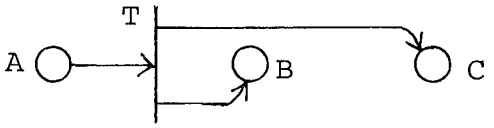
A	B	C	$A \Longrightarrow (B \wedge C)$	A	B	C	Netz
w	w	w	w	m	m	m	b
w	w	f	f	m	m	u	s (T_1)
w	f	w	f	m	u	m	s (T_2)
w	f	f	f	m	u	u	s (T_1,T_2)
f	w	w	w	u	m	m	b
f	w	f	w	u	m	u	b
f	f	w	w	u	u	m	b
f	f	f	w	u	u	u	b

$(A \vee B) \Longrightarrow C$



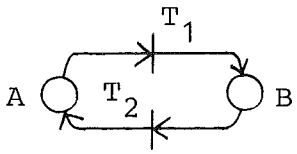
A	B	C	$(A \vee B) \Longrightarrow C$	A	B	C	Netz
w	w	w	w	m	m	m	b
w	w	f	f	m	m	u	s (T_1,T_2)
w	f	w	w	m	u	m	b
w	f	f	f	m	u	u	s (T_1)
f	w	w	w	u	m	m	b
f	w	f	f	u	m	u	s (T_2)
f	f	w	w	u	u	m	b
f	f	f	w	u	u	u	b

$A \Longrightarrow (B \vee C)$



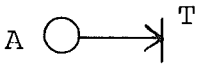
A	B	C	$A \Longrightarrow (B \vee C)$	A	B	C	T
w	w	w	w	m	m	m	b
w	w	f	w	m	m	u	b
w	f	w	w	m	u	m	b
w	f	f	f	m	u	u	s
f	w	w	w	u	m	m	b
f	w	f	w	u	m	u	b
f	f	w	w	u	u	m	b
f	f	f	w	u	u	u	b

$A \Longleftrightarrow B$



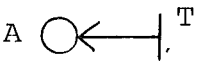
A	B	A ⟺ B	A	B	Netz
w	w	w	m	m	b
w	f	f	m	u	s (T_1)
f	w	f	u	m	s (T_2)
f	f	w	u	u	b

$A \Longrightarrow \neg A$



A	¬A	A ⟹ ¬A	A	T
w	f	f	m	s
f	w	w	u	b

$\neg A \Longrightarrow A$



A	¬A	¬A ⟹ A	A	T
w	f	w	m	b
f	w	f	u	s

Die beiden letzten Beispiele zeigen, daß $A \Longrightarrow \neg A$ gleichbedeutend mit $\neg A$ und $\neg A \Longrightarrow A$ gleichbedeutend mit A ist.
Für die anderen Beispiele gilt:

$$(A \wedge B) \Rightarrow C \quad \Longleftrightarrow \quad (A \Rightarrow C) \vee (B \Rightarrow C)$$

$$A \Rightarrow (B \wedge C) \quad \Longleftrightarrow \quad (A \Rightarrow B) \wedge (A \Rightarrow C)$$

$$(A \vee B) \Rightarrow C \quad \Longleftrightarrow \quad (A \Rightarrow C) \wedge (B \Rightarrow C)$$

$$A \Rightarrow (B \vee C) \quad \Longleftrightarrow \quad (A \Rightarrow B) \vee (A \Rightarrow C)$$

$$A \Leftrightarrow B \quad \Longleftrightarrow \quad (A \Rightarrow B) \wedge (B \Rightarrow A)$$

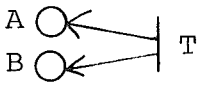
Berücksichtigt man weiter, daß gilt:

A	B	A ⟹ B	¬A	B	¬A∨B	(A⟹B)⟺(¬A∨B)
w	w	w	f	w	w	w
w	f	f	f	f	f	w
f	w	w	w	w	w	w
f	f	w	w	f	w	w

also daß (A⟹B)⟺(¬A∨B) allgemeingültig (d. h. immer wahr) ist, so zeigt sich, daß eine "oder"-Funktion genau einer Transition entspricht.

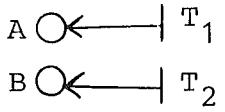
Damit (und mit A ⟺ (A∨A) lassen sich die aussagenlogischen "oder"- und "und"-Funktionen sowie die immer wahre Funktion (Tautologie: A∨¬A) und die immer falsche Funktion (A∧¬A) als Petri-Netz darstellen:

(A∨B)



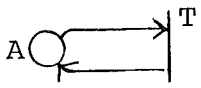
A	B	A ∨ B	A	B	T
w	w	w	m	m	b
w	f	w	m	u	b
f	w	w	u	m	b
f	f	f	u	u	s

(A ∧ B) <⟹> (¬A => A) ∧ (¬B => B)
<⟹> (A ∨ A) ∧ (B ∨ B)



A	B	A∧B	A	B	Netz
w	w	w	m	m	b
w	f	f	m	u	s (T_2)
f	w	f	u	m	s (T_1)
f	f	f	u	u	s (T_1,T_2)

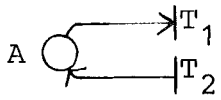
$A \vee \neg A$



A	¬A	A∨¬A	A	T
w	f	w	m	b
f	w	w	u	b

$A \wedge \neg A \longleftrightarrow (\neg A \Rightarrow A) \wedge (A \Rightarrow \neg A)$

$A \; A \longleftrightarrow (A \vee A) \wedge (\neg A \vee \neg A)$

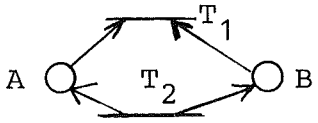


A	¬A	A∧¬A	A	Netz = T_1 oder T_2
w	f	f	m	s (T_1)
f	w	f	u	s (T_2)

damit läßt sich nun jede aussagenlogische Formel als Petri-Netz darstellen:

<u>Exklusives oder:</u>

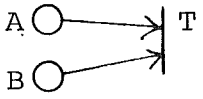
$A \text{ .xor. } B \Longleftrightarrow (A \wedge \neg B) \vee (\neg A \wedge B) \Longleftrightarrow$
$(A \vee \neg A) \wedge (A \vee B) \wedge (\neg B \vee \neg A) \wedge (\neg B \vee B) \Longleftrightarrow$
$(A \vee B) \wedge (\neg A \vee \neg B)$



A	B	A.xor.B	A	B	Netz
w	w	f	m	m	s (T_1)
w	f	w	m	u	b
f	w	w	u	m	b
f	f	f	u	u	s (T_2)

Nand:

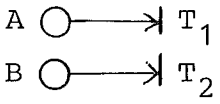
$A.nand.\ B \Longleftrightarrow \neg(A \wedge B) \Longleftrightarrow \neg A \vee \neg B$



A	B	A.nand.B	A	B	T
w	w	f	m	m	s
w	f	w	m	u	b
f	w	w	u	m	b
f	f	w	u	u	b

Nor:

$A.nor.B \Longleftrightarrow \neg(A \vee B) \Longleftrightarrow \neg A \wedge \neg B \Longleftrightarrow (\neg A \vee \neg A)_{\wedge} (\neg B \vee \neg B)$



A	B	A.nor.B	A	B	Netz
w	w	f	m	m	s (T_1, T_2)
w	f	f	m	u	s (T_1)
f	w	f	u	m	s (T_2)
f	f	w	u	u	b

Um zu einer Petri-Netz-Darstellung zu gelangen, empfiehlt es sich, den vorgegebenen aussagenlogischen Ausdruck in die "clause-Normalform" zu bringen, d. h. den Ausdruck so umzuwandeln, daß er nur noch aus einer Reihe durch "und" verknüpfter "oder" besteht. Für die (elementaren) Aussagen zeichnet man dann Stellen und für die "oder"-Verknüpfungen Transitionen. Die durch "oder" verknüpften (elementaren) Aussagen (Stellen) verbindet man durch Pfeile mit den entsprechenden "oder" (Transitionen) und zwar bei negierten Elementaraussagen von der Stelle zur Transition (z. B. $\neg A$: ) sonst von der Transition zur Stelle (z. B. A: ).

$$((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$$

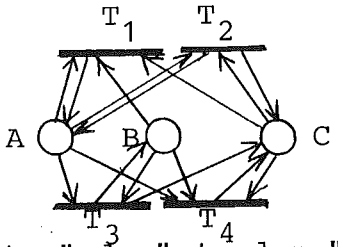
$$\Longleftrightarrow \neg((\neg A \vee B) \wedge (\neg B \vee C)) \vee (\neg A \vee C)$$

$$\Longleftrightarrow \neg(\neg A \vee B) \vee \neg(\neg B \vee C) \vee \neg A \vee C$$

$$\Longleftrightarrow (A \wedge \neg B) \vee (B \wedge \neg C) \vee \neg A \vee C$$

$$\Longleftrightarrow (A \vee B \vee \neg A \vee C) \wedge (A \vee \neg C \vee \neg A \vee C) \wedge (\neg B \vee B \vee \neg A \vee C) \wedge (\neg B \vee \neg C \vee \neg A \vee C)$$

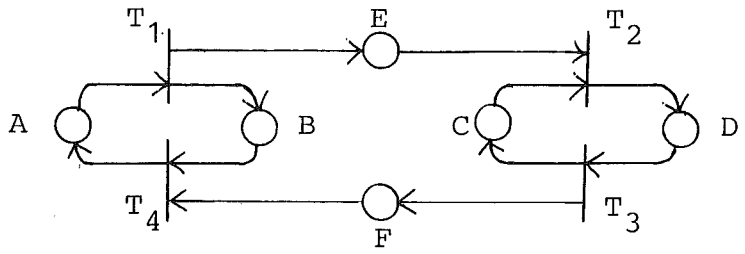
$$[\Longleftrightarrow T1 \wedge T2 \wedge T3 \wedge T4]$$



(Enthält ein "oder" in der "clause-Normalform" eine Tautologie (z. B. Av¬A), so ist sie immer wahr, d. h. die entsprechende Transition ist immer blockiert. Sind alle Transitionen immer blockiert, wie im vorangegangenen Beispiel, so ist der entsprechende aussagenlogische Ausdruck allgemeingültig.)

Will man zu einem Petri-Netz eine der möglichen aussagenlogischen Formulierungen erhalten, so bildet man für jede Transition einen Minterm der "clause-Normalform", umgekehrt zu dem oben beschriebenen Vorgehen.

Beispiel 1.2



T1 : ¬A ∨ B ∨ E
T2 : ¬C ∨ ¬E ∨ D
T3 : ¬D ∨ C ∨ F
T4 : ¬B ∨ ¬F ∨ A

<---> A => (B ∨ E)
<---> (C ∧ E) => D
<---> D => (C ∨ F)
<---> (B ∧ F) => A

T1 ∧ T2 ∧ T3 ∧ T4 <--->
(¬A ∨ B ∨ E) ∧ (¬C ∨ ¬E ∨ D) ∧ (¬D ∨ C ∨ F) ∧ (¬B ∨ ¬F ∨ A)

2. VERKLEMMUNGEN UND FORTSETZBARKEIT

In einem binären Petri-Netz mit n Stellen sind 2^n Markierungen möglich. Diese Markierungen kann man einteilen in

(i) Markierungen, bei denen keine Transition des Netzes schalten kann,
(ii) Markierungen, bei denen mindestens eine Transition schaltfähig ist,
(iii) Markierungen, die sicher zu totalen Verklemmungen führen,
(iv) Markierungen, die zu totalen Verklemmungen führen können,
(v) Markierungen, die zu partiellen Verklemmungen führen (1-lebendig),
(vi) Markierungen, bei denen das Netz lebendig ist (2-lebendig).

In den Fällen (i) und (ii) handelt es sich um "statische" Aussagen über die Markierungen, d. h. die Klassifizierung von Markierungen bezüglich der Fälle (i) und (ii) kann ohne Schalten von Transitionen, also ohne Dynamik durchgeführt werden. Für die Fälle (iii) und (iv) ist zur Klassifizierung von Markierungen Schalten erforderlich. Bei bestimmten Arten von Petri-Netzen kann man die Fälle (v) und (vi) statisch, d. h. ohne Schalten bestimmen, sonst muß der Fluß im Netz betrachtet werden.

Mit Hilfe der aussagenlogischen Darstellung von Petri-Netzen ist der Nachweis einiger Eigenschaften von Markierungen und von bestimmten Petri-Netz-Formen möglich:

- Ist die aussagenlogische Darstellung eines Petri-Netzes eine allgemeingültige Formel, so ist bei jeder möglichen Markierung des Netzes jede Transition blockiert.

- Ist die Negation der aussagenlogischen Darstellung eines Petri-Netzes eine allgemeingültige Formel, so ist bei jeder möglichen Markierung des Netzes mindestens eine Transition schaltfähig, d. h. das Netz ist lebendig (1- oder 2-lebendig), unabhängig von einer Anfangsmarkierung.

- Alle Markierungen, für die die aussagenlogische Darstellung "wahr" ergibt, sind totale Verklemmungen.

- Rückwärtsschalten:
  Man kann die Pfeilrichtungen in einem Petri-Netz ändern, indem man in der aussagenlogischen Darstellung jede Elementaraussage (≙Stelle) negiert. Alle Markierungen, durch die jede Transition blockiert ist,(d. h. alle Markierungen für die aussagenlogische Darstellung "wahr" ergibt), sind dann nicht erreichbare Markierungen. Interessant ist dabei insbesondere die Erkennbarkeit nicht erreichbarer totaler Verklemmungen und deren Aussonderungsmöglichkeit. Außerdem können nicht erreichbare Markierungen keinen Schaltzyklen angehören.

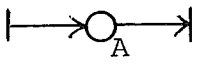
Beispiel 2.1

Ein Petri-Netz, in dem jede Transition bei jeder Markierung blockiert ist, entspricht einer allgemeingültigen aussagenlogischen Formel, die auf "clause"-Normalform gebracht in jedem Term eine Tautologie der Form $A \vee \neg A$ enthält. Ein solches Netz wurde im Beispiel 1.1 behandelt.

Beispiel 2.2

Folgende Petri-Netze enthalten bei jeder möglichen Markierung eine schaltfähige Transition:

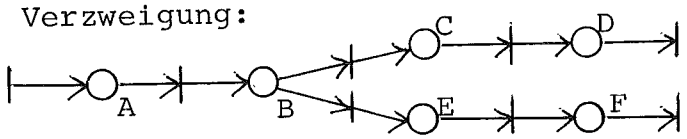
a) die immer falsche Formel $A \wedge \neg A$, die negiert eine Tautologie darstellt:



b) aneinandergereihte immer falsche Formeln

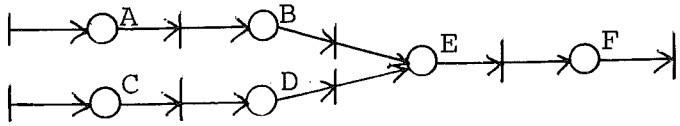
$$A \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee D) \wedge \neg D$$

c) die Verzweigung:



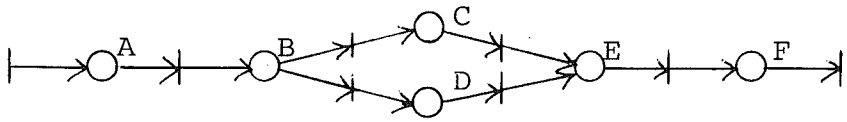
$$A \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee D) \wedge \neg D$$
$$\wedge (\neg B \vee E) \wedge (\neg E \vee F) \wedge \neg F$$

d) die Zusammenführung



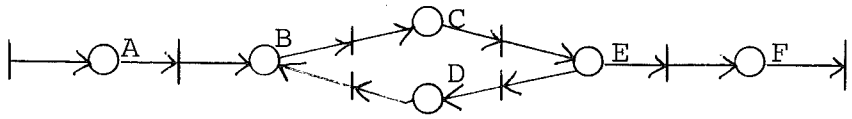
$$A \wedge (\neg A \vee B) \wedge (\neg B \vee E) \wedge C \wedge (\neg C \vee D) \wedge (\neg D \vee E)$$
$$\wedge (\neg E \vee F) \wedge \neg F$$

e) die bedingte Verzweigung



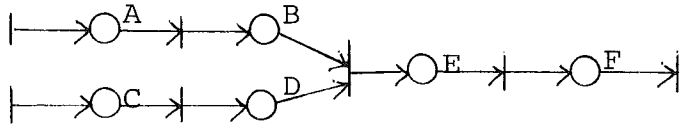
$$A \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee E) \wedge (\neg B \vee D) \wedge (\neg D \vee E) \wedge (\neg E \vee F) \wedge \neg F$$

f) die Schleife



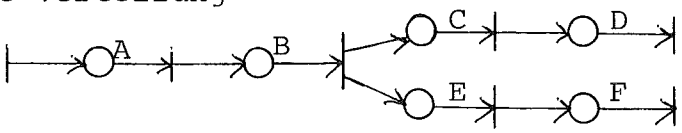
$$A \wedge (\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee E) \wedge (\neg E \vee D) \wedge (\neg D \vee B) \wedge (\neg E \vee F) \wedge \neg F$$

g) die Zusammenfassung



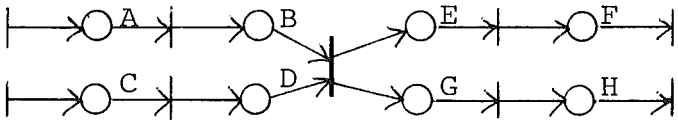
$$A \wedge (\neg A \vee B) \wedge C \wedge (\neg C \vee D) \wedge (\neg B \vee \neg D \vee E) \wedge (\neg E \vee F) \wedge \neg F$$

h) die Verteilung



$$A \wedge (\neg A \vee B) \wedge (\neg B \vee C \vee E) \wedge (\neg C \vee D) \wedge \neg D \wedge (\neg E \vee F) \wedge \neg F$$

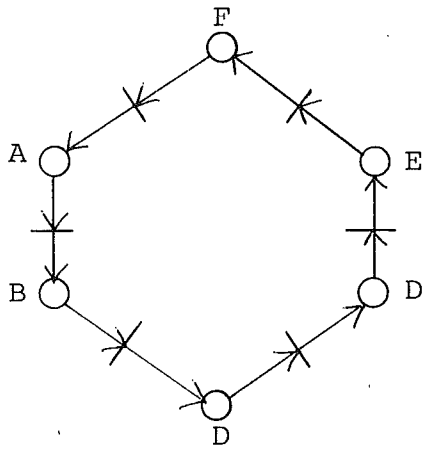
i) die Synchronisation



$$A \wedge (\neg A \vee B) \wedge C \wedge (\neg C \vee D) \wedge (\neg B \vee \neg D \vee E \vee G) \wedge (\neg E \vee F) \wedge \neg F \wedge (\neg G \vee H) \wedge \neg H$$

Beispiel 2.3

Der Kreis:



$(\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee D) \wedge (\neg D \vee E) \wedge (\neg E \vee F) \wedge (\neg F \vee A)$

Verklemmungen bei

$A \wedge B \wedge C \wedge D \wedge E \wedge F$ (d.h. alle Stellen markiert)

$\neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg F$ (d.h. keine Stelle markiert)

Rückwärtsschalten:

$(A \vee \neg B) \wedge (B \vee \neg C) \wedge (C \vee \neg D)$

$\wedge (D \vee \neg E) \wedge (E \vee \neg F) \wedge (F \vee \neg A)$

nicht erreichbare Markierungen:

$\neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg F$ (d.h. keine Stelle markiert)

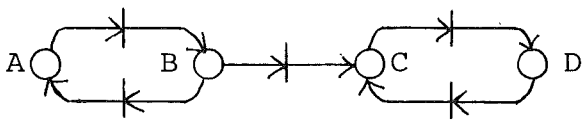
$A \wedge B \wedge C \wedge D \wedge E \wedge F$ (d.h. alle Stellen markiert)

alle Verklemmungen sind nicht erreichbare Markierungen,
d. h. außer in den Anfangsmarkierungen

$A \wedge B \wedge C \wedge D \wedge E \wedge F$ $\qquad$ $\neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg F$

ist das Petri-Netz lebendig.

Beispiel 2.4



$$(\neg A \vee B) \wedge (\neg B \vee A) \wedge (\neg B \vee C) \wedge (\neg C \vee D) \wedge (\neg D \vee C)$$

Verklemmungen bei

$$\begin{array}{l} \neg A \wedge \neg B \wedge C \wedge D \\ A \wedge B \wedge C \wedge D \\ \neg A \wedge \neg B \wedge \neg C \wedge \neg D \end{array}$$

Rückwärtsschalten:

$$(A \vee \neg B) \wedge (B \vee \neg A) \wedge (B \vee \neg C) \wedge (C \vee \neg D) \wedge (D \vee \neg C)$$

nicht erreichbare Markierungen

$$\begin{array}{l} A \wedge B \wedge \neg C \wedge \neg D \\ \neg A \wedge \neg B \wedge \neg C \wedge \neg D \\ A \wedge B \wedge C \wedge D \end{array}$$

nicht erreichbare Verklemmungen

$$= \{\text{Verklemmungen}\} \cap \{\text{nicht erreichbare Markierungen}\} :$$

$\neg A \wedge \neg B \wedge \neg C \wedge \neg D$ (d. h. keine Stelle markiert)

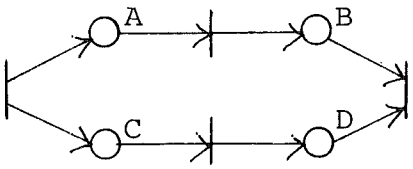
$A \wedge B \wedge C \wedge D$ (d. h. alle Stellen markiert)

erreichbare Verklemmungen

$$= \{\text{Verklemmungen}\} \setminus \{\text{nicht erreichbare Verklemmungen}\}$$

$$\neg A \wedge \neg B \wedge C \wedge D$$

Beispiel 2.5:



$$(A \vee C) \wedge (\neg A \vee B) \wedge (\neg C \vee D) \wedge (\neg B \vee \neg D)$$

Verklemmungen bei

$$\begin{array}{l} A \wedge B \wedge \neg C \wedge \neg D \\ \neg A \wedge \neg B \wedge C \wedge D \end{array}$$

Rückwärtsschalten:

$$(\neg A \vee \neg C) \wedge (A \vee \neg B) \wedge (C \vee \neg D) \wedge (B \vee D)$$

nicht erreichbare Markierungen:

$$\begin{array}{l} \neg A \wedge \neg B \wedge C \wedge D \\ A \wedge B \wedge \neg C \wedge \neg D \end{array}$$

nicht erreichbare Verklemmungen

$$= \{\text{Verklemmungen}\} \cap \{\text{nicht erreichbare Markierungen}\}$$

$$= \{ (\neg A \wedge \neg B \wedge C \wedge D) , (A \wedge B \wedge \neg C \wedge \neg D) \}$$

= alle Verklemmungen

3. EIN ANWENDUNGSBEISPIEL

Am Beispiel einer einfachen Prozeßsteuerung wird eine Anwendungsmöglichkeit vorgestellt. Um den Rahmen dieses Papiers nicht zu sprengen, wird zunächst ein idealisiertes Modell behandelt und dann im Hinweis auf reale Anwendungsmöglichkeiten gegeben.

a) Idealisiertes Modell einer Prozeßsteuerung:

Vorgegeben sei ein technischer Preozß, ein Steuerungssystem und eine Schnittstelle zwischen technischem Prozeß und Steuerungssystem folgender Art:

Technischer Prozeß (TP):
Die Tätigkeiten des TP sind:

t_1: Heranführen eines Prozeßgutes,

t_2: Melden des Vorhandenseins eines Prozeßgutes und Weiterführen des Prozeßgutes,

t_3: Bearbeiten des Prozeßgutes auf Anweisung und Weiterführung des Prozeßgutes,

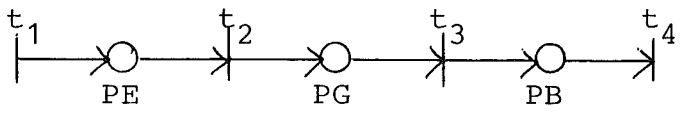
t_4: Auswurf oder Weiterleitung des Prozeßgutes.

Die Zustände des TP sind:

PE: Prozeßgut herangeführt,

PG: Vorhadensein des Prozeßgutes gemeldet und Prozeßgut zur Bearbeitung weitergeführt.

PB: Prozeßgut bearbeitet und weitergeführt.



Steuerungssystem (SS):
Die Tätigkeiten des SS sind:

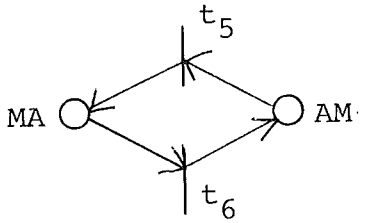
t_5: Meldung (von TP) annehmen

t_6: Anweisung (an TP) geben.

Die Zustände des SS sind:

AM: Anweisung gegeben und nächste Meldung erwarten.

MA: Meldung empfangen und Anweisung erforderlich.

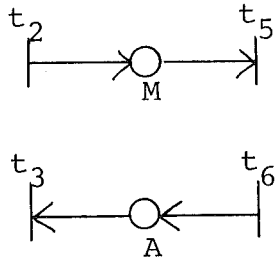


Schnittstelle:

Die Zustände der Schnittstelle sind

M: Es liegt eine Meldung vor.

A: Es liegt eine Anweisung vor.



b) <u>Verklemmungen in den Komponenten:</u>

TP: in TP gibt es keine Verklemmungen, da

$$\neg(PE \wedge (\neg PE \vee PG) \wedge (\neg PG \vee PB) \wedge \neg PB)$$

$$\langle\text{---}\rangle \quad (\neg PE \vee (PE \wedge \neg PG) \vee (PG \wedge \neg PB) \vee PB)$$

$$\begin{aligned} \langle\text{---}\rangle \quad & (\neg PE \vee PE \vee PG \vee PB) \wedge \\ & (\neg PE \vee PE \vee \neg PB \vee PB) \wedge \\ & (\neg PE \vee \neg PG \vee PG \vee PB) \wedge \\ & (\neg PE \vee \neg PG \vee \neg PB \vee PB) \end{aligned}$$

eine allgemeingültige Formel ist.

SS: in SS liegen zwei Verklemmungsfälle vor:

$$MA \wedge AM \quad \text{und} \quad \neg MA \wedge \neg AM$$

Schnittstelle:
In der Schnittstelle gibt es keine Verklemmungen, da

$$\neg(M \wedge \neg M) \quad <--> \quad (\neg M \vee M)$$
$$\neg(\neg A \wedge A) \quad <--> \quad (A \vee \neg A)$$

allgemeingültige Formeln sind.

c) Nichterreichbare Verklemmungen in SS:

Die nicht erreichbaren Markierungen in SS sind

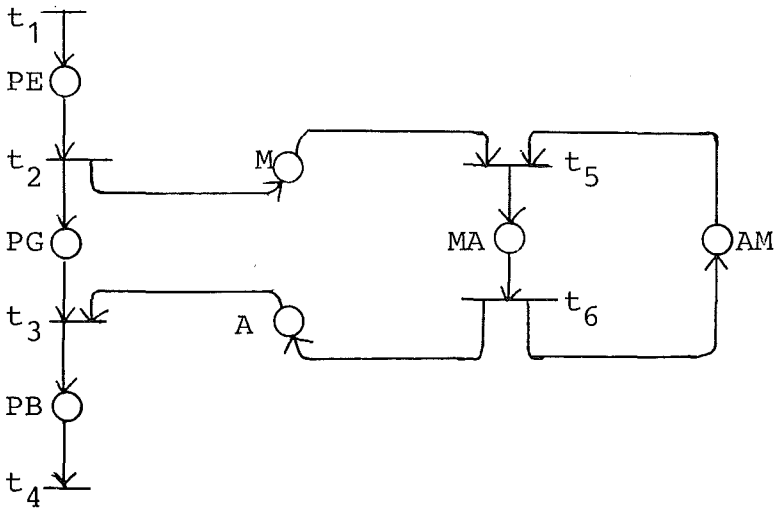
$$\neg AM \wedge \neg MA \text{ und } AM \wedge MA$$

Damit sind die nicht erreichbaren Verklemmungen in SS

$$= \{\text{Verklemmungen}\} \cap \{\text{nicht erreichbare Markierungen}\}$$
$$= \{\text{Verklemmungen}\} = \{(MA \wedge AM), (\neg MA \wedge \neg AM)\}$$

d. h. beide Verklemmungen sind erreichbar.

d) Gesamtdarstellung des Prozeßsteuerungssystems:



Aussagenlogische Darstellung:

$$PE \wedge (PE \Rightarrow (PG \vee M)) \wedge ((PG \wedge A) \Rightarrow PB) \wedge \neg PB \wedge ((M \wedge AM) \Rightarrow MA) \wedge (MA \Rightarrow (A \vee AM))$$

oder (im $(\wedge,\vee,\neg)$-Kalkül)

$$PE \wedge (\neg PE \vee PG \vee M) \wedge (\neg PG \vee \neg A \vee PB) \wedge \neg PB \wedge (\neg M \vee \neg AM \vee MA) \wedge (\neg MA \vee A \vee AM)$$

Negation der aussagenlogischen Darstellung:

$$\neg PE \vee (PE \wedge \neg PG \wedge \neg M) \vee (PG \vee A \vee \neg PB) \vee PB \vee (M \wedge AM \wedge \neg MA) \vee (MA \wedge \neg A \wedge \neg AM)$$

e) Verklemmungen in der Gesamtdarstellung:

Der Theorem-Prover liefert die folgenden Minterme ohne Tautologie:

$\neg PE \vee \neg PG \vee A \vee PB \vee M \vee MA$	$\vee (AM \wedge \neg AM)$
$\neg PE \vee \neg PG \vee A \vee PB \vee M \vee \neg AM$	$\vee (MA \wedge \neg MA)$
$\neg PE \vee \neg PG \vee A \vee PB \vee AM \vee MA$	$\vee (M \wedge \neg M)$
$\neg PE \vee \neg PG \vee A \vee PB \vee \neg MA \vee \neg AM$	$\vee (M \wedge \neg M)$
$\neg PE \vee \neg M \vee PG \vee PB \vee AM \vee MA$	$\vee (A \wedge \neg A)$
$\neg PE \vee \neg M \vee PG \vee PB \vee AM \vee \neg A$	$\vee (MA \wedge \neg MA)$
$\neg PE \vee \neg M \vee PG \vee PB \vee \neg MA \vee \neg A$	$\vee (AM \wedge \neg AM)$
$\neg PE \vee \neg M \vee PG \vee PB \vee \neg MA \vee \neg AM$	$\vee (A \wedge \neg A)$
$\neg PE \vee \neg M \vee A \vee PB \vee AM \vee MA$	$\vee (PG \wedge \neg PG)$
$\neg PE \vee \neg M \vee A \vee PB \vee \neg MA \vee \neg AM$	$\vee (PG \wedge \neg PG)$

Das ergibt die folgenden Verklemmungen:

PE ∧ PG ∧ ¬PB	∧ ¬M ∧ ¬A	∧ AM ∧ ¬MA		
PE ∧ PG ∧ ¬PB	∧ ¬M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ PG ∧ ¬PB	∧ ¬M ∧ ¬A	∧ ¬AM ∧ MA		
PE ∧ PG ∧ ¬PB	∧ ¬M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ PG ∧ ¬PB	∧ M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ PG ∧ ¬PB	∧ ¬M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ PG ∧ ¬PB	∧ M ∧ ¬A	∧ AM ∧ MA		*
PE ∧ PG ∧ ¬PB	∧ ¬M ∧ ¬A	∧ AM ∧ MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ A	∧ ¬AM ∧ ¬MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ A	∧ ¬AM ∧ MA		
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ A	∧ ¬AM ∧ ¬MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ A	∧ AM ∧ MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ A	∧ ¬AM ∧ MA		
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ A	∧ AM ∧ MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ ¬A	∧ AM ∧ MA		*
PE ∧ PG ∧ ¬PB	∧ M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ ¬A	∧ ¬AM ∧ ¬MA		*
PE ∧ PG ∧ ¬PB	∧ M ∧ ¬A	∧ AM ∧ MA		*
PE ∧ ¬PG ∧ ¬PB	∧ M ∧ ¬A	∧ AM ∧ MA		*

Mit * sind die Verklemmungen bezeichnet, in denen auch SS verklemmt ist.

Schließt man die totalen Verklemmungen des SS aus, so ergeben sich die Verklemmungen

$$
\begin{array}{llll}
PE \wedge & PG \wedge \neg PB & \wedge \neg M \wedge \neg A & \wedge AM \wedge \neg MA \\
PE \wedge & PG \wedge \neg PB & \wedge \neg M \wedge \neg A & \wedge AM \wedge \neg MA \\
PE \wedge \neg & PG \wedge \neg PB & \wedge M \wedge A & \wedge \neg AM \wedge MA \\
PE \wedge \neg & PG \wedge \neg PB & \wedge M \wedge A & \wedge \neg AM \wedge MA
\end{array}
$$

Faßt man die identischen Verklemmungsfälle zusammen, so ergibt sich:

$$
\begin{array}{llll}
PE \wedge & PG \wedge \neg PB & \wedge \neg M \wedge \neg A & \wedge AM \wedge \neg MA \\
PE \wedge \neg & PG \wedge \neg PB & \wedge M \wedge A & \wedge \neg AM \wedge MA
\end{array}
$$

Bemerkung:
Zu diesem Ergebnis kann man mit Hilfe eines Theorem-Provers automatisch kommen, indem man

$$
\begin{array}{l}
PE \wedge (PE \Rightarrow (PG \vee M)) \wedge ((PG \wedge A) \Rightarrow PB) \wedge \neg PB \\
\quad \wedge ((M \wedge AM) \Rightarrow MA) \wedge (MA \Rightarrow (A \vee AM)) \\
\wedge \neg (AM \wedge MA) \wedge \neg (\neg AM \wedge \neg MA)
\end{array}
$$

eingibt, also die Netzbeschreibung und die negierten nicht erreichbaren Verklemmungen der Komponenten (hier der Komponente SS).

f) Interpretation der Verklemmungen:

Verklemmungen der Komponente SS:

$\neg AM \wedge \neg MA$	das Steuersystem wurde nicht eingeschaltet oder es stirbt unterwegs.
$AM \wedge MA$	??? (zuviel System?)

Verklemmungen des Gesamtsystems (außer SS)

$PE \wedge PG \wedge \neg PB \wedge \neg M \wedge \neg A \wedge AM \wedge \neg MA$

Es wurde nichts gemeldet
(Meldeorgan funktioniert nicht)

$PE \wedge \neg PG \wedge \neg PB \wedge M \wedge A \wedge \neg AM \wedge MA$

Es wurde in t_2 oder in t_3 nicht weitergeführt.

g) Rückwärtsschalten/nicht erreichbare Verklemmungen

Systemsteuerung (SS):
Da es sich bei dieser Komponente um einen Kreis handelt sind alle Verklemmungen nicht erreichbar.

In den Komponenten TP und Schnittstelle gibt es keine Verklemmungen.

Gesamtsystem:
Da in allen Verklemmungen des Gesamtsystems PE markiert und PB unmarkiert sind, sind beim Rückwärtsschalten t_1 und t_4 schaltfähig. Läßt man jedoch die Transitionen t_1 und t_4 weg, so sind die Verklemmungen des Gesamtsystems nicht erreichbar.

h) Fehlersituationen:

Man kann Fehlersituationen systematisch erkennen, indem man die einzelnen Transitionen falsch schalten läßt und das Ergebnis interpretiert

t_2:
- Weder Meldung noch Weiterleitung,
- Meldung aber keine Weiterleitung,
- keine Meldung aber Weiterleitung,
- Meldung und/oder Weiterleitung, obwohl nichts da ist
- Überschreiben der letzten Meldung,
- Weiterleitung, obwohl letztes Prozeßgut noch nicht bearbeitet und weitergleitet.

t_3: - Bearbeitung und Weiterleitung ohne Prozeßgut und ohne Anweisung,
- Bearbeitung und Weiterleitung ohne Anweisung,
- Bearbeitung und Weiterleitung ohne Prozeßgut,
- Weiterleiten auf falsches Prozeßgut.

(- Falsche Bearbeitung)

t_5: - Meldung annehmen, obwohl keine da,
- Meldung verarbeiten, aber nicht löschen,
- Meldung nicht verarbeiten,
- Meldung löschen, aber nicht verarbeiten,
- Auflaufen auf letzte Meldungsverarbeitung.

t_6: - Keine Anweisung geben, trotz Meldung,
- Ohne Meldungsempfang Anweisung geben,
- Ohne Verarbeitung Anweisung geben (noise),
- Anweisung korrekt geben, aber keine neue Anweisung mehr erwarten,

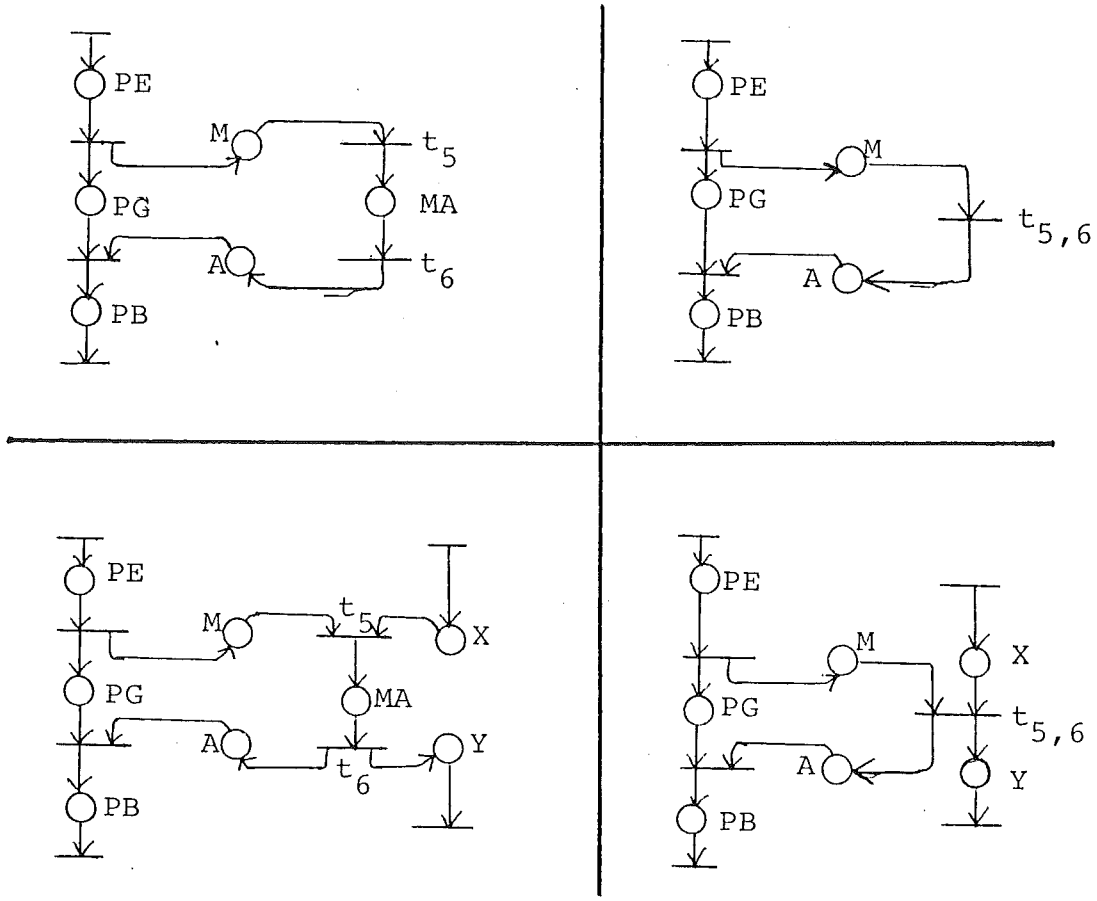
(- Falsche Anweisung)

t_1: - Es wird nie ein Prozeßgut eingegeben,
- Ein nicht existentes Prozeßgut wird weitergeleitet,
- Auflauf von Prozeßgütern.

t_4: - Es wird kein Prozeßgut ausgegeben.

(- Ein nicht existentes Prozeßgut wird weitergeleitet)

i) Andere Darstellungsmöglichkeiten derselben Steuerung:

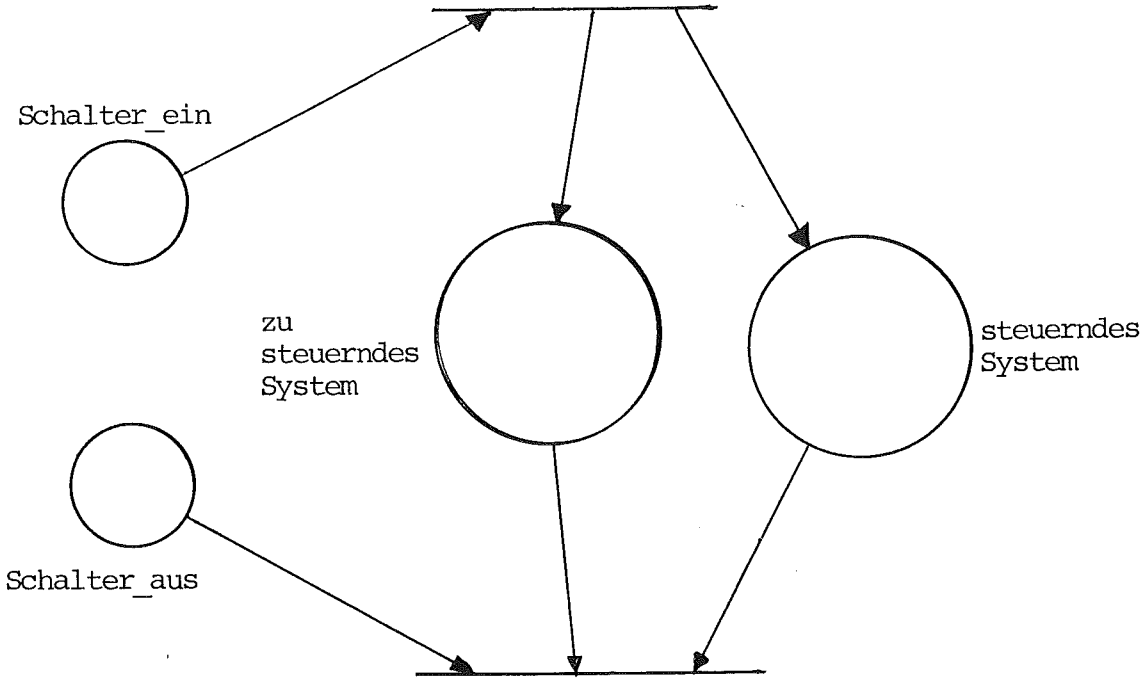


Teil II: Allgemeines Modell

4. ALLGEMEINES ZUR PETRI-NETZ-DARSTELLUNG VON PROZESSSTEUERUNGSANLAGEN

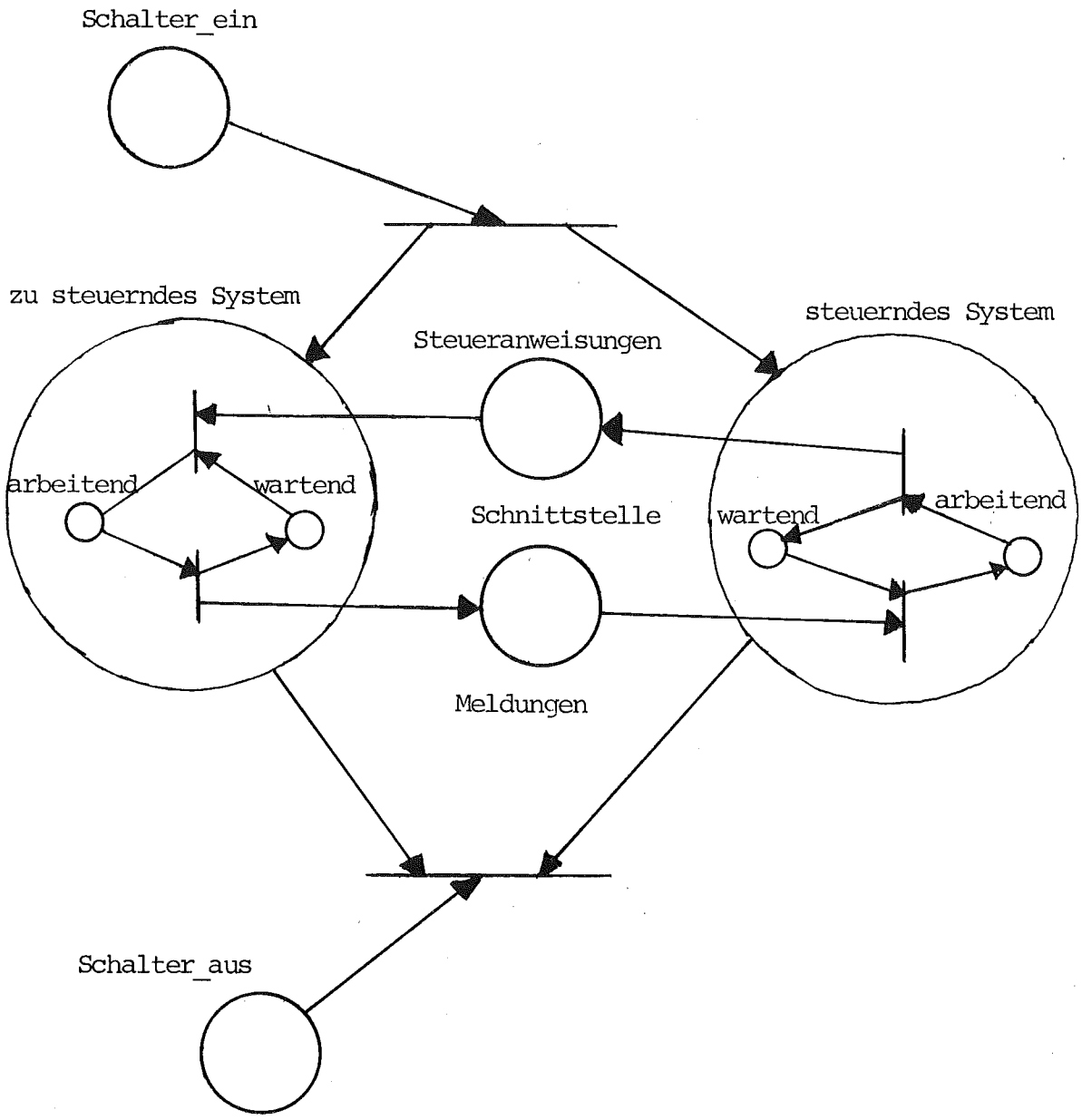
Eine Prozeßsteuerungsanlage besteht aus zwei Teilen, einem zu steuernden System (i.a. ein technischer Prozeß) und einem steuernden System (i.a. eine Rechenanlage). Allgemein kann angenommen werden, daß es einen Schalter gibt, mit dem man die gesamte Anlage ein- und ausschalten kann. Damit ergibt sich folgende grobe Petri-Netz-Darstellung:

BILD 1:



Zur Steuerung kommunizieren die beiden Systeme, indem das steuernde System Steueranweisungen an das zu steuernde System gibt und das zu steuernde System Meldungen an das steuernde System sendet. Bei beiden Systemen handelt es sich also um Automaten mit Ein- und Ausgabe und mit zwei Klassen von Zuständen, "wartend (auf Eingabe)" und "arbeitend".

BILD 2:



Nach diesem einfachen Modell ist eine Prozeßsteuerungsanlage ein wechselseitiges "Producer-Consumer" System, die Steuerungsanweisungen und Meldungen bilden die Schnittstelle zwischen dem steuernden und dem zu steuernden System.

Durch Schalter_ein kann eine der folgenden Initialisierungen der Systeme erfolgen (Anfangszustände):

	im zu steuernden System		im steuernden System		
Nr.	arbeitend	wartend	wartend	arbeitend	Bemerkung
1	1	0	1	0	-
2	1	0	0	1	-
3	0	1	1	0	Verklemmung!
4	0	1	0	1	-

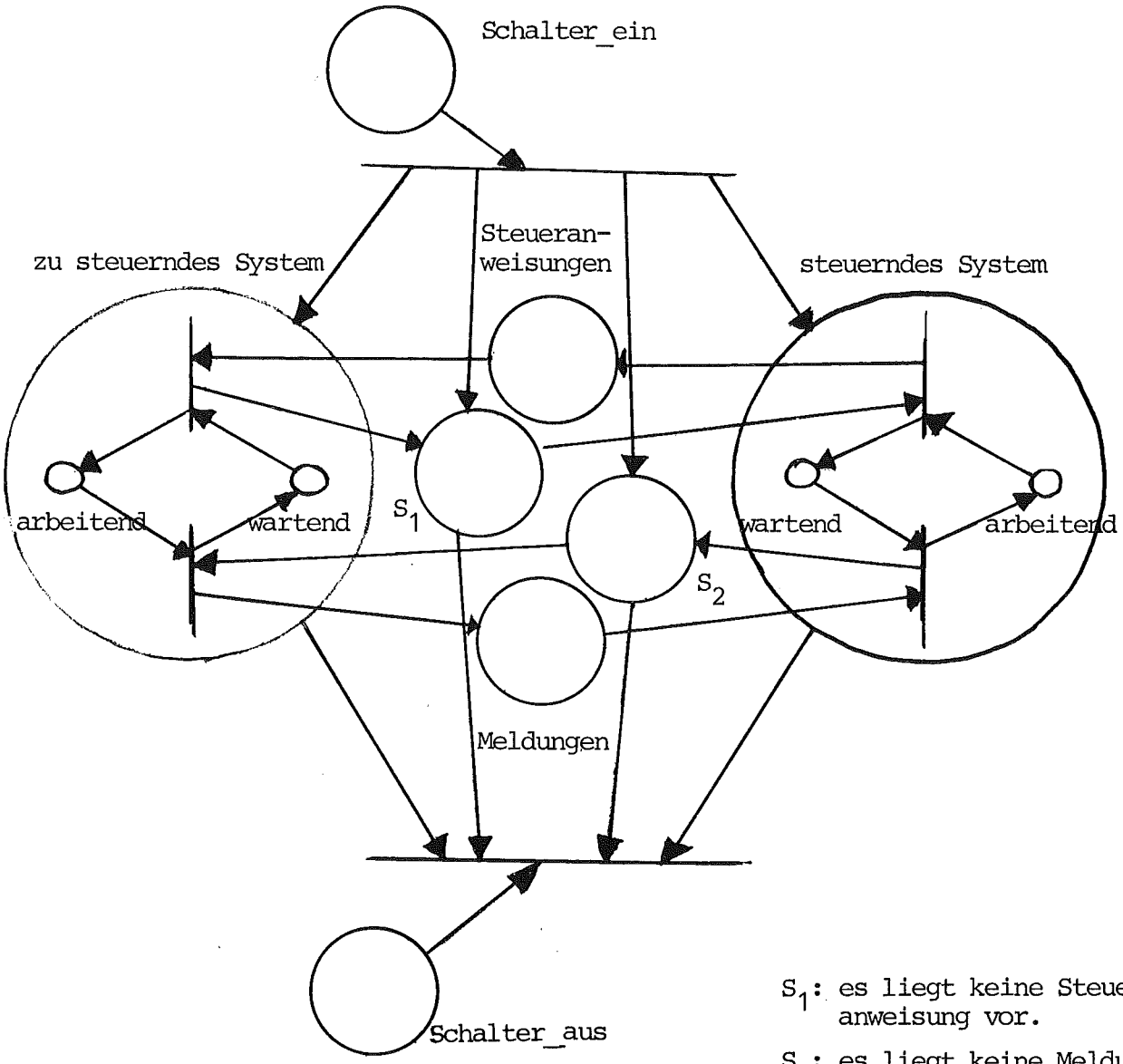
Schalter_aus kann das Gesamtsystem in folgenden Zuständen unterbrechen (Endzustände):

	zu steuerndes Systen		Schnittstelle		steuerndes System	
Nr.	arbeitend	wartend	Steueranweisung	Meldg.	wartend	arbeitend
1	1	0	0	0	1	0
2	1	0	0	0	0	1
3	0	1	0	0	1	0
4	0	1	0	0	0	1
5	0	1	0	1	1	0
6	0	1	1	0	1	0
7	0	1	0	1	0	1
8	1	0	1	0	1	0
9	0	1	1	1	1	0
10	0	1	1	0	0	1
11	1	0	0	1	1	0
12	0	1	0	2	1	0
13	0	1	2	0	1	0

Genügen zur Darstellung binäre Petri-Netze, so entsprechen die Endzustände 12 und 13 den Endzuständen 5 und 6.

In der Regel stoppt man eine Prozeßsteuerungsnalage (wenn überhaupt) im Anfangszustand, sodaß ein "Neustart" im selben Zustand wieder möglich ist. Dies ist insbesondere wegen zusätzlicher (Zwischen-)Zustände nötig, die wie "own-variables" fungieren, in diesen hier betrachteten vereinfachten Modellen jedoch nicht in Erscheinung treten. Um das Anhalten einer Prozeßsteuerungsanlage in Petri-Netz-Form darstellen zu können, muß man das Modell wie folgt erweitern:

BILD 3:



S_1: es liegt keine Steuerungsanweisung vor.

S_2: es liegt keine Meldung vor.

Für die beiden wohl häufigsten Anfangszustände 1 und 4 ergeben sich aus dem letzten Bild folgende Auflösungen:

BILD 4:

Anfangszustand 1

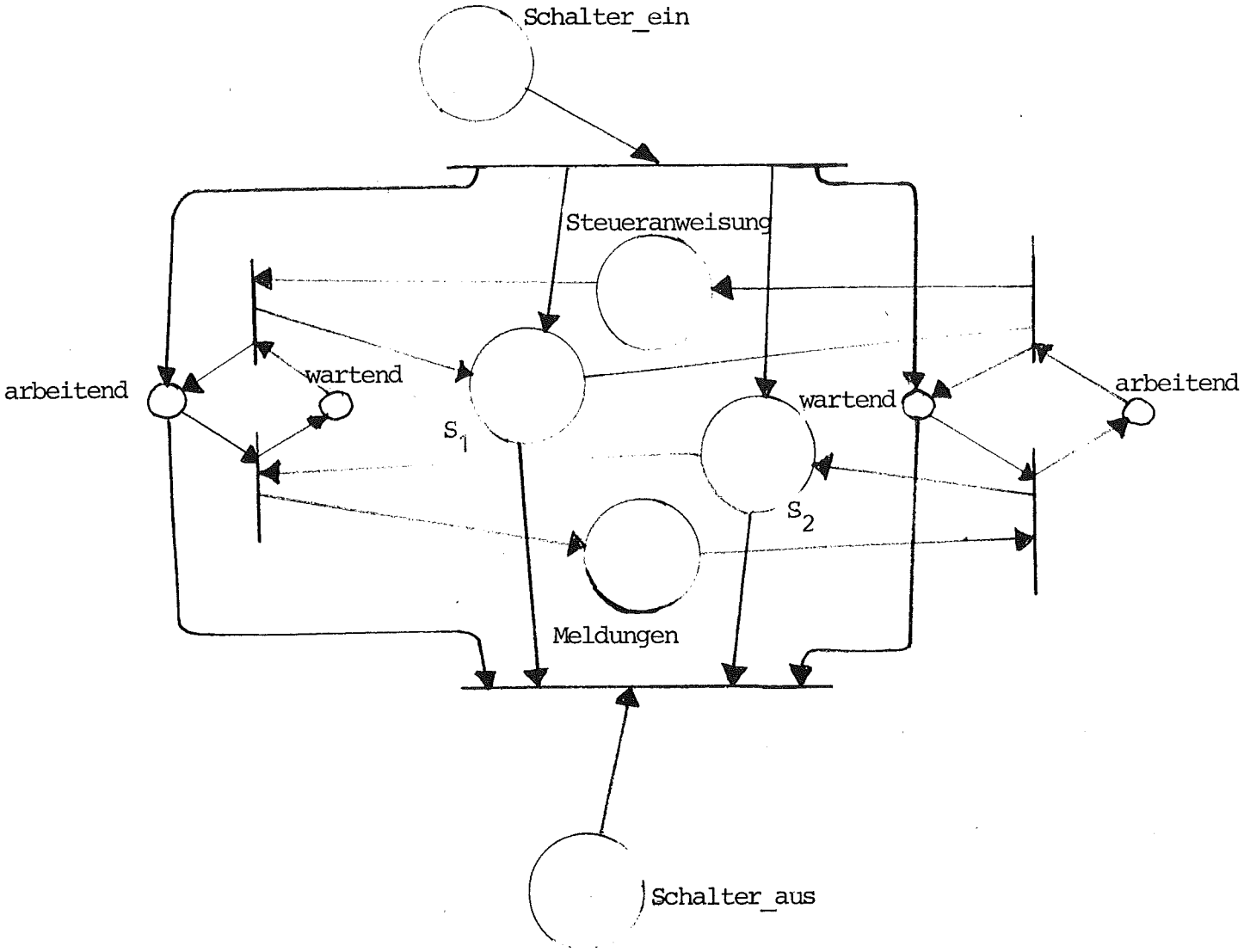
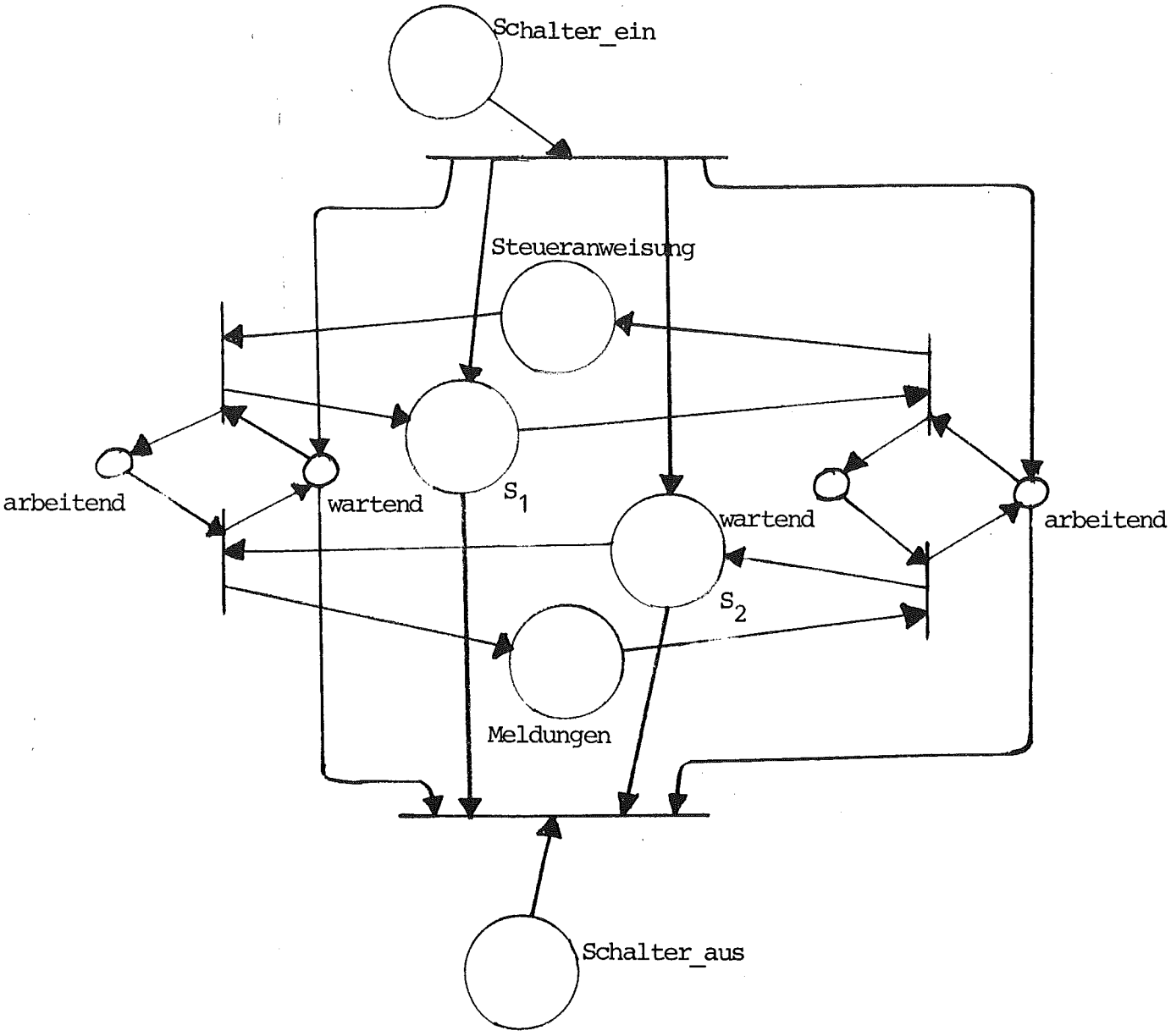


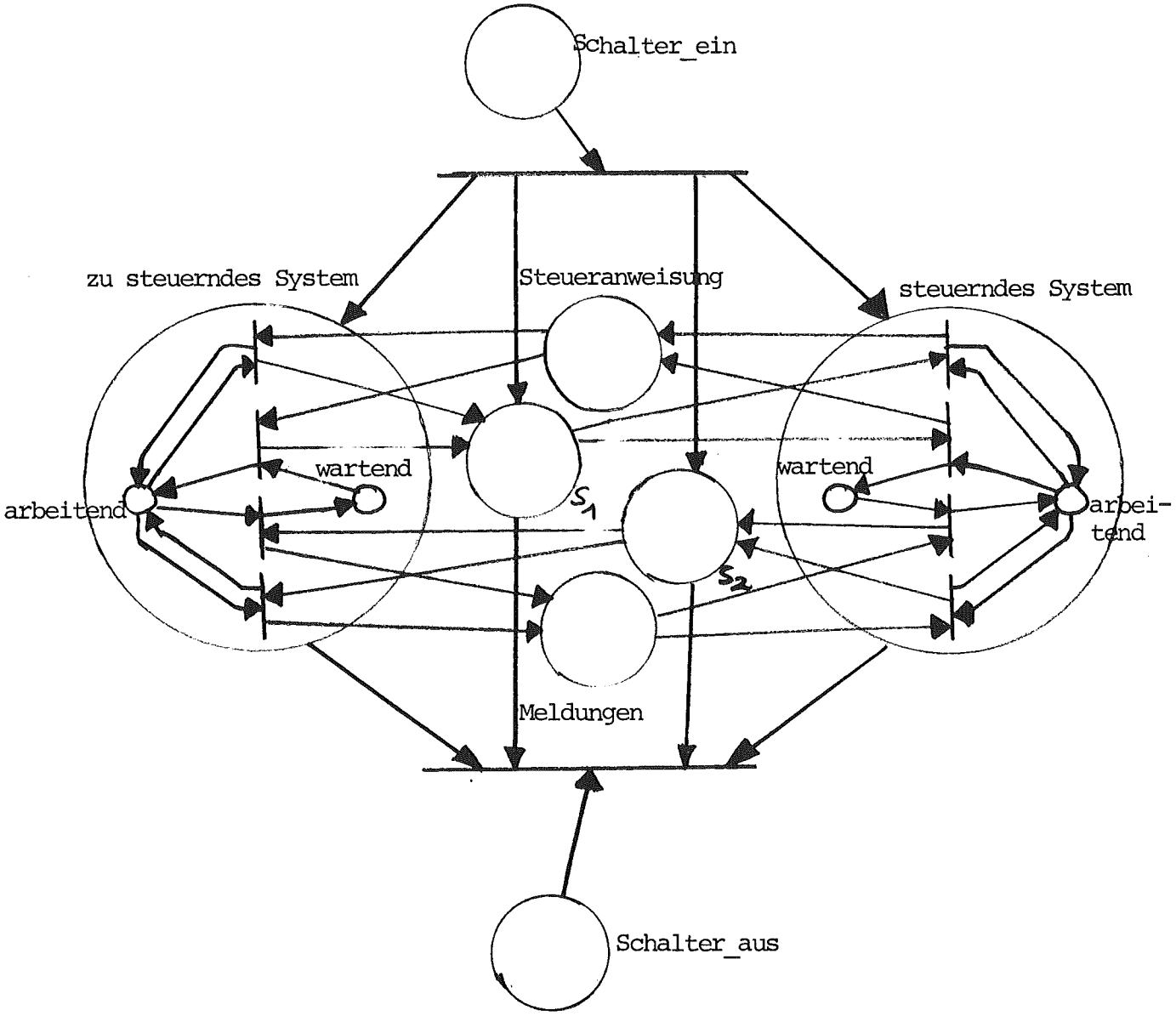
BILD 5:

Anfangszustand 4:



Das zu steuernde System sowie das steuernde System können aus mehreren parallel oder quasiparallel ablaufenden Teilen bestehen. Dies führt zu folgendem Modell (Bild 6):

BILD 6:



Bei den bisher betrachteten Modellen handelt es sich um abgeschlossene, selbständige Systeme. In der Praxis werden Prozeßsteuerungsanlagen durch den Menschen beeinflußt. Dabei kann der Mensch als Teil

- des zu steuernden Systems
- des steuernden Systems

und/oder

- der Schnittstelle

auftreten.

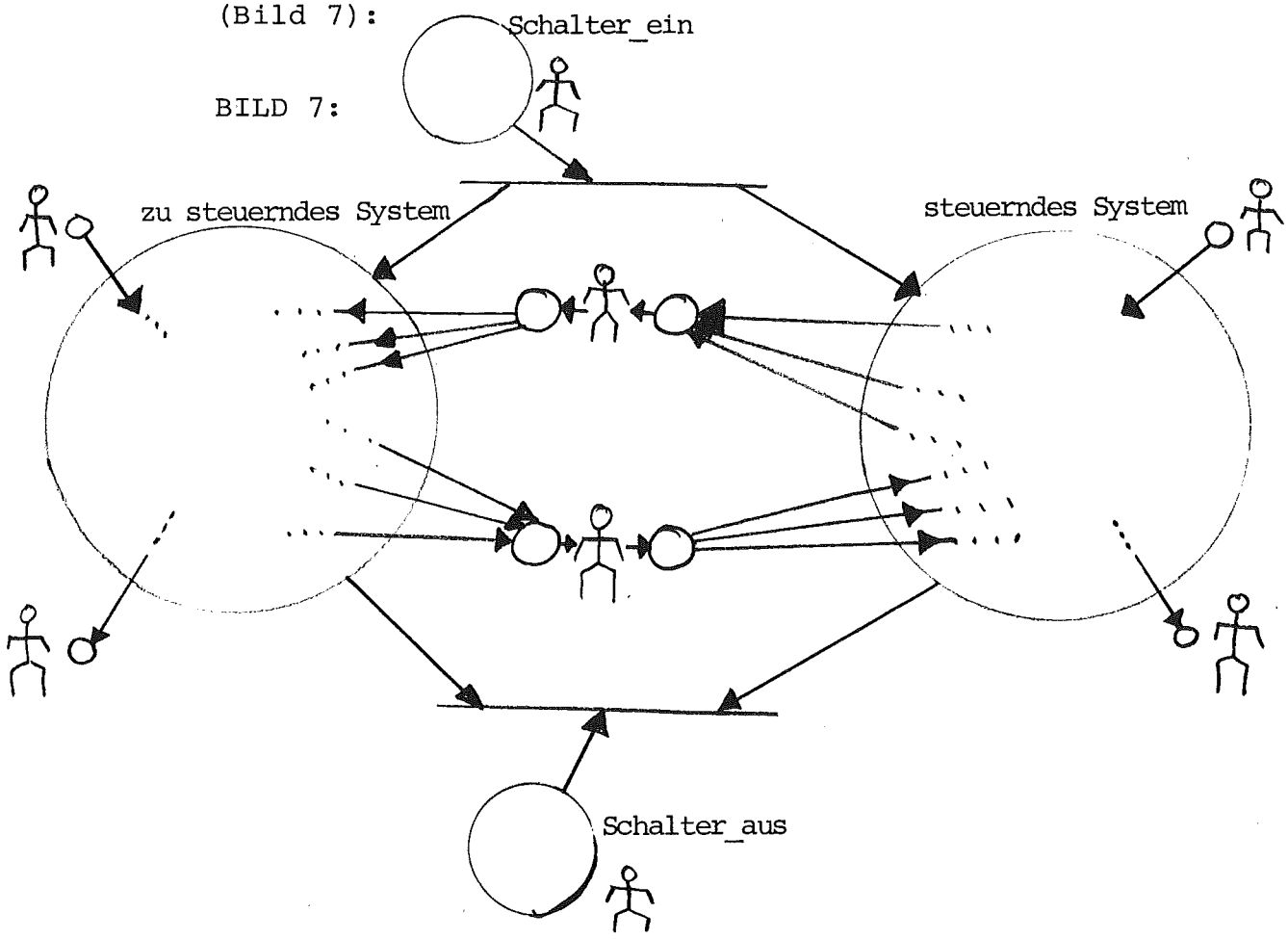
Beispiele für die Mitwirkung des Menschen im zu steuernden System sind die Eingabe und die Entnahme von Waren und/oder Materialien.

Bei der Bedienung von Maschinen tritt der Mensch als Teil des steuernden Systems auf.

In off-line Systemen fungiert der Mensch als Teil der Schnittstelle.

Das off-line System läßt sich mit Hilfe des bisher verwendeten vereinfachten Modells wie folgt darstellen (Bild 7):

BILD 7:



In diesem Bild wird zusätzlich angedeutet, daß der Mensch als Bediener des Schalters ebenfalls auf die Prozeßsteuerung Einfluß nimmt.

Teil III: Die Paketverteilanlage

5. METHODEN ZUR AUFSCHLÜSSELUNG DER AUFGABENSTELLUNG

Ziel der hier vorgestellten Methode ist es, möglichst systematisch ein formalisierbares Modell des Gesamtsystems und seiner Komponenten zu gewinnen. Dabei wird (analog zu SREM/SREP) die vorgegebene Aufgabenstellung satzweise analysiert. Die einzelnen Sätze werden zu Aussagen zerlegt und diese dann eingeteilt in

- Aussagen über den zu steuernden Prozeß
- Aussagen über den steuernden Prozeß und
- Aussagen über die Schnittstelle.

Diese Aussagen lassen sich (oft eins zu eins) als Beschreibung von Zustandsübergängen interpretieren und als Bedingung - Ereignis - Netz (eine Sonderform der Petri-Netze) darstellen. Dadurch wird eine formale Beschreibung gewonnen, die sich, wie in Teil I angedeutet, untersuchen läßt.

Der Übergang von der natürlichsprachlichen Aufgabenbeschreibung zur Zustands-Ereignis-Beschreibung ist ausschließlich manuell und erfordert sicherlich einige Intuition und Erfahrung. Alle weiteren Schritte können durch Tools unterstützt werden, wie

- Eingabe der Zustände und Zustandsübergänge
- Ableiten des Simulationsmodells (und Ablaufsimulation)
- Übergang von Bedingungs-Ereignis-Netzen zu aussagenlogischen Formeln
- Herausarbeiten der Fehlersituationen und Fehlermöglichkeiten
- Erkennung und Untersuchung der Verklemmungssituationen und -möglichkeiten
- graphische Darstellung
- Vereinfachung/Umordnung der Netze.

Für die Erstellung solcher Werkzeuge und auch auf der theoretischen Seite ist noch viel Entwicklungsarbeit nötig.

Als Leitfaden bei der Aufschlüsselung der Aufgabenstellung diente folgendes Vorgehensschema:

- Text der Aufgabenstellung (satzweise)
- Interpretation
- Bedingungen/Zustände und Ereignisse
- graphische Darstellung
- aussagenlogische Formel
- Fehlermöglichkeiten
- Verklemmungssituationen.

Um den Rahmen nicht zu sprengen, und wegen des Fehlens der oben erwähnten Tools, wurde diese Vorgehensweise in den folgenden Abschnitten nicht immer eingehalten oder nicht vollständig ausgeführt.

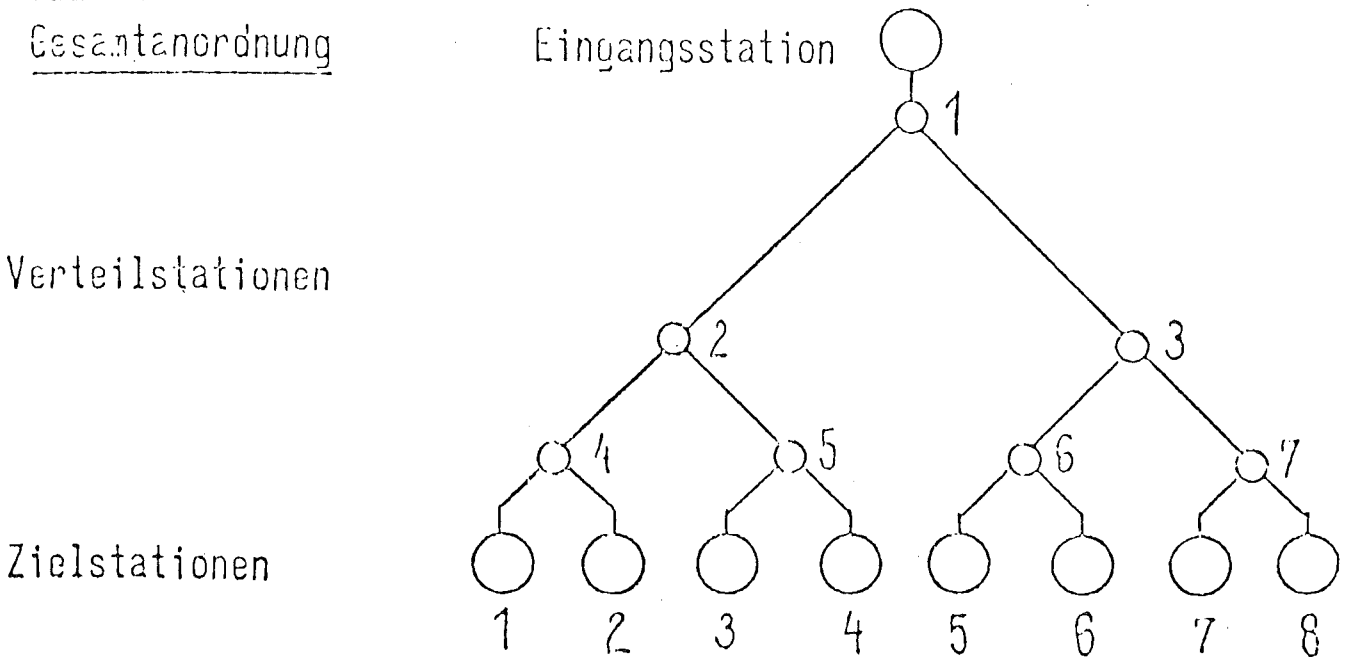
Somit können die folgenden Ausführungen nur einen Eindruck vermitteln und sollen nicht als Muster verstanden werden.

6. DAS GESAMTSYSTEM

Die Aufgabenstellung lautet:

Bild 1

Gesamtanordnung



1.

Bild 1 zeigt die Gesamtanordnung. Die in die Eingangsstation einlaufenden Pakete sind durch ein Codezeichen markiert, das die Zielstation angibt. Das Steuersystem liest das Codezeichen und steuert danach die einzelnen Verteilstationen, welche das Durchlaufen des Paketes melden.

Satzweise Aufschlüsselung:

1. Satz:

Bild 1 zeigt die Gesamtanordnung.

Interpretation

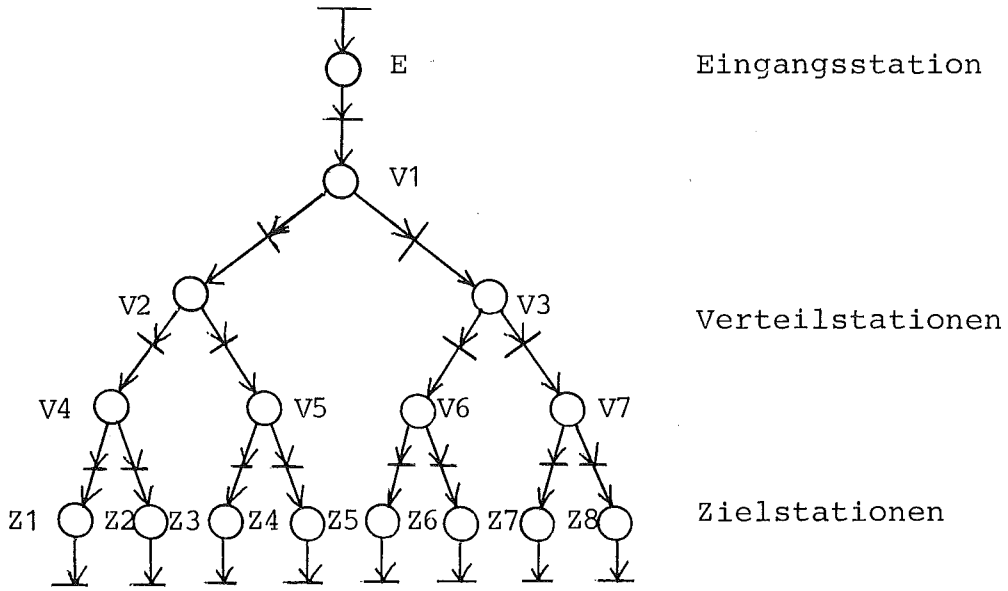
Bei dem Bild 1 handelt es sich um eine schematische Darstellung des zu steuernden (d.h. technischen) Prozesses.

Bemerkung

Da es sich bei diesem Satz um einen Hinweis auf eine grapische Darstellung handelt, die sich leicht in Petri-Netz-Form überführen läßt, wird mit der Netz-Darstellung begonnen.

Petri-Netz-Darstellung

BILD 8:



Formallogische Formulierung

Obwohl nach Teil I, Beispiel 2.2 c) total lebendig ist, lohnt sich die formallogische Formulierung in diesem Fall, da dadurch eine "Minimalkonfiguration" wie folgt gewonnen werden kann:

aussagelogisch:

$$E \wedge (E \Rightarrow V1) \wedge$$
$$(V1 \Rightarrow (V2 \wedge V3)) \wedge$$
$$(V2 \Rightarrow (V4 \wedge V5)) \wedge (V3 \Rightarrow (V6 \wedge V7)) \wedge$$
$$(V4 \Rightarrow (Z1 \wedge Z2)) \wedge (V5 \Rightarrow (Z3 \wedge Z4)) \wedge$$
$$(V6 \Rightarrow (Z5 \wedge Z6)) \wedge (V7 \Rightarrow (Z7 \wedge Z8)) \wedge$$
$$\neg Z1 \wedge \neg Z2 \wedge \neg Z3 \wedge \neg Z4 \wedge \neg Z5 \wedge \neg Z6 \wedge \neg Z7 \wedge \neg Z8$$

prädikatenlogisch:

mit n = 8:

$$E \wedge (E \Rightarrow V[1]) \wedge$$
$$(\forall i \in 1..(\tfrac{n}{2}-1): (V[i] \Rightarrow (V[2i] \wedge V[2i+1]))) \wedge$$
$$(\forall j \in \tfrac{n}{2}..(n-1): (V[j] \Rightarrow (Z[2(j-(\tfrac{n}{2}-1))-1] \wedge Z[2(j-(\tfrac{n}{2}-1))]))) \wedge$$
$$(\forall k \in 1..n: \neg Z[k])$$

Minimalkonfiguration mit n = 4 d.h.

$$E \wedge (E \Rightarrow V1) \wedge$$
$$(V1 \Rightarrow (V2 \wedge V3)) \wedge$$
$$(V2 \Rightarrow (Z1 \wedge Z2)) \wedge (V3 \Rightarrow (Z3 \wedge Z4)) \wedge$$
$$\neg Z1 \wedge \neg Z2 \wedge \neg Z3 \wedge \neg Z4$$

Bei allen weitern Betrachtungen des Gesamtsystems kann man sich also auf eine Konfiguration aus 4 Ziel- und 3 Verteilstationen beschränken.

<u>Fehlermöglichkeiten</u>

Um Fehlersituationen und implizite Behauptungen zu entdecken, betrachtet man die Bedingungs-Ereignis-Darstellung:

$$E \wedge (E \Rightarrow V1) \wedge$$
$$(V1 \Rightarrow (V2 \vee V3)) \wedge$$
$$(V2 \Rightarrow (Z1 \vee Z2)) \wedge (V3 \Rightarrow (Z3 \vee Z4)) \wedge$$
$$\neg Z1 \wedge \neg Z2 \wedge \neg Z3 \wedge \neg Z4$$

Implizite Behauptungen:

$E \Rightarrow V1$
$V1 \Rightarrow (V2 \vee V3)$
$V2 \Rightarrow (Z1 \vee Z2)$
$V3 \Rightarrow (Z3 \vee Z4)$
und allgemein
$E \Rightarrow (Z1 \vee Z2 \vee Z3 \vee Z4)$

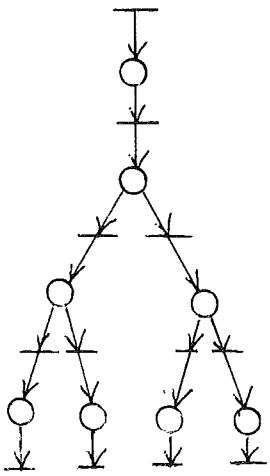
Rückfrage an den Aufgabensteller:

Ist es möglich, daß
$E \not\Rightarrow V1$
$V1 \not\Rightarrow (V2 \vee V3)$
$V2 \not\Rightarrow (Z1 \vee Z2)$ und/oder
$V3 \not\Rightarrow (Z3 \vee Z4)$ und/oder allgemein
$E \not\Rightarrow (Z1 \vee Z2 \vee Z3 \vee Z4)$
d.h. daß ein Paket "verschwindet"/"gestohlen wird" und welche Möglichkeiten gibt es dafür?

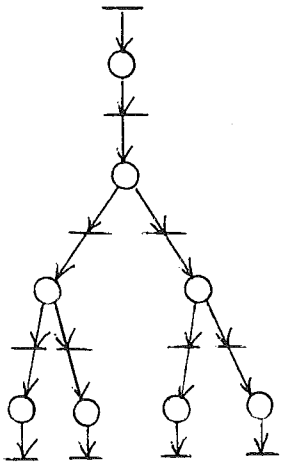
Eingliederung in das Gesamtmodell

Aufgrund der Interpretation ist das bisher aus der Aufgabenstellung gewonnene Netz als zu steuernder Prozeß in die allgemeine Darstellung von Prozeßsteuerungsanlagen (Teil II) einzuordnen. Ein aus Komplexitätsgründen gescheiterter Versuch, allgemeine Überlegungen, die hier zu weit führen würden, und die Betrachtung anderer Software-Engineering-Methoden (PSL/PSA, MASCOT, EPOS, etc) haben zu dem Ergebnis geführt, daß man am besten das steuernde System ebenso gliedert wie den technischen Prozeß. Das heißt, daß die Darstellung (der Gesamtanordnung) des zu steuernden Systems der Darstellung des steuernden Systems gleicht. Das ergibt für das bisher Erarbeitete folgendes Bild:

BILD 9:



zu steuerndes System



steuerndes System

2. Satz

Die in die Eingangsstation einlaufenden Pakete sind durch ein Codezeichen markiert, das die Zielstation angibt.

Interpretation

—

Bedingungen/Zustände und Ereignisse

Paket vor Eingangsstation

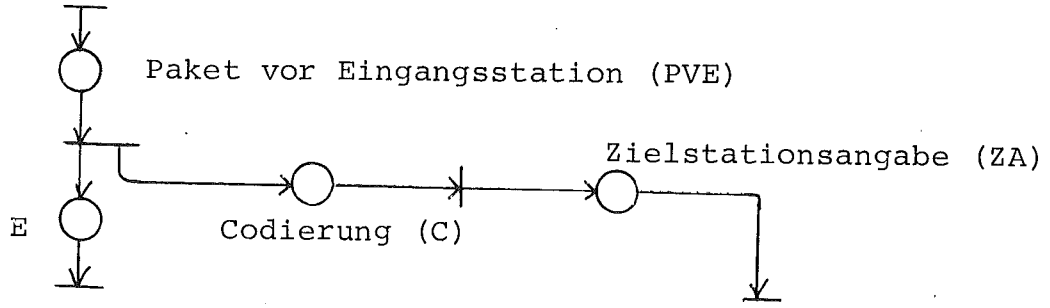
$\longrightarrow$ Paket in Eingangsstation $\wedge$ Codezeichen

Codezeichen

$\longrightarrow$ Zielstationsangabe

Netzdarstellung

BILD 10:



Aussagenlogische Formulierung

PVE $\wedge$ ($\neg$ PVE $\vee$ E $\vee$ C) $\wedge \neg$ E $\wedge$ ($\neg$ C $\vee$ ZA) $\wedge \neg$ ZA
Immer falsche Formel d.h. totale Lebendigkeit.

Fehlermöglichkeiten ($\rightarrow$ Rückfragen an den Aufgabensteller)

Paket ohne Codezeichen
Codezeichen ist keine Zielstationangabe
Codezeichen ohne Paket (?)
Zielstationsangabe ohne Codewort (?)

3. Satz

Das Steuersystem liest das Codezeichen und steuert danach die einzelnen Verteilstationen, welche das Durchlaufen des Paketes melden.

Interpretation

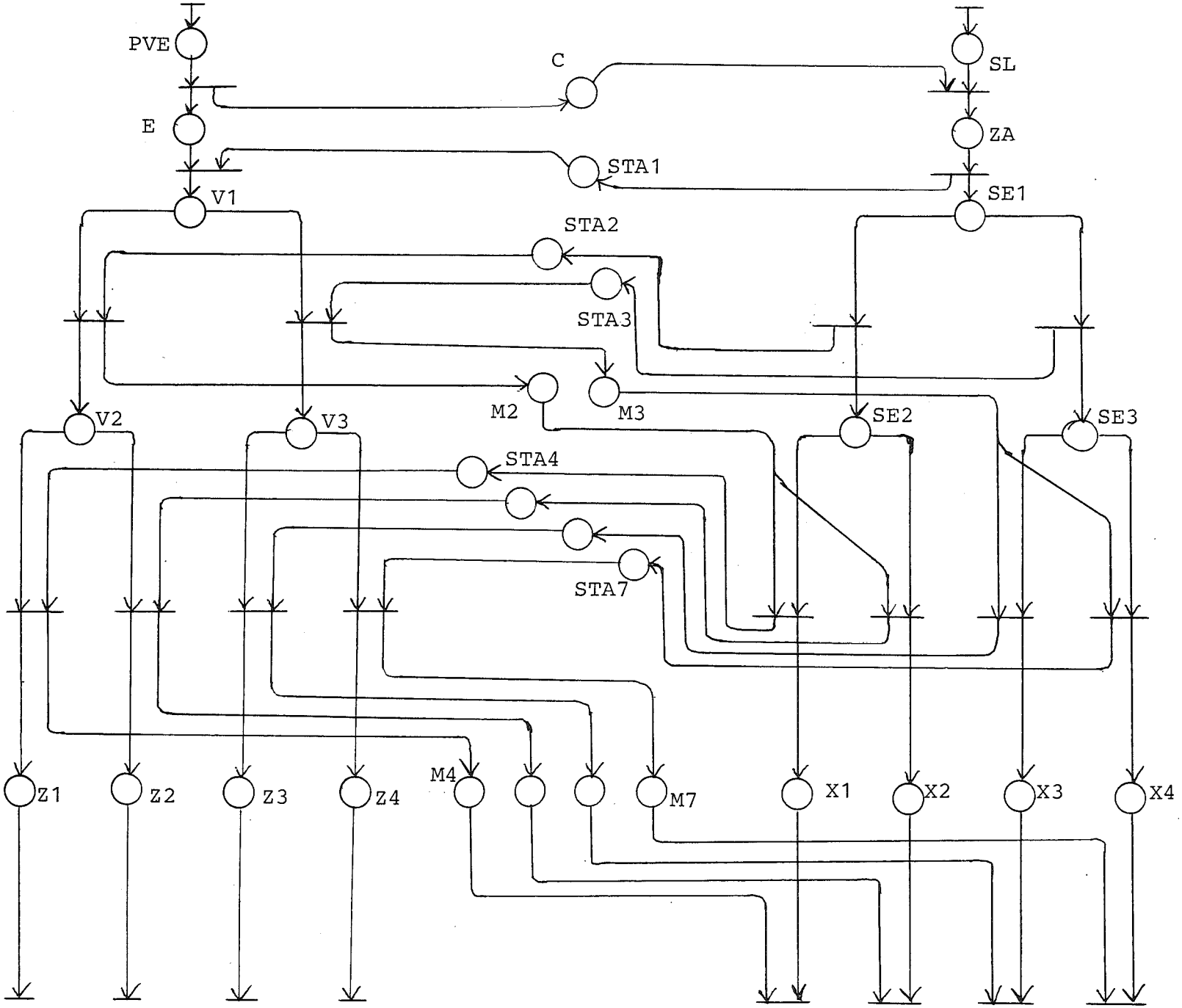
Das Melden des Durchlaufens eines Paketes durch eine Verteilstation entspricht einer Meldung des zu steuernden Systems an das steuernde System. Das Steuern der einzelnen Verteilstationen durch das Steuersystem geschieht über Steueranweisungen an das zu steuernde System. Über einen Zusammenhang zwischen den Meldungen und den Steueranweisungen wird in diesem Satz abgesehen von dem Zusammenhang zwischen Codezeichen und Steueranweisungen ("danach") nicht ausgesagt. Damit ist das Lesen des Codezeichens eine Meldung an das Steuersystem, das Code-

zeichen selbst eine Schnittstelle. Zusammen mit dem 2. Satz folgt daraus, daß die Marken, die im Bild 9 durch das steuernde System laufen, aus den Codezeichen gewonnene Zielstationsangaben sind.

Bedingungen/Zustände und Ereignisse

Paket vor Eingangsstation:	PVE	zu steuerndes System
Paket in Eingangstation:	E	
Paket in Verteilstation i:	Vi	
Paket in Zielstation j:	Zj	
Steuersystem lesebereit:	SL	steuerndes System
Zielstationsangabe ermittelt:	ZA	
Steuerungsentscheidung i:	SEi	
Steuerung nach Ziel j ausgeführt:	Xj	
Steueranweisung k:	STAk	Schnittstellen
Meldung l / Codezeichen:	Ml / C	

$$PVE \wedge (PVE \Rightarrow (E \wedge C)) \wedge SL \wedge ((SL \wedge C) \Rightarrow ZA) \wedge$$
$$(ZA \Rightarrow (SE1 \wedge STA1)) \wedge ((E \wedge STA1) \Rightarrow V1$$
$$(SE1 \Rightarrow ((STA2 \wedge SE2) \vee (STA3 \wedge SE3)) \wedge$$
$$((V1 \wedge STA2) \Rightarrow (V2 \wedge Ms)) \wedge ((V1 \wedge STA3) \Rightarrow (V3 \wedge M3)) \wedge$$
$$((SE2 \wedge M2) \Rightarrow ((STA4 \wedge X1) \vee (STA5 \wedge X2)) \wedge$$
$$((SE3 \wedge M3) \Rightarrow ((STA6 \wedge X3) \vee (STA7 \wedge X4)) \wedge$$
$$((V2 \wedge STA4) \Rightarrow (Z1 \wedge M4)) \wedge ((V2 \wedge STA5) \Rightarrow (Z2 \wedge M5)) \wedge$$
$$((V3 \wedge STA6) \Rightarrow (Z3 \wedge M6)) \wedge ((V3 \wedge STA7) \Rightarrow (Z4 \wedge M7)) \wedge$$
$$\neg(M4 \wedge X1) \wedge \neg(M5 \wedge X2) \wedge \neg(M6 \wedge X3) \wedge \neg(M7 \wedge X4) \wedge$$
$$\neg Z1 \wedge \neg Z2 \wedge \neg Z3 \wedge \neg Z4$$


PVE
C
SL
E
ZA
STA1
V1
SE1
STA2
STA3
M2
M3
V2
V3
SE2
SE3
STA4
STA7
Z1
Z2
Z3
Z4
M4
M7
X1
X2
X3
X4

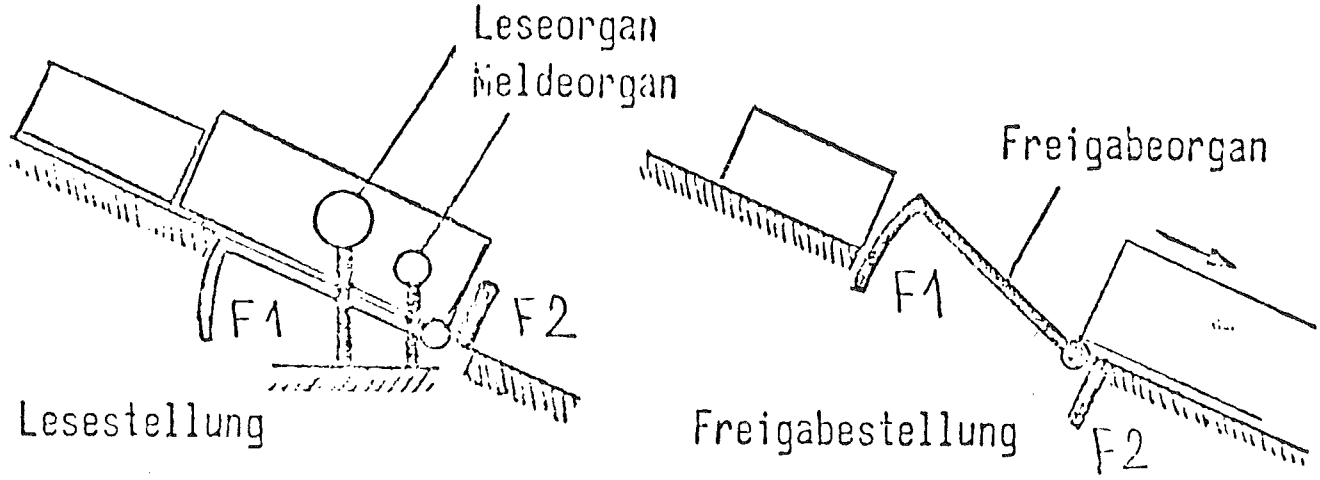
Aus dieser Darstellung der Gesamtanordnung lassen sich globale Aussagen über Fehlersituationen und totale Verklemmungsmöglichkeiten des Gesamtsystems ableiten. Da jedoch die Abläufe der einzelnen Stationen (Eingangs- und Verteilstationen) noch näher beschrieben und dadurch detailliertere Angaben über Fehler- und Verklemmungssituationen gewonnen werden, wird an dieser Stelle auf eine weitere Ausarbeitung verzichtet.

7. DIE EINGANGSSTATION

Die Aufgabenstellung lautet:

Bild 2

Eingangsstation



2.

Bild 2 zeigt die Anordnung der Eingangsstation.

Sie besteht aus einem Freigabeorgan mit den Teilen F1 und F2. F2 hält das einlaufende Paket solange fest, bis das Meldeorgan die Ankunft an das Steuersystem gemeldet hat und dieses mit Hilfe des Leseorgans das Codezeichen aufgenommen hat. Danach gibt das Steuersystem einen Auftrag an das Freigabeorgan, die Sperre F2 gibt den Weiterlauf für das Paket frei und das Beschleunigungsteil F1 neigt sich. Dadurch gleitet das Paket weiter, gleichzeitig wird das nachfolgende Paket solange am Einlaufen gehindert, bis die Sperre F2 wieder eingetreten ist.

Bei der Behandlung des nachfolgenden Pakets ist zu prüfen, ob sich sein Ziel von dem des Vorläufers unterscheidet. In diesem Fall ist die Freigabe seines Weiterlaufs solange zu verzögern, daß in dieser Zeitspanne, die dadurch zwischen dem Passieren der einzelnen Verteilstationen durch die Pakete liegt, die Umstellung der Lenkorgane durchgeführt werden kann. Im anderen Fall können die Pakete unmittelbar aufeinander folgen.

Satzweise Aufschlüsselung

1. Satz:

Bild 2 zeigt die Anordnung der Eingangsstation.

Interpretation:

Für das zu steuernde System wird die Eingabestation detaillierter beschrieben. Sie besteht aus einem Meldeorgan, einem Leseorgan und einem Freigabeorgan. Es wird keine Aussage über den Zusammenhang dieser Organe gemacht, weshalb keine weitere Aufschlüsselung an dieser Stelle vorgenommen werden kann.

2. Satz:

Sie (die Eingangsstation oder die Anordnung der Eingangsstation) besteht aus einem Freigabeorgan mit den Teilen F1 und F2.

Interpretation:

Dieser Satz ist nach der bildlichen Darstellung (Bild 2 der Aufgabenstellung) für die Aufschlüsselung redundant.

3. Satz:

F2 hält das einlaufende Paket solange fest, bis das Meldeorgan die Ankuft an das Steuersystem gemeldet hat und dieses mit Hilfe des Leseorgangs das Codezeichen aufgenommen hat.

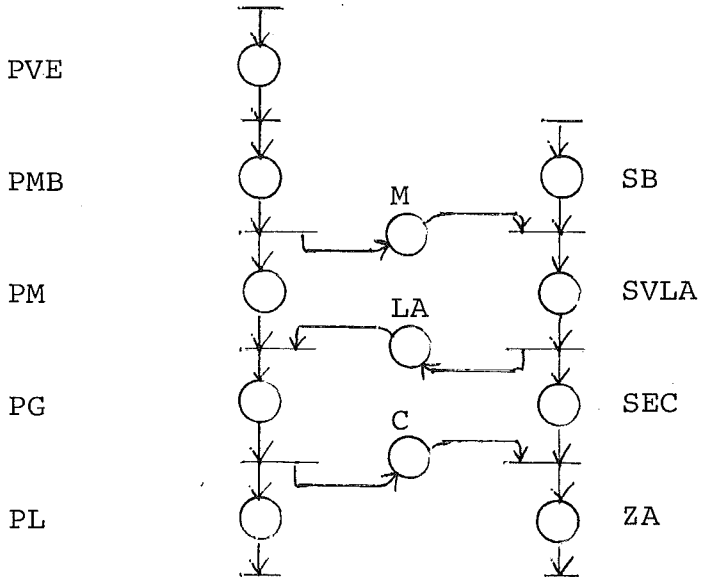
Interpretation:

Der Teil F2 des Freigabeorgans kann die Zustände "sperren" und "freigeben", also zwei Zustände annehmen. Im Zustand "sperren" hält es einlaufende Pakete fest. Das Meldeorgan meldet dem zu steuernden System die Ankunft der einlaufenden Pakete. Dieses (das Steuersystem) gibt aufgrund einer Meldung des Meldeorgans eine Leseanweisung (in Form einer Steueranweisung) an das Leseorgan. Das Leseorgan übermittelt das Codezeichen an das zu steuernde System. Damit ist die Bedeutung des Verbs "lesen" am Anfang des dritten Satzes im ersten Abschnitt der Aufgabenstellung geklärt. Die Beschreibung des Zeitverhaltens durch die Worte "solange" und "bis" ist unvollständig und für die hier verwendete Aufschlüsselungsmethode irreführend, da sie einen Vorgang andeutet, der in den folgenden Sätzen detaillierter und etwas anders dargestellt wird.

Bedingungen/Zustände und Ereignisse

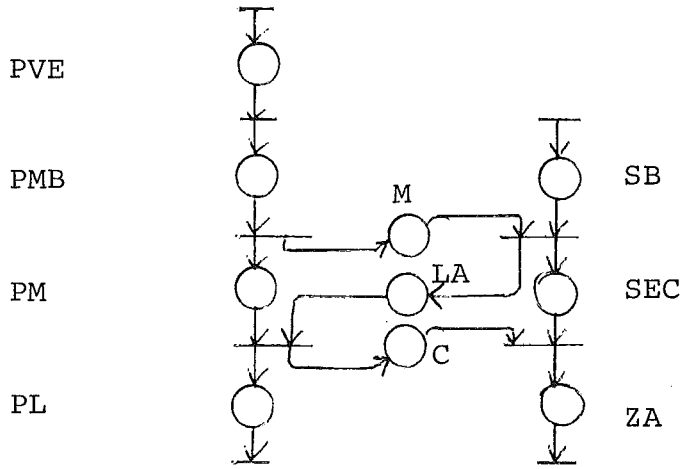
PVE: Paket vor Eingangsstation
PMB: Paket meldebereit
PM: Paket gemeldet
PG: Paket wird gelesen
PL: Paket gelesen
M: Meldung liegt vor
LA: Leseanweisung liegt vor
C: Codewort liegt vor
SB: System bereit
SVLA: System vor Leseanweisung
SEC: System erwartet Codewort
ZA: Zielstationangabe ermittelt.

Bild 12:



Dies läßt sich vereinfachen zu:

Bild 13:



4. Satz:

Danach gibt das Steuersystem einen Auftrag an das Freigabeorgan, die Sperre F2 gibt den Weiterlauf für das Paket frei und das Beschleunigungsteil F1 neigt sich.

Interpretation

Nach der Auswertung des Codezeichens gibt das Steuersystem eine Steueranweisung an das Freigabeorgan. Diese Steueranweisung bewirkt das Aufheben der Sperre F2 und die Beschleunigung des Pakets durch Heben von F1.

Bedingungen/Zustände und Ereignisse

Abkürzungen: wie oben und

AFO: Anweisung an das Freigabeorgan
F2 S: F2 sperrend
F2 F: F2 freigebend
F1 B: F1 beschleunigend (geneigt/gehoben)
F1 N: F1 nachholend (gesenkt)

$[PM] \rightarrow PL \wedge C$
$C\ [\wedge SEC] \rightarrow ZA \wedge AFO$
$AFO \wedge PL \wedge F2S \rightarrow F2F$
$F2F \wedge F1N \rightarrow F1B$

5. Satz:

Dadurch gleitet das Paket weiter, gleichzeitig wird das nachfolgende Paket solange am Einlaufen gehindert, bis die Sperre F2 wieder eingetreten ist.

Interpretation

Daß das Paket weitergleitet ist an dieser Stelle nur von sekundärer Bedeutung, aber berücksichtigenswert. Wichtig ist in diesem Satz insbesondere, daß die Sperre, die sonst durch das Teil F2 gewährleistet wird, während des F2-Zustandes "freigeben" das Teil F1 übernimmt. Außerdem besagt dieser Satz, daß nachdem ein Paket die Eingangsstation verlassen hat, zuerst die Sperre durch F2 wiederhergestellt wird und sich dann F1 wieder senkt.

Bedingungen/Zustände_und_Ereignisse

Abkürzungen wie oben und

PWG: Paket weitergeleitet
EF: Eingangsstation frei

PVE $\rightarrow$ EF $\wedge$ PMB
F1B $\rightarrow$ PWG $\wedge$ F2S $\wedge$ F1N $\wedge$ EF

Bemerkung:

Da sich der erste Absatz (Sätze 1 - 5) des zweiten Abschnitts der Aufgabenstellung fast ausschließlich mit dem zu steuernden System beschäftigt und im Gegensatz dazu der zweite Absatz insbesondere mit dem steuernden System befaßt, soll nun eine (Zwischen-) Gesamtdarstellung der Eingabestation und ihrer Steuerung angeführt werden (Bild 14).

Die Stellen EF, F2S, F1N und SB müssen zu Beginn (Schalter_ein) initialisiert werden. Sie stellen zugleich einen Zustand dar, indem das System auageschaltet werden kann (Schalter_aus). (siehe Teil II).

Anhand dieses Bildes lassen sich nun die impliziten Behauptungen, die Fehlersituationen und die Rückfragen an den Aufgabensteller systematisch ermitteln, analog zu den vorausgegangenen Beispielen in Teil III und Teil I 3. h). Zur Untersuchung der Verklemmungssituationen empfiehlt sich die in Bild 15 dargestellte Vereinfachung.

Bild 14:

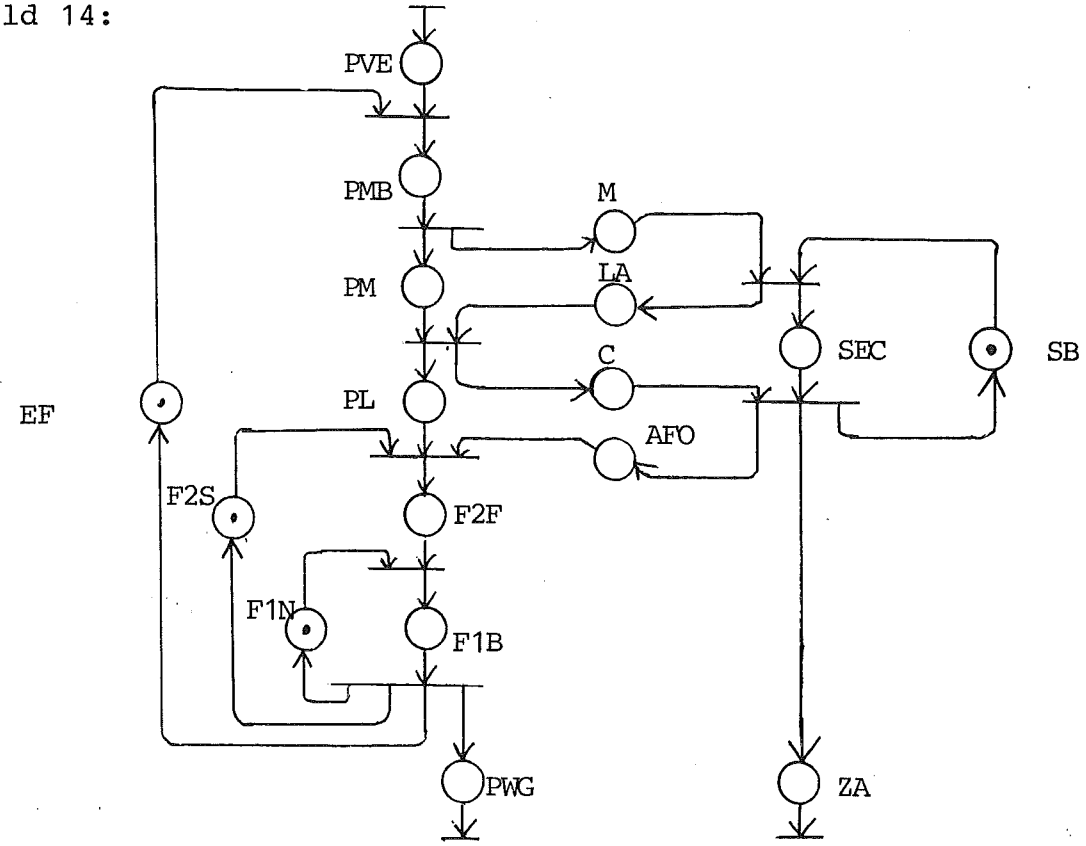
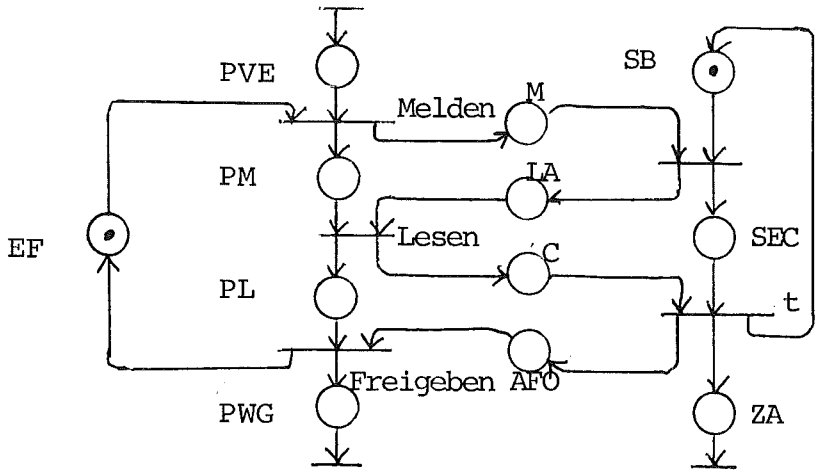


Bild 15:

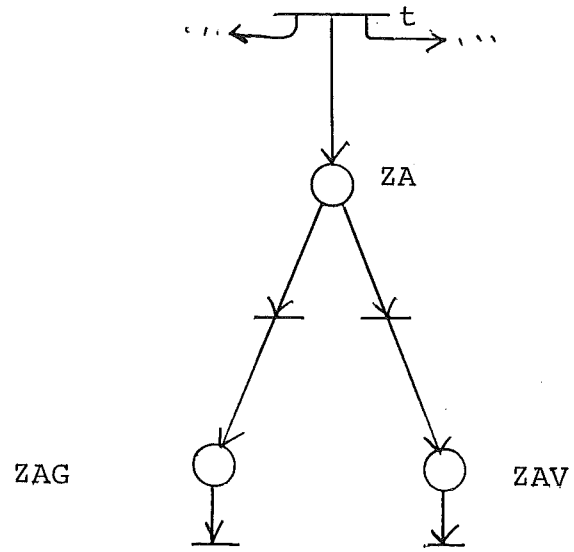


Der Rest der Beschreibung der Eingangsstation handelt von einer Fallunterscheidung und einer Reaktion (eventuelles Warten). Das Warten erfolgt in der linearen Verarbeitung, welche (im Bild 15) durch die Transition t dargestellt wird. Sollte die Fallunterscheidung für den weiteren Ablauf (und dessen Steuerung) wichtig sein, läßt sie sich durch

ZA ⟶ Zielstationangabe_gleich (= ZAG) ∨
Zielstationangabe_verschieden (=ZAV)

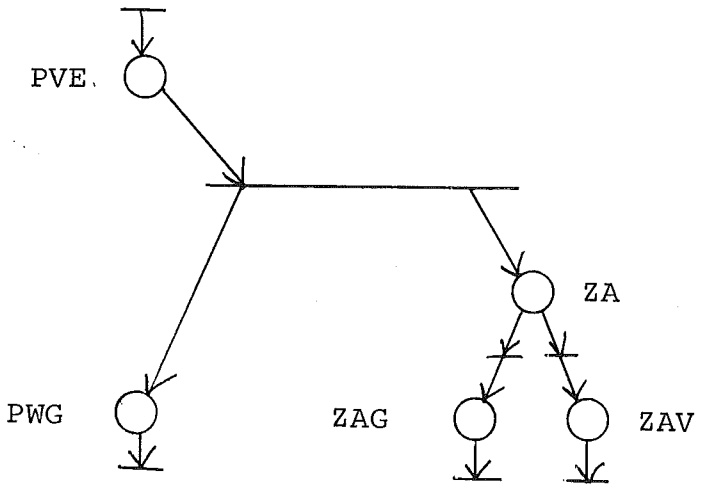
darstellen:

Bild 16:



Hat man alle Untersuchungen (Fehlersituationen, Verklemmungen etc.) abgeschlossen, kann man Bild 15 weiter vereinfachen zu

Bild 17:

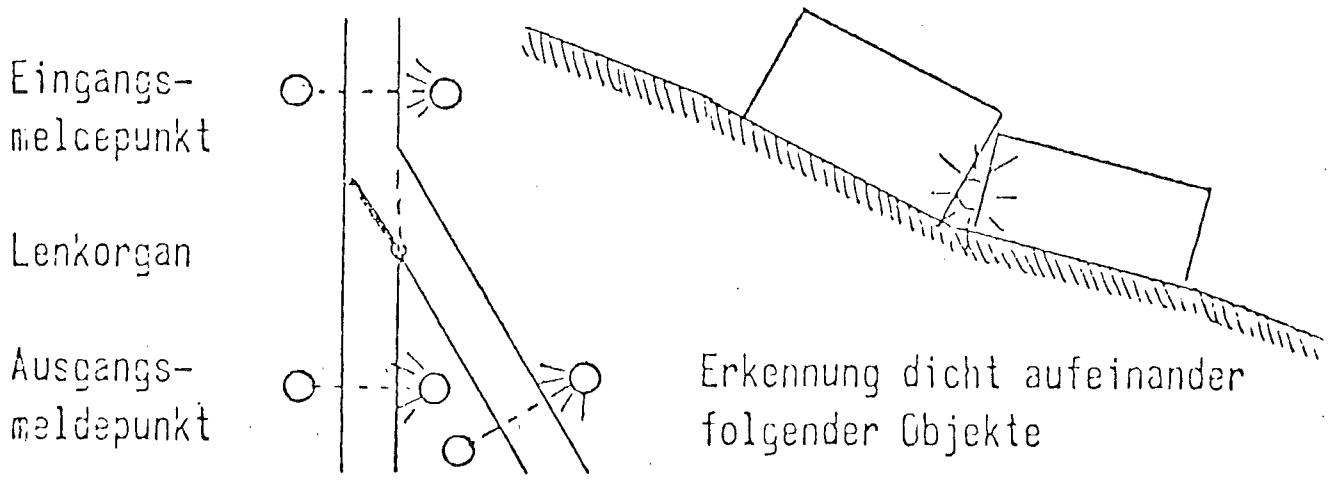


8. DIE VERTEILSTATIONEN

Die Aufgabenstellung lautet:

Bild 3

Verteilstation



3.
Bild 3 zeigt die Verteilstation

Eingangs- und Ausgangspunkte jeder Verteilstation sind mit Lichtschranken versehen. Diese können das Passieren der einzelnen Pakete mit Sicherheit erkennen, auch wenn diese dicht aufeinander folgen. Die Meldungen werden im Steuersystem zur Laufwegverfolgung jedes einzelnen Pakets ausgewertet. Dadurch kann in Verbindung mit dem bereits markierten Ziel der Steuerauftrag für das nächste Lenkorgan ermittelt und ausgegeben werden.

Beim Ausgeben des Steuerauftrages ist darauf zu achten, daß alle Vorläufer die betreffende Verteilstation passiert haben. Die Verteilstation selbst muß frei sin, d.h. zwischen Eingangs- und Ausgangslichtschrnke darf sich kein Paket befinden. Es könnten u.U. mehrere sein, wenn es sich um Pakete mit gleichen Zielstationen handelt.

Tritt auf Grund unterschiedlicher Geschwindigkeiten der Fall ein, daß ein Paket, für das eine Verteilstation umzustellen ist, dessen Eingangsmeldepunkt erreicht, bevor der Vorläufer diese verlassen hat, muß die Umstellung unterbleiben, der Falschläufer bekommt das Ziel seines Vorläufers und der Vorgang wird ausgedruckt.

Die Verteilstationen und ihre Steuerung stellen den kompliziertesten Teil der Paketverteilanlage dar. Eine ausführliche Behandlung würde den Rahmen sprengen. Deshalb sollen die wesentlichsten graphischen Darstellungen genügen.

Darstellung des zu steuernden Prozesses:

Bezeichnungen:

EM: Eingangsmeldung
MAL: Meldung Ausgang links
MAR: Meldung Ausgang rechts
LOL: Lenkorgan in Stellung links
LOR: Lenkorgan in Stellung rechts.

Annahme: Das steuernde System reagiert immer rechtzeitig.

Fall 1: Es wird für jedes Paket eine Steueranweisung an das Lenkorgan gegeben (Bild 18).
L: Steueranweisung "nach links"
R: Steueranweisung "nach rechts".

Fall 2: Es wird nur bei erforderlicher Umstellung des Lenkorgans eine Steueranweisung gegeben (Bild 19).
KST: keine Steueranweisung an das Lenkorgan
ST: Steueranweisung an das Lenkorgan.

Annahme: Das steuernde System kann auch zu langsam reagieren: Bild 20 für Fall 2.

Bild 18:

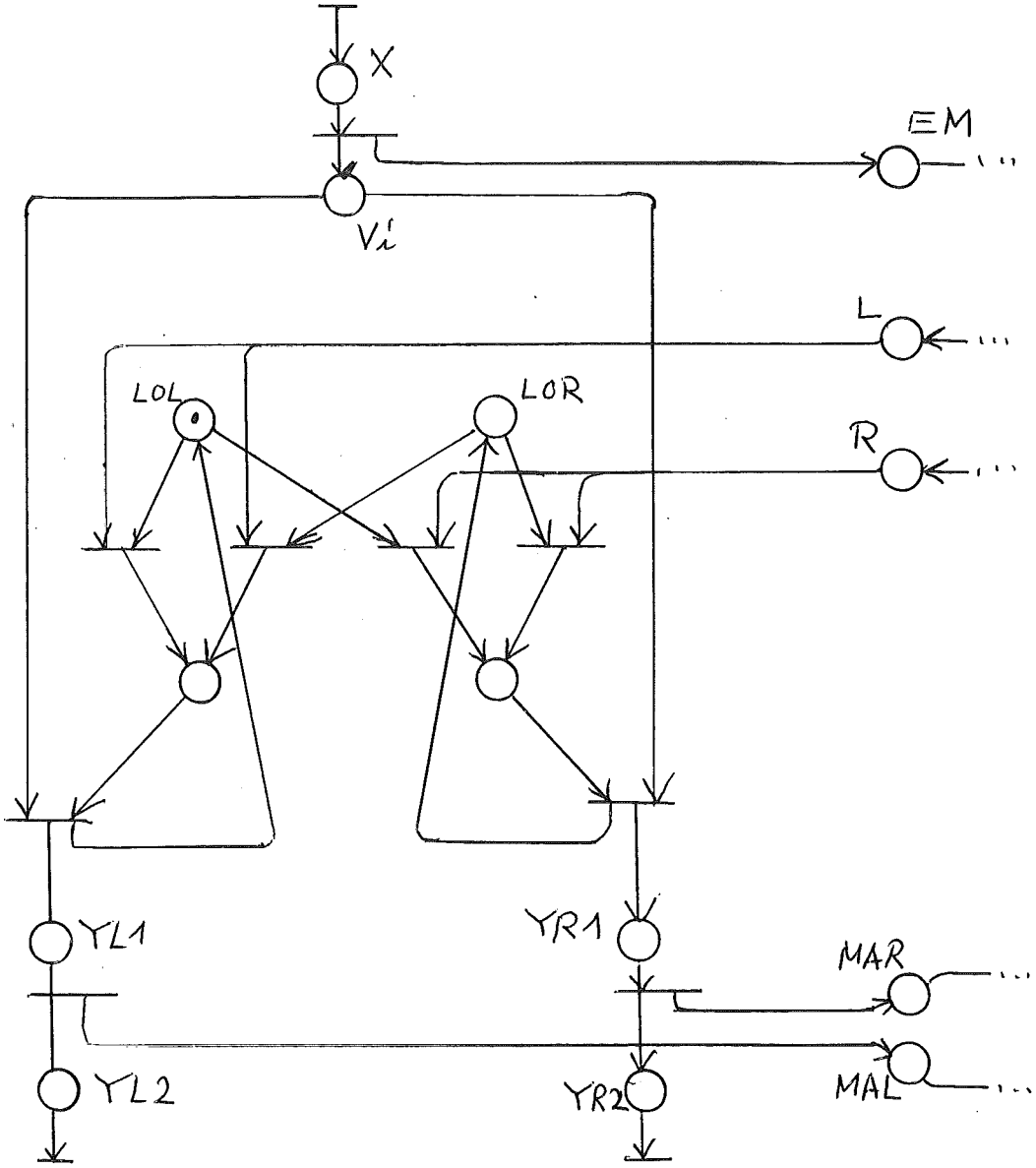


Bild 19:

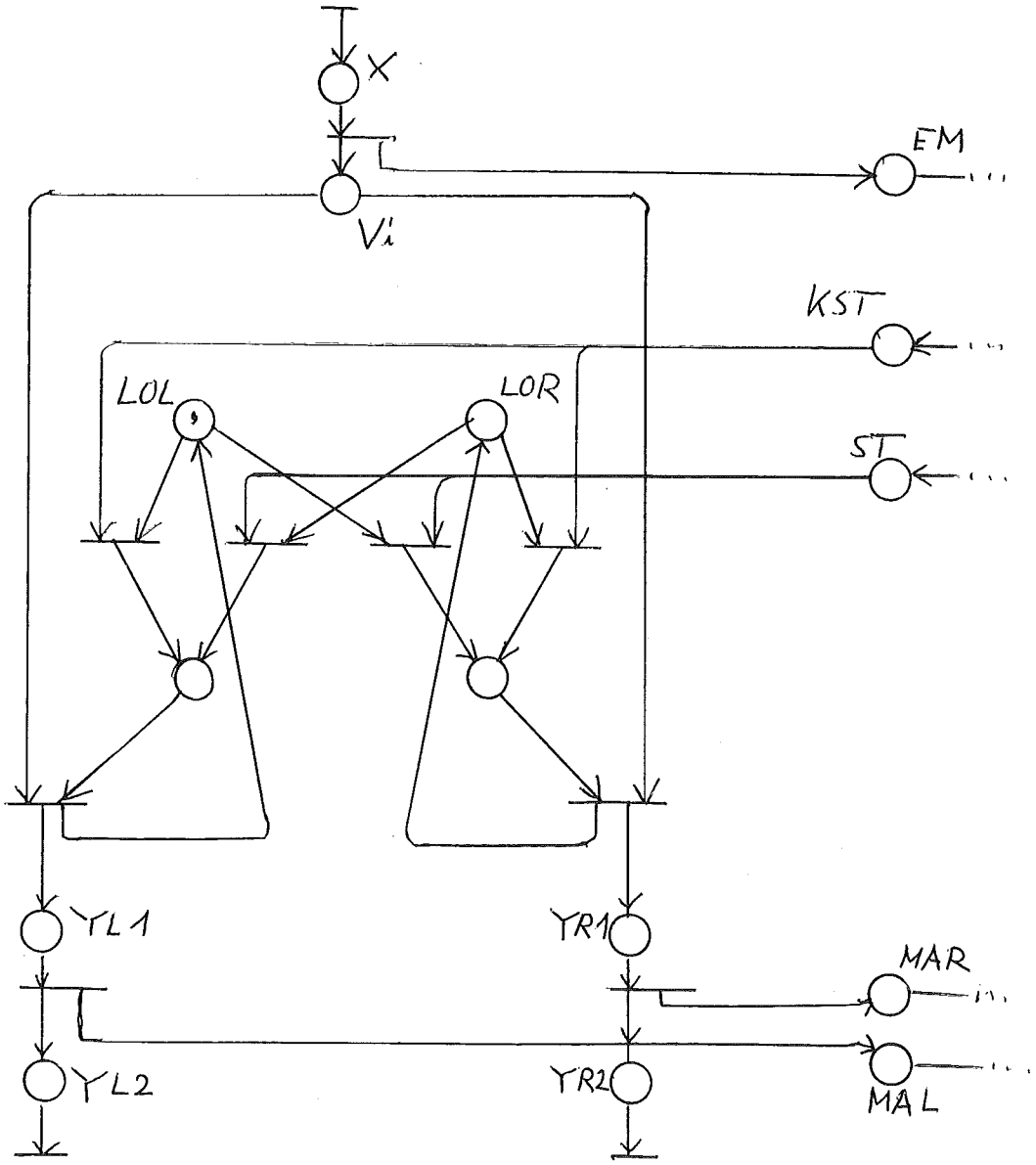
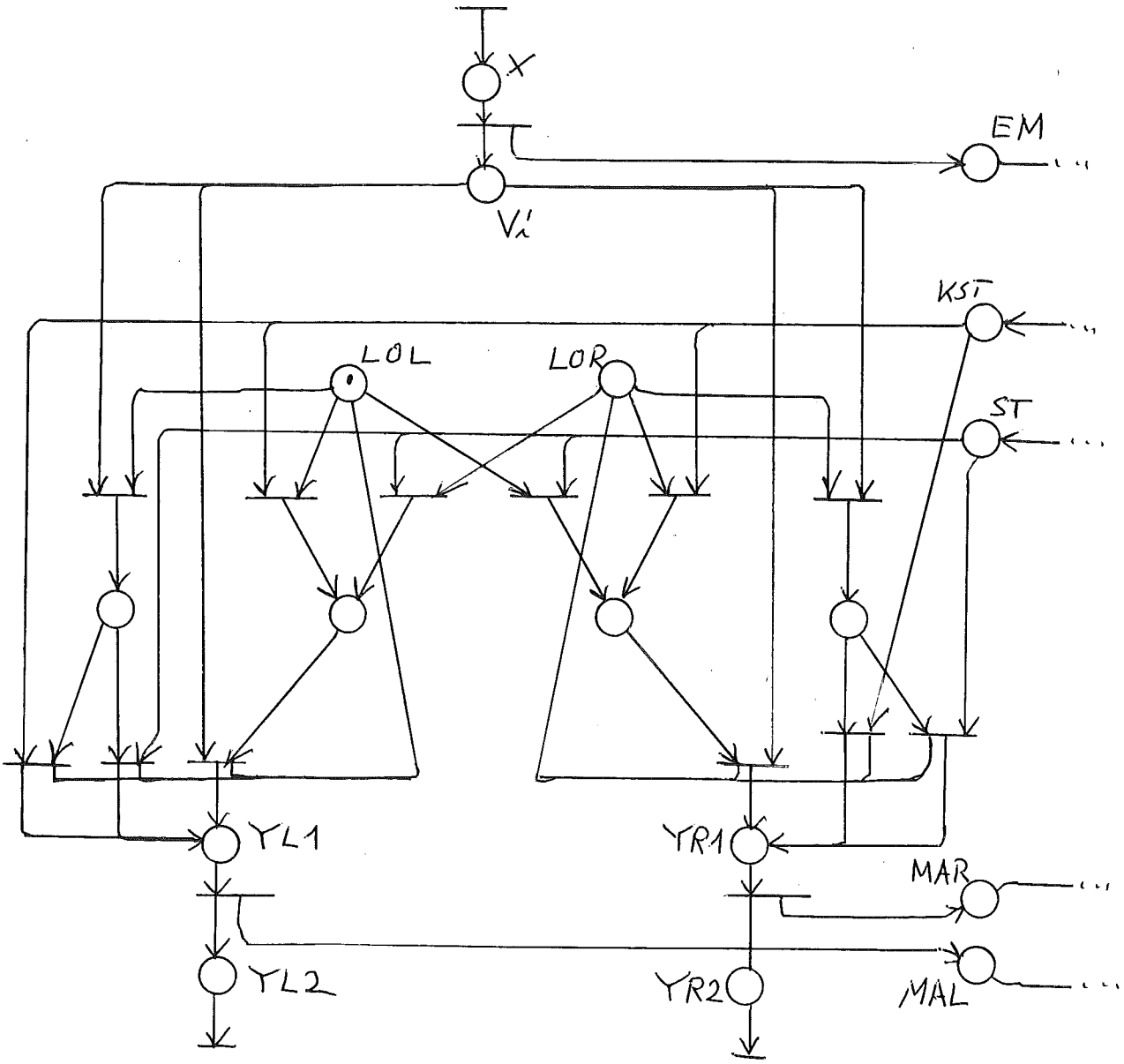
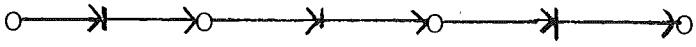


Bild 20:



Bei den Bildern 18, 19 und 20 wurde nur dargestellt, daß ein Paket in einer Verteilstation sein können. Andernfalls werden die Stellen X, Vi, YL1, YL2, YR1 und YR2 zu Puffern der Form

 , wenn z.B. 4 Pakete darin Platz haben.

Darstellung des steuernden Systems

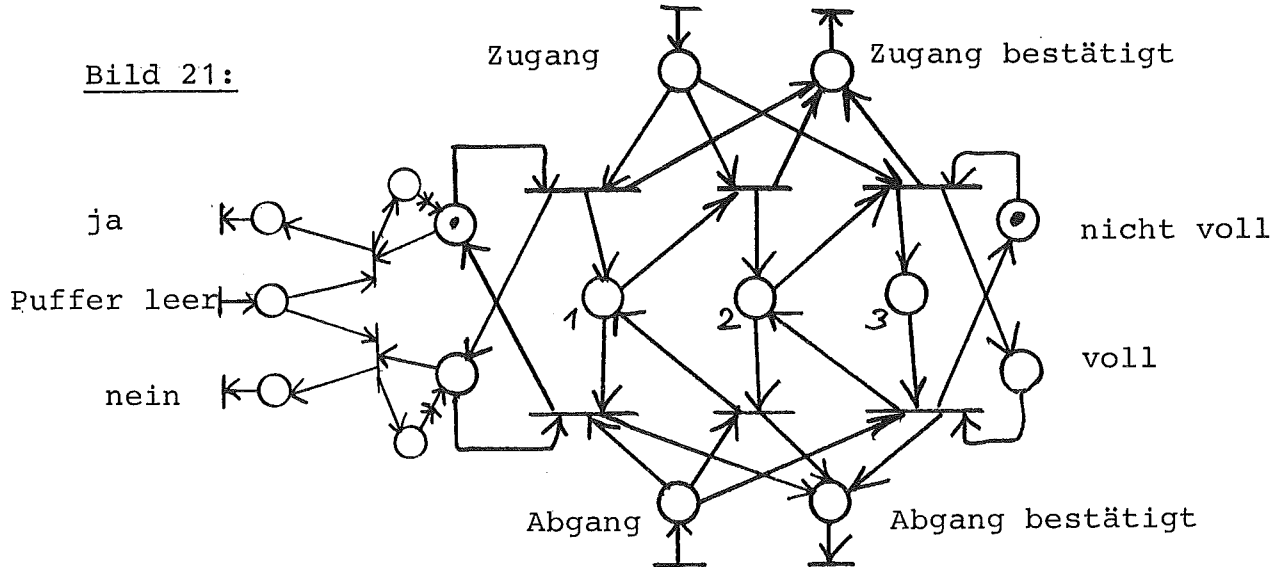
Hauptproblem:

Mitzählen, wieviele Pakete in Verteilstation

Beispiel:

Höchstens 3 Pakete haben in einer Verteilstation Platz.
(angeregt durch und analog /4/)

Bild 21:

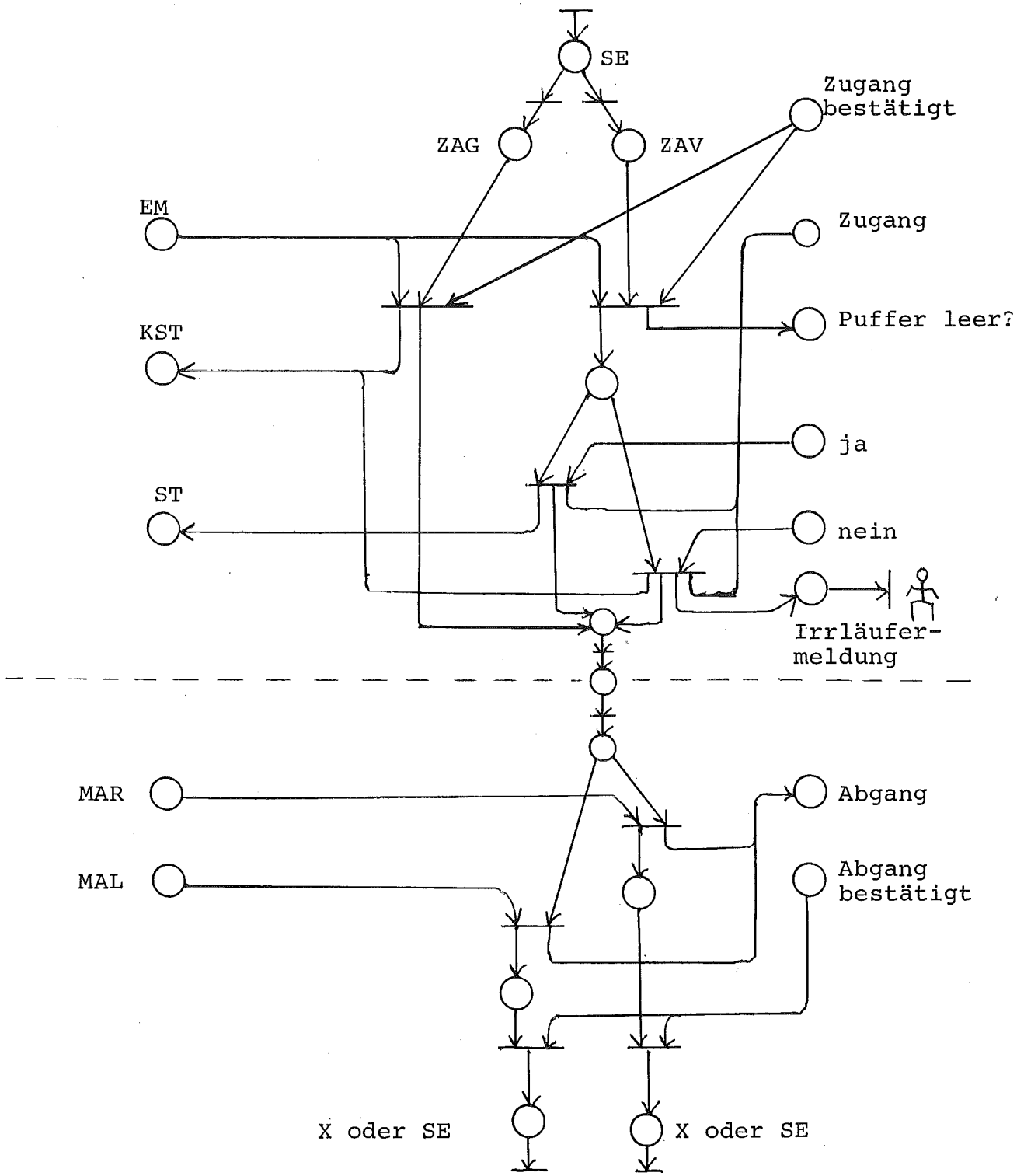


Anmerkung:

Mit Hilfe von Fakten analog zu den Prädikat-Transitions-Netzen /1,2,3/ läßt sich mit einem Theorenprover untersuchen, ob das Petri-Netz im Bild 21 verklemmungsfrei ist.

für Fall 2

Bild 22:



Schlußbemerkung

Wie in Teil III, 5. erwähnt, ist auch auf der theoretischen Seite noch Entwicklungsarbeit nötig. Dies bezieht sich insbesondere auf die Modularisierungsaspekte in Petri-Netzen /2/, und die "Fakten-Netze", /1, 3/. Bezüglich Prozeßkommunikation beinhalten /5, 6/ interessante Aspekte. Gerade letztere gingen in diesen Bericht nicht ein.

Literaturhinweise:

/1/ C. A. Petri: "Über einige Anwendungen der Netztheorie"
GI-9. Jahrestagung, Springer Verlag, 1979

/2/ J. L. Peterson: "Petri Nets", ACM
Computer Surveys, Vol. 9, No. 3, September 1977

/3/ GI-Fachgruppe Petri-Netze und verwandte Systemmodelle
Newsletter Nr. 4, Februar 1980

/4/ L. W. Cooprider: "Petri Nets and the Representation of Standard Synchronizations"
Department of Computing Design, University of Pittsburgh, TA Pennsylvania, 1976

/5/ M. Devy - M. Diaz: "Multilevel specification and validation of the control in communication system"
First International Conference on Distributed Computing Systems, Huntsville Alabama, 1979

/6/ A. Petrenko: "On the Specification and Verification of Communication Protocols"
International Institute for Applied Systems Analysis
Working paper, April 1980.

Entwurf der Steuerung einer Paketverteilanlage durch SSP

(SSP = Methode zum strukturierten Systementwurf von Steuerungen für diskrete technische Prozesse)

von W. Gottschalk

1. Prinzip des Verfahrens

Das Prinzip des Verfahrens besteht darin, daß man bei dem zu entwerfenden Steuersystem einen Unterschied macht zwischen den Funktionen, die das System auf Grund der Anforderungen des Prozesses ausführen muß, und den zusätzlichen Funktionen, die sich aus den Erfordernissen der angewandten Technik ergeben. Diese Trennung erleichtert die erste kreative Entwurfsphase, in welcher man sich zunächst darauf konzentrieren kann, den Ablauf der Steuervorgänge allein aus der Sicht des Prozesses zu entwickeln, losgelöst von den Problemen der späteren Realisierung. Erst wenn von dieser Entwicklung ein überprüftes Ergebnis vorliegt, wird dieses in der zweiten Phase als eine schon weitgehend fehlerfreie Aufgabenstellung für die technische Realisierung verwendet.

Der Einsatzbereich von SSP umfaßt diese erste Phase, in welcher ausschließlich manuell vorgegangen wird. Dabei wird Wert darauf gelegt, daß der Arbeitsaufwand relativ gering ist und daß, was besonders hervorgehoben werden soll, schon in den ersten Konzepten ausreichende Prüfmöglichkeiten geboten werden. Denn je früher Fehler entdeckt werden, desto weniger kostet ihre Beseitigung. Ebenso wurde angestrebt, daß das Verfahren leicht verständlich und leicht anwendbar

Anschrift des Verfassers: Dr.-Ing. Werner Gottschalk
Berliner Platz 1 D, 3300 Braunschweig

ist, um den Benutzerkreis möglichst groß ziehen zu können und damit eine hohe Effizienz des Verfahrens zu erreichen. Das Ergebnis ist das technisch neutrale Konzept der Steuerfunktionen in Form eines Modells. Dabei bedient man sich einer Entwurfssprache, die sowohl für die Fachleute des Prozesses, als auch für die des Steuersystems in gleicher Weise verständlich ist. Dieses interdisziplinäre Kommunikationsmittel gestattet es, den Entwurf von beiden beteiligten Fachrichtungen auf Vollständigkeit, Widerspruchsfreiheit und Eindeutigkeit zu überprüfen. Bis zu diesem Zeitpunkt braucht noch keine Entscheidung über die Form der technischen Realisierung - Hardware oder Software - getroffen zu werden. Die zweite Phase kann dann mit weiteren Entwurfsmethoden bearbeitet werden, die der jeweils gewählten Technik angepaßt sind.

Die Einführung eines technisch neutralen Modells verlangt eine ebensolche Denkweise. SSP benutzt dazu die Grundgedanken der Automatentheorie, welche einen Automanten, wie beispielsweise ein Prozeßsteuersystem, als eine Funktionseinheit auffaßt, deren Ausgabe zu einem bestimmten Zeitpunkt nicht nur von der Eingabe zu diesem Zeitpunkt, sondern auch von vorangegangenen Eingaben abhängig ist. Die dazu erforderliche Speicherung der Eingaben wird als Zustand des Systems definiert. Der Datenfluß, der sich von den Eingaben über die Zustandsmarkierungen bis zu den Ausgaben verzweigt, stellt ein durch die Verarbeitungsfunktionen vielfältig verknüpftes Netz dar und wird als Träger des Funktionsablaufes betrachtet. Das Erarbeiten von Vorstellungen über die Struktur dieses Datenflusses und über die damit verbundene Folge von Zuständen ist die grundlegene Denkweise von SSP.

Komplexe Probleme werden durch hierarchische Strukturierung nach dem allgemein bekannten Top-down-Verfahren überschaubar gemacht. Die praktischen Erfahrungen zeigen, daß die Zielvorstellungen eines

solchen Entwurfsganges nicht immer geradlinig und unmittelbar konkretisiert werden können. Wachsendes Problemverständnis und zunehmende Einsicht in die Möglichkeiten der Mittel führen bei der Detaillierung der unteren Abstraktionsebenen oft zu Erkenntnissen, die eine Änderung der oberen Ebenen einschließlich des Modells notwendig machen, so daß der Arbeitsvorgang rekursiv abläuft. Gerade hier erweist sich der besondere Vorteil von SSP dadurch, daß das technisch neutrale Funktionsmodell als Diskussionsbasis verbunden mit der neutralen Entwurfssprache die Verständigung zwischen den beteiligten Fachgebieten erleichert.

2. Entwurf des Steuersystems für eine Paketverteilanlage

Die bei der Prozeßsteuerung auftretenden Vorgänge und Beziehungen sind derart vielfältig und komplex, daß es zweckmäßig ist, zu ihrer Beschreibung verschiedene Darstellungstechniken zu gebrauchen. Jedes Beschreibungshilfsmittel hat bestimmte Stärken und Schwächen. Die SSP-Entwurfssprache stellt eine Kombination aus verschiedenen Darstellungstechniken dar, die so gewählt sind, daß sie sich gegenseitig ergänzen. Am Beispiel der gestellten Aufgabe soll ihre Anwendung erläutert werden.

2.1 Datenflußnetz

Die obere Abstraktionsebene des Systems (Bild 1) wird als Datenflußnetz dargestellt und mit Hilfe der "Instanzennetztechnik" /1/ beschrieben. Unter der Bezeichnung "Instanz" (quadratisches Symbol) wird ein Systemelement verstanden, in welchem eine Verarbeitung von Daten stattfindet, wie die Steuerung der Eingangsstation und die Steuerung der Verteilstation. Die Instanzen sind durch Funktionseinheiten verbunden, welche den Transport der Daten ausführen. Wobei dieser Transport sofort erfolgen kann (Schnittstelle) oder verzögert eintreten kann (Speicher). Sie werden unter dem gemeinsamen Oberbegriff

"Kanal" zusammengefaßt (Kreissymbol). Im Beispiel stellen die Symbole "Prozeßdaten" Schnittstellen dar und die Symbole "Warteschlange" und "Adaptives Prozeßmodell" Speicher.

Zur Bildung einer hierarchischen Struktur muß zunächst untersucht werden, wieweit sich Instanzen oder Kanäle, vom Problem her gesehen, in Teilfunktionen zerlegen lassen. Nach den Regeln der Instanzennetztechnik kann man dann diese Teilfunktionen als Instanzen bzw. Kanäle einer tieferen Abstraktionsebene interpretieren. Das Verfahren wird solange fortgesetzt, bis die sich ergebenden Funktionselemente eine für die technische Realisierung handhabbare Größenordnung erreicht haben. Im vorliegenden Fall ist dieses Ziel wegen des geringen Umfangs der Aufgabe bereits mit der ersten Abstraktionsstufe erreicht.

2.2 Steuerung Eingangsstation

Eine Verarbeitungsfunktion ist zeitabhängig, darum müssen zu ihrer Beschreibung die Informationen des Datenflusses durch ein zeitorientiertes Element ergänzt werden. Das geschieht dadurch, daß die Folge der Zustandsmarkierungen mit in die Betrachtungen einbezogen und mit Hilfe von Zustandsgraphen nach dem Petrinetz-Verfahren erklärt wird.

In Bild 2 stellen die "Plätze" (Kreissymbol) B, Z, F, L und R die Schnittstelle zum Prozeß dar. Die Plätze AB, LZ und WL bedeuten die Zustandsmarkierungen und sind zu einem Zustandsgraph verbunden. Auf den Verbindungslinien sind Querstriche ES 1, ES 2,3,4 und ES 5 angeordnet, die den Übergang zwischen den benachbarten Zuständen symbolisieren und "Transition" genannt werden.

Zur Kennzeichnung des Betriebsablaufes können auf die einzelnen Plätze Marken aufgelegt werden. Beispielsweise würde das System in Ruhestellung aufnahmebereit sein und auf dem Platz AB eine Marke tragen. Trifft die Besetztmeldung ein, wird auch auf den Platz B eine Marke gelegt. Damit sind beide Plätze, deren Richtungspfeil auf die Transition ES 1 hinweist, markiert und die Vorbedingung erfüllt, daß die Transition "Schalten" kann, d.h. die Marken von AB und B werden entnommen und auf LZ eine Marke gelegt, weil der Lesezustand eingetreten ist.

Nach dem gleichen Prinzip wird bei den übrigen Transitionen verfahren. ES 2,3,4 schaltet, wenn LZ markiert ist. Das "A" bedeutet, daß in einem Attribut eine zusätzliche Aussage gemacht wird, in diesem Fall die Zielinformation des zugeordneten Paketes. Das Schalten bewirkt die Markierung des Weiterlaufs, die Ausgabe des Freigabeauftrags sowie die Weitergabe der Paketmarkierung in die Ablaufwarteschlange. Durch ES 5 wird die Grundstellung wieder hergestellt, wenn das Paket die Station verlassen hat.

Diese Darstellungsform wird auch als Transitionsnetz bezeichnet und bietet die Möglichkeit, bereits beim ersten konzeptförmigen Entwurf eines Systems eine manuelle Simulation vorzunehmen. Damit ist eine von den anfangs genannten Prüfmöglichkeiten gegeben.

Der Ablauf von Funktionsfolgen kann von Bedingungen abhängen, die sich mit den beschriebenen Mitteln nur recht umständlich darstellen ließen. Beispielsweise kann die Ausgabe des Freigabeauftrages sofort beim Eintreffen der Zielinformation erfolgen, wenn die Ablaufwarteschlange leer ist. Das Gleiche gilt, wenn sie noch mit einem Paket besetzt ist, welches das gleiche Ziel hat. Sind dagegen die Ziele verschieden, muß der Freigabeauftrag verzögert werden, damit zwischen den Paketen eine genügend große Zeitspanne zum Umstellen der Weichen bleibt.

Damit die Transitions-Netze eine möglichst große Übersichtlichkeit bieten und in ihrem Umfang nicht zu groß werden, müssen sie von solchen Einzelheiten der Schaltbedingungen der Transitionen möglichst frei gehalten werden. Aus diesem Grunde werden den Transitionsnetzen Entscheidungstabellen zugeordnet, in denen jede Regel eine Transition beschreibt. Tabelle 1 zeigt die Entscheidungstabelle für das Transitionnetz "Eingangsstation".

Die Tabelle enthält alle Bedingungen und alle Aktionen, die beim Schalten der Transitionen mitwirken, wodurch eine gewisse Redundanz entsteht. Sie kann aber in Kauf genommen werden, in Anbetracht des Vorteils, daß Gelegenheit ist, vielfältige Angaben zu ergänzen, die das Netz nicht aufnehmen kann, wie z. B. die Behandlung von Attributen, Zeitangaben, Hinweise auf den Steuerfluß usw. Zum schnellen Verständnis der Tabelle liefert eine Regelübersicht für jede Regel im Klartext eine kurze Angabe, um welchen Betriebsfall es sich handelt. Bedarfsweise können zusätzliche Erklärungen, Formeln usw. zu den Bedingungen, Aktionen und Regeln im Abschnitt "Kommentar" in jedem beliebigen Umfang ergänzt werden.

Ein besonderer Vorteil ist, daß die Tabelle nach den Vorschriften der Entscheidungstabellentechnik eine Prüfung auf Widerspruch, Redundanz und Vollständigkeit gestattet. Dadurch ist hier eine weitere Prüfmöglichkeit des Entwurfs bereits in einer sehr frühen Phase gegeben.

2.2 Steuerung Verteilstation

Das Transitionsnetz der Steuerung der Verteilstation (Bild 3) zeigt zwei parallel ablaufende Vorgänge:

1. Die Markierung des Paketdurchlaufs in den Warteschlangen ZW, DW und AW 1 bzw. AW 2 und

2. Die Markierung der Zustände der Verteilstation SB, SA und PD.

Beide Vorgänge sind über Kommunikationsverbindungen (gestrichelte Linien) miteinander verbunden, die folgendes aussagen:

Die Transition VS 1 schaltet dann, wenn sowohl SB als auch ZW markiert sind. Dabei ist jedoch die Verbindung zu ZW nur informativ, d.h. beim Schalten wird nur die Marke aus SB entnommen, die in ZW bleibt liegen.

Die Transition VS 3 schaltet nur dann, wenn PD markiert ist, dagegen DW leer ist (Bemerkung "=0" neben der Transition). Die Transition VS 2 dagegen schaltet, wenn SA markiert ist aber DW besetzt ist. Zur Unterstreichung des Gegensatzes gegenüber VS 3 ist hier die Bemerkung "≠0" neben der Transition angebracht.

Der Verlauf der übrigen Vorgänge dürfte nach den vorangegangenen Erklärungen aus dem Transitionsnetz (Bild 3) und den zugehörigen Entscheidungstabellen (Tabelle 2 und 3) leicht zu entnehmen sein.

Literatur:

/1/ Oberquelle, H.: "Grobe" Beschreibungen von Systemen durch Netze. Universität Hamburg, Berichte des Instituts für Informatik Nr. 62, 1978.

/2/ Strunz, H.: Entscheidungstabellentechnik, Carl Hanser Verlag, München/Wien, 1977.

/3/ Ullrich, G.: Der Entwurf von Steuerstrukturen für parallele Abläufe mit Hilfe von Petri-Netzen. Universität Hamburg, Berichte des Instituts für Informatik Nr. 36, 1977.

/4/ Gottschalk, W.: SSP, eine Methode zum strukturierten Systementwurf von Steuerungen für diskrete technische Prozesse. Kernforschungszentrum Karlsruhe, Forschungsbericht KfK-PDV 176, 1979.

Steuerung einer Paketverteilanlage - Datenflußnetz

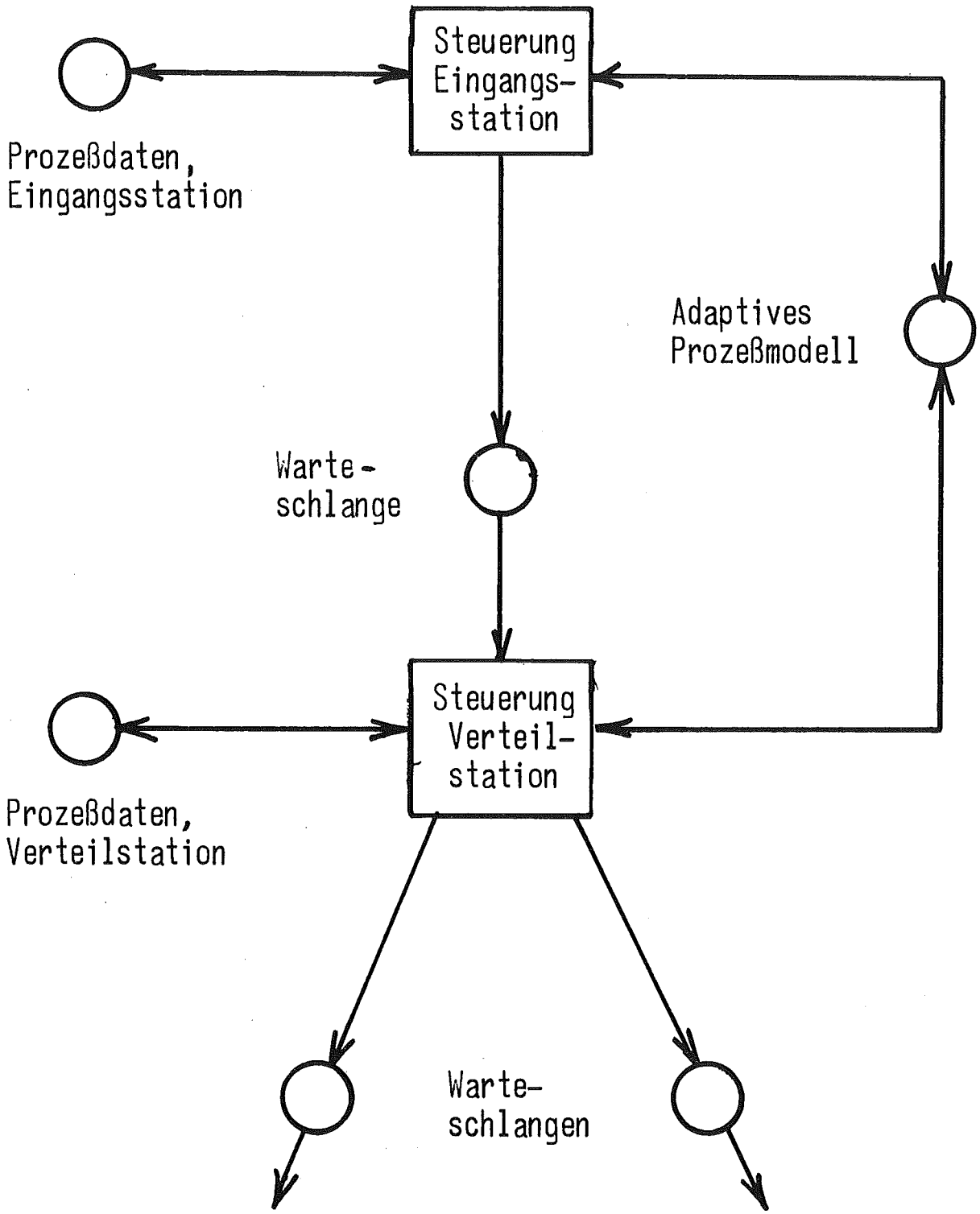


Bild 1

Steuerung Eingangsstation

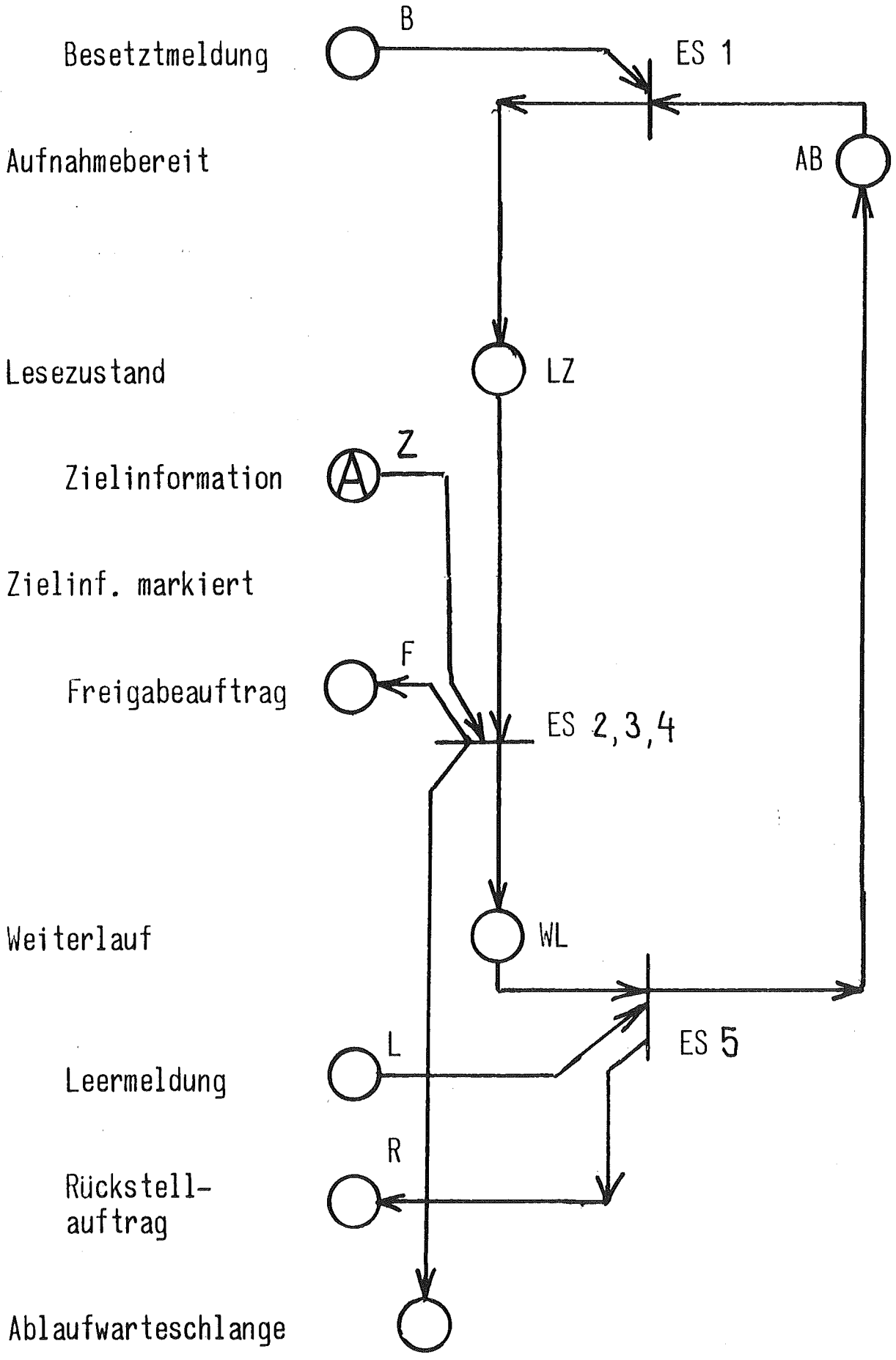


Bild 2

Steuerung Verteilstation

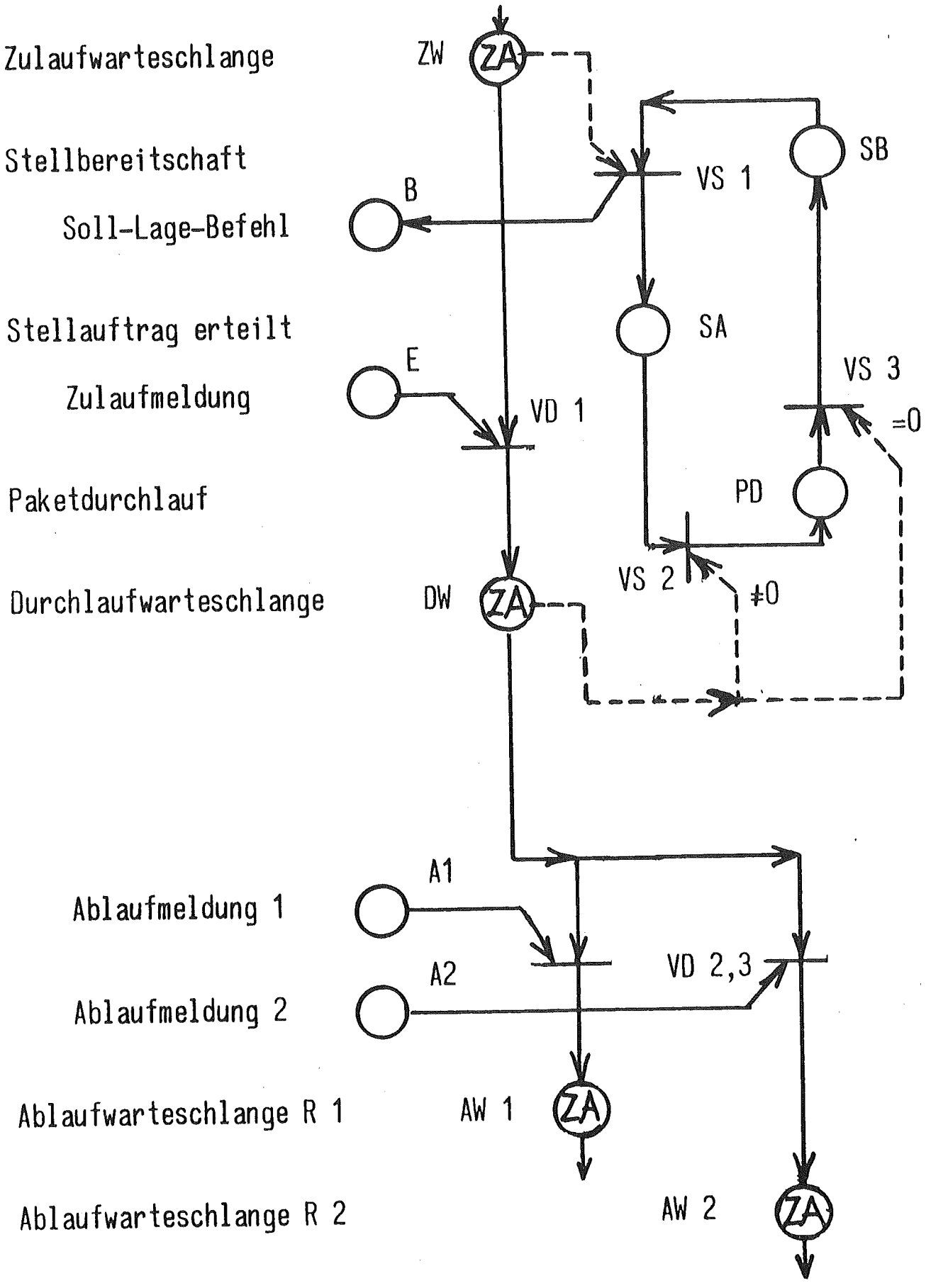


Bild 3

ES	Eingangsstation	1	2	3	4	5	E
B1	Alter Zustand	AB	LZ	LZ	LZ	WL	-
B2	Eingangsinform.	B	-	-	-	L	-
B3	Vorläufer vor nächstem Verteiler	-	N	J	J	-	-
B4	Gleiche Richtung	-	-	J	N	-	-
A1	Mark. in Warteschl.	-	x	x	x	-	-
A2	Auftragsausgabe	-	F	F	-	R	-
A3	Verzögerte Auftrags-ausgabe	-	-	-	F	-	-
A4	Neuer Zustand	LZ	WL	WL	WL	AB	-
A5	Warten auf Meldung	Z	L	L	L	B	-
A6	Tabelle wiederholen	x	x	x	x	x	-
A7	Keine Aktion	-	-	-	-	-	x

Regelübersicht

1: Paket läuft ein

Eintreffen der Zielinformation aktualisiert eine der drei nachfolgenden Regeln:

2: Weiterlauf, Weg zum nächsten Verteiler ist frei
3: Weiterlauf, Vorläufer hat gleiche Richtung
4: Weiterlauf, Vorläufer hat andere Richtung
5: Eingangsstation kommt in Grundstellung

Kommentar

R4: Der Weiterlauf muß verzögert werden, damit der Abstand zwischen den Paketen genügend groß ist, um Zeit zum Umstellen der Weichen zu lassen.

Tabelle 1

VS	Verteilstation, Steuerauftrag	1	2	3	E
B1	Alter Zustand	SB	SA	PD	-
B2	Zuläufer markiert	J	-	-	-
B3	Durchlaufwarteschlange leer	J	N	J	-
A1	Soll-Lage-Befehl ausgeben	x	-	-	-
A2	Neuer Zustand	SA	PD	SB	-
A3	Warten auf Zuläufer	-	-	x	-
A4	Tabelle wiederholen	x	x	x	-
A5	Keine Aktion	-	-	-	x

Regelübersicht

1: Zuläufer löst Soll-Lage-Befehl aus

2: Warten auf Paketeinlauf

3: Grundstellung wenn Station frei

Kommentar

R2: Paketeinlauf muß abgewartet werden, damit neue Stellbereitschaft erst nach Durchlauf eintreten kann.

A1: Die Soll-Lage wir aus dem Attribut des in ZW markierten Paketes ermittelt.

Tabelle 2

VD	Verteilstation, Durchlauf	1	2	3	E
B1	Zulaufmarkierung	ZW	DW	DW	-
B2	Paketmeldung	E	A1/2	A1/2	-
B3	Attribut entspricht Richtung	-	J	N	-
A1	Durchlaufzähler erhöhen	x	-	-	-
A2	Durchlaufzähler erniedrigen	-	x	x	-
A3	Falschläuferbehandlung	-	-	x	-
A4	Ablaufmarkierung	DW	AW1/2	AW1/2	-
A5	Keine Aktion	-	-	-	x

Regelübersicht

1: Paket läuft ein

2: Paket läuft aus (Regelfall)

3: Paket läuft aus (Falschläufer)

Kommentar

A3: Das Attribut der Markierung des falsch gelaufenen Paketes wird durch das des Vorläufers ersetzt.

Der Vorgang wird ausgedruckt.

Tabelle 3

S A D T -

Structured Analysis and Design Technique
Eine kurze Einführung zum Spezifikationsbeispiel "Paketverteilungsanlage"

U. Thielmann
Abteilung Prozeßrechnersysteme
G. Woysch
Forschungszentrum der Standard Elektrik Lorenz AG
Hellmuth-Hirth-Str. 42, 7000 Stuttgart-40

Einleitung

SADT ist eine Methode für die systematische Analyse komplexer Systeme, die von der Firma SOFTECH, USA (1), entwickelt und in verschiedenen Anwendungsgebieten bereits erfolgreich eingesetzt wurde. SADT stellt eine Reihe von Interview -,Analyse - und Dokumentationstechniken zusammen, um damit eine effektive Teamarbeit zu bewirken.
Ausführlich wird SADT in (2,3,4) besprochen, Firmenunterlagen sind bei SOFTECH erhältlich.

Übersicht über die Darstellungsmethodik

Das Beispielproblem "Paketverteilanlage" wurde mit Hilfe von SADT analysiert und dokumentiert. Einige kurze Hinweise sollen das Lesen des Modells erleichtern.
SADT geht bei der Analyse top-down vor und dokumentiert entsprechend.Es entstehen hierarchische Schichten der Systembeschreibung, bei denen von oben nach unten streng nach strukturierter Verfeinerung vorgegangen wird, innerhalb einer gegebenen Ebene ist aber eine Netzstruktur mit Aufspaltung der Schnittstellen und Rückführungen (Schleifen) z.B. von Daten erlaubt.
Dargestellt wird graphisch die sogenannte Funktions- und Aktivitätszerlegung. Bei der Funktionszerlegung wird jede Funktion durch ein "Rechteck" repräsentiert.

Alle Schnittstellen zwischen Funktionen werden durch Pfeile dargestellt und entsprechen den zwischen den Funktionen ausgetauschten Daten. Jede Rechteckseite hat eine spezielle Bedeutung: Links werden die Eingangsdatenpfeile angegeben, rechts erscheinen die Ausgangsdatenpfeile. Von oben zeigt man einschränkende Bedingungen, Syntaxregeln, Triggerbedingungen u.a., während von unten sogenannte Mechanismen eingezeichnet werden. Darunter versteht man bereits feststehende Gegebenheiten, die entweder als Modell existieren können oder bereits implementiert sind (Unterprogramme, Betriebssystem) oder Hardware, die benützt wird (Terminal, Massenspeicher, Rechnersystem). Alle Pfeile und "Rechtecke" sind numeriert und können in ihrer Verfeinerung durch die verschiedenen Ebenen anhand der Numerierung weiterverfolgt werden.
Die Bedeutung aller Elemente eines Diagramms ist in den jedes Diagramm begleitenden Beschreibungsblättern aufgeführt, so daß die grafischen Symbole durch diese Erklärungen inhaltlich definiert werden.
Man liest das SADT-Analysemodell am besten ebenenweise und geht an interessierenden Stellen in eine darunterliegende Ebene.
Jedes Diagramm besitzt einen Rahmen, der Angaben über das Projekt enthält und dazu dient, die entstandenen SADT-Blätter in einer Projektbibliothek zu ordnen und miteinander zu verbinden.

Entstehung des SADT-Modells "Paketverteilungsanlage"

Das SADT-Modell, wie es hier vorgelegt wird, stellt nur den am Ende einer Systemanalyse sichtbaren Teil der SADT-Methodik dar. Vorausgegangen waren ähnliche Entwürfe und Diskussionen zwischen dem "Autor" der Entwürfe und einem "Leser".
SADT selbst gibt Vorschriften, wie bei der Aufstellung des Modells vorzugehen ist, welche Aufgaben wie zu verteilen sind und wann es sinnvoll ist, das Modell abzubrechen. SADT hilft zusätzlich, indem Hinweise für die Erstellung, die Korrektur und Dokumentation der gesamten Schritte zur Erstellung einer Entwicklungsebene gegeben werden. Es ist also jederzeit möglich, den Entwicklungszyklus zurückzuverfolgen und die Gründe für bestimmte Entwurfsentscheidungen nocheinmal zu durchdenken.

(1) SOFTECH, Inc. 460 Totten Pond Road, Wattham, MA 02154,USA

(2) SOFTECH, Inc. An Introduction to SADT, Structured Analysis and Design Technique.

(3) Douglas T.Ross, Kenneth E. Schomann, Jr.: Structured Analysis for Requirements Definition, IEEE Trans. on Software Engineering, Jan. 1977.

(4) Douglas T.Ross, Stuctured Analysis: A Language for Communicating Ideas, IEEE Trans. on Software Engineering, Jan. 1977.

S E L A L R I G H T S R E S E R V E D	USED AT:	AUTHOR: DR.THIELMANN	DATE: 18.12.79	WORKING DRAFT RECOMMEND PUBLIC.	READER	DATE	CONTEXT	
		PROJEKT: PAKET	REV:					
		NOTES: 1 2 3 4 5 6 7 8 9 10						

PAKETVERTEILUNGSANLAGE

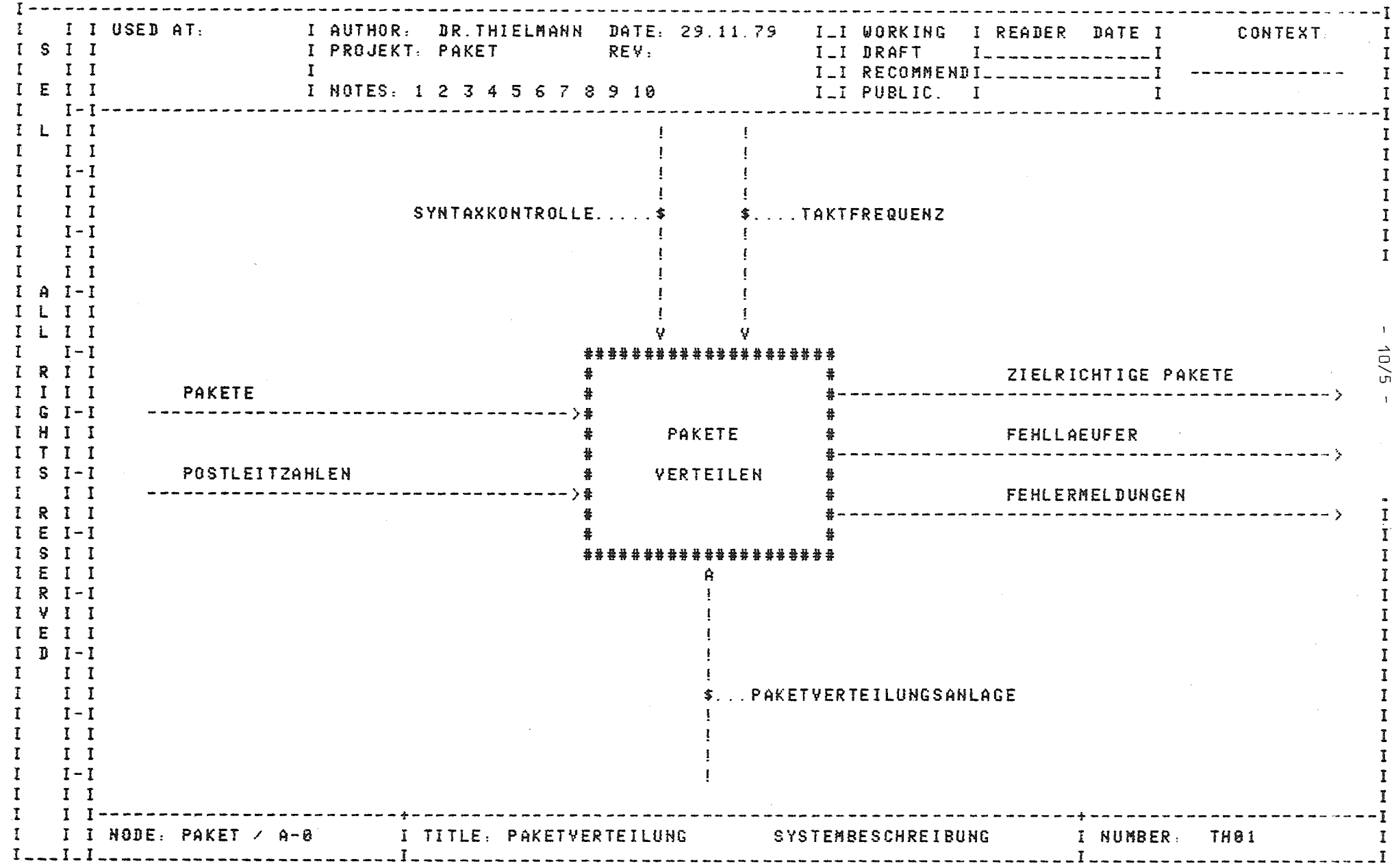
EINE SYSTEMSPEZIFIKATION MIT HILFE VON SADT

STANDARD ELEKTRIK LORENZ A.G. 7000 STUTTGART 40

AUTHOR: U.THIELMANN ZK /PZR

READER: G.WOYSCH CS/FZNH

NODE:	TITLE: PAKETVERTEILUNG	NUMBER: 1


USED AT:
AUTHOR: DR.THIELMANN DATE: 29.11.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMEND
PUBLIC.
READER DATE
CONTEXT
SYNTAXKONTROLLE
TAKTFREQUENZ
PAKETE
POSTLEITZAHLEN
PAKETE VERTEILEN
ZIELRICHTIGE PAKETE
FEHLLAEUFER
FEHLERMELDUNGEN
PAKETVERTEILUNGSANLAGE
NODE: PAKET / A-0
TITLE: PAKETVERTEILUNG SYSTEMBESCHREIBUNG
NUMBER: TH01

USED AT:
AUTHOR: DR.THIELMANN DATE: 10.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT RECOMMEND PUBLIC.
READER DATE
CONTEXT:

S E L A L L A R I G H T S R E S E R V E D

A-0: PAKETE VERTEILEN

DAS GESAMTSYSTEM PAKETVERTEILUNGSANLAGEN SOLL PAKETE NACH BELIEBIG VIELEN ZIELSTATIONEN AUFTEILEN. DIE INFORMATIONEN UEBER DIE ZIELSTATION IST JEWEILS IN DER POSTLEITZAHL DER PAKETE ENTHALTEN.

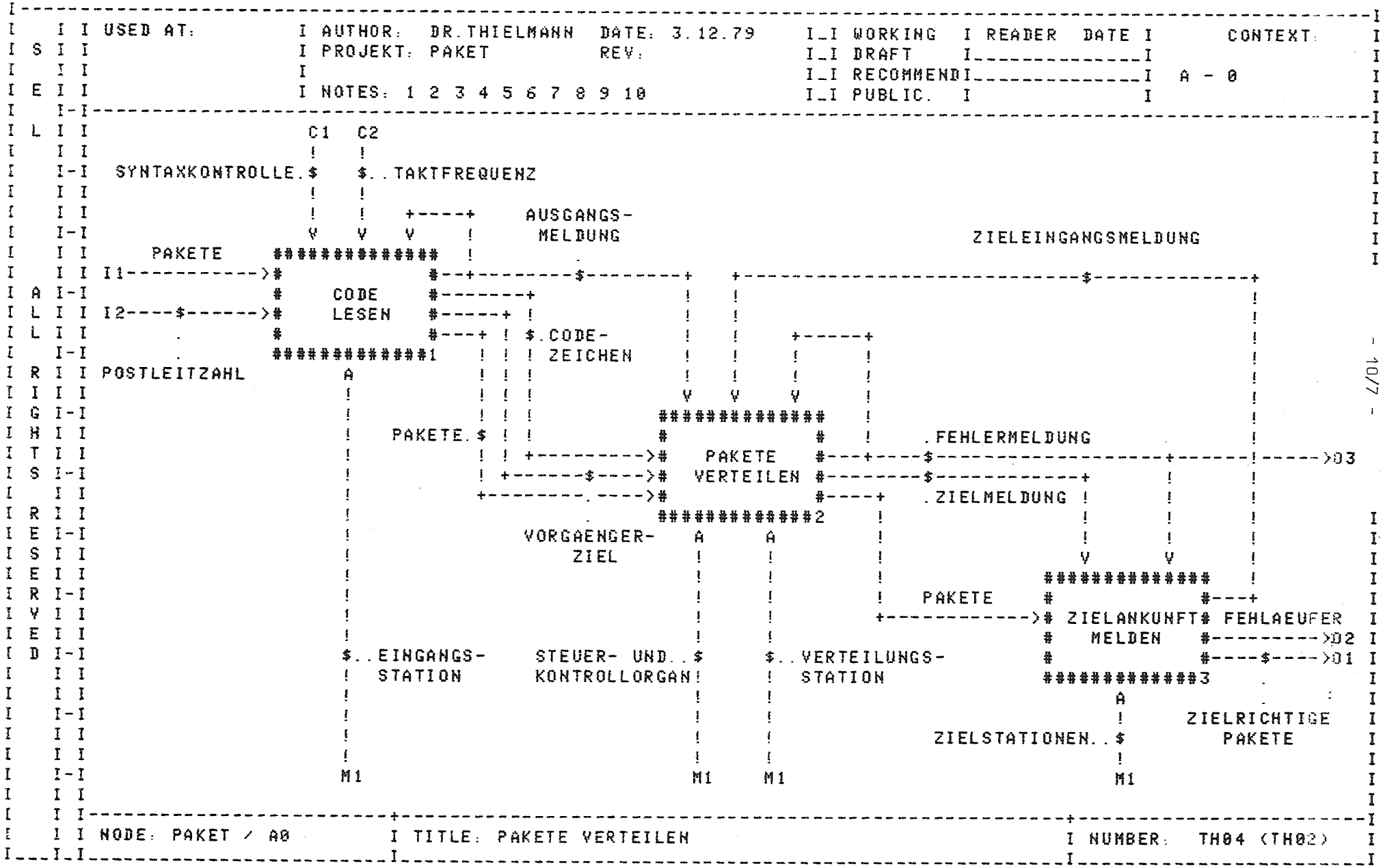
EINGAENGE:
I1 - ANKOMMENDE PAKETE
I2 - POSTLEITZAHL DER ANKOMMENDEN PAKETE

KONTROLL-GROESSEN:
C1 - SYNTAXKONTROLLE
ES SOLL UEBERPRUEFT WERDEN, OB ZU DEN EINGELESENEN POSTLEITZAHLEN AUCH IMMER EINE ZIELSTATION EXISTIERT.
C2 - TAKTFREQUENZ
DER ZEITTAKT WIRD ZUR STEUERUNG DER PAKETE BENOETIGT.

AUSGAENGE:
O1 - ZIELRICHTIG GELEITETE PAKETE
DIE PAKETE, DIE IN DIE RICHTIGE ZIELSTATION GELEITET WURDEN
O2 - FEHLLAEUFER
PAKETE, DEREN POSTLEITZAHL KEINER ZIELSTATION ZUGEORDNET WERDEN KOENNEN ODER DIE IN DEN VERTEILERSTATIONEN FEHLERHAFT GELEITET WORDEN SIND.
O3 - FEHLERMELDUNGEN
FEHLGELEITETE PAKETE UND FEHLER IN DER VERTEILANLAGE WERDEN PROTOKOLLIERT.

MECHANISMEN:
M1 - PAKETVERTEILUNGSANLAGE
DIE PAKETVERTEILUNGSANLAGE BESTEHT AUS EINER EINGANGSSTATION, IN DER DIE ZIELINFORMATION EINGELESEN WIRD, EINER BAUMSTRUKTUR VON VERTEILSTATIONEN UND DEN ZIELSTATIONEN.

NODE: PAKET / A-0 TITLE: PAKETVERTEILUNG SYSTEMBESCHREIBUNG NUMBER: 1


USED AT:
AUTHOR: DR.THIELMANN DATE: 3.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMEND
PUBLIC.
READER DATE
CONTEXT:
A - 0
SYNTAXKONTROLLE.
TAKTFREQUENZ
AUSGANGS-
MELDUNG
ZIELEINGANGSMELDUNG
PAKETE
POSTLEITZAHL
CODE
LESEN
$.CODE-
ZEICHEN
PAKETE.
.FEHLERMELDUNG
PAKETE
VERTEILEN
.ZIELMELDUNG
VORGAENGER-
ZIEL
PAKETE
ZIELANKUNFT
MELDEN
FEHLAEUFER
EINGANGS-
STATION
STEUER- UND
KONTROLLORGAN
VERTEILUNGS-
STATION
ZIELSTATIONEN..
ZIELRICHTIGE
PAKETE
NODE: PAKET / A0
TITLE: PAKETE VERTEILEN
NUMBER: TH04 (TH02)

USED AT:	AUTHOR: DR.THIELMANN DATE: 12.12.79 PROJEKT: PAKET REV: NOTES: 1 2 3 4 5 6 7 8 9 10	WORKING / DRAFT / RECOMMEND / PUBLIC.	READER DATE
CONTEXT: A - 0			

SEL

ALL RIGHTS RESERVED

A0: PAKETE VERTEILEN

DIE FUNKTION 'PAKETE VERTEILEN' WIRD IN DREI UNTERFUNKTIONEN ZERLEGT

-A1: CODE LESEN
DIE POSTLEITZAHL DER EINLAUFENDEN PAKETE WIRD EINGELESEN UND ALS CODE AN DIE VERTEILEINHEIT WEITERGEMELDET.ES WIRD GLEICH UEBERPRUEFT, OB DAS AKTUELLE PAKET DASSELBE ZIEL WIE SEIN VORGAENGER HAT.

-A2: PAKETE VERTEILEN
DIE PAKETE WERDEN IN DER ANLAGE IN DIE VERSCHIEDENEN RICHTUNGEN VERTEILT

-A3:ZIELANKUNFT MELDEN
DIE ZIELSTATIONEN MELDEN DIE ANKUNFT DER PAKETE UND TEILEN SIE AUFGRUND DER MELDUNGEN DES KONTROLLORGANS IN FEHLLAEUFER UND RICHTIG GELEITETE PAKETE EIN.

NODE: PAKET / A0	TITLE: PAKETE VERTEILEN FUNKTIONSBESCHREIBUNG A0	NUMBER: 1

		USED AT:	AUTHOR: DR.THIELMANN	DATE: 12.12.79	WORKING	READER	DATE	CONTEXT
	S		PROJEKT: PAKET	REV:	DRAFT			
					RECOMMEND			A - 0
	E		NOTES: 1 2 3 4 5 6 7 8 9 10		PUBLIC.			

L A L L R I G H T S E R V E D

FUNKTION 1: CODE LESEN

EINGAENGE: I1 - ANKOMMENDE PAKETE
I2 - POSTLEITZAHLEN DER ANKOMMENDEN PAKETE

KONTROLL-GROESSEN: C1 - SYNTAXKONTROLLE
ES SOLL UEBERPRUEFT WERDEN, OB ZU DEN EINGELESENEN POSTLEITZAHLEN AUCH IMMER EINE ZIELSTATION EXISTIERT.
C2 - TAKTFREQUENZ
DER ZEITTAKT WIRD ZUR ABSTANDSBESTIMMUNG DER PAKETE BENOETIGT.
C3 - AUSGANGSMELDUNG DER EINGANGSSTATION
SOWIE DIE AUSGANGSMELDUNG ERFOLGT, KANN DAS NAECHSTE PAKET IN DIE EINGANGS-STATION AUFGENOMMEN WERDEN.

AUSGAENGE: O1 - AUSGANGSMELDUNG
DER AUSGANG DER PAKETE WIRD DER VERTEILEINHEIT BEKANNTGEGEBEN.
O2 - CODEZEICHEN
DIE IN CODEZEICHEN UMGEFORMTEN POSTLEITZAHLEN WERDEN DER VERTEILEINHEIT ZUR STEUERUNG DER PAKETE UEBERMITTELT.
O3 - VORGAENGERZIEL
FALLS DASSELBE ZIEL WIE FUER DAS VORGAENGERPAKET VORLIEGT, WIRD DIESER SACHVERHALT DER VERTEILEINHEIT MITGETEILT.
O4 - PAKETE
DIE PAKETE WERDEN NACHDEM DER CODE EINGELESEN IST AN DIE VERTEILEINHEIT WEITERGELEITET.

MECHANISMEN: M1 - EINGANGSSTATION
DIE EINGANGSSTATION BESTEHT AUS MELDEORGAN, LESEORGAN UND FREIGABEORGAN.

NODE: PAKET / A0	TITLE: PAKETE VERTEILEN	PFEILBESCHREIBUNG A0 NUMBER: 1

USED AT:	AUTHOR: DR.THIELMANN DATE: 12.12.79 PROJEKT: PAKET REV: NOTES: 1 2 3 4 5 6 7 8 9 10	WORKING / READER / DATE DRAFT RECOMMEND PUBLIC.

CONTEXT: A - 0

SELECTED / ALL RIGHTS RESERVED

FUNKTION 2: PAKETE VERTEILEN

EINGAENGE:
I1 - CODEZEICHEN ,   SIEHE O2 VON FKT.1
I2 - VORGAENGERZIEL ,   SIEHE O3 VON FKT.1
I3 - PAKETE ,   SIEHE O4 VON FKT.1

KONTROLL-GROESSEN:
C1 - AUSGANGSMELDUNG DER EINGANGSSTATION , SIEHE O1 VON FKT.1
C2 - ZIELEINGANGSMELDUNG
DER ZIELEINGANG WIRD DEM STEUERORGAN GEMELDET, DER VERTEILVORGANG IST FUER DAS BETREFFENDE PAKET DAMIT BEENDET, DIE PAKETDATEN WERDEN GELOESCHT.
C3 - FEHLERMELDUNG
FALLS INNERHALB DES VERTEILSYSTEMS EIN HARDWAREFEHLER AUFTRITT, WIRD DIESER DEM STEUERORGAN GEMELDET.
C4 - DIE EINGANGS- UND AUSGANGSMELDUNGEN DER EINZELNEN VERTEILSTATIONEN DIENEN DER KONTROLLE DES VERTEILVORGANGS.

AUSGAENGE:
O1 - FEHLERMELDUNGEN
HARDWAREFEHLER, DIE INNERHALB DER VERTEILEINHEIT AUFTRETEN WERDEN PROTOKOLLIERT.
O2 - ZIELMELDUNG
DIE ZIELSTATIONEN WERDEN UEBER DIE ANKUNFT EINES PAKETES INFORMIERT.
O3 - PAKETE
DIE AUFGETEILTEN PAKETE WERDEN AN DIE ZIELSTATIONEN WEITERGELEITET.

MECHANISMEN:
M1 - STEUER- UND KONTROLLORGAN
DIESE EINHEIT STEUERT UND UEBERWACHT DEN GESAMTEN VERTEILVORGANG SOWIE DIE GESAMTHEIT DER VERTEILSTATIONEN.
M2 - VERTEILSTATIONEN
DIE VERTEILANLAGE BESTEHT AUS EINER BAUMSTRUKTUR VON VERTEILSTATIONEN.

- 10/10 -

USED AT:
AUTHOR: DR.THIELMANN DATE: 12.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT RECOMMEND PUBLIC.
READER DATE
CONTEXT
A - 0

SEIELLARIERVED

FUNKTION 3: ZIELANKUNFT MELDEN

EINGAENGE: I1 - PAKETE
DIE BEI DEN ZIELSTATIONEN ANKOMMENDEN PAKETE

KONTROLL-GROESSEN: C1 - ZIELMELDUNG , SIEHE 03 VON FKT.2
C2 - FEHLERMELDUNG
FALLS EIN PAKET FEHLGELEITET WURDE, WIRD DIES DER ZIELSTATION MITGETEILT,
BEI DER DIESER FEHLLAEUFER ANKOMMT.

AUSGAENGE: 01 - ZIELEINGANGSMELDUNG
JEDER PAKETEINGANG BEI EINER ZIELSTATION WIRD DEM STEUER- UND KONTROLLORGAN
MITGETEILT.
02 - FEHLLAEUFER
FEHLGELEITETE PAKETE WERDEN BEI DEN ZIELSTATIONEN AUSGESONDERT.
03 - ZIELRICHTIGE PAKETE
ANHAND DER ZIELMELDUNG WERDEN ZIELRICHTIGE PAKETE ALS SOLCHE IDENTIFIZIERT.

MECHANISMEN: M1 - ZIELSTATION

NODE: PAKET / A0 TITLE: PAKETE VERTEILEN PFEILBESCHREIBUNG A0 NUMBER: 3

```
I---------------------------------------------------------------------------------------------------------------I
I   I I USED AT:         I AUTHOR:  DR.THIELMANN  DATE: 3.12.79    I_I WORKING   I READER  DATE I      CONTEXT:   I
I S I I                  I PROJEKT: PAKET         REV:             I_I DRAFT     I______________I                 I
I   I I                  I                                         I_I RECOMMENDI______________I  FKT.1 VON A0    I
I E I I                  I NOTES: 1 2 3 4 5 6 7 8 9 10             I_I PUBLIC.   I              I                 I
I   I-I---------------------------------------------------------------------------------------------------------I
I L I I                                    C1                                               C2                  I
I   I I                 +----------------------!---------------------------------------------!----+             I
I   I-I  PAKETE         !                      !                                             !    !             I
I   I I      .          V                      !                                             !    !             I
I   I I      .  ############## EINGANGSMELDUNG !                                             !    !             I
I   I-I I1--$->#             #-------------+   $...SYNTAXKONTROLLE               TAKTFREQUENZ...$    !             I
I   I I        # PAKETEING.  #             !   !                                             !    !             I
I   I I        #  MELDEN     #---+         !   !                                             !    !             I
I A I-I        #             #   !         !   !                                             !    !
I L I I        ##############1   !         V   V                                             !    !
I L I I              A           !     ##############    CODEZEICHEN                         !    !
I   I-I              !           !     #            #----+-----------------------------------!----!------->O2
I R I I              !           !     # CODE LESEN #    !                                   !    !
I I I I I2--$--------!-----------!---->#UND MELDEN  #    !          +--------+               !    !
I G I-I    .         !           !     #            #    !          !        !               !    !
I H I I    .         !           !     ##############2   !          V        $.VORGAENGERZIEL!    !
I T I I POSTLEITZAHL !           !           A           !    ##############  !              !    !
I S I-I              !           !           !           !    #            #-+---------------!----!------->O3
I   I I              !           !           !           !    #  ABSTAND   # .ABSTANDSMELDUNG!    !
I R I I              !           !           !           +---># BESTIMMEN  #-$-------+       !    !
I E I-I              !          $..PAKETE    !                #            #         !       !   $.AUSGANGS-  I
I S I I              !           !           !                ##############3        V       V   ! MELDUNG    I
I E I I              !           !           !                                  ############## !              I
I R I-I              !           !           !                                  #            #-+------->O1    I
I V I I              !           !           !                                  #   PAKET    #                I
I E I I              !           +-----------!--------------------------------->#  FREIGEBEN #--$------>O4    I
I D I-I              !                       !                                  #            #  .             I
I   I I              !                       !                                  ##############4  .PAKETE      I
I   I I              !                       !                                       A      A                 I
I   I-I              $..EINGANGSMELDEORGAN   $..CODELESER                            !      !                 I
I   I I              !                       !                          TAKTGEBER...$      $..FREIGABEORGAN   I
I   I I              !                       !                                       !      !                 I
I   I-I              M1                      M1                                      M1     M1                I
I   I I                                                                                                       I
I   I I------------------------+--------------------------------------------------+-------------------------I
I   I I NODE: PAKET / A1       I TITLE: PAKETCODE LESEN                           I NUMBER:  TH05 (TH03)    I
I___I_I________________________I__________________________________________________I_________________________I
```

- 10/12 -

		USED AT:	AUTHOR: DR.THIELMANN	DATE: 12.12.79	WORKING	READER DATE	CONTEXT
	S		PROJEKT: PAKET	REV:	DRAFT		
					RECOMMEND		FKT.1 VON A0
	E		NOTES: 1 2 3 4 5 6 7 8 9 10		PUBLIC.		

L

A1: PAKETCODE LESEN

DIE FUNKTION 'PAKETCODE LESEN' UMFASST ALLE FUNKTIONEN, DIE IN DER EINGANGSSTATION ABGEARBEITET WERDEN.

-A11: PAKETE EINGANG MELDEN
DER PAKETEINGANG WIRD GEMELDET, DAMIT DER LESEVORGANG BEGINNEN KANN

A
L
L

-A12: CODE LESEN UND MELDEN
DIE POSTLEITZAHLEN WERDEN GELESEN UND DER ERZEUGTE CODE AN DAS STEUER- UND KONTROLLORGAN WEITERGEMELDET. FALLS AUFGRUND DER SYNTAXKONTROLLE FESTGESTELLT WIRD, DASS DAS PAKET KEINER ZIELSTATION ZUGEORDNET WERDEN KANN, WIRD EIN BESTIMMTER CODE (Z.B. 000000) ERZEUGT, MIT DEM DAS PAKET ALS FEHLLAEUFER KLASSIFIZIERT WIRD.

R
I
G
H
T

-A13: ABSTAND BESTIMMEN
ANHAND DER CODEZEICHEN WIRD ABGEPRUEFT, OB DIESELBE ZIELSTATION WIE FUER DAS VORGAENGERPAKET VORLIEGT. FALLS DIESELBE ZIELSTATION VORLIEGT, KANN DAS PAKET IN WESENTLICH KUERZEREM ZEITLICHEN ABSTAND ALS GEWOENLICH FREIGEGEBEN WERDEN.

S
I
D
E

-A14: PAKETFREIGABE
DIE PAKETE WERDEN IN BESTIMMTEN ZEITLICHEN ABSTAENDEN FREIGEGEBEN. AUSSCHLAGGEBEND FUER DEN ABSTAND ZUM VORGAENGER IST, OB DIESELBE ZIELSTATION WIE FUER DEN VORGANEGER ANGESTEUERT WERDEN SOLL.

NODE: PAKET / A1	TITLE: CODE LESEN	FUNKTIONSBESCHREIBUNG A1 NUMBER: 1

USED AT:	AUTHOR: DR.THIELMANN DATE: 12.12.79	WORKING	READER	DATE	CONTEXT
	PROJEKT: PAKET REV:	DRAFT			
		RECOMMEND			FKT.1 VON A0
	NOTES: 1 2 3 4 5 6 7 8 9 10	PUBLIC.			

FUNKTION 1: PAKET EINGANG MELDEN

EINGAENGE: I1 - ANKOMMENDE PAKETE

KONTROLL-GROESSEN: C1 - AUSGANGSMELDUNG DER EINGANGSSTATION
DAMIT IST DER ZUGANG FUER DAS NAECHSTE PAKET FREI.

AUSGAENGE: O1 - EINGANGSMELDUNG
DER PAKETEINGANG WIRD GEMELDET.
O2 - PAKETE

MECHANISMEN: M1 - EINGANGSMELDEORGAN

FUNKTION 2: CODE LESEN UND MELDEN

EINGANEGE: I1 - POSTLEITZAHL

KONTROLL-GROESSEN: C1 - EINGANGSMELDUNG
C2 - SYNTAXKONTROLLE
ES SOLL UEBERPRUEFT WERDEN, OB ZU DEN EINGELESENEN POSTLEITZAHLEN
AUCH IMMER EINE ZIELSTATION EXISTIERT.

AUSGAENGE: O1 - CODEZEICHEN
DIE CODEZEICHEN BEINHALTEN DIE STEUERINFORMATIONEN FUER DIE VERTEIL-
ANLAGE, SIE WERDEN AN DAS STEUER- UND KONTROLLORGAN WEITERGEMELDET.

MECHANISMEN: M1 - CODELESER

NODE: PAKET / A1	TITLE: CODE LESEN PFEILBESCHREIBUNG A1	NUMBER: 1

		USED AT:	AUTHOR: DR.THIELMANN	DATE: 12.12.79	I_I WORKING	READER DATE	CONTEXT	
	S		PROJEKT: PAKET	REV:	I_I DRAFT			
					I_I RECOMMEND		FKT.1 VON A0	
	E		NOTES: 1 2 3 4 5 6 7 8 9 10		I_I PUBLIC.			

L	
	FUNKTION 3: ABSTAND BESTIMMEN

	EINGANEGE: I1 - CODEZEICHEN
	DIE CODEZEICHEN WERDEN BENOETIGT UM FESTZUSTELLEN OB DASSELBE ZIEL WIE FUER DEN VORGAENGER VORLIEGT.
A	
L	KONTROLL- C1 - VORGAENGERZIEL
L	GROESSEN: DIE INFORMATION, OB DASSELBE ZIEL VORLIEGT, DIENT ZUR ABSTANDSBESTIMMUNG.
R	AUSGAENGE: O1 - VORGAENGERZIEL
I	INFORMATION, OB DASSELBE ZIEL WIE FUER DEN VORGANEGER VORLIEGT ODER NICHT.
G	O2 - ABSTANDSMELDUNG
H	INFORMATION, OB KLEINER ODER GROSSER ABSTAND.
T	
S	MECHANISMEN: KEINE

R	
E	
S	
E	
R	
V	
E	
D	

NODE: PAKET / A1	TITLE: CODE LESEN PFEILBESCHREIBUNG A1	NUMBER: 2

USED AT:

AUTHOR: DR.THIELMANN DATE: 12.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

WORKING | READER | DATE
DRAFT
RECOMMEND
PUBLIC.

CONTEXT:
FKT.1 VON A0

FUNKTION 4: PAKETE FREIGEBEN

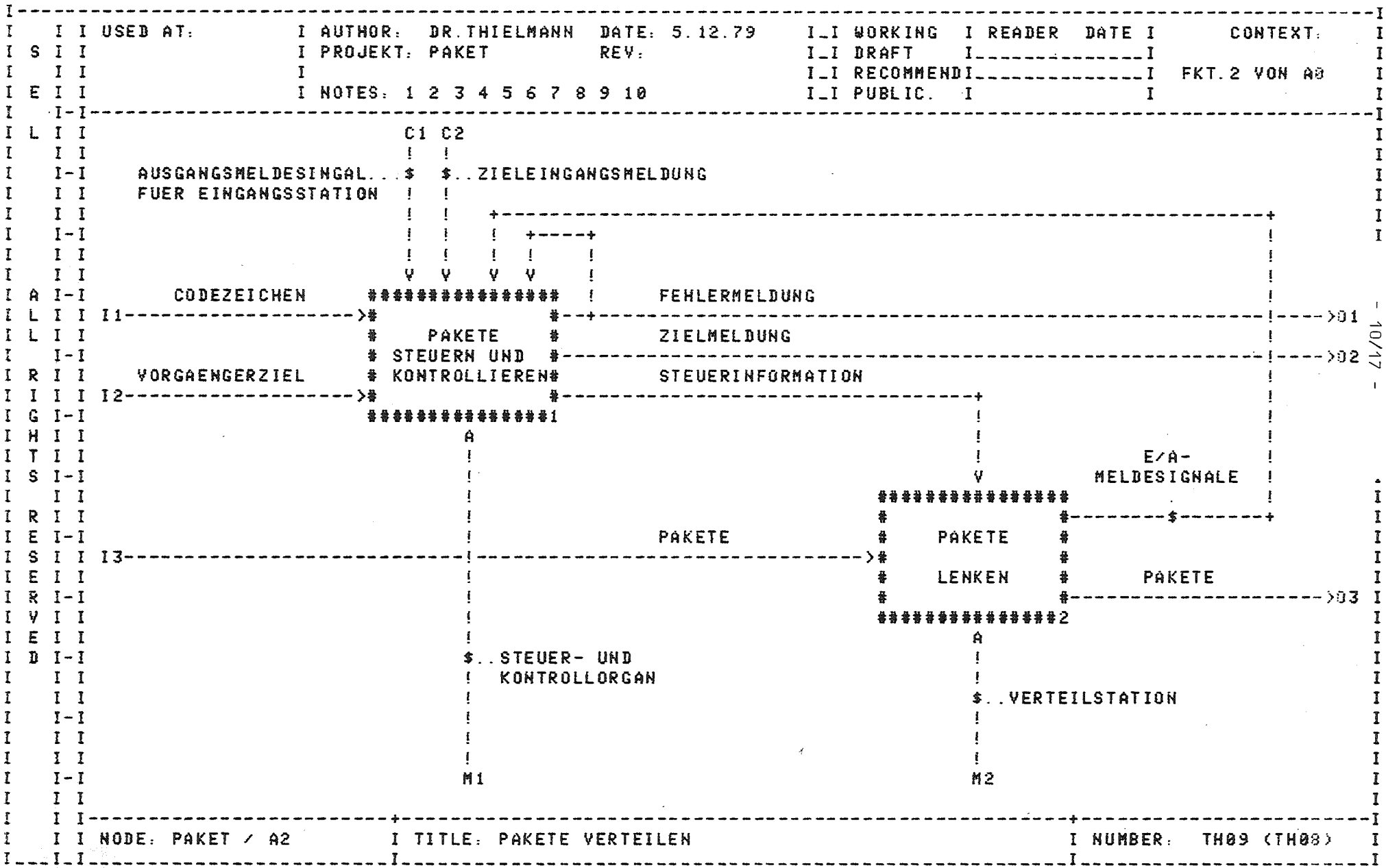
EINGAENGE: I1 - PAKETE

KONTROLL-GROESSEN:
C1 - ABSTANDSMELDUNG, SIEHE O2 VON FKT.3
C2 - TAKTFREQUENZ
DIE TAKTFREQUENZ WIRD BENOETIGT UM DEN ZEITLICHEN ABSTAND FESTZULEGEN, IN DEM DIE PAKETE DIE EINGANGSSTATION VERLASSEN.

AUSGAENGE:
O1 - AUSGANGSMELDUNG
DER PAKETAUSGANG ODER DIE PAKETFREIGABE WIRD DER VERTEILEINHEIT MITGETEILT
O2 - PAKETE

MECHANISMEN:
M1 - TAKTGEBER
M2 - FREIGABEORGAN

NODE: PAKET / A1 | TITLE: CODE LESEN PFEILBESCHREIBUNG A1 | NUMBER: 3


USED AT:
AUTHOR: DR.THIELMANN DATE: 5.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING
DRAFT
RECOMMEND
PUBLIC.
READER DATE
CONTEXT:
FKT.2 VON A0
C1 C2
AUSGANGSMELDESINGAL...
FUER EINGANGSSTATION
ZIELEINGANGSMELDUNG
CODEZEICHEN
FEHLERMELDUNG
I1
01
PAKETE
STEUERN UND
KONTROLLIEREN
ZIELMELDUNG
02
VORGAENGERZIEL
STEUERINFORMATION
I2
E/A-
MELDESIGNALE
PAKETE
I3
PAKETE
LENKEN
PAKETE
03
STEUER- UND
KONTROLLORGAN
VERTEILSTATION
M1
M2
NODE: PAKET / A2
TITLE: PAKETE VERTEILEN
NUMBER: TH09 (TH08)

USED AT:
AUTHOR: DR.THIELMANN DATE: 11.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

WORKING DRAFT / RECOMMEND / PUBLIC. READER DATE

CONTEXT: FKT.2 VON A0

SLEITRAGSHEFT

A2: PAKETE VERTEILEN

DIE FUNKTION 'PAKETE VERTEILEN' WIRD HIER IN 'SOFTWARE'- UND 'HARDWARE'-FUNKTIONEN AUFGESPLITTET.

-A21: PAKETE STEUERN UND KONTROLLIEREN
DIE GEMELDETEN CODEZEICHEN WERDEN IN STEUERINFORMATIONEN FUER DIE LENKORGANE UMGEWANDELT UND DIESEN MITGETEILT. DER DURCHLAUF DER PAKETE WIRD MIT HILFE DER MELDUNGEN DER EIN- UND AUSGANGSMELDEORGANE DER EINZELNEN STATIONEN KONTROLLIERT.

-A22: PAKETE LENKEN
DIE PAKETE WERDEN DURCH DIE BAUMSTRUKTUR DER VERTEILSTATIONEN GESCHLEUSST.

NODE: PAKET / A2 TITLE: PAKETE VERTEILEN FUNKTIONSBESCHREIBUNG A2 NUMBER: 1

USED AT:	AUTHOR: DR.THIELMANN DATE: 11.12.79	WORKING	READER DATE	CONTEXT:
	PROJEKT: PAKET REV:	DRAFT		
		RECOMMEND		FKT.2 VON A0
	NOTES: 1 2 3 4 5 6 7 8 9 10	PUBLIC.		

SEITE
ALL RIGHTS RESERVED

FUNKTION 1: PAKETE STEUERN UND KONTROLLIEREN

EINGANEGE:

I1 - CODEZEICHEN
DIE CODEZEICHEN WERDEN ZUR STEUERUNG DER LENKORGANE VERWENDET
I2 - VORGANEGERZIEL
ANGABE, OB DASSELBE ZIEL FUER DAS VORGAENGERPAKET VORLIEGT ODER NICHT.

KONTROLL-GROESSEN:

C1 - AUSGANGSMELDESIGNAL DER EINGANGSSTATION
DEM STEUERORGAN WIRD MITGETEILT, DASS EIN PAKET DIE EINGANGSSTATION VERLAESST UND SOMIT DURCH DIE VERTEILEINHEIT GESCHLEUSST WERDEN MUSS.

C2 - ZIELEINGANGSMELDUNG
DIE ZIELSTATIONEN MELDEN JEDEN PAKETEINGANG. DAMIT WIRD DAS BETREFFENDE PAKET AUS DER AKTUELLEN BUCHHALTUNG GELOESCHT.

C3 - EIN-AUSGANGSMELDESIGNAL
PAKETEIN- UND -AUSGANEGE BEI DEN VERTEILSTATIONEN WERDEN AN DAS STEUERORGAN GEMELDET, UM DIE DURCHLAUFKONTROLLE DER PAKETE ZU GEWAEHRLEISTEN.

C4 - FEHLERMELDUNGEN
WAEHREND DES VERTEILVORGANGES AUFTRETENDE FEHLER WERDEN IN DIE STEUERUNG MIT EINBEZOGEN, UM EINEN MOEGLICHST REIBUNGSLOSEN WEITERLAUF ZU GEWAEHRLEISTEN.

AUSGAENGE:

O1 - FEHLERMELDUNGEN
FEHLER WERDEN PROTOKOLLIERT UND IN DER WEITEREN STEUERUNG BERUECKSICHTIGT. FEHLLAEUFER WERDEN DER BETREFFENDEN ZIELSTATION ALS SOLCHE GEMELDET.

- 10/19 -

NODE: PAKET / A2	TITLE: PAKETE VERTEILEN PFEILBESCHREIGUNG A2	NUMBER: 1

USED AT:	AUTHOR: DR.THIELMANN DATE: 11.12.79	WORKING	READER DATE	CONTEXT.
	PROJEKT: PAKET REV:	DRAFT		
		RECOMMEND		FKT.2 VON A0
	NOTES: 1 2 3 4 5 6 7 8 9 10	PUBLIC.		

02 - ZIELMELDUNG
DER BETREFFENDEN ZIELSTATION WIRD MITGETEILT, DASS EIN PAKET ANKOMMT.

03 - STEUERINFORMATION
DEN LENKORGANEN WIRD DIE STEUERINFORMATION UEBERMITTELT.

MECHANISMEN: M1 - STEUER- UND KONTROLLORGAN
DAS STEUER- UND KONTROLLORGAN WIRD AUS EINEM RECHNER BESTEHEN.

FUNKTION 2: PAKETE LENKEN

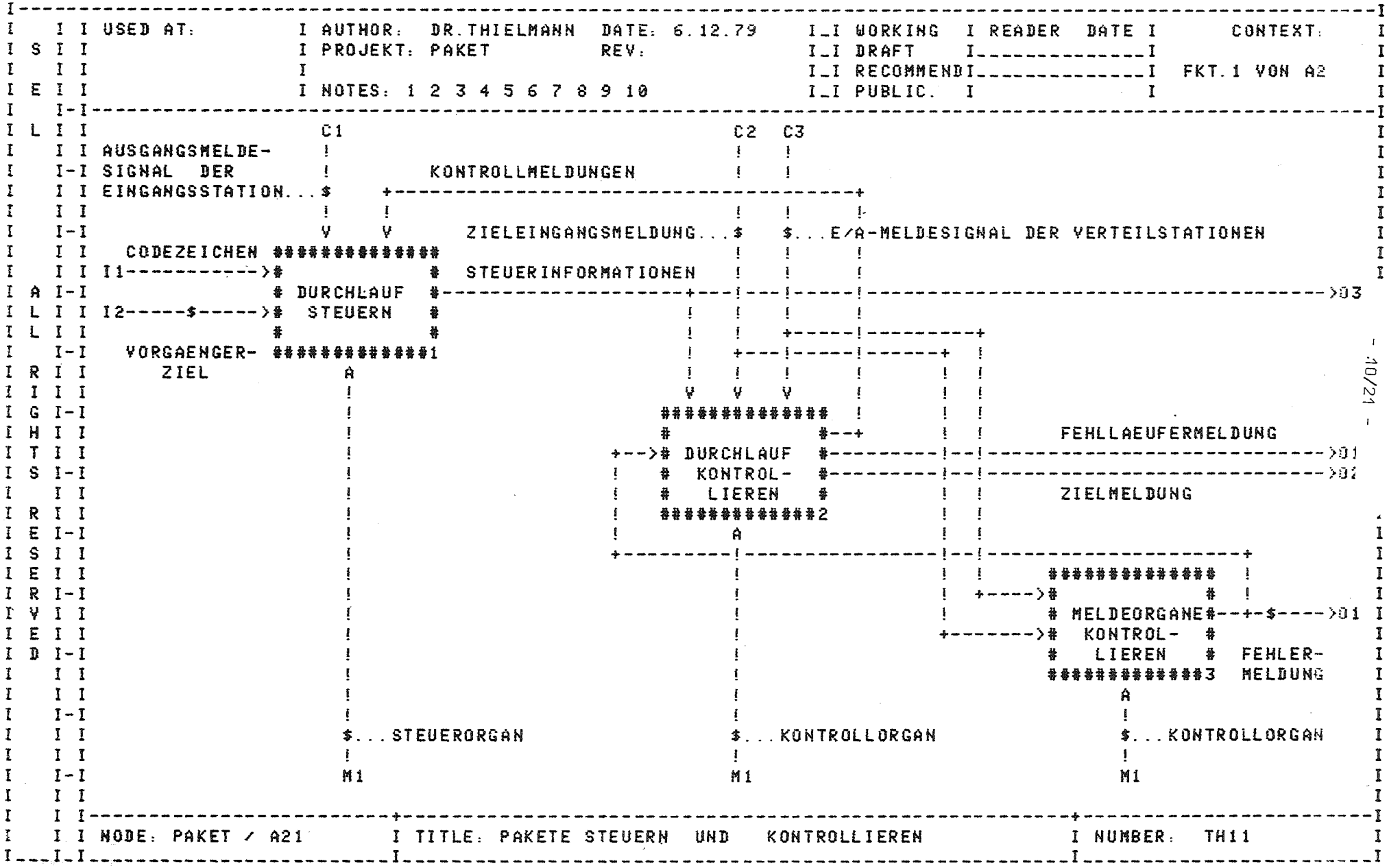
EINGAENGE: I1 - PAKETE

KONTROLL-GROESSEN: C1 - STEUERINFORMATION
MIT DER STEUERINFORMATION WIRD DEN LENKORGANEN DIE ENTSPRECHENDE WEICHENSTELLUNG MITGETEILT.

AUSGANEGE: O1 - EIN-AUSGANGSMELDESIGNALE
DIE EIN- UND AUSGANEGE WERDEN VON JEDER VERTEILSTATION AN DAS STEUER-ORGAN GEMELDET.
O2 - PAKETE

MECHANISMEN: M1 - VERTEILSTATIONEN
DIESER PFEIL UMFASST DIE GESAMTE BAUMSTRUKTUR DER VERTEILSTATIONEN.
JEDE VERTEILSTATION BESTEHT AUS EINEM EINGANGSMELDEORGAN, EINER WEICHE, DIE DIE PAKETE NACH RECHTS ODER LINKS LENKT, UND AUS ZWEI AUSGANGS-MELDEORGANEN.

NODE: PAKET / A2	TITLE: PAKETE VERTEILEN PFEILBESCHREIGUNG A2	NUMBER: 2


USED AT:
AUTHOR: DR.THIELMANN DATE: 6.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT RECOMMEND PUBLIC.
READER DATE
CONTEXT:
FKT.1 VON A2
AUSGANGSMELDE-SIGNAL DER EINGANGSSTATION...
KONTROLLMELDUNGEN
ZIELEINGANGSMELDUNG...
E/A-MELDESIGNAL DER VERTEILSTATIONEN
CODEZEICHEN
STEUERINFORMATIONEN
DURCHLAUF STEUERN
VORGAENGER-ZIEL
DURCHLAUF KONTROL-LIEREN
FEHLLAEUFERMELDUNG
ZIELMELDUNG
MELDEORGANE KONTROL-LIEREN
FEHLER-MELDUNG
STEUERORGAN
KONTROLLORGAN
KONTROLLORGAN
NODE: PAKET / A21
TITLE: PAKETE STEUERN UND KONTROLLIEREN
NUMBER: TH11

USED AT:
AUTHOR: DR.THIELMANN DATE: 13.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT RECOMMEND PUBLIC.
READER DATE
CONTEXT
FKT.1 VON A2

S E L A L L R I G H T S R E S E R V E D

A21: PAKETE STEUERN UND KONTROLLIEREN

DIE GESAMTE STEUERFUNKTION WIRD IN DIE FUNKTIONEN 'DURCHLAUF STEUERN', 'DURCHLAUF KONTROLLIEREN' UND 'MELDEORGANE KONTROLLIEREN' AUFGETEILT.

-A211: DURCHLAUF STEUERN
DURCH DIE AUSGANGSMELDUNG DER EINGANGSSTATION UND DEM UEBERMITTELTEN CODEZEICHEN FINDET EINE EINDEUTIGE ZUORDNUNG ZWISCHEN PAKET UND CODEZEICHEN STATT.DAS CODEZEICHEN WIRD IN DIE EIGENTLICHE STEUERINFORMATION FUER DIE LENKORGANE UMGEWANDELT. DIESE STEUERINFORMATION WIRD AN DIE EINZELNEN VERTEILSTUFEN IM TAKT MIT DEN KONTROLLMELDUNGEN, DIE DIE MOMENTANE LAGE DES PAKETES BEINHALTEN, WEITERGEGEBEN. FALLS DASSELBE ZIEL WIE FUER DAS VORGAENGERPAKET VORLIEGT,WIRD KEINE ERNEUTE STEUERINFORMATION HERAUSGEGEBEN.
FUER DEN FALL, DASS DAS PAKET BEREITS IN DER EINGANGSSTATION ALS FEHLLAEUFER KLASSIFIZIERT WURDE, ALSO MIT DEM DAFUER BESTIMMTEN CODE (Z.B. 000000) GEKENNZEICHNET IST,WIRD KEINE NEUE STEUERINFORMATION AN DIE VERTEILSTATIONEN ABGEGEBEN. JEDOCH WIRD DEM KONTROLLORGAN MITGETEILT, DASS ES SICH UM EINEN FEHLLAEUFER HANDELT.

-A212: DURCHLAUF KONTROLLIEREN
DER DURCHLAUF VON JEDEM PAKET DURCH JEDE VERTEILSTATION WIRD ANHAND DER EIN- UND AUSGANGSMELDUNGEN MIT DER STEUERINFORMATION VERGLICHEN UND SOMIT KONTROLLIERT.
SOWOHL DER RICHTIGE, WIE AUCH DER FALSCHE DURCHGANG WIRD DEM STEUERZENTRUM MITGETEILT (KONTROLLMELDUNGEN). IM FALLE EINER FEHLSTEUERUNG UNTERBLEIBEN DIE STEUERINFORMATIONEN VOM STEUERORGAN. ES WIRD NUR DER WEITERE VERLAUF KONTROLLIERT UND DER BETREFFENDEN ZIELSTATION MITGETEILT, DASS EIN FEHLLAEUFER ANKOMMT.

NODE: PAKET / A21
TITLE: STEUERN UND KONTROLL. FUNKTIONSBESCHREIBUNG A2
NUMBER: 1

USED AT:

AUTHOR: DR.THIELMANN DATE: 13.12.79 WORKING DRAFT RECOMMEND PUBLIC. READER DATE CONTEXT

PROJEKT: PAKET REV: FKT.1 VON A2

NOTES: 1 2 3 4 5 6 7 8 9 10

S E L L A L I R I G H T S R E S E R V E D

-A213: MELDEORGANE KONTROLLIEREN

DIE EIN- UND AUSGANGSMELDUNGEN WERDEN LAUFEND AUF VOLLSTAENDIGKEIT KONTROLLIERT, UM DEN AUSFALL EINES MELDEORGANS SOFORT FESTZUSTELLEN.

NODE: PAKET / A21 TITLE: STEUERN UND KONTROLL. FUNKTIONSBESCHREIBUNG A2 NUMBER: 2

USED AT:
AUTHOR: DR.THIELMANN DATE: 13.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT / RECOMMEND / PUBLIC.
READER DATE
CONTEXT
FKT.1 VON A2

S E L A L L R I G H T S R E S E R V E D

FUNKTION 1: DURCHLAUF STEUERN

EINGAENGE:

I1 - CODEZEICHEN
DIE CODEZEICHEN WERDEN IN STEUERINFORMATIONEN FUER DIE LENKORGANE UMGEWANDELT.

I2 - VORGAENGERZIEL
FALLS DASSELBE ZIEL WIE FUER DEN VORGAENGER VORLIEGT, WERDEN DEN LENKORGANEN FUER DEN FOLGENDEN DURCHLAUF KEINE STEUERINFORMATIONEN UEBERMITTELT.

KONTROLL-GROESSEN:

C1 - AUSGANGSMELDUNG DER EINGANGSSTATION
DAMIT WIRD DAS CODEZEICHEN MIT DEM RICHTIGEN PAKET SYNCHRONISIERT.

C2 - KONTROLLMELDUNGEN
DIE KONTROLLMELDUNGEN BEINHALTEN DIE AKTUELLE LAGE DES PAKETES IM BAUM DER VERTEILSTATIONEN. FALLS EIN PAKET FEHLGELEITET WURDE, MUESSEN DIE STEUERINFORMATIONEN AKTUALISIERT WERDEN.

AUSGAENGE:

O1 - STEUERINFORMATION
DIE STEUERINFORMATIONEN WERDEN SOWOHL DIREKT DEN VERTEILSTATIONEN, ALS AUCH DEM KONTROLLORGAN UEBERMITTELT.

O2 - FEHLLAEUFERMELDUNG
DIE TATSACHE, DASS EIN FEHLLAEUFER VORLIEGT, WIRD DEM KONTROLLORGAN GEMELDET.

MECHANISMEN:

M1 - STEUERORGAN

NODE: PAKET / A21
TITLE: STEUERN UND KONTROLL. PFEILBESCHREIBUNG A21
NUMBER: 1

USED AT:	AUTHOR: DR.THIELMANN DATE: 13.12.79 PROJEKT: PAKET REV: NOTES: 1 2 3 4 5 6 7 8 9 10	I_I WORKING I_I DRAFT I_I RECOMMEND I_I PUBLIC.	READER DATE

CONTEXT

FKT.1 VON A2

FUNKTION 2: DURCHLAUF KONTROLLIEREN

EINGAENGE:

I1 - VORGAENGERZIEL
BEI GLEICHEN ZIELEN WIE FUER DAS VORGAENGERPAKET ERHAELT DAS KONTROLLORGAN KEINE NEUE STEUERINFORMATION. ANHAND DER EIN- UND AUSGANGSMELDUNGEN DER VERTEILSTATIONEN WIRD LEDIGLICH DER WIEDERHOLTE DURCHGANG KONTROLLIERT.

I2 - FEHLERMELDUNG
FALLS EINES DER EIN- UND AUSGANGSMELDEORGANE EINER VERTEILSTATION AUS-FAELLT, WIRD DIES DER KONTROLLFUNKTION GEMELDET, DAMIT DIESE NICHT FAELSCHLICHERWEISE EINE FEHLSTEUERUNG MELDET.

KONTROLL-GROESSEN:

C1 - FEHLLAEUFERMELDUNG
PAKETE, DIE KEINER ZIELSTATION ZUZUORDNEN SIND, WERDEN ALS FEHLLAEUFER GEMELDET.DAS KONTROLLORGAN UEBERWACHT DANN NUR DEN DURCHLAUF.

C2 - STEUERINFORMATION
ANHAND DER STEUERINFORMATION MUSS DER DURCHLAUF KONTROLLIERT WERDEN

C3 - ZIELEINGANGSMELDUNG
WENN EIN PAKET EINE ZIELSTATION ERREICHT HAT, WIRD DIE DURCHLAUF KONTROLLE BEENDET.

C4 - EIN- UND AUSGANGSSIGNALE DER VERTEILSTATIONEN
DIESE SIGNALE WERDEN ZUR DURCHLAUFKONTROLLE VERWENDET.

AUSGAENGE:

O1 - KONTROLLMELDUNGEN
DIE KONTROLLMELDUNGEN BEINHALTEN DIE INFORMATION, OB DER DURCHLAUF RICHTIG ODER FALSCH ERFOLGT IST.

O2 - FEHLLAEUFERMELDUNG
FALLS EINE FEHLLEITUNG AUFTRITT, WIRD DIESE PROTOKOLLIERT UND DER BETREF-FENDEN ZIELSTATION ALS SOLCHE ANGEKUENDIGT.

O3 - ZIELMELDUNG
DER ZIELSTATION WIRD EIN RICHTIG GELEITETES PAKET ALS SOLCHES GEMELDET.

- 10/25 -

NODE: PAKET / A21	TITLE: STEUERN UND KONTROLL. PFEILBESCHREIBUNG A21	NUMBER: 2

USED AT: AUTHOR: DR.THIELMANN DATE: 13.12.79 WORKING DRAFT RECOMMEND PUBLIC. READER DATE CONTEXT.
PROJEKT: PAKET REV: NOTES: 1 2 3 4 5 6 7 8 9 10 FKT.1 VON A2

SEL ALLE ALLGEMEIN ERSTELLEN VERIFIED

MECHANISMEN: M1 - KONTROLLORGAN

FUNKTION 3: MELDEORGANE KONTROLLIEREN

EINGAENGE: I1 - ZIELEINGANGSMELDUNG
I2 - EIN- UND AUSGANGSMELDESIGNALE DER VERTEILSTATIONEN

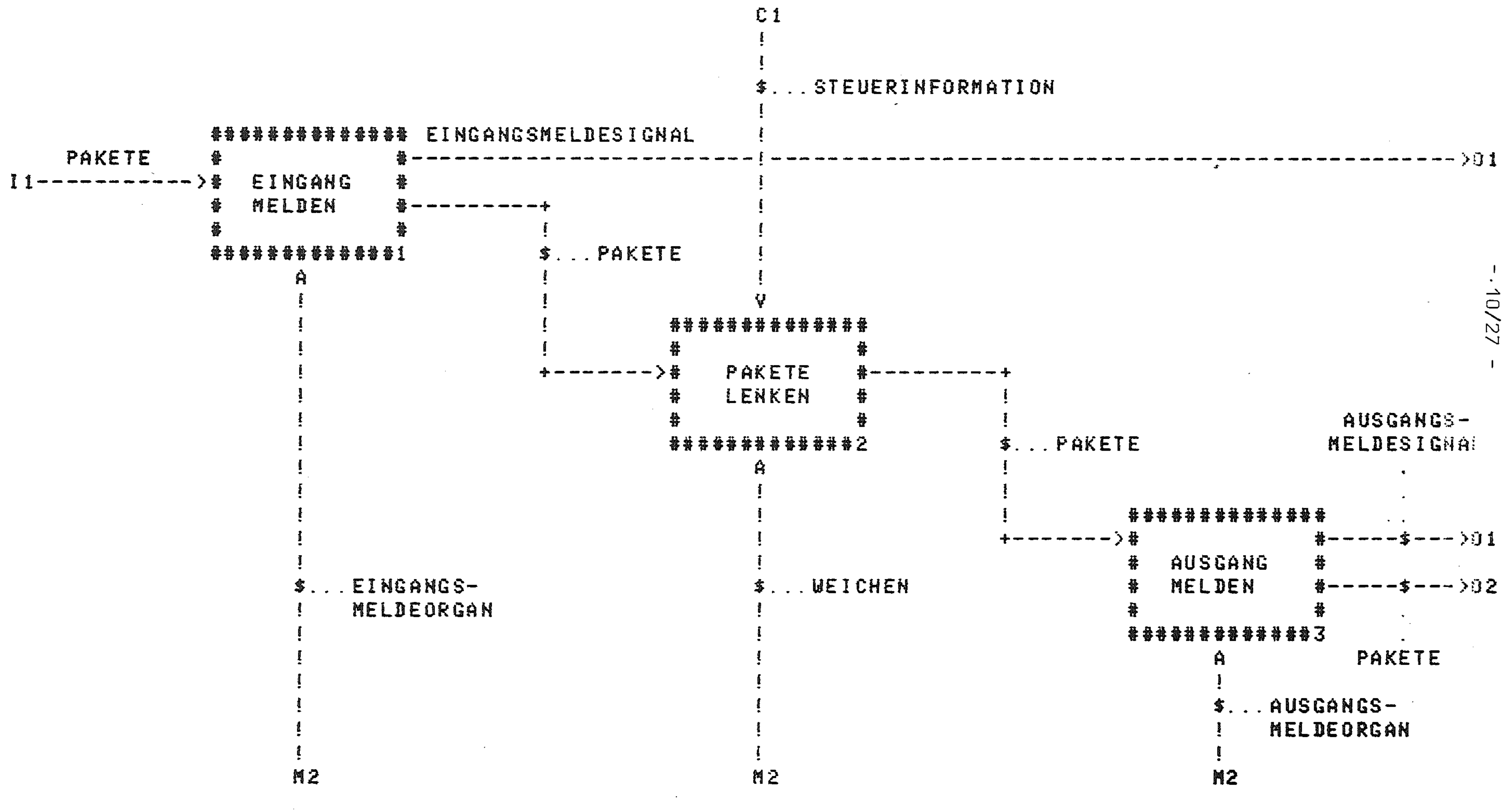
AUSGAENGE: O1 - FEHLERMELDUNGEN
FEHLER DER MELDEORGANE WERDEN PROTOKOLLIERT UND DEM KONTROLLORGAN MITGETEILT.

MECHANISMEN: M1 - KONTROLLORGAN

NODE: PAKET / A21 TITLE: STEUERN UND KONTROLL. PFEILBESCHREIBUNG A21 NUMBER: 3

SEE ALL RIGHTS RESERVED

USED AT:	AUTHOR: DR.THIELMANN DATE: 6.12.79 PROJEKT: PAKET REV: NOTES: 1 2 3 4 5 6 7 8 9 10	WORKING / READER DATE DRAFT RECOMMEND PUBLIC.	CONTEXT: FKT.2 VON A2



NODE: PAKET / A22	TITLE: PAKETE LENKEN	NUMBER: TH10

USED AT:
AUTHOR: DR.THIELMANN DATE: 13.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT RECOMMEND PUBLIC.
READER DATE
CONTEXT: FKT.2 VON A2

SLEIS

LALLE

ARBEITSREIHENFOLGE

A22: PAKETE LENKEN

DIE FUNKTION 'PAKETE LENKEN' BEINHALTET DEN EIGENTLICHEN VERTEILVORGANG IN DEN EINZELNEN VERTEILSTATIONEN. DIE FUNKTIONEN, DIE INNERHALB JEDER VERTEILSTATION AUFTRETEN, WERDEN HIER EINZELN AUFGEFUEHRT.

-A221: EINGANG MELDEN
DER EINGANG IN EINE VERTEILSTATION WIRD DEM KONTROLLORGAN GEMELDET.

-A222: PAKETE LENKEN
DIE PAKETE WERDEN IN JEDER VERTEILSTATION ENTWEDER NACH RECHTS ODER LINKS GELENKT.

-A223: AUSGANG MELDEN
VON DEMJENIGEN DER BEIDEN AUSGAENGE, DURCH DEN DAS PAKET DIE VERTEILSTATION VERLAESST, GEHT EINE AUSGANGSMELDUNG AN DAS KONTROLLORGAN.

NODE: PAKET / A22 TITLE: PAKETE LENKEN FUNKTIONSBESCHREIBUNG NUMBER: 1

	I	USED AT:	AUTHOR: DR.THIELMANN	DATE: 13.12.79	WORKING	READER DATE	CONTEXT.
S	I		PROJEKT: PAKET	REV:	DRAFT		
	I				RECOMMEND		FKT.2 VON A2
E	I		NOTES: 1 2 3 4 5 6 7 8 9 10		PUBLIC.		

ALLGEMEINE SICHT

FUNKTION 1: EINGANG MELDEN

EINGAENGE: I1 - PAKETE

AUSGAENGE: O1 - EINGANGSMELDESIGNAL

MECHANISMEN: M1 - EINGANGSMELDEORGAN

FUNKTION 2: PAKETE LENKEN

EINGAENGE: I1 - PAKETE

KONTROLL-GROESSEN: C1 - STEUERINFORMATION
ENTSPRECHEND DER STEUERINFORMATION WIRD DIE WEICHE GESTELLT.

AUSGAENGE: O1 - GELENKTE PAKETE

MECHANISMEN: M1 - WEICHEN

FUNKTION 3: AUSGANG MELDEN

EINGAENGE: I1 - GELENKTE PAKETE

AUSGAENGE: O1 - AUSGANGSMELDESIGNAL
O2 - PAKETE

MECHANISMEN: M1 - AUSGANGSMELDEORGAN

NODE: PAKET / A22	TITLE: PAKETE LENKEN	PFEILBESCHREIBUNG A22 NUMBER: 1

FORMULARBEARBEITUNG "FORMUL" V-NR. 2   17-JAN-80 11:28:51 SEITE: 27
DATEI-NAME: PAKET.DKM;1

```
I----------------------------------------------------------------------------------------------------------------I
I   I I USED AT:               I AUTHOR:  DR.THIELMANN  DATE: 4.12.79    I_I WORKING  I READER  DATE I      CONTEXT:    I
I S I I                        I PROJEKT: PAKET         REV:             I_I DRAFT    I_____________I                   I
I   I I                        I                                         I_I RECOMMENDI_____________I  FKT.3 VON A0     I
I E I I                        I NOTES: 1 2 3 4 5 6 7 8 9 10             I_I PUBLIC.  I             I                   I
I   I-I----------------------------------------------------------------------------------------------------------I
I L I I                                                                       C1  C2                             I
I   I I                                                                       !   !                              I
I   I-I                                                                       !   !                              I
I   I I                                                                       !   !                              I
I   I I                                                         ZIELMELDUNG...$   $...FEHLLAEUFERMELDUNG         I
I   I-I                                                                       !   !                              I
I   I I                                                                       !   !                              I
I   I I                                                                       !   !                              I
I A I-I                       ################                                !   !                              I
I L I I                       #              #    ZIELEINGANGSMELDUNG         !   !
I L I I       PAKETE          #   ANKUNFT    #--------------------------------!---!-------------------------->01
I   I-I I1 ------------------>#              #                                !   !
I R I I                       #   MELDEN     #-------------+                  !   !
I I I I                       #              #             !                  !   !
I G I-I                       ###############1             !                  !   !
I H I I                               A                    !                  !   !
I T I I                               !                    $...PAKETE         !   !
I S I-I                               !                    !                  V   V
I   I I                               !                    !            ################    FEHLLAEUFER
I R I I                               !                    !            #              #--------------------->02
I E I-I                               !                    !            #  FEHLLAEUFER #
I S I I                               !                    +----------->#              #                         I
I E I I                               !                                 #  ABTRENNEN   #  ZIELRICHTIGE PAKETE    I
I R I-I                               !                                 #              #--------------------->03 I
I V I I                               !                                 ###############2                         I
I E I I                               !                                         A                                I
I D I-I                               !                                         !                                I
I   I I                               !                                         !                                I
I   I I                               !                                         !                                I
I   I-I                               $...MELDEORGAN                            $...WEICHE                       I
I   I I                               !                                         !                                I
I   I I                               !                                         !                                I
I   I-I                               M1                                        M1                               I
I   I I                                                                                                          I
I   I I-------------------------+-------------------------------------------------------+------------------------I
I   I I NODE: PAKET / A3        I TITLE: ZIELANKUNFT MELDEN                             I NUMBER:   TH12         I
I___I_I_________________________I_______________________________________________________I________________________I
```

- 10/30 -

USED AT:
AUTHOR: DR.THIELMANN DATE: 14.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT / RECOMMEND / PUBLIC.
READER DATE
CONTEXT: FKT.3 VON A0

S E L L A L R I G H T S R E S E R V E D

A3: ZIELANKUNFT MELDEN

DIE ZIELSTATIONEN MELDEN DIE ANKUNFT DER PAKETE UND TEILEN SIE AUFGRUND DER MELDUNGEN DES KONTROLLORGANS IN FEHLLAEUFER UND ZIELRICHTIG GELEITETE PAKETE EIN.

-A31: ANKUNFT MELDEN
DIE ANKUNFT DER PAKETE AN DER ZIELSTATION WIRD AN DAS STEUER- UND KONTROLL-ORGAN GEMELDET UM ES AUS DER BUCHFUEHRUNG ZU STREICHEN.

-A32: FEHLLAEUFER ABTRENNEN
IN DER ZIELSTATION WERDEN DIE FEHLGELEITETEN PAKETE VON DEN RICHTIG GELEITETEN PAKETEN GETRENNT.

NODE: PAKET / A3 TITLE: ZIELANKUNFT MELDEN FUNKTIONSBESCHREIBUNG A3 NUMBER: 1

USED AT:
AUTHOR: DR.THIELMANN DATE: 14.12.79
PROJEKT: PAKET REV:
NOTES: 1 2 3 4 5 6 7 8 9 10
WORKING DRAFT / RECOMMEND / PUBLIC. READER DATE
CONTEXT: FKT.3 VON A0

SICHT A1

FUNKTION 1: ANKUNFT MELDEN

EINGAENGE: I1 - PAKETE

AUSGAENGE: O1 - ZIELEINGANGSMELDUNG
DIE ZIELEINGANGSMELDUNG WIRD AN DAS STEUER- UND KONTROLLORGAN GEMELDET.
O2 - PAKETE

MECHANISMEN: M1 - MELDEORGAN FUER ZIELEINGANGSMELDUNG

FUNKTION 2: FEHLLAEUFER ABTRENNEN

EINGAENGE: I1 - PAKETE

KONTROLL-GROESSEN: C1 - ZIELMELDUNG
ZIELRICHTIGE PAKETE WERDEN ALS SOLCHE VOM KONTROLLORGAN ANGEKUENDIGT.

AUSGAENGE: O1 - FEHLLAEUFER
DIE FEHLGELEITETEN PAKETE WERDEN GESAMMELT.
O2 - ZIELRICHTIGE PAKETE
DIE ZIELRICHTIGEN PAKETE WERDEN GESAMMELT.

MECHANISMEN: M1 - WEICHE
DIE WEICHE DIENT ZUR TRENNUNG DER FEHLLAEUFER VON DEN ZIELRICHTIG GELEITETEN PAKETEN.

NODE: PAKET / A3 TITLE: ZIELANKUNFT MELDEN PFEILBESCHREIBUNG A3 NUMBER: 1

Hierarchy plus Input-Process-Output (HIPO)

Bearbeiter: Hans Keutgen, GEI

HIPO wurde als grafische Entwurfshilfe und Dokumentationstechnik von IBM entwickelt und ist eine Methode im Rahmen der Improved Programming Techniques.

HIPO-Kurzbeschreibung

Die Dokumentationstechnik dient zur Unterstützung

- des Designs,
- der Realisierung,
- der Wartung

komplexer Programmsysteme. Dabei wird besonders betont

- die hierarchische Struktur,
- die Beschreibung des Kontrollflusses,
- die Beschreibung des Datenflusses.

Als Beschreibungsmittel stehen zur Verfügung:

- Baumdiagramme,
- Ebenendiagramme,
  * Übersichtsdiagramme,
  * Detaildiagramme.

Das Baumdiagramm enthält Namen und Identifizierungsnummern aller Übersichts- und Detaildiagramme, die Funktionsblöcke enthalten. Es beschreibt den hierarchischen Aufbau und die Beziehung zwischen den logischen Einheiten.

Die Ebenendiagramme beschreiben die Abhängigkeiten zwischen dem Ein-/Ausgabeverhalten und der funktionellen Zergliederung des Systems. Sie haben einen einheitlichen Aufbau.

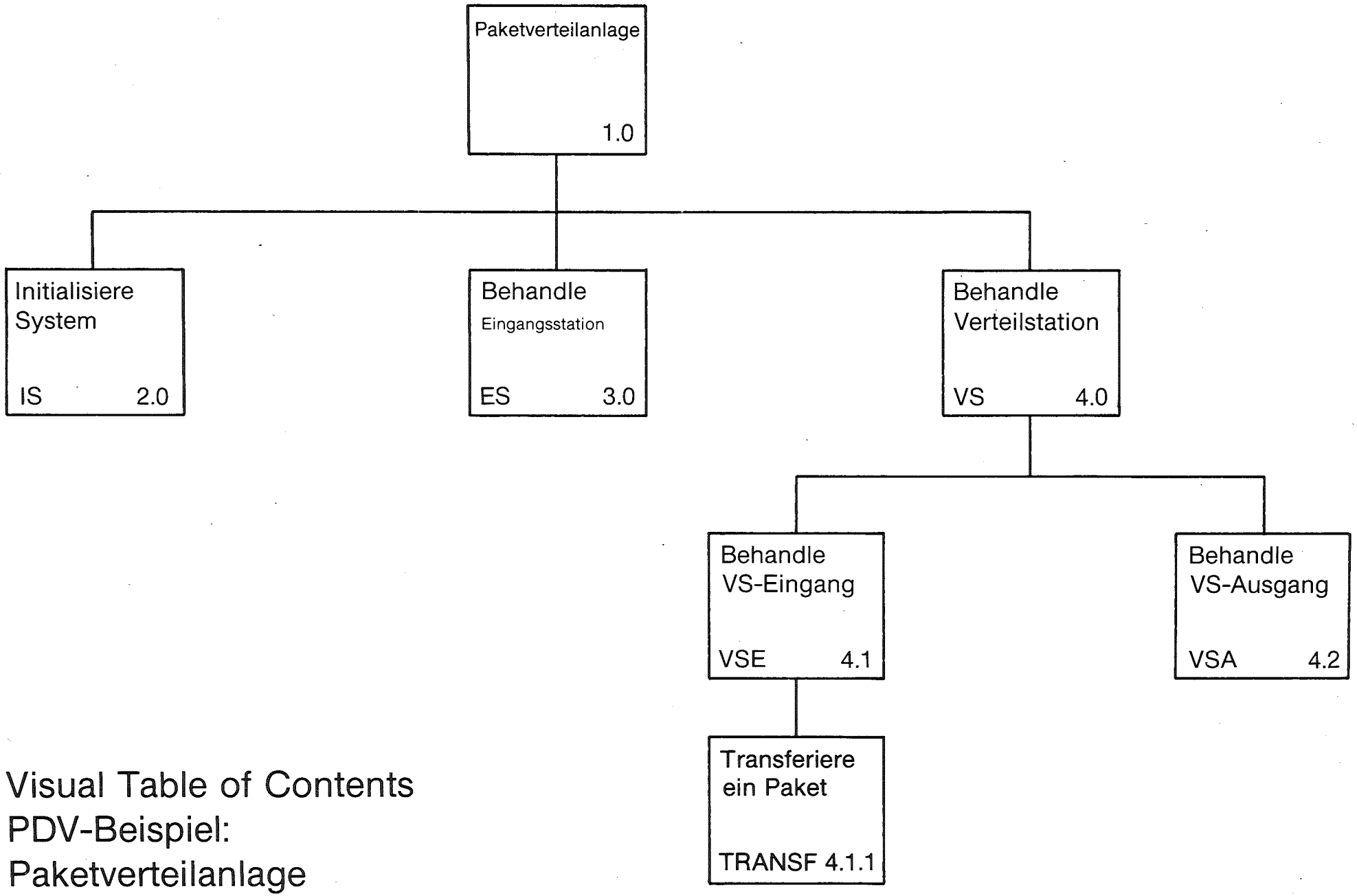
Die Übersichtsdiagramme beschreiben die Hauptfunktionen eines Systems und geben den Zusammenhang der untergeordneten Funktionen (Detaildiagramme) an.

Die Detaildiagramme beinhalten Teilfunktionen einer übergeordneten logischen Einheit, die in einem Übersichts- oder übergeordneten Detaildiagramm beschrieben sind.

Für jeden Diagrammtyp gibt es eine eingeführte Menge von Regeln, die deren Aufbau beschreiben. Dabei verwendete grafische Notationen sind im wesentlichen selbsterklärend. Als Hilfsmittel werden Formblätter und eine Schablone angeboten.

Literatur

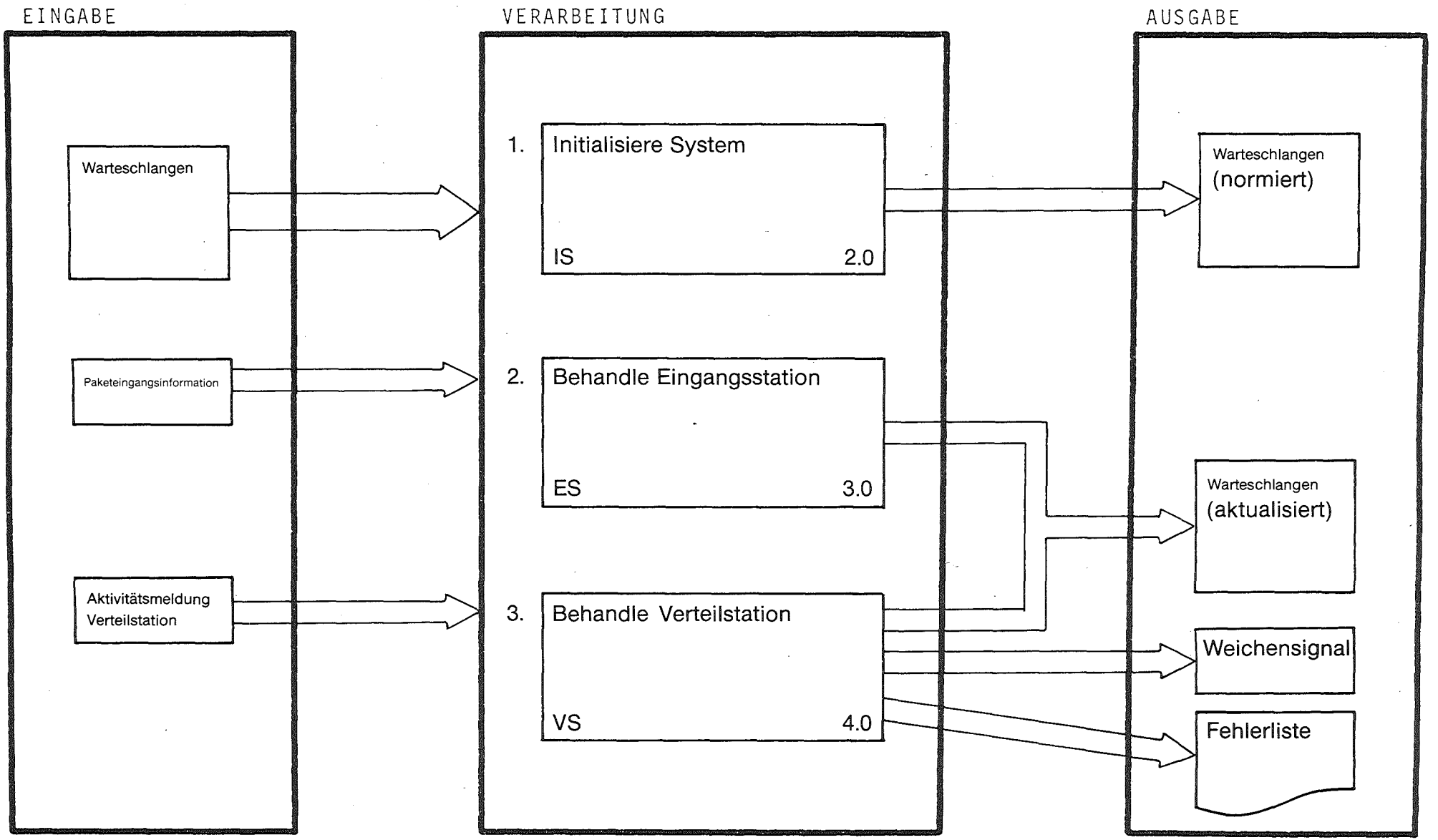
HIPO - A Design Aid and Documentation Technique;
IBM-Form GC20-1851



Visual Table of Contents
PDV-Beispiel:
Paketverteilanlage

Autor: H. Keutgen, GEI | System: Paketverteilanlage | Datum: 3. April 1980

Diagramm-Id: 1.0 | Systemteil/Programm:



Erweiterte Beschreibung zu 1.0

Der Verarbeitungsschritt 1. wird nur einmal vor Start der Verarbeitungsteile 2 und 3 durchlaufen. Die Schritte 2 und 3 können nach Verarbeitung des ersten Paketes in der Eingangsstation parallel ablaufen.

Erweiterte Beschreibung zu 2.0

Das Systemteil 'Initialisiere System' dient lediglich dem Löschen alter, möglicherweise noch vorhandener Warteschlangeneinträge. Zu diesem Zweck wird der Zähler der vorhandenen Einträge zurückgesetzt für alle Eingangs- und Durchlaufwarteschlangen aller Verteilstationen.

Erweiterte Beschreibung zu 3.0

Die Paketeingangsinformation besteht aus dem Interrupt Eingangsstation und dem Codezeichen. Das Eingangsdatum 'Ziel Vorläufer' entspricht dem Ausgabedatum 'Ziel für Nachfolger' aus dem vorigen Schritt.

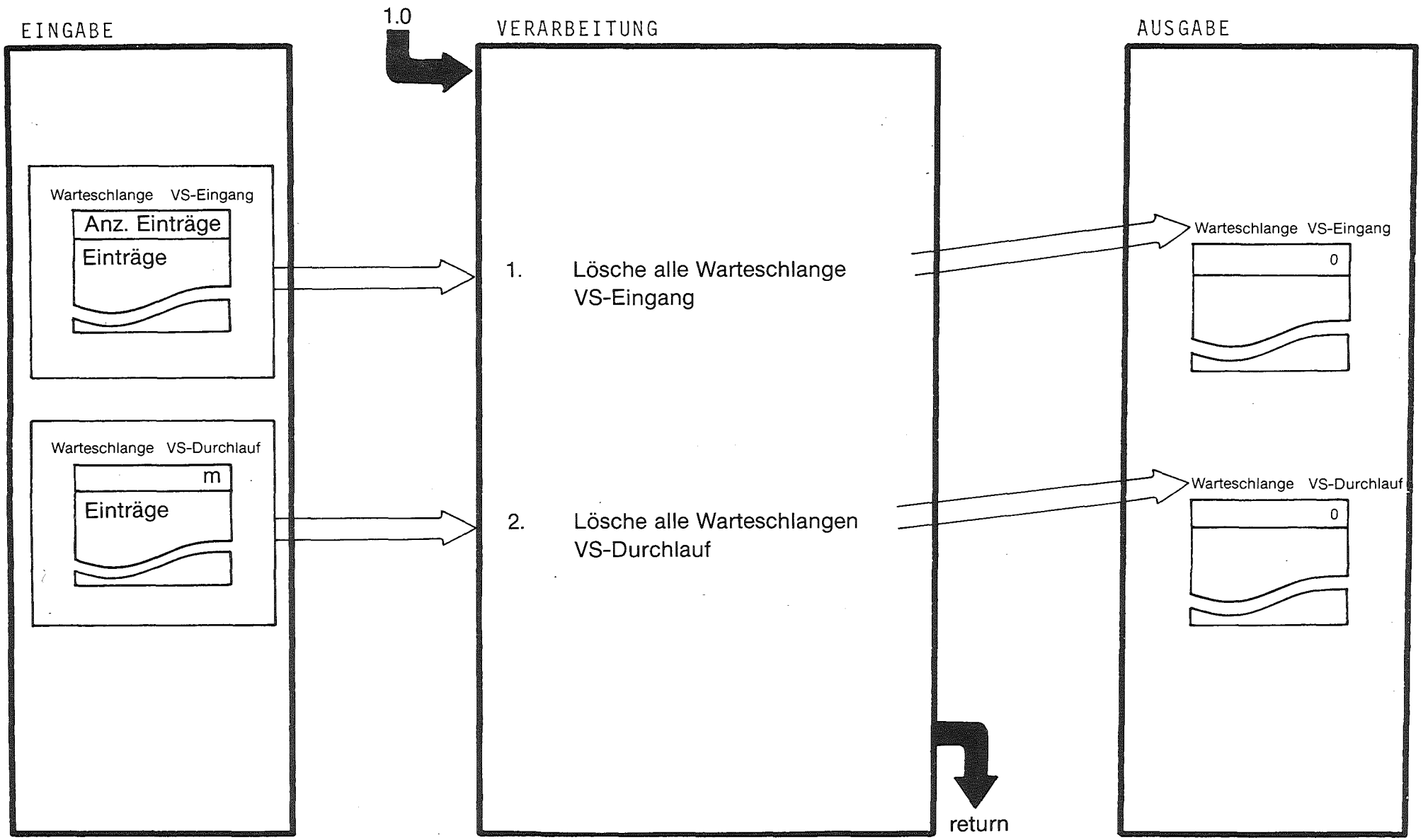
Der Verarbeitungsblock hat keinen eingehenden bzw. ausgehenden Kontrollfluß, da der Prozeß vom Interrupt getriggert anläuft und mit Abschluß des Verarbeitungsschrittes 7. beendet ist.

Das Ausgabedatum 'Paketinformation' entspricht einem Eintrag in den Warteschlangen. Dabei ist das Datum 'Richtungsvorgabe' für den Systemteil 'Behandle Eingangsstation' nicht von Bedeutung.

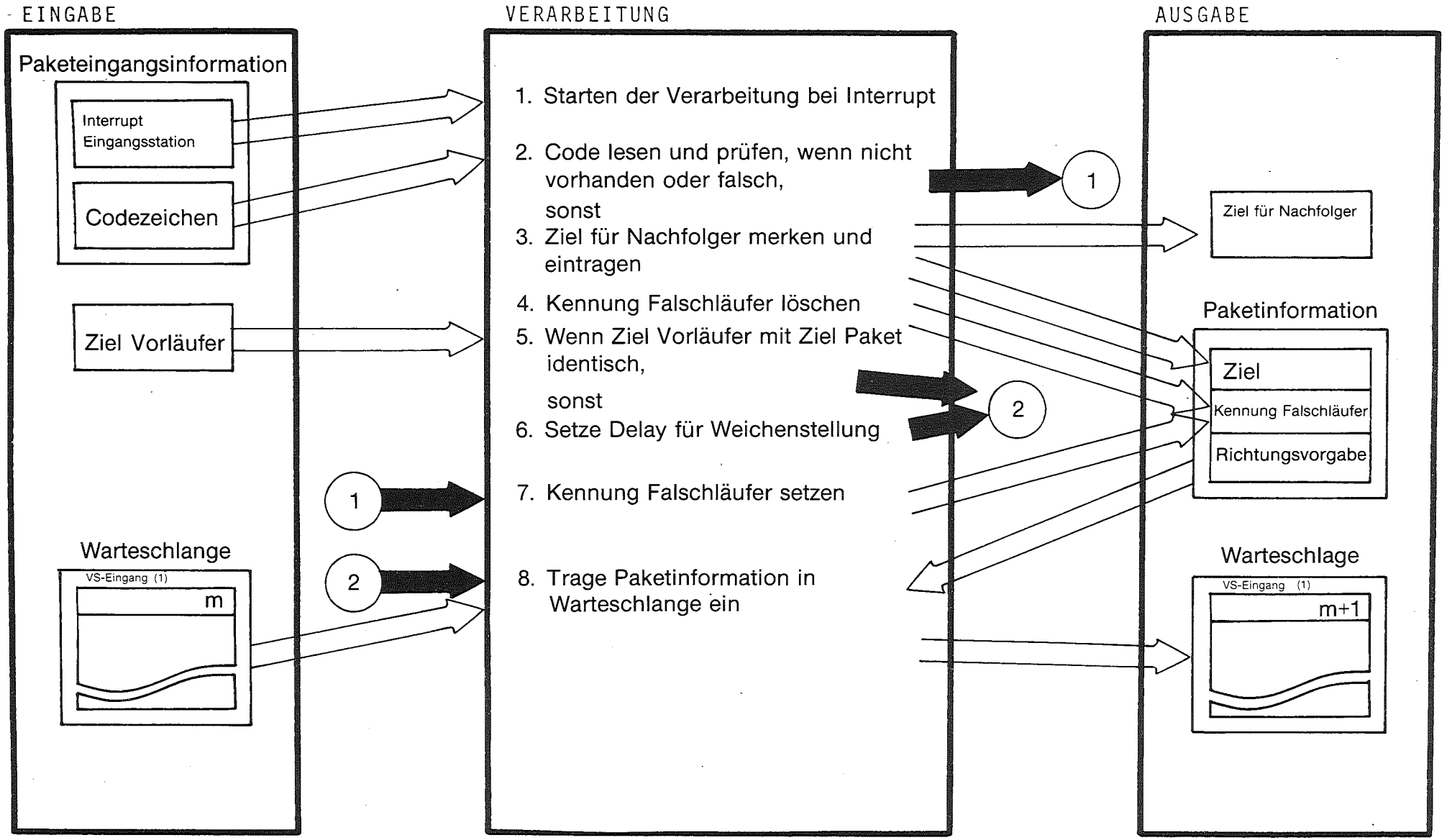
Autor: H. Keutgen, GEI
Diagramm-Id: 2.0

System: Paketverteilanlage
Systemteil/Programm: Initialisiere System /IS

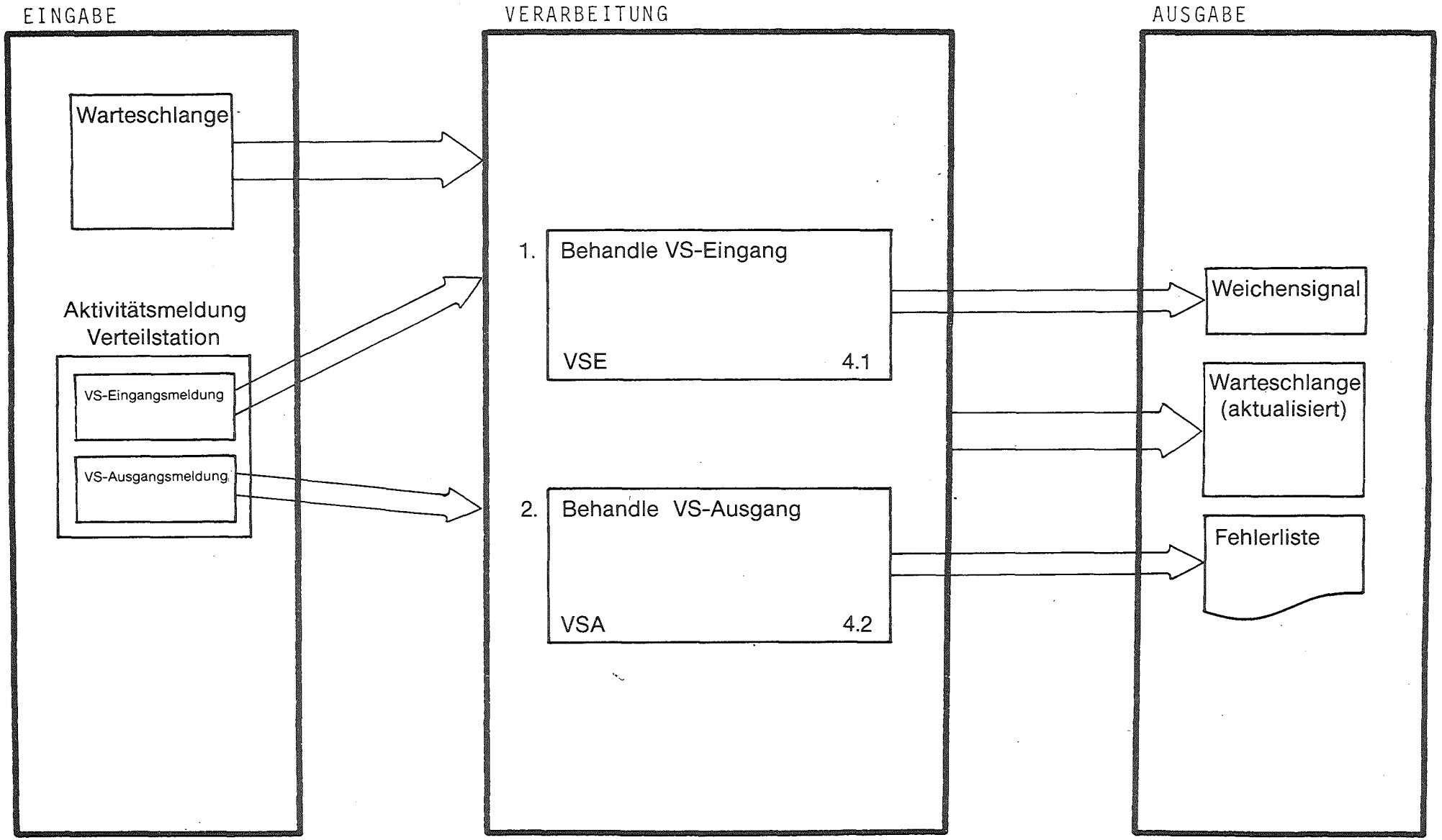
Datum: 3. April 1980



Autor: H. Keutgen, GEI
System: Paketverteilanlage
Datum: 3. April 1980
Diagramm-Id: 3.0
Systemteil/Programm: Behandle Eingangsstation /ES



Autor: H. Keutgen, GEI
System: Paketverteilanlage
Datum: 3. April 1980
Diagramm-Id: 4.0
Systemteil/Programm: Behandle Verteilstation /VS



Erweiterte Beschreibung zu 4.0

Das Eingabedatum 'Aktivitätsmeldung Verteilstation' besteht aus den Untermeldungen für Verteilstation-Eingang und -Ausgang.

Der Verarbeitungsblock hat keinen eingehenden bzw. ausgehenden Kontrollfluß, da der Prozeß vom Interrupt getriggert anläuft und nach Abarbeiten von allen Schritten aus 4.1 bzw. 4.2 beendet ist. Die Teile 4.1 und 4.2 sind als parallel ablauffähig anzusehen.

Erweiterte Beschreibung zu 4.1

Bei dem Eingabedatum 'VS-Eingangsmeldung' handelt es sich um den Eingangsinterrupt, der das Systemteil 'Behandle VS-Eingang' triggert.

Der Verarbeitungsblock hat keinen eingehenden Kontrollfluß, da er vom Interrupt 'VS-Eingangsmeldung' angestoßen wird. Er hat jedoch zwei Verarbeitungsschritte (3 und 6), an denen der Kontrollfluß ggfs. vorzeitig beendet wird.

Das Datum 'Paketinformation' entspricht einem Eintrag der Warteschlangen. Das 'Weichensignal' setzt die Verteilerweiche auf die vorbestimmte Richtung.

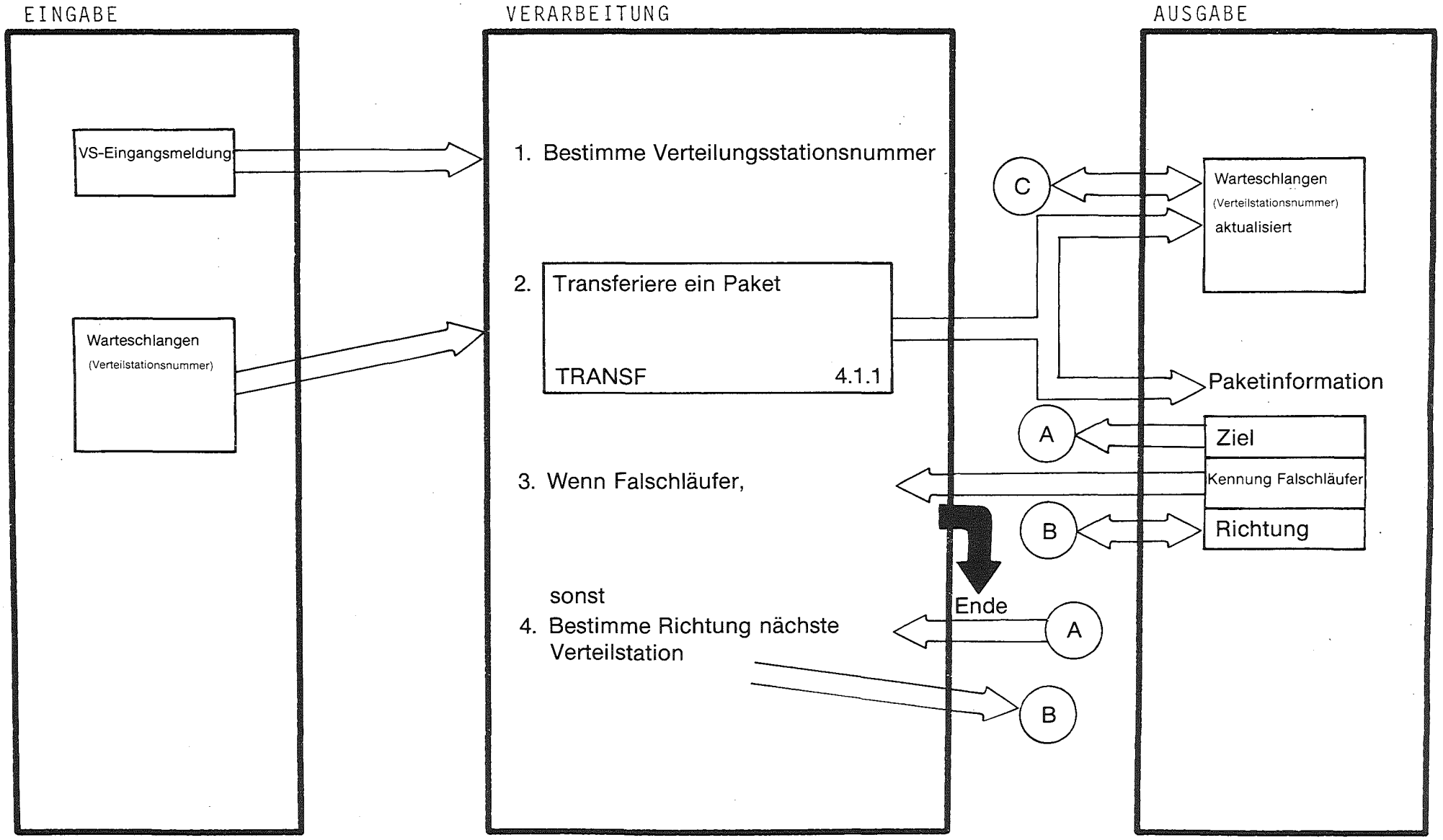
Erweiterte Beschreibung zu 4.2

Bei dem Eingabedatum 'VS-Ausgabemeldung' handelt es sich um die 'Interruptmeldung Verteilerstation-Ausgang' und um die zugehörige 'Richtungsanzeige'. Der Interrupt triggert den Systemteil 'Behandle VS-Ausgang'.

Der Verarbeitungsblock hat keinen Kontrollfluß, da er von 'Interruptmeldung Verteilerstation-Ausgang' angestoßen wird. Er hat jedoch zwei Verarbeitungsschritte (7 und 8), an denen der Kontrollfluß ggfs. beendet wird.

Autor: H. Keutgen, GEI | System: Paketverteilanlage | Datum: 3. April 1980

Diagramm-Id: 4.1 | Systemteil/Programm: Behandle VS-Eingang /VSE | Seite 1 von 2



Hommel, G. (Hrsg.)

Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage

Kernforschungszentrum Karlsruhe GmbH
PDV-Bericht, KfK-PDV 186, August 1980, Teil 2
335 Seiten

Verschiedene bekannte oder in der Entwicklung befindliche Spezifikationsverfahren werden an einem konkreten Beispiel, einer Paketverteilanlage erprobt. Die Beiträge entstanden im Rahmen des PDV-Arbeitskreises "Systematische Entwicklung von PDV-Systemen".

Hommel, G. (Hrsg.)

Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage

Kernforschungszentrum Karlsruhe GmbH
PDV-Bericht, KfK-PDV 186, August 1980, Teil 2
335 Seiten

Verschiedene bekannte oder in der Entwicklung befindliche Spezifikationsverfahren werden an einem konkreten Beispiel, einer Paketverteilanlage erprobt. Die Beiträge entstanden im Rahmen des PDV-Arbeitskreises "Systematische Entwicklung von PDV-Systemen".

Hommel, G. (Hrsg.)

Comparison of different specification methods using the example of a parcels distribution system

Kernforschungszentrum Karlsruhe GmbH
PDV-Report, KfK-PDV 186, August 1980, Part 2
335 pages

Various known or developing specification methods are used for the practical example of a parcels distribution system. The papers were contributed to a working group with the subject "Systematic development of process control systems".

Hommel, G. (Hrsg.)

Comparison of different specification methods using the example of a parcels distribution system

Kernforschungszentrum Karlsruhe GmbH
PDV-Report, KfK-PDV 186, August 1980, Part 2
335 pages

Various known or developing specification methods are used for the practical example of a parcels distribution system. The papers were contributed to a working group with the subject "Systematic development of process control systems".

Autor: H. Keutgen, GEI | System: Paketverteilanlage | Datum: 3. April 1980

Diagramm-Id: 4.1. | Systemteil/Programm: Behandle VS-Eingang /VSE | Seite 2 von 2

EINGABE

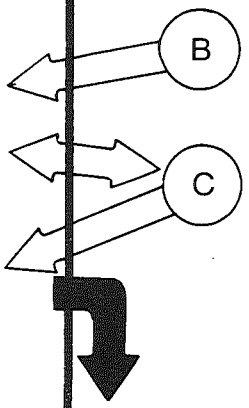
VERARBEITUNG

5. Trage Richtung ein in letztes eingelaufenes Paket

6. Wenn Weiche nicht leer,

sonst

7. Setze Weiche (Verteilstationsnummer) in ermittelte Richtung

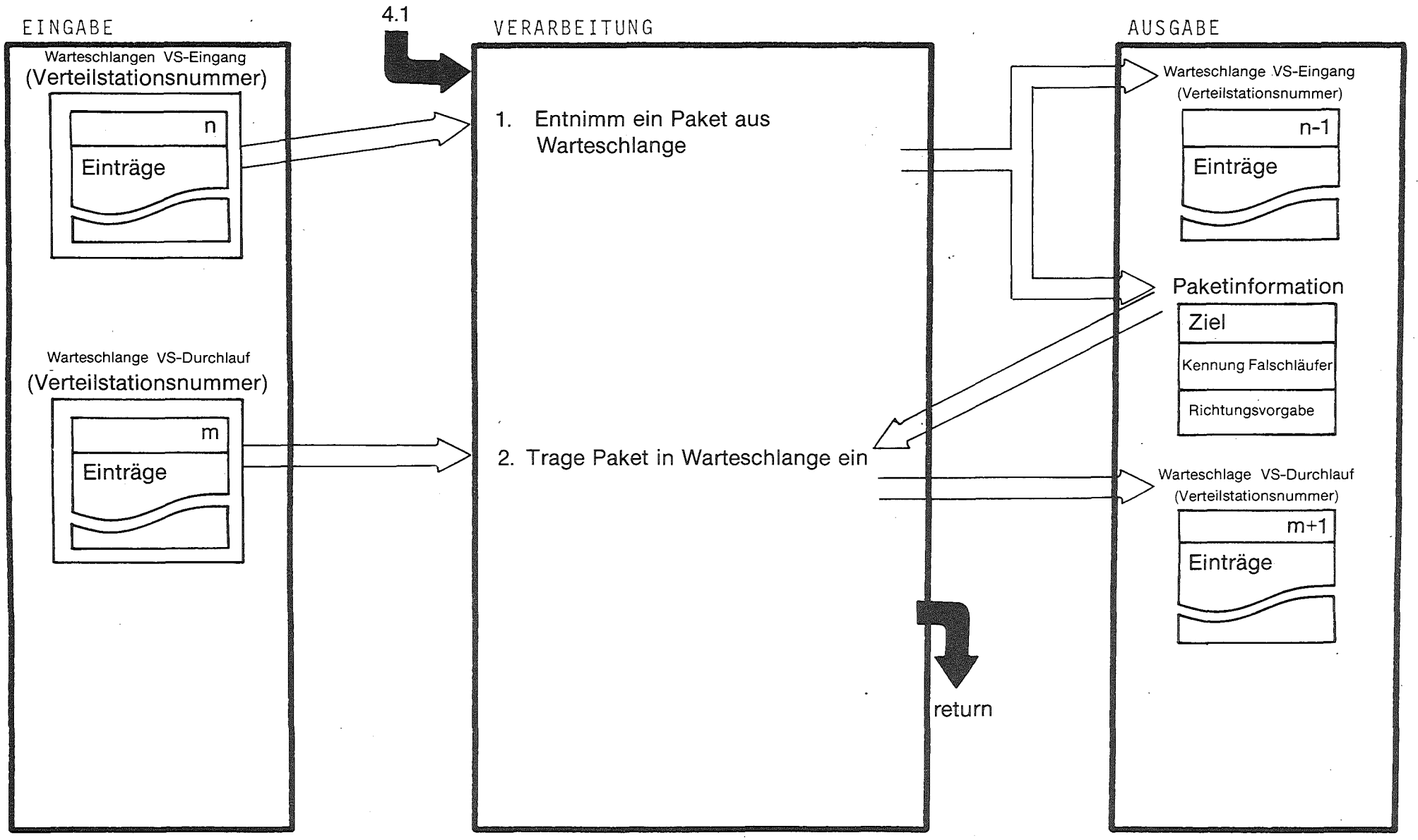


AUSGABE

Weichensignal

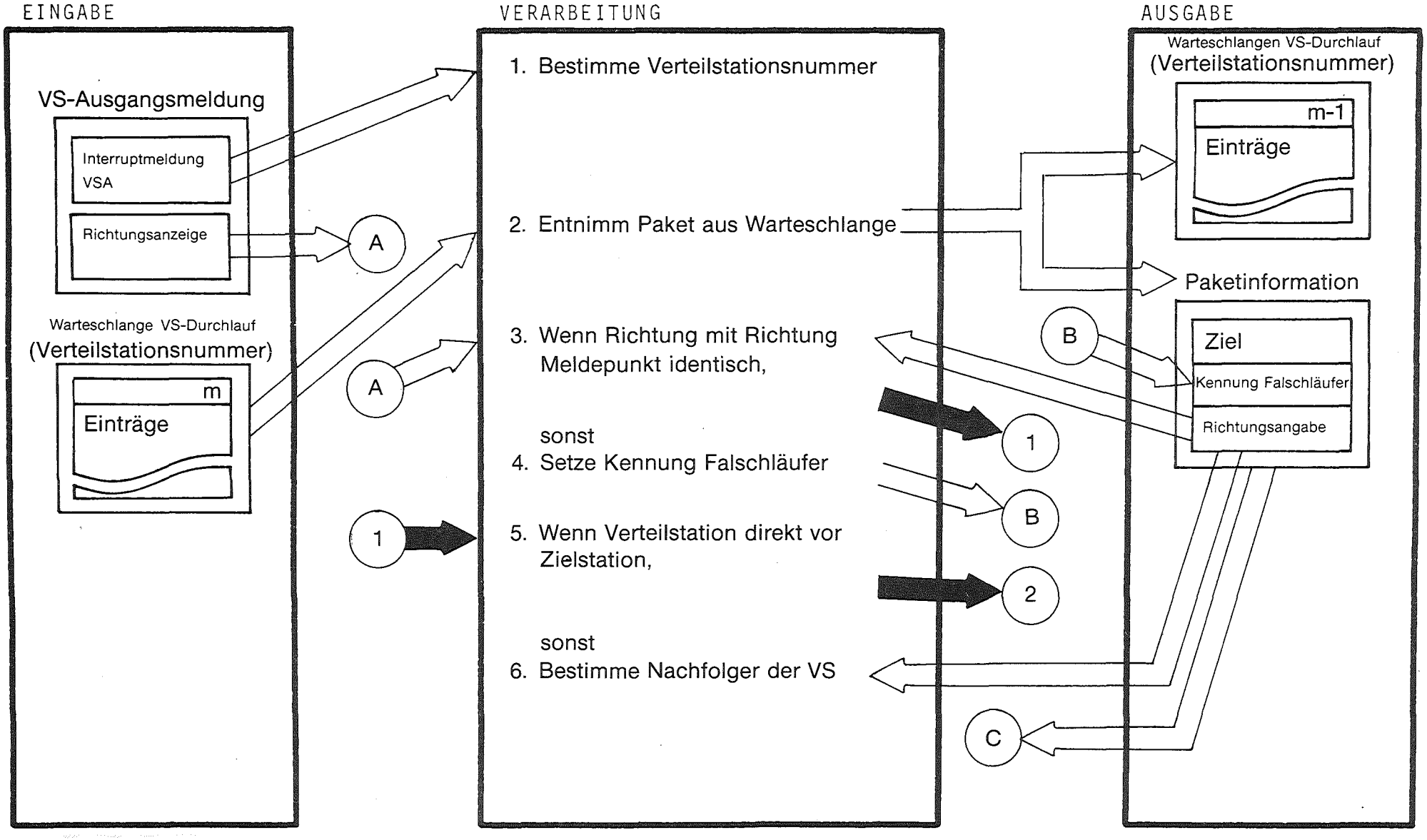
Autor: H. Keutgen, GEI | System: Paketverteilanlage | Datum: 3. April 1980

Diagramm-Id: 4.1.1 | Systemteil/Programm: Transferiere ein Paket /TRANSF



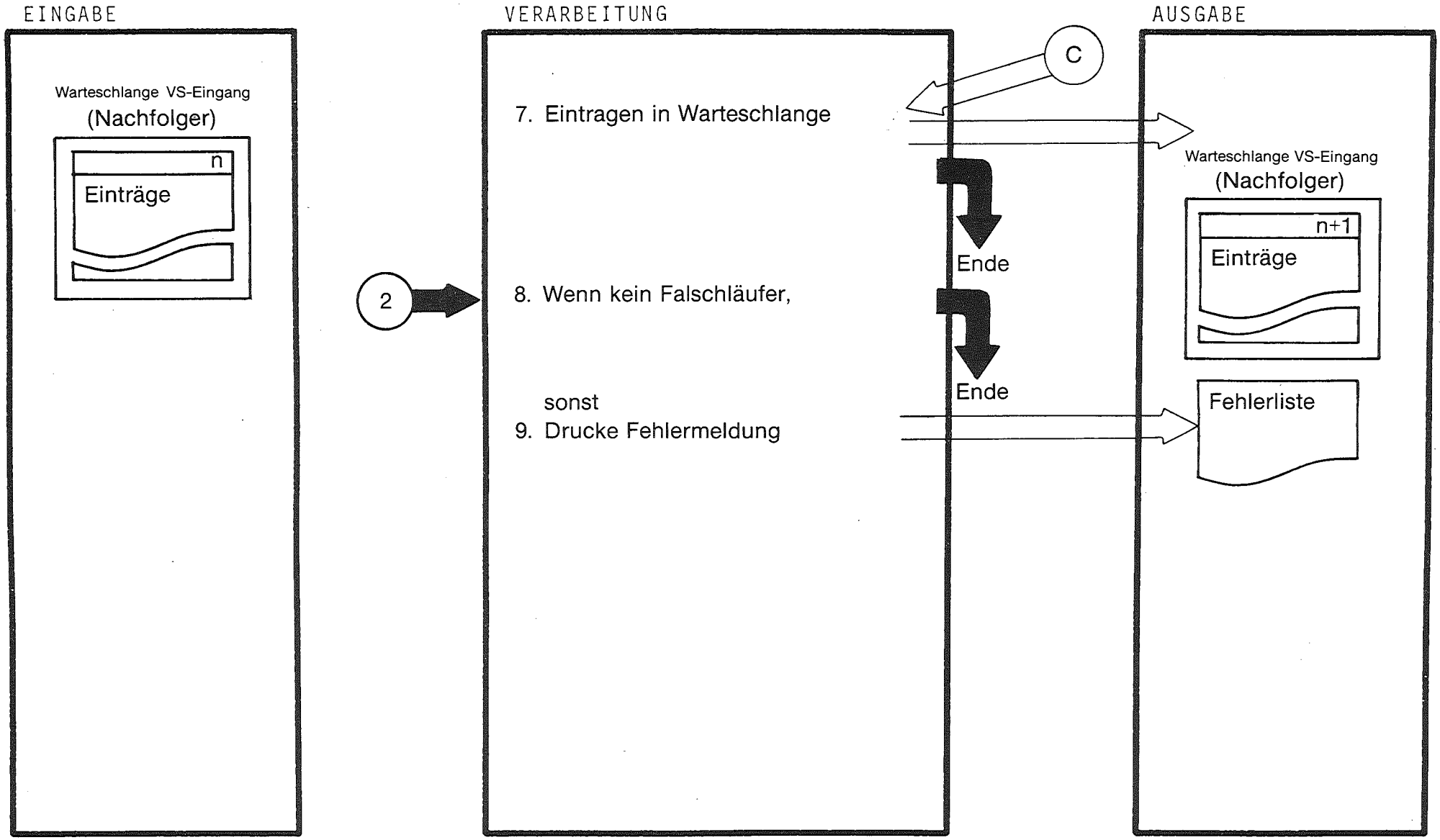
Autor: H. Keutgen, GEI | System: Paketverteilanlage | Datum: 8. April 1980

Diagramm-Id: 4.2 | Systemteil/Programm: Behandle VS-Ausgang /VSA | Seite 1 von 2



Autor: H. Keutgen, GEI | System: Paketverteilanlage | Datum: 8. April 1980

Diagramm-Id: 4.2 | Systemteil/Programm: Behandle VS-Ausgang /VSA | Seite 2 von 2



DATENSTRUKTURIERTER ENTWURF DER STEUERSOFTWARE FÜR EINE PAKETVERTEILANLAGE (METHODE JACKSON)

Bernd K ü h n e l und Horst R ü h r n s c h o p f
Siemens AG, Erlangen

Zusammenfassung

Im PDV*)-Arbeitskreis "Systematische Entwicklung von PDV-Systemen" wurde eine vereinbarte Aufgabenstellung - Steuerung einer Verteilanlage - nach mehreren Methoden für Spezifikation und Entwurf von Software bearbeitet. In diesem Beitrag wird der datenstrukturierte Entwurf der Lösung nach Methode Jackson vorgestellt. Im ersten Teil wird die Methode erläutert, im zweiten Teil wird der Entwurf der Steuersoftware beschrieben.

*) PDV : Projekt Prozeßlenkung mit DV-Anlagen des BMFT

Vorbemerkung

Der vorliegende Bericht besteht aus zwei Teilen. In Teil I wird die verwendete Methode erläutert, in Teil II wird dann der Software-Entwurf nach dieser Methode für eine spezielle Aufgabe beschrieben.

Inhalt

. . .

Teil I:

Erläuterungen zur Methode des datenstrukturierten Software-Entwurfs (Methode Jackson)

---

1. Abgrenzung und Einordnung

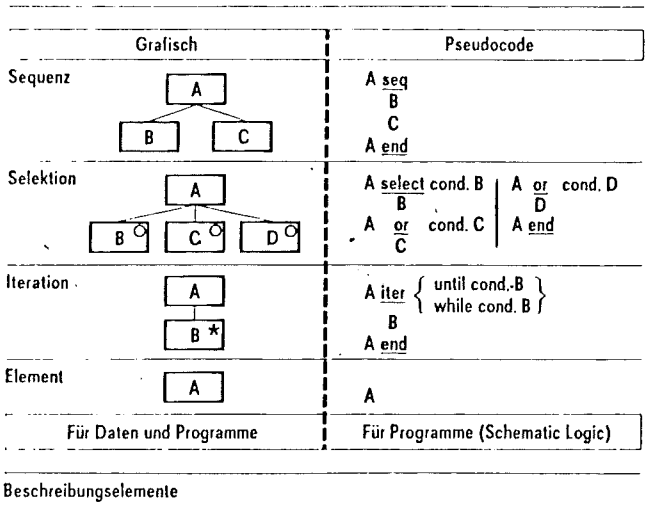
Die nach M.A. Jackson benannte Methode /1, 2/ (JSP - Jackson Structured Programming) ist im Spektrum der praktizierten Methoden explizit als Entwurfsmethode einzuordnen (vgl. auch /3/). Aus der Sicht eines software-life-cycles hat sie ihren Schwerpunkt in der Entwurfsphase.
Mit schrittweisem Vorgehen, Regelwerk und begleitender Dokumentation führt JSP zu einem Lösungsmodell, dem Entwurf, als Ergebnis des Entwerfens. Ein Prinzip der Methode ist es, die problemgerechte Struktur eines Programms systematisch aus den Strukturen der beteiligten (sequentiellen) Daten zu entwickeln. Deshalb heißt der Entwurf datenstrukturiert. Ein Ziel der Methode JSP ist es, "einfache" Programme zu entwerfen. Bei komplexen Problemen wird dies durch methodische Zerlegung (vgl.3.) erreicht, durch Herausarbeiten der wesentlichen sequentiellen Zusammenhänge.
Die JSP hat ihren Ursprung in der konventionellen Datenverarbeitung, sie ist jedoch ohne Schwierigkeiten auf Belange der Prozeß-Datenverarbeitung übertragbar, wenn man sie verallgemeinert und als Methode zur Modellbildung versteht.

Im folgenden wird die JSP nur grob skizziert, es werden ausgewählte Gesichtspunkte erläutert. Zum ausführlichen Studium muß auf die Literatur und auf Kurse und Kursunterlagen bei in- und ausländichen Lehrinstituten verwiesen werden.

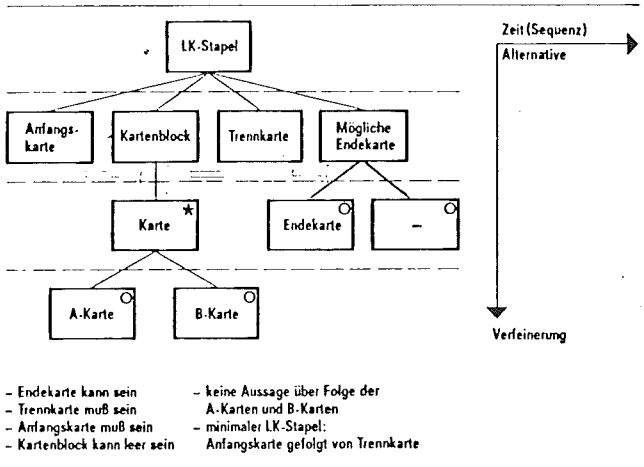
. . .

2. Schritte der Methode

Ausgehend von der Aufgabenstellung läuft das Entwerfen mit JSP in 5 Schritten ab, im realen Fall u.U. iterativ.



Beschreibungselemente



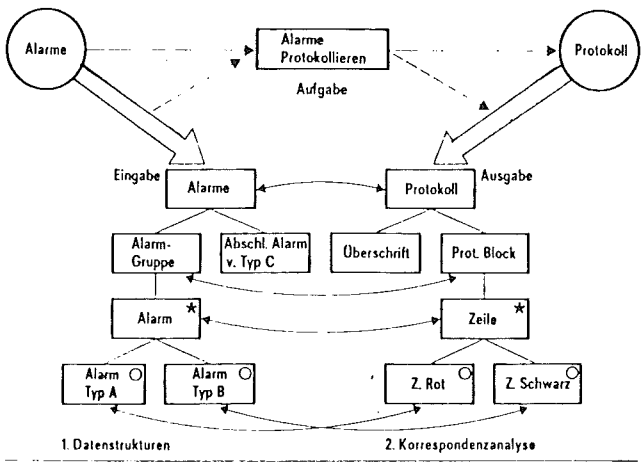
Beispiel: Struktur

Bild 1

Bild 2

Es gibt graphische Sprachmittel zur Darstellung von Baumstrukturen, sowohl für Daten als auch für Programme (Bild 1). Ferner gibt es einen Pseudocode zur abschließenden Beschreibung von Programmen (Bild 1). In Bild 2 ist eine Datenmenge LK-Stapel strukturiert. Die mit dieser Struktur ausgedrückten Aussagen über den Bezug zum Problem sind unter der Struktur aufgelistet.

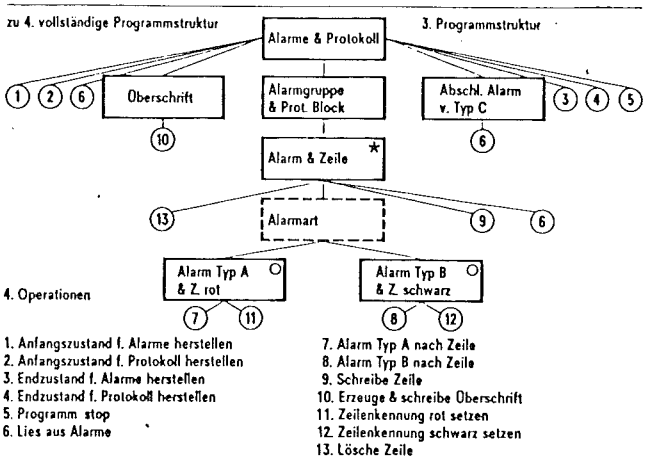
. . .



Beispiel: Alarme protokollieren

Schritt 1 und Schritt 2

Bild 3



Beispiel: Alarme protokollieren

Schritt 3 und Schritt 4

Bild 4

```
ALARME & PROTOKOLL seq
        Anfangszustand f. ALARME herstellen                ( 1)
        Anfangszustand f. PROTOKOLL herstellen             ( 2)
        Lies aus Alarme                                    ( 6)
        Erzeuge und schreibe Überschrift                   (10)
        ALARMGRUPPE & PROTBLOCK iter while Alarmtyp A oder B
                ALARM & ZEILE seq
                        Lösche Zeile                       (13)
                        ALARMART select Alarmtyp A
                                Alarmtyp A nach Zeile      ( 7)
                                Zeilenkennung rot setzen   (11)
                        ALARMART or Alarmtyp B
                                Alarmtyp B nach Zeile      ( 8)
                                Zeilenkennung schwarz setzen (12)
                        ALARMART end
                                Schreibe Zeile             ( 9)
                                Lies aus Alarme            ( 6)
                ALARM & ZEILE end
        ALARMGRUPPE & PROTBLOCK end
        ABSCHL. ALARM v. TYP C seq
                Lies aus Alarme                            ( 6)
        ABSCHL. ALARM v. TYP C end
                Endzustand f. ALARME herstellen            ( 3)
                Endzustand f. PROTOKOLL herstellen         ( 4)
                Programm stop                              ( 5)
ALARME & PROTOKOLL end
```

Beispiel: Alarme protokollieren
Schematic Logic

Schritt 5

Bild 5

Zur Erläuterung der Entwurfsschritte dienen die Bilder 3, 4 und 5 mit dem Beispiel "Alarme Protokollieren".

. . .

Schritt 1: Datenstrukturen erstellen

Die Aufgabe wird hinsichtlich ihrer Eingangs- und Ausgangsdaten analysiert. Die "wesentlichen" Daten werden als problemgerechte Baumstrukturen graphisch dargestellt (Bild 3, 1.)

Schritt 2: Korrespondenzanalyse

Die Überlagerbarkeit der Strukturen wird untersucht und gekennzeichnet (Bild 3, 2., Pfeile). Korrespondenz zwischen zwei Knoten bzw. den zugehörigen Daten liegt vor, wenn die Daten gleichoft, in gleicher Reihenfolge und gemeinsam verarbeitet werden können. Bei dieser Analyse können sich Strukturkonflikte zeigen, die dann Standardlösungen haben (vgl. 3.).

Schritt 3: Programmstruktur bilden

Konfliktfreie Strukturen werden nach Regeln zusammengefaßt und überlagert. Es entsteht die Programmstruktur (Bild 4, ohne "Kreise").

Schritt 4: Operationen spezifizieren, auflisten, einhängen

Die Operationen werden aus der Aufgabenstellung und aus den Strukturen heraus auf der gewählten Abstraktionsstufe festgelegt. Sie werden als Blätter dem Programmbaum nach Regeln für Ort und Sequenz angelagert (Bild 4, Operationen in Kreisen). Die entstandene Struktur kann statisch (in Schichten, Verfeinerung) oder dynamisch (entlang den Blättern) interpretiert werden.

Schritt 5: Schematic logic formulieren

Die gewonnene Programmstruktur wird als Pseudocode formuliert. Dabei werden Bedingungen ausgearbeitet und u.U. Backtracking-Behandlung eingefügt (Bild 5).

. . .

Damit liegt der Entwurf des Programms auf der gewählten Abstraktionsebene vor; der Entwurf kann nun implementiert oder, falls nötig, verfeinert werden.

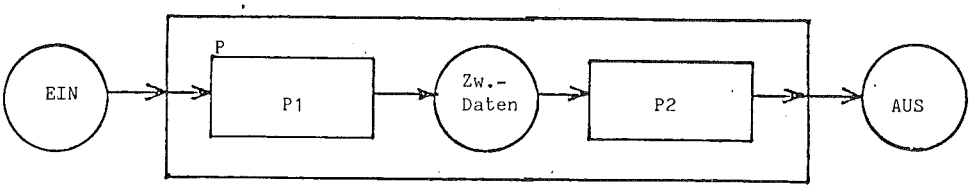
Das dargestelle Vorgehen deckt den Programmentwurf ab. Die Behandlung komplexerer Probleme wird im folgenden gestreift; die Anwendung im Teil II verdeutlicht sie.

3. Strukturkonflikte, Systeme

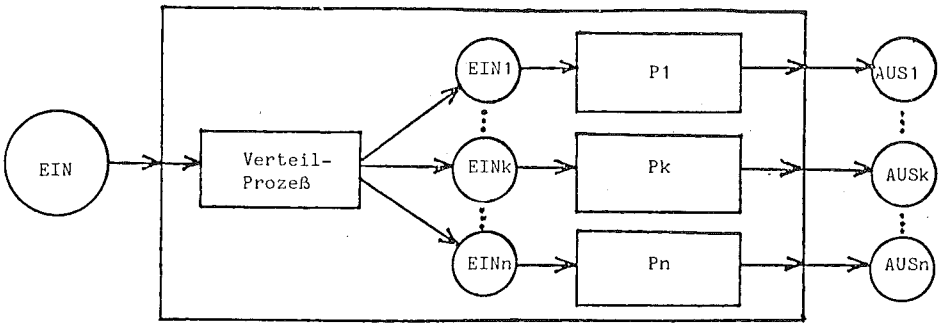
Die bei einer Korrespondenzanalyse festgestellten Strukturkonflikte sind nach JSP klassifizierbar als

- o Reihenfolgekonflikt: unverträgliche Reihenfolge der Daten
- o Abgrenzungskonflikt: unverträgliche Blockungsgrenzen
- o Misch- bzw. Einschubkonflikt: separat benötigte Datengruppen sind vermischt (ineinandergeschoben)

Die Behandlung dieser Konflikte führt im wesentlichen zu den in Bild 6 dargestellten Lösungen und liefert Systeme: Sequentialisierung mit Zwischendaten oder Verteilung in mehrere Datenmengen, die dann getrennt bearbeitet werden. Die entstehenden Programme - im weiteren auch Prozesse (auf Entwurfsebene) genannt - werden dann nach Schritt 1 bis 5 entworfen.



System durch Sequentialisierung



System durch Verteilung

Bild 6: Auflösung von Strukturkonflikten

. . .

Die Verteilung stellt einen Ansatz zur Parallelisierung der einzelnen Prozesse dar. Sie liefert u.U. eine Klasse "ähnlicher" Prozesse, von der nur ein Repräsentant entworfen werden muß. Die Kommunikations- und Verschaltungsstrategien solcher Systeme sind über die (Zwischen-) Datenmengen im Entwurf allgemein gehalten; sie werden bei Realisierung spezialisiert.

'Wesentliche' Sequenzen in einem System werden durch Betrachtung der Lebensgeschichte (life history) von Objekten des Systems gefunden. Es sind Prozesse zu installieren, die diese life histories abwickeln. Auch dabei entstehen vielfach Klassen "ähnlicher" Prozesse.

4. Realisierung von Entwürfen

Realisierung bedeutet, einen oder mehrere Schritte auf die Zielebene (Maschine, Sprache, Rechner) hin durchzuführen.
Bei der Realisierung eines JSP-Entwurfs ist zu diskutieren und festzulegen

- wie Prozeß-Klassen abzubilden sind, z.B.
  einzelne Prozesse oder ein umfassender Repräsentant, der die Prozeßsteuerblöcke verarbeitet
- wie die Prozesse zu verschalten sind, z.B.
  Pufferkonzepte, Koroutinen oder Ersatzkonzepte, Zerlegung in Prozeduren
- welche Prozesse parallel arbeiten (evtl. parallele Prozessoren)
- wie die Realzeitbedingungen zwischen Prozessen erfüllt werden können,
- letztlich, wie die Prozesse, Daten, Operationen und Bedingungen zu implementieren sind.

. . .

5. Erfahrungen

Wir setzen die JSP seit einigen Jahren erfolgreich ein /6, 7/, speziell für Prozeßrechner-Software. Ein erweitertes Datenverständnis, z.B. Strukturierung von zeitabgeleiteten Daten, Berücksichtigung zeitkritischer Daten bei Korrespondenzen, Behandlung von Ähnlichkeitsklassen und Parallelisierung von Prozessen runden das datenstrukturierte Entwerfen ab /8/.

Für viele Anwendungen ist über die Daten ein schneller und intensiver Problemzugang möglich. Die im Entwurf entwickelten Systemstrukturen sind weitgehend sequentialisiert, deshalb transparent, diskutierbar und bei Realisierung anpaßbar und verformbar je nach Randbedingungen, wie z.B. Platz, Laufzeit, Anzahl der Prozessoren oder Realzeitverhalten.

Teil II:

Datenstrukturierter Entwurf des Steuersystems einer Paketverteilanlage (nach Methode Jackson)

Inhalt

. . .

1. Erläuterungen zur Aufgabenstellung

Nach Aufgabenstellung ist ein Steuersystem (Software) für eine Paketverteilanlage zu entwerfen. Als Anhang 2 ist die originale Beschreibung der Aufgabe angefügt. Ein grober Überblick wird wie folgt gegeben.
Die Anlage hat eine Eingangsstation, mehrere Verteilstationen und Zielstationen. Die Anlage ist baumartig aufgebaut; jede Verteilstation hat einen Eingang und zwei Ausgänge. Einlaufende Pakete werden von der Eingangsstation dem Steuersystem gemeldet, ihr Zielcode wird gelesen und die Pakete werden - bei gleichem Ziel in dichter Folge, sonst verzögert - in das Verteilsystem freigegeben. Dort werden sie zu ihrem Ziel gesteuert. Ein Verzögern der Paketfolge ist nur in der Eingangsstation möglich. Die einzelnen Verteilstationen haben am Eingang und an den Ausgängen Lichtschranken, die das Passieren eines Paketes sicher erkennen und der Steuerung melden. Der Zielcode kann nicht erneut gelesen werden. Die Lenkung in einer Verteilstation kann nur dann geschaltet werden, wenn sich kein zweites Paket innerhalb des Lichtschrankenbereiches der Station befindet.

Beim Entwerfen entstanden einige Fragen zur Präzisierung der Aufgabenstellung. Die vereinbarten Aussagen sind als Anhang 3 angefügt. Sie präzisieren im wesentlichen die Fehlerbehandlung, Durchsatz und Fassungsvermögen des Steuersystems, die Abwicklung auf einem Monoprozessor und den Level der Zielsprache.

Für die Verteilanlage werden mindestens 2 Zielstationen vorausgesetzt, eine davon als Fehler-Zielstation. Eine Paketerkennung der Zielstationen ist in der Aufgabe nicht erwähnt; sie wird entweder als Zusatz eingeführt, oder Zielmeldungen und -auswertungen werden nach Durchlaufen der letzten Verteilstation ausgeführt.

. . .

2. Allgemeine Vorbemerkungen

In den folgenden Abschnitten wird der datenstrukturierte Entwurf nach Methode Jackson für das Steuersystem dargestellt. Dabei wird das Ergebnis, der Entwurf, gezeigt und beschrieben. Es werden nicht alle Einzelschritte und Iterationen des Entwerfens als Modellbildungsprozeß erläutert. Die Entwurfsunterlagen sind in diesem Sinne nicht vollständig.

Abschnitt 3 beschreibt die Systemzerlegung in einzelne Prozesse und das Systembild. Unter einem Prozeß verstehen wir hier eine zusammengehörige Sequenz von Aktionen auf einer Abstraktionsebene des Entwurfs (Vermeidung des Begriffs Funktion oder Programm).

Zu jedem Prozeß ist in Abschnitt 4 angegeben

- Übersichtsbild ("Datenfluß-Bild")
- Datenstrukturen mit Korrespondenzen
- Programmstruktur mit Operationen
- Liste der Operationen
- Liste der Bedingungen
- Liste der Strukturknoten mit ausgeführtem Text
- Schematic logic (Pseudocode),

mit Zuordnung der Operationsnummern, z.B. (5), und der Knotenbeschreibungen, z.B. -3-, sowie vorab Erläuterungen zum Übersichtsbild und zum Entwurf.

Der Entwurf wurde von uns in ein PASCAL-Programm umgesetzt und realisiert, das die Verteilersteuerung simuliert (siehe hierzu Abschnitt 5).

Noch einige Bemerkungen zur Notation:
In den Übersichtsbildern bedeuten Rechtecke Verarbeitungseinheiten (z.B. Prozesse) auf dem jeweiligen Level, Kreise bezeichnen für die Strukturierung wesentliche Datenmengen, während Ovale Datenmengen bezeichnen, deren Struktur als unwesentlich erachtet wird.

. . .

(H) weist auf hardware-Schnittstellen hin;

(H/I) kennzeichnet evtl. zwischengeschaltete Interruptbearbeitung.

Korrespondenzen werden durch Doppelpfeile bezeichnet, "einseitige" Korrespondenzen (Implikation) durch einseitige Pfeile.
Eine Liste der in den Strukturen verwendeten Abkürzungen ist im Anhang 1 angegeben.

Die Strukturdiagramme wurden mit dem Programmsystem DIPROTOR (Prototyp-Version) der Gesellschaft für Mathematik und Datenverarbeitung mbH, Bonn, erstellt.

3. Strukturierungsansatz, Systemzerlegung

Es ist ein Steuersystem zu entwerfen. In Bild 1 ist der erste Ansatz als Verteilsteuerung dargestellt. Die für die Strukturierung wichtigsten Datenmengen sind

- o eingabeseitig
  - Paketmeldungen zur Eingangssteuerung
  - die Codezeichen der Pakete zur Zielangabe
- o ausgabeseitig
  - die Freigaben an das Eingangselement
  - die zielsortierten Pakete (-Identifikationen)

Die weiteren in Bild 1 aufgeführten Daten sind zunächst nicht wesentlich für die Systemstrukturierung. Bei Zerlegung der Verteilsteuerung treten weitere (innere) Datenmengen auf, die für die Strukturierung von Teilen des Systems herangezogen werden.

Die (physikalischen) Pakete werden in der Verteilsteuerung durch zugeordnete Daten - die Paket-Identifikationen (abgekürzt: Paket-Idn) - repräsentiert. Diese Zuordnung muß über den gesamten Steuerungsvorgang für ein Paket bestehen bleiben, da außer in der Eingangsstation der Zielcode des Paketes nicht gelesen werden kann. Die Paketverteilanlage kann also nicht völlig dezentralisiert gesteuert werden.

. . .

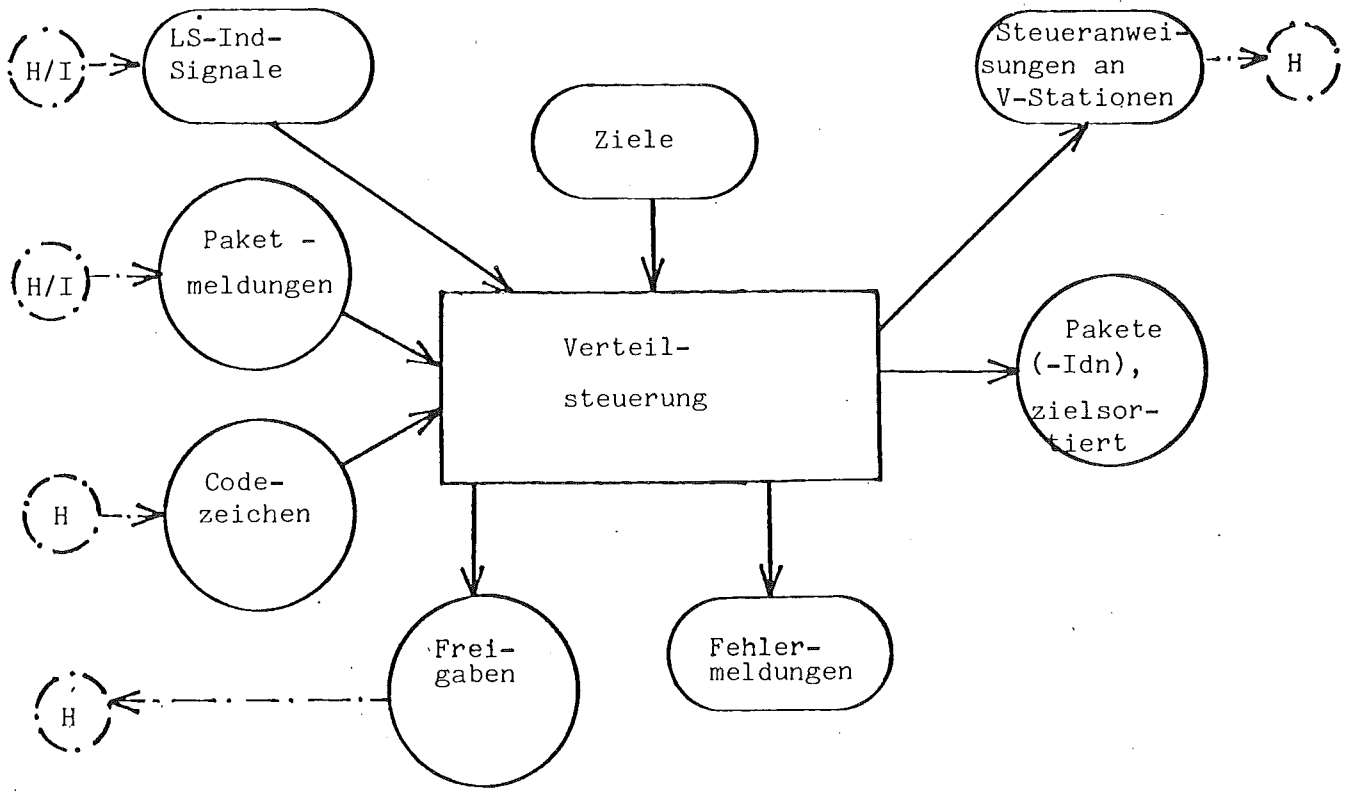
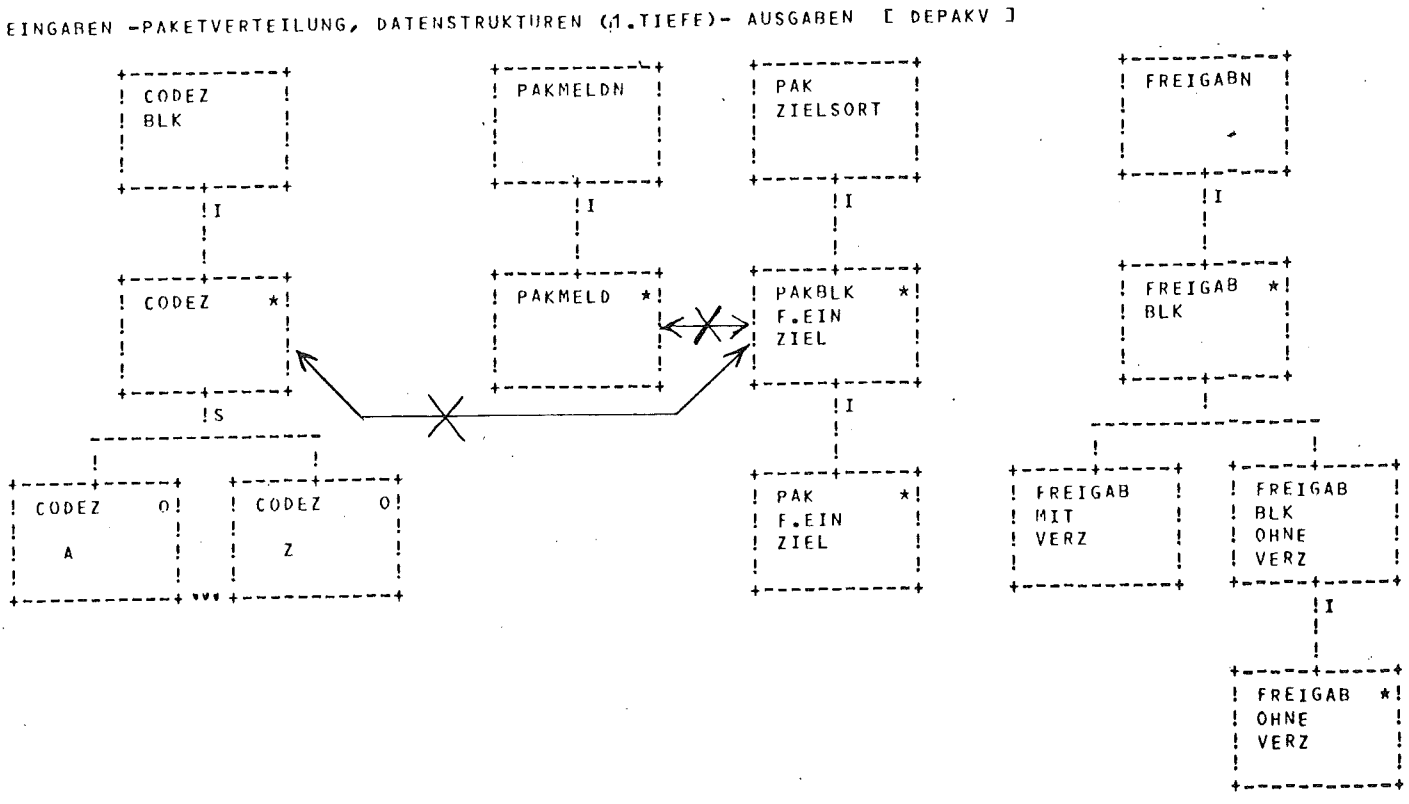


Bild 1: 1. Ansatz zum Steuersystem



Die Analyse der nach Bild 1 angegebenen Strukturen der "wichtigen" Datenmengen liefert folgende Erkenntnisse:

- o Nicht-Korrespondenz von Codezeichen bzw. Paketmeldungen und ausgabeseitig zielsortierten Pakete-Idn (Einschub- bzw. Interleaving-Konflikt). Die Standardlösung führt zur Verteilung der Eingabedaten entsprechend den Ausgabeanforderungen, also zur Aufteilung der aus Paketmeldungen resultierenden Datenmenge in zielspezifische Teilmengen. Ferner muß es Prozesse zur Verarbeitung dieser Daten geben.
- o Die Freigaben an die Eingangssteuerung sind geblockt (ohne Verzögerung), die Eingabestrukturen lassen sich dem anpassen (siehe 4.1.).
- o Die wesentlichen Objekte, für die das System arbeitet, sind die Paket-Idn. Folglich muß es Prozesse geben, die die Lebensgeschichte (life history) einer jeden Paket-Id abwickeln. Dies führt zu potentiell unendlich vielen paketspezifischen Prozessen; jeder solcher Prozeß steuert "sein" Paket P auf "seinem" Weg zu "seinem" Ziel X ("Paketprozeß").

Diese Überlegungen führen, zusammen mit den folgenden, zur Zerlegung der Verteilsteuerung, entsprechend dem Systembild Bild 2.

Um von den zielsortierten Paket-Idn zu den einzelnen Paketsteuerungsprozessen zu gelangen, sind die (trivialen) Verteiler für Paket-Idn notwendig; sie verteilen entsprechend ihrer Eingangssequenz.
Bei der Strukturierung eines Paketprozesses (vgl. 4.4.) findet man, daß Informationen der einzelnen Verteilstationen (V-Stationen) auf dem Weg des Paketes benötigt werden. Folglich gibt es für jede V-Station einen Prozeß (Stationsprozeß), der die Lichtschrankensignale der V-station bearbeitet, das Gedächtnis der V-Station verwaltet (Ziel des Vorgänger-Paketes und Anzahl der Pakete in der Station) und diese Information den Paketprozessen aufbereitet. Für einen speziellen Paketprozeß arbeiten in diesem Sinne genau die Stationsprozesse auf dem Weg des Paketes.

. . .

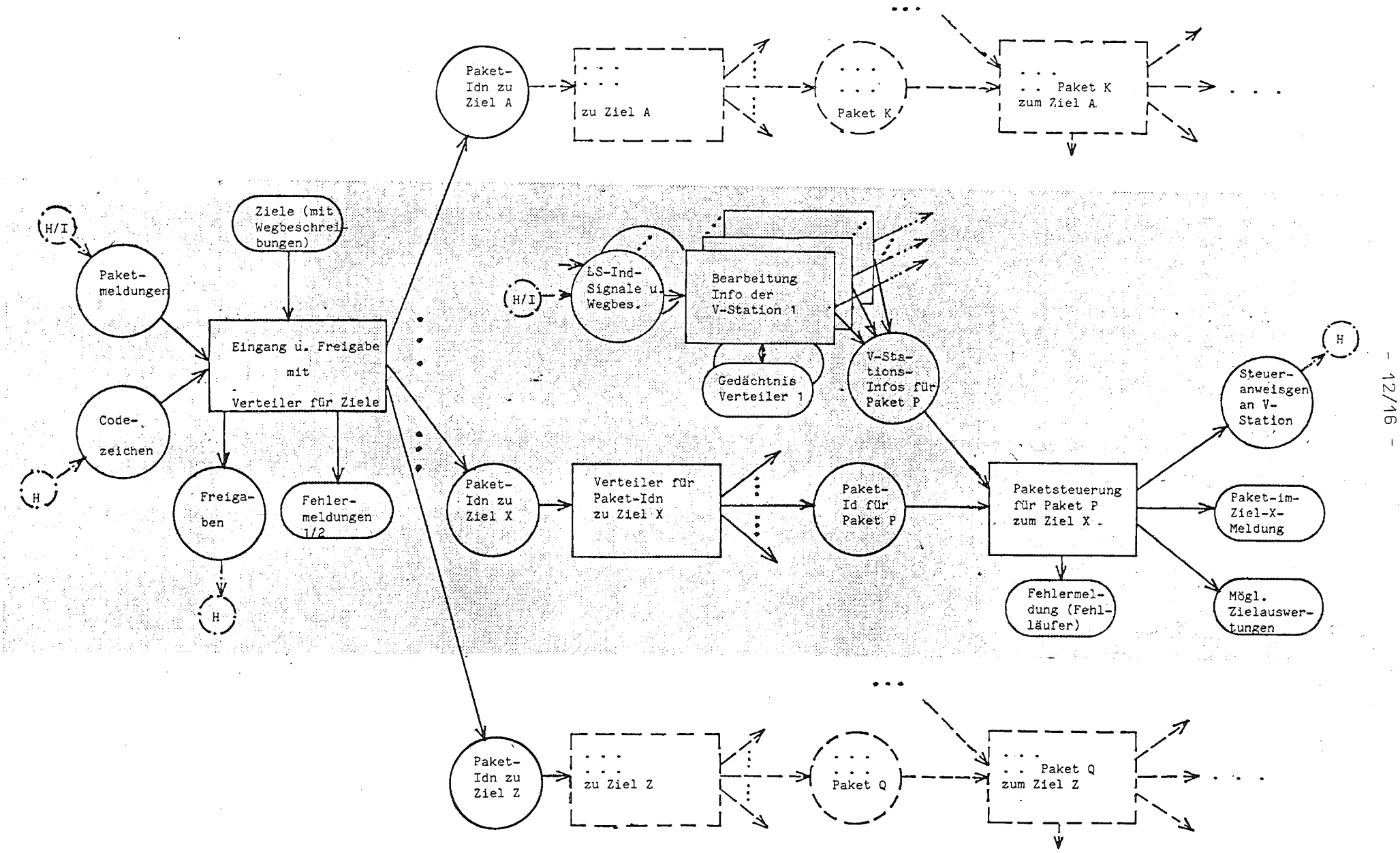


Bild 2 : Übersicht zum Gesamtsystem (Software)

Somit gibt es im Entwurf (siehe Bild 2) neben dem Prozeß "Eingang und Freigabe mit Verteiler für Ziele" mehrere (Ähnlichkeits-)Klassen von Prozessen

- o die Klasse der Verteiler für Paket-Idn
- o die Klasse der Paketprozesse
- o die Klasse der Stationsprozesse.

Die Entwurfsmethode führt dazu, daß die wesentlichen sequentiellen Vorgänge herausgearbeitet, strukturiert, werden (auf System- und auf Prozeßebene) und daß eine saubere funktionale Trennung bezüglich der Bearbeitung der Objekte - V-Stationen und Pakete - erreicht wird.

Aufgabe bei der Realisierung (Implementierung) des Entwurfs ist es dann, festzulegen, welche Prozesse als "reale" Prozesse implementiert werden - z.B. je ein Repräsentant jeder Klasse - und welche nur als "Prozeßsteuerblock" abgearbeitet (interpretiert) werden. Hierbei müssen dann auch Realzeitbelange zwischen den Prozessen berücksichtigt werden, sowie gewünschte Verschaltungen, wie z.B. Coroutinenkonzepte oder Ersatzkonstruktionen dazu (Inversion, Zerlegung u. Prozeduren etc.). Von einem Paketprozeß z.B. muß mindestens die Paketidentifikation, der zugeordnete Weg zum Ziel sowie die Kennzeichnung des aktuellen Verteilers in der Wegbeschreibung bei der Implementierung vorhanden sein.

Zusammenfassung: Der Systementwurf wurde unterstützt durch folgende Vorgehensweisen

- o Standardlösung von Strukturkonflikten (interleaving)
- o Strukturierung der "Lebensgeschichte" (life history) von wesentlichen Objekten im System
- o Zerlegung (Vereinfachung wegen schwacher funktionaler Bindung, siehe z.B. 4.1.)

Im folgenden werden die Jackson-Entwürfe der einzelnen Prozesse des Steuersystems der Paketverteilanlage beschrieben. Dabei werden auch die Fehlerfälle, wie z.B. Fehlläufer (in 4.4.) behandelt.

. . .

4. Entwurf der einzelnen Prozesse

4.1. Eingang und Freigabe

Die Steuerung der Eingangsstation hat wenig mit der Verteilung der Ziele zu tun. Folglich werden beide Funktionen der Klarheit wegen als getrennte Prozesse entworfen (siehe Bild 3). Der Prozeß EINGANG UND FREIGABE nimmt die Paketmeldungen entgegen, liest die Codezeichen, erzeugt die Freigaben für die Pakete (mit oder ohne Verzögerung), behandelt fehlerhafte Codezeichen und liefert fehlerfreie Paketidentifikationen mit Code an den VERTEILER FÜR ZIELE.

Der mögliche Fehlerfall 1 - fehlendes oder verstümmeltes Codezeichen zu einer Paketmeldung - führt nach erweiteter Aufgabenstellung zur Zuweisung eines Ersatzcodezeichens (z.B. FA) mit anschließender "normalen" Steuerung des Paketes durch die Anlage zu dem zu FA gehörigen Ziel. Aus diesem Grund ist die Meldung und Behandlung solcher Fehler der Operation "lies Codezeichen" zugeordnet (Filterwirkung).

Die Datenmengen Paketmeldungen und Codezeichen sind von der Aufgabenstellung her unterschieden (getrennte Erkennungsgeräte). Ihre Strukturen sind aufeinander abzustimmen. Da ohne Paketmeldung kein Codezeichen gelesen wird, ist die Führungsstruktur die der Paketmeldungen.

Die mächtigere Struktur der Datenmenge Freigaben (Blöcke mit unverzögerter Freigabe) erfordert eine Anpassung der Eingabestrukturen, um Korrespondenzen zu ermöglichen. EINGANG UND FREIGABE bearbeitet somit Paketgruppen gleichen Codezeichens (mindestens ein Element pro Gruppe).

Anmerkung: Die Operation "lies Paketmeldung" kann je nach Realisierung des Prozesses (Sprache, Betriebssystem usw.) ein "warte auf Paketmeldung" sein, bzw. die Stelle im Prozeß bezeichnen, wo dieser durch die Hardware oder einen Interruptverwalter jeweils angestoßen (aktiviert) wird.

Nachfolgend ist der Entwurf für EINGANG UND FREIGABE sowie für die Operation LIESCODEZEICHEN angefügt.

. . .

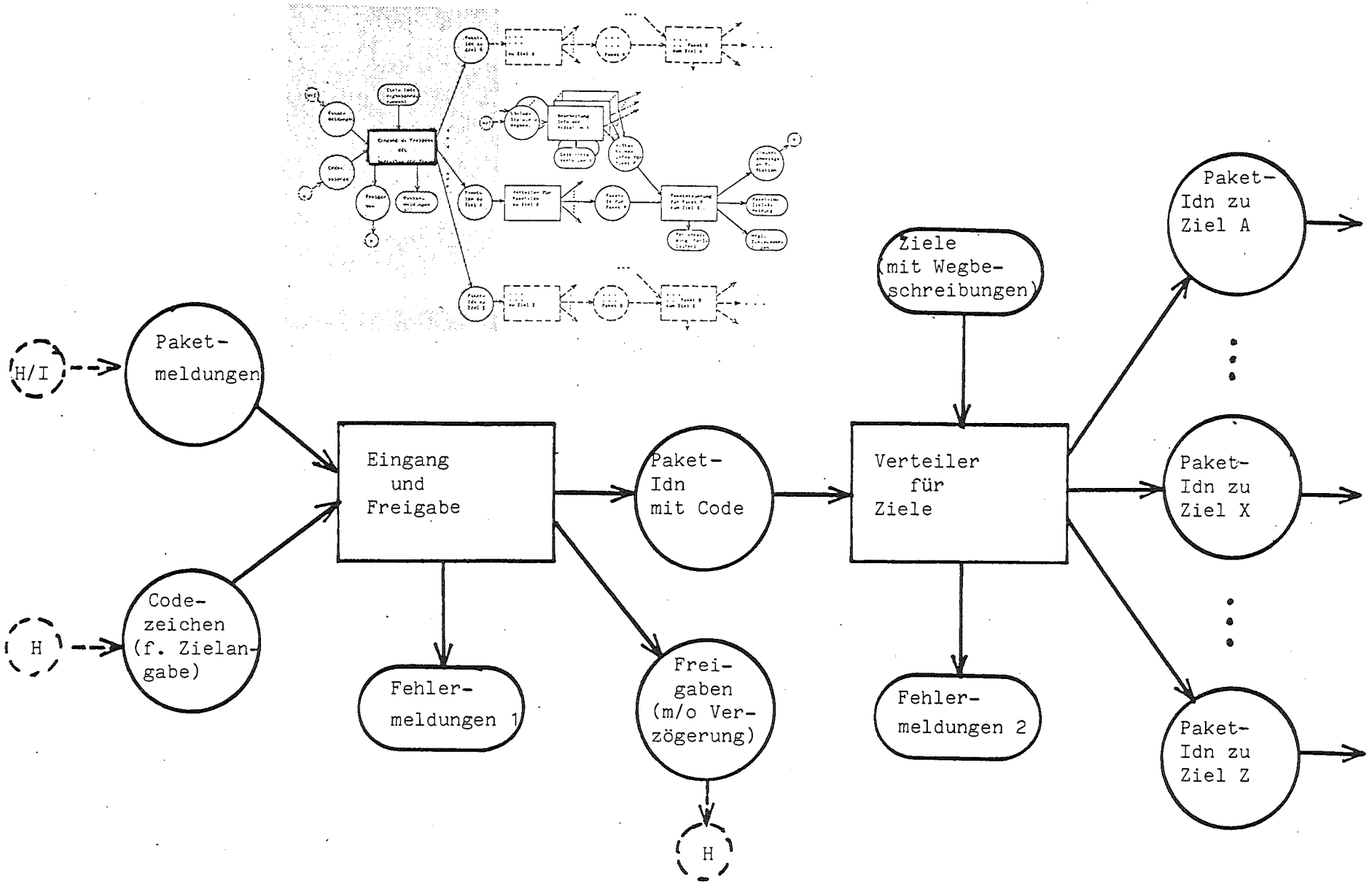
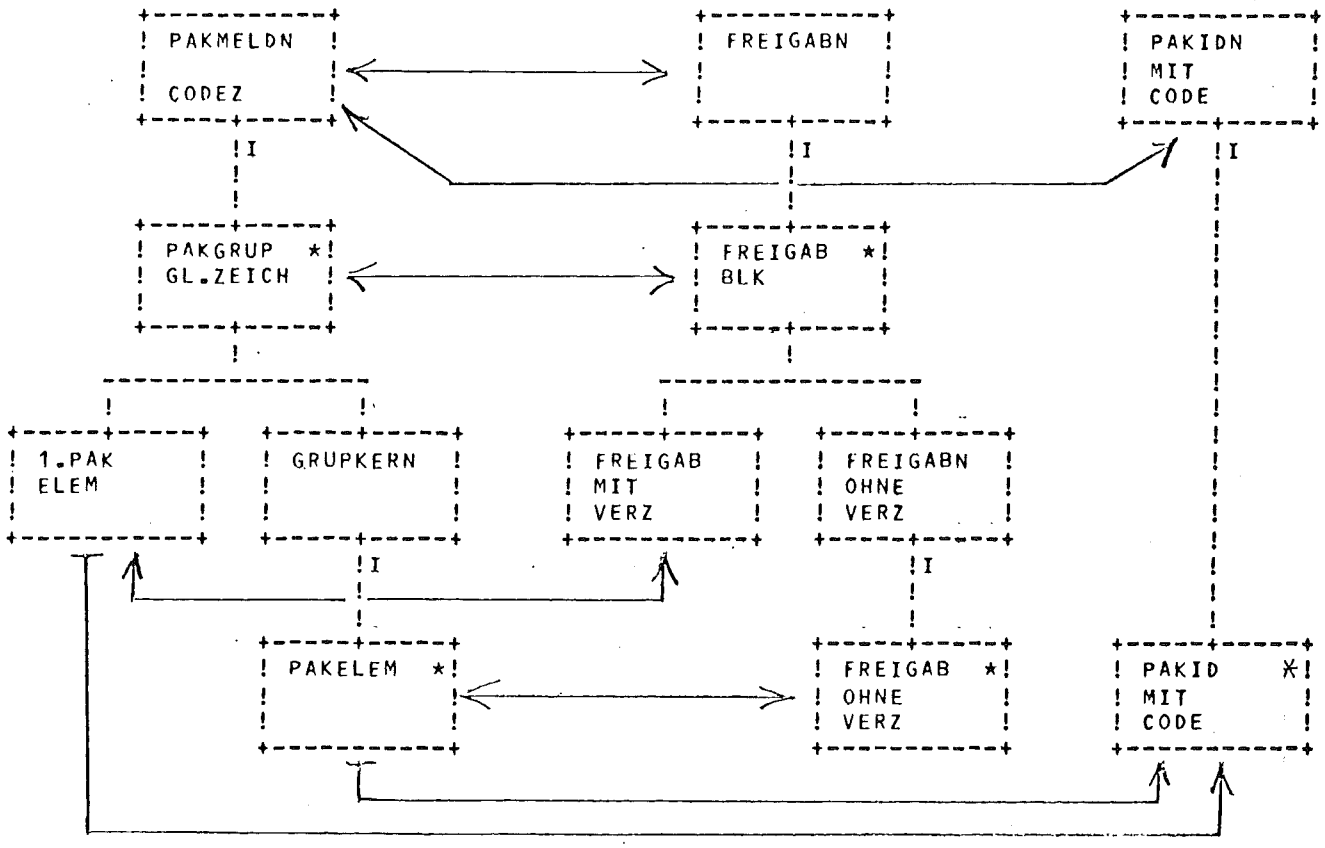
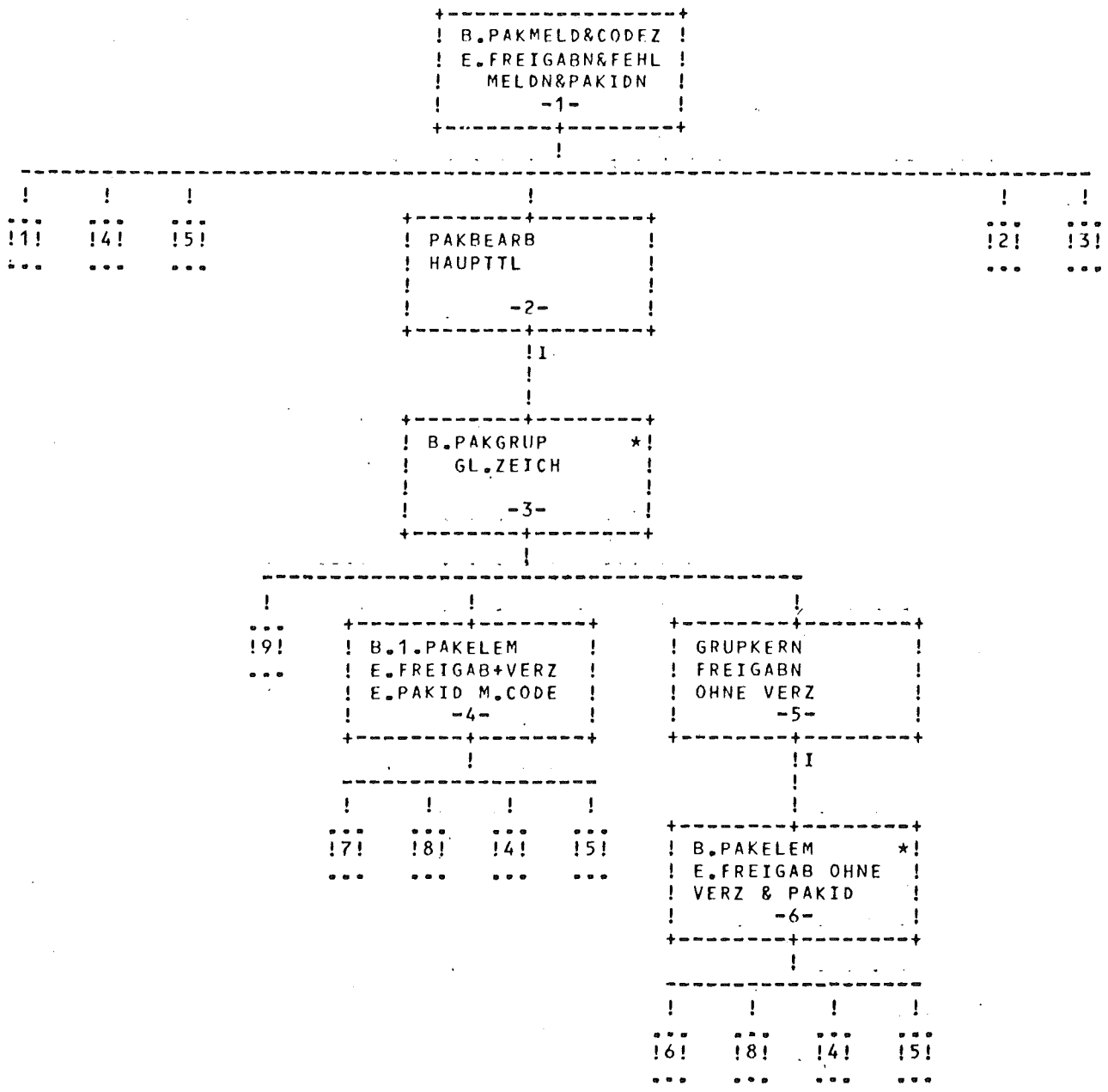


Bild 3: Übersicht zu EINGANG UND FREIGABE mit VERTEILER FÜR ZIELE

DATENSTRUKTUREN FUER EINGANG UND FREIGABE [DEINFR]



PROGRAMMSTRUKTUR MIT OPERATIONEN FUER EINGANG UND FREIGABE [PEINFR]



OPERATIONENLISTE

1 anfangsbearbeitung fuer datenobjekte durchfuehren
(datenobjekte sind - meldungen
- codezeichen
- fehlermeldungen
- paketidentifikation mit code
- freigaben)
2 endebearbeitung fuer datenobjekte (siehe 1) durchfuehren
3 programm stop
4 lies paketmeldung
5 lies codezeichen (mit fehlermeldung 1 und fehlerbehandlung)
6 erzeuge freigabe ohne verzoegerung
7 erzeuge freigabe mit verzoegerung
8 paketidentifikation mit code ausgeben
9 vergleichszeichen := codezeichen

ITERATIONSBEDINGUNGEN

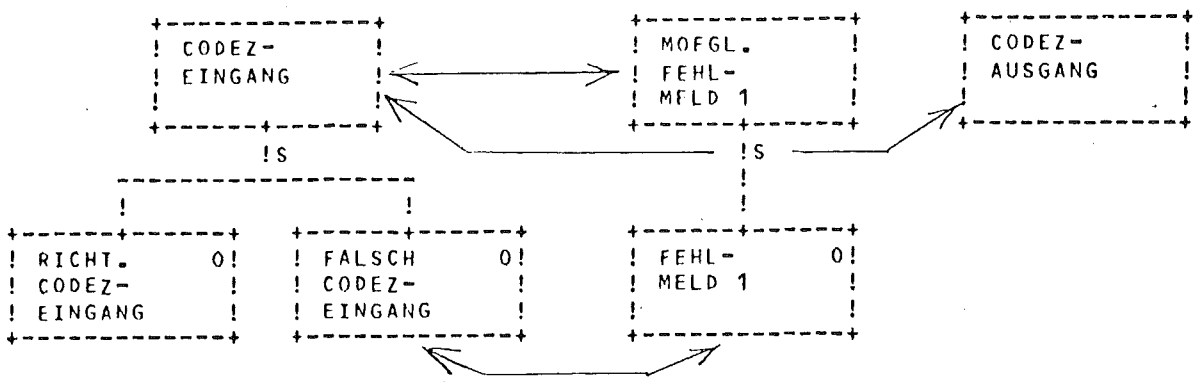
-2- while not eof-Meldung
-5- while not eof-Meldung und Vergleichszeichen = Codezeichen

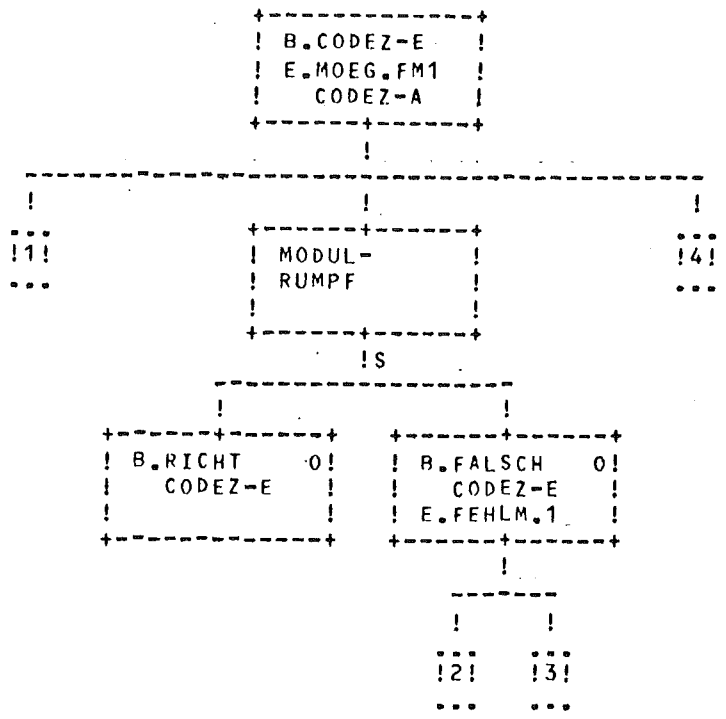
```
KNOTENBESCHREIBUNG
-1- B.PAKETMELDUNGEN UND CODEZEICHEN
    E.FREIGABEN, PAKETIDENTIFIKATIONEN
-2- PAKETBEARBEITUNGS-HAUPTTEIL
-3- B.PAKETGRUPPE GLEICHEN ZEICHENS
-4- B.ERSTES PAKETELEMENT
    E.FREIGABEN MIT VERZOEGERUNG, PAKETIDENTIFIKATION MIT CODE
-5- GRUPPENKERN UND FREIGABEN OHNE VERZOEGERUNG
-6- B.PAKETELEMENT
    E.FREIGABE OHNE VERZOEGERUNG, PAKETIDENTIFIKATION MIT CODE
==========================================================================
```

SCHEMATIC LOGIC

```
B.PAKMELDN,CODEZ/E.FREIGABN,FEHLMELDN,PAKIDN * SEQ *                -1-
  anfangsbearbeitung fuer datenobjekte durchfuehren                  (1)
  lies paketmeldung                                                  (4)
  lies codezeichen ( mit fehlermeldung 1 und -behandlung )           (5)
  PAKBEARB-HAUPTTL * ITER * while not eof-Meldung                    -2-
    B.PAKGRUP GL.ZEICH * SEQ *                                       -3-
      vergleichszeichen := codezeichen                               (9)
      B.1.PAKELEM * SEQ *                                            -4-
        erzeuge freigabe mit verzoegerung                            (7)
        paketidentifikation mit code ausgeben                        (8)
        lies paketmelung                                             (4)
        lies codezeichen ( mit fehlermeldung 1 und -behandlung )     (5)
      B.1.PAKELEM * END *
      B.GPUPKERN * ITER * while not eof-Meld u. Vgl-Zeich = Codez    -5-
        B.PAKELEM * SEQ *                                            -6-
          erzeuge freigabe ohne verzoegerung                         (6)
          paketidentifikation mit code ausgeben                      (8)
          lies paketmeldung                                          (4)
          lies codezeichen ( mit fehlermeldung 1 und -behandlung )   (5)
        B.PAKELEM * END *
      B.GRUPKERN * END *
    B.PAKGRUP GL.ZEICH * END *
  PAKBEARB-HAUPTTL * END *
  endebearbeitung fuer datenobjekte durchfuehren                     (2)
  programm stop                                                      (3)
B.PAKMELDN * END *
--------------------------------------------------------------------------
```

DATEN- UND PROGRAMMSTRUKTUR FUER DEN MODUL LIESCODEZEICHEN [LIESCZ]





==========

OPERATIONENLISTE

1 lies codezeichen-eingang
2 ersetze falsches codezeichen durch codezeichen fa
3 fehlermeldung 1
4 codezeichen-ausgang ausgeben

==========

SCHEMATIC LOGIC

```
B.CODEZ-EINGANG, E.MOEGL.FEHLMELD 1, E.CODEZ-AUSGANG   * SEQ *
  lies codezeichen eingang                                        (1)
  B.MODUL-RUMPF  * SELECT *  richtiges Codezeichen
    B.RICHTIGES CODEZ-EINGANG  * SEQ *
    B.RICHTIGES CODEZ-EINGANG  * END *
  B.MODUL-RUMPF  * OR *  falsches Codezeichen
    B.FALSCHES CODEZ-EINGANG, E.FEHLMELD 1  * SEQ *
      ersetze falsches codezeichen durch codezeichen fa           (2)
      fehlermeldung 1                                             (3)
    B.FALSCHES CODEZ-EINGANG, E.FEHLMELD 1  * END *
  B.MODUL-RUMPF  * END *
    codezeichen-ausgang ausgeben                                  (4)
B.CODEZ-EINGANG, E.MOEGL.FEHLMELD 1, E.CODEZ-AUSGANG   * END *
```

4.2. Verteiler für Ziele

Der Prozeß VERTEILER FÜR ZIELE (siehe Bild 3) bestimmt zu jeder Paketidentifikation mit Zielcode (von EINGANG UND FREIGABE) das zugehörige Ziel und verteilt die Paketidentifikationen zielspezifisch.

Der mögliche Fehlerfall 2 - zu einem Code existiert kein Ziel im System - führt nach erweiterter Aufgabenstellung zur Zuweisung eines Ersatzzieles. Hierzu wird das aus 4.1. schon bekannte Ersatzziel zu FA gewählt.

Ohne Einschränkung der Allgemeinheit und in Hinblick auf spätere Realisierung sind die Ziele jeweils mit einer Wegbeschreibung zum Ziel (Folge der Nummern der Verteilerstationen) versehen.

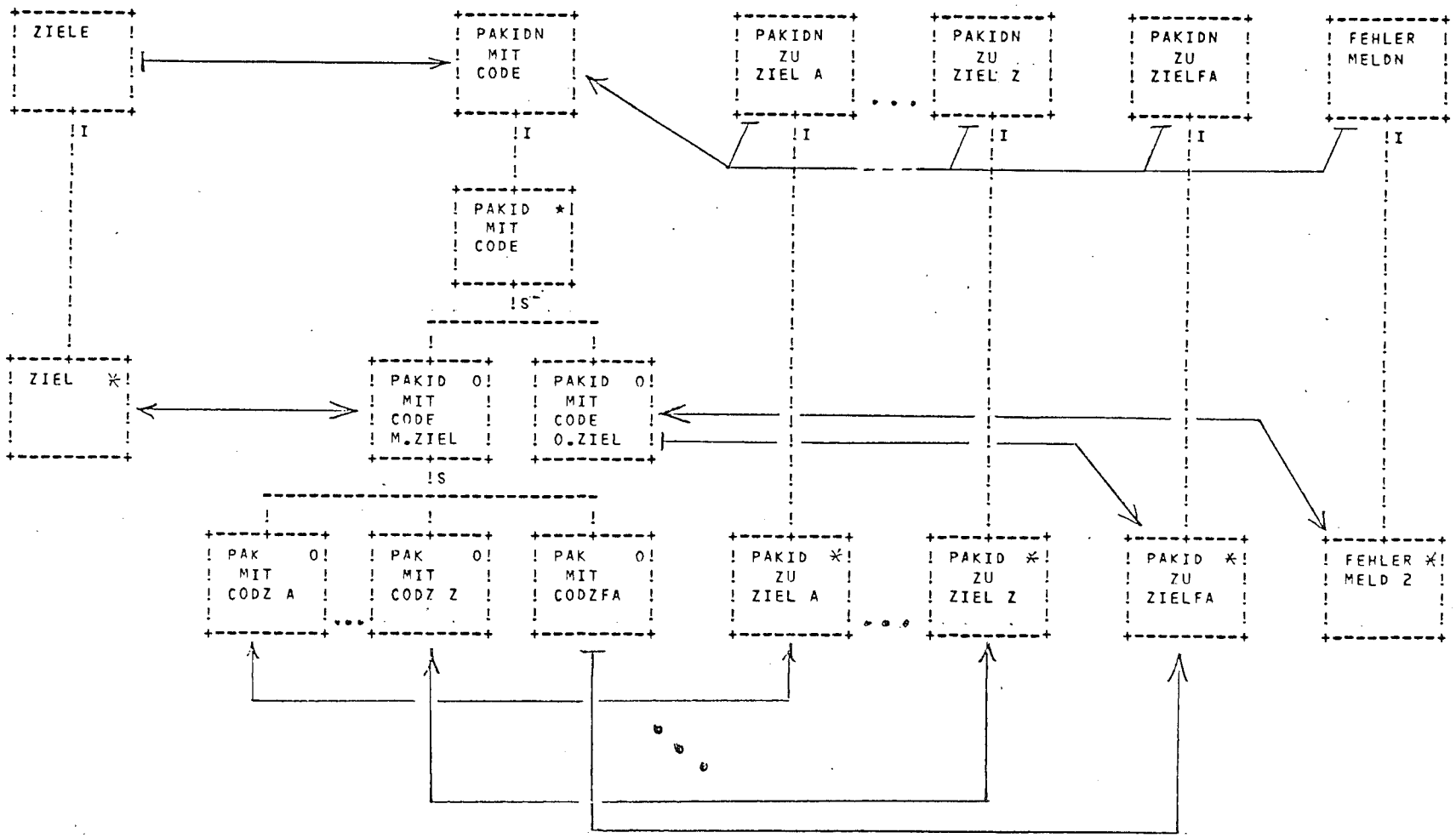
Anmerkung 1:
Nach der Bearbeitung durch den VERTEILER FÜR ZIELE ist jede Paketidentifikation mit ihrem Ziel und o.B.d.A der Wegbeschreibung dorthin versehen.

Anmerkung 2:
Eine Änderung der Code - Ziel - Zuordnung im System betrifft nur den VERTEILER FÜR ZIELE bzw. die von ihm verwalteten Daten.

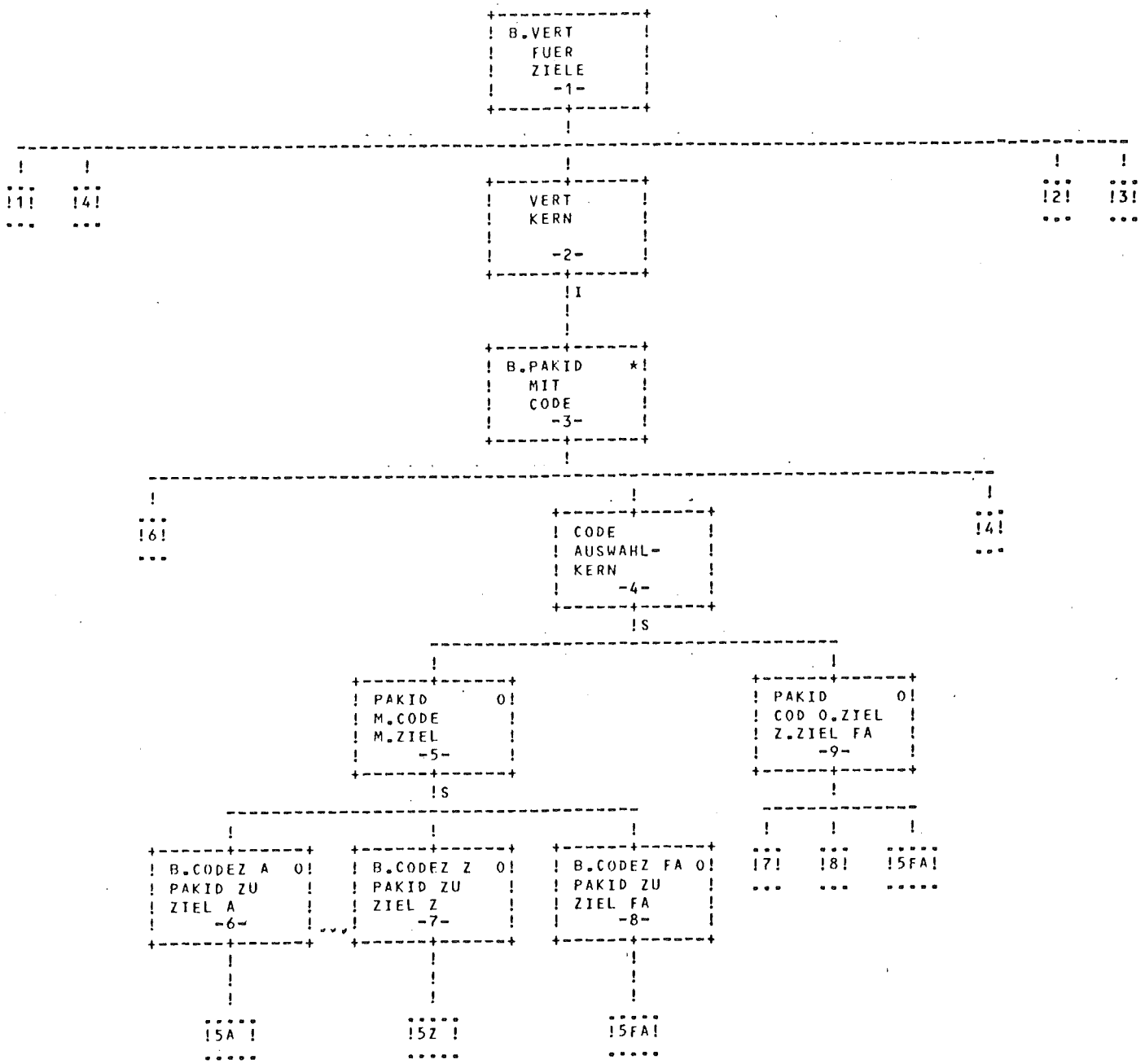
Nachfolgend ist der Entwurf für den VERTEILER FÜR ZIELE angegeben.

. . .

DATENSTRUKTUR -VERTEILER FUER ZIELE- [DVERFZ]



PROGRAMMSTRUKTUR -VERTEILER FUER ZIELE- [PVERFZ]



OPERATIONENLISTE

1 anfangsbearbeitung fuer datenobjekte durchfuehren
(datenobjekte sind - paketidentifikationen mit code
- ziele mit wegbeschreibungen
- fehlermeldungen 2
- paketidentifikationen zu ziel a,...,z,fa)

2 endebearbeitung fuer datenobjekte (siehe 1) durchfuehren

3 programm ston

4 lies paketidentifikationen mit code

5A schreibe paketidentifikationen zu ziel a nachdem ziel zu code festgestellt

. . .

5Z schreibe paketidentifikationen zu ziel z nachdem ziel zu code festgestellt

5FA schreibe paketidentifikationen zu ziel fa nachdem ziel zu code festgestellt

6 lies ziel zu paketidentifikationen mit code

7 fehlerbehandlung durchfuehren

8 fehlermeldung 2

```
ITERATIONSBEDINGUNGEN
  -2- while Paketidentifikation mit Code vorhanden
SELEKTIONSBEDINGUNGEN
  -4- Ziel zu Codezeichen ( nicht ) vorhanden
  -5- Codezeichen A ( ...Z, FA )
=============================================================================
KNOTENBESCHREIBUNG
-1- B.VERTEILER FUER ZIELE
-2- VERTEILER-KERN
-3- B.PAKETIDENTIFIKATION MIT CODE
-4- B.PAKETIDENTIFIKATION MIT CODE UND MIT ZIEL
-5- B.CODE-AUSWAHL
-6- B.CODEZEICHEN A FUER PAKET-ID ZUM ZIEL A
-7- B.CODEZEICHEN Z FUER PAKET-ID ZUM ZIEL Z
-8- B.CODEZEICHEN FA FUER PAKET-ID ZUM ZIEL FA
-9- B.PAKETIDENTIFIKATION MIT CODE OHNE ZIEL ( FEHLERMELDUNG 2 )
=============================================================================
```

SCHEMATIC LOGIC

```
B.VERT FUER ZIELE * SEQ *                                          -1-
   anfangsbearbeitung fuer datenobjekte durchfuehren               (1)
   lies paketidentifikation mit code                               (4)
   B.VERT-KERN * ITER * while Paket-ID mit Code vorhanden          -2-
      B.PAKID MIT CODE * SEQ *                                     -3-
        lies ziel zu paketidentifikation mit code                  (6)
        B.CODE-AUSWAHL KERN * SELECT * Ziel zu Codez vorhanden     -4-
          B.PAKID MIT CODE U.M.ZIEL * SELECT * Codezeichen A       -5-
            B.CODEZ ZU ZIEL A * SEQ *                              -6-
              schreibe paketidentifikation zu ziel a               (5A)
            B.CODEZ ZU ZIEL A * END *
              . . .                                                .
              . . .                                                .
              . . .                                                .
          B.PAKID MIT CODE U.M.ZIEL * OR * Codezeichen Z           -5-
            B.CODEZ ZU ZIEL Z * SEQ *                              -7-
              schreibe paketidentifikation zu ziel z               (5Z)
            B.CODEZ ZU ZIEL Z * END *
          B.PAKID MIT CODE U.M.ZIEL * OR * Codezeichen FA          -5-
            B.CODEZ ZU ZIEL FA * SEQ *                             -7-
              schreibe paketidentifikation zu ziel fa              (5FA)
            B.CODEZ ZU ZIEL FA * END *
          B.PAKID MIT CODE U.M.ZIEL * END *
        B.CODE-AUSWAHL KERN * OR * Ziel zu Codez nicht vorhanden   -4-
          B.PAKID MIT CODE O.ZIEL ( FEHLMELD 2 ) * SEQ *           -9-
            fehlerbehandlung durchfuehren                          (7)
            fehlermeldung 2                                        (8)
            schreibe paketidentifikation zu ziel fa                (5FA)
          B.PAKID MIT CODE O.ZIEL ( FEHLMELD 2 ) * END *
        lies paketidentifikation mit code                          (4)
      B.PAKID MIT CODE * END *
   B.VERT-KERN * END *
   endebearbeitung fuer datenobjekte durchfuehren                  (2)
   programm stop                                                   (3)
B.VERT FUER ZIELE * END *
```

4.3. Verteiler für Paket-Identifikationen zu einem Ziel

Einen Prozeß VERTEILER FÜR PAKETE-IDN ZU EINEM ZIEL gibt es für jede zielspezifische Menge von Paketidentifikationen, die der VERTEILER FÜR ZIELE erzeugt.

Alle diese Prozesse sind ähnlich, ein Repräsentant wird nachfolgend betrachtet.

In Bild 4 (s.u.) ist die Übersicht dargestellt; zur Strukturierung sind nur die Datenstrukturen angegeben. Die triviale Funktion des Prozesses ist die Auflösung der Iteration von Paketidentifikationen in die Sequenz dieser Einzeldaten.

Anmerkung:

Die Funktion eines solchen Prozesses kann jeweils der Paket-Idn-Ausgabeoperation des VERTEILERS FÜR ZIELE zugeordnet werden.

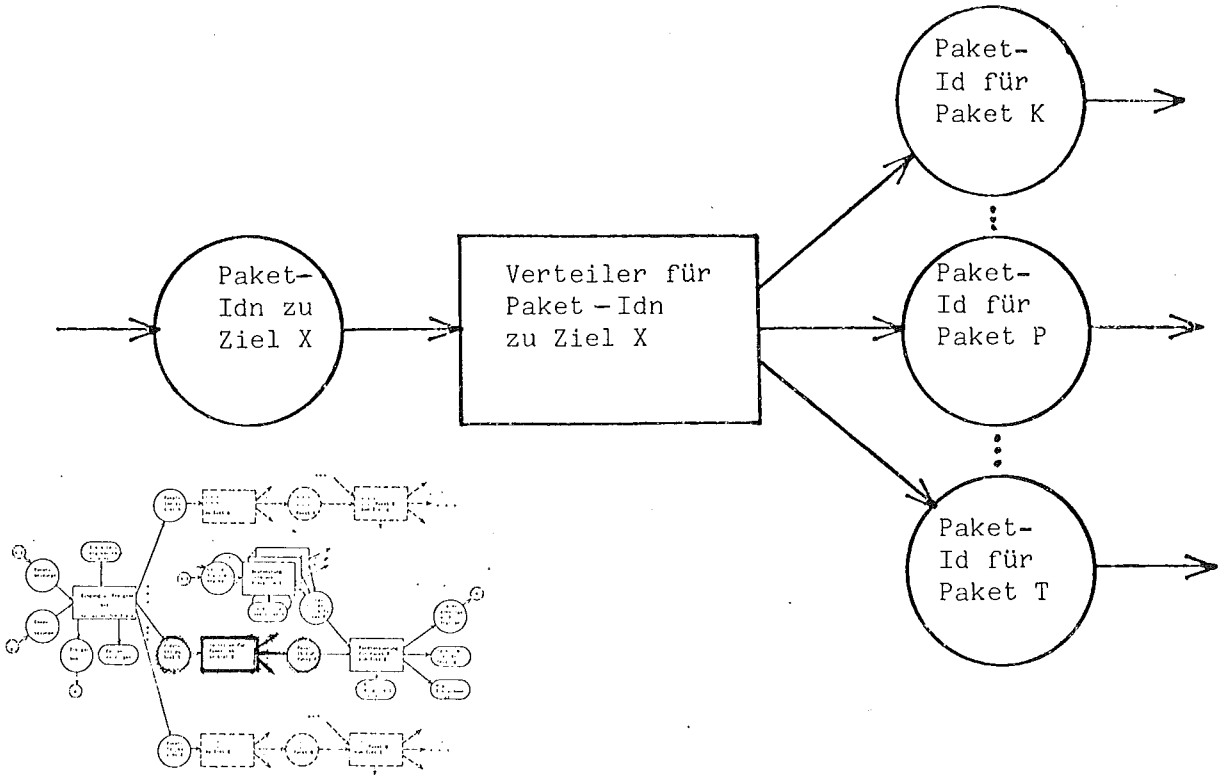
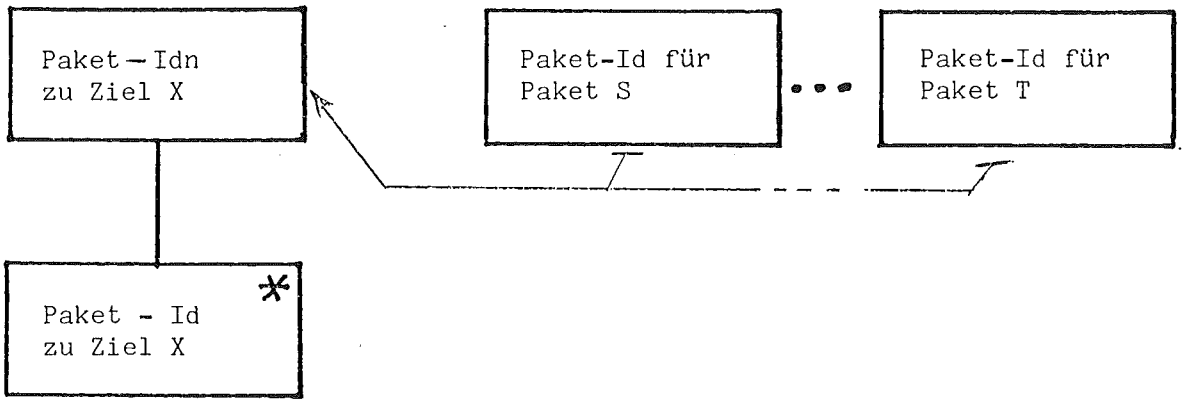


Bild 4 : Übersicht zu VERTEILER FÜR PAKETE-IDENTIFIKATIONEN ZU EINEM ZIEL



Datenstrukturen zu VERTEILER FÜR PAKET-IDN

4.4. Paketsteuerung für ein Paket

Entsprechend den Grundgedanken des Entwurfs wird für jedes Paket ein eigener Steuerprozeß zu einem Ziel beschrieben - er wickelt die Lebensgeschichte des Paketes bzw. dessen Identifikation in dem Verteilsystem ab. Diese Prozesse bilden eine Klasse "ähnlicher" Prozesse. Sie unterscheiden sich "nur" durch Paketidentifikation und Ziel bzw. Wegbeschreibung.
Im folgenden wird der Entwurf eines Repräsentanten dieser Klasse beschrieben, des Prozesses PAKETSTEUERUNG FÜR PAKET P ZUM ZIEL X, kurz PAKETPROZESS genannt (vgl. Bild 5).

Der PAKETPROZESS bearbeitet die Paketidentifikation des Paketes P. Er erzeugt Steueranweisungen an die Verteilstationen (V-Stationen) längs des Weges zum Ziel X, meldet die Zielankunft des Paketes P und veranlaßt mögliche Zielauswertungen (Fehlerfälle usf.). Um Steueranweisungen (Schaltaufträge) an die V-Stationen erzeugen zu können, benötigt der PAKETPROZESS Informationen der einzelnen V-Stationen. Er liest diese stationsspezifischen Zustandsdaten (Freizustand und Schaltzustand der Station, sowie Vorgängerwegbeschreibung) zusammen mit der Information, daß das Paket P die Eingangslichtschranke der Station erreicht hat. Diese Daten werden von V-Stationsprozessen (siehe 4.5.) erzeugt, und zwar genau von denjenigen, die den Weg zum Ziel X abdecken (vgl. Bild 5).
Die Information über den Durchgang, auch die Ausgangslichtschranke eines Verteilers, kann zur Wegkontrolle (z.B. HW-Fehler) benutzt werden.

Die Kernfunktion des PAKETPROZESSES - Bearbeitung der V-Stationsdaten und Ableitung der Schaltaufträge - ist iteriert über der Wegbeschreibung zum Ziel X. Der PAKETPROZESS meldet die Zielankunft und beendet sich nach Bearbeitung der letzten V-Station.

Der mögliche Fehlerfall - Fehlläufer infolge Auflaufens auf Vorgänger mit anderem Teilziel in einer V-Station - wird im Entwurf durch "Systembacktracking" beschrieben: der akutelle PAKETPROZESS bricht ab und wird durch einen neuen PAKETPROZESS zum Ziel des Vorgängers abgelöst, der genau in der kritischen V-Station seine Funktion übernimmt.

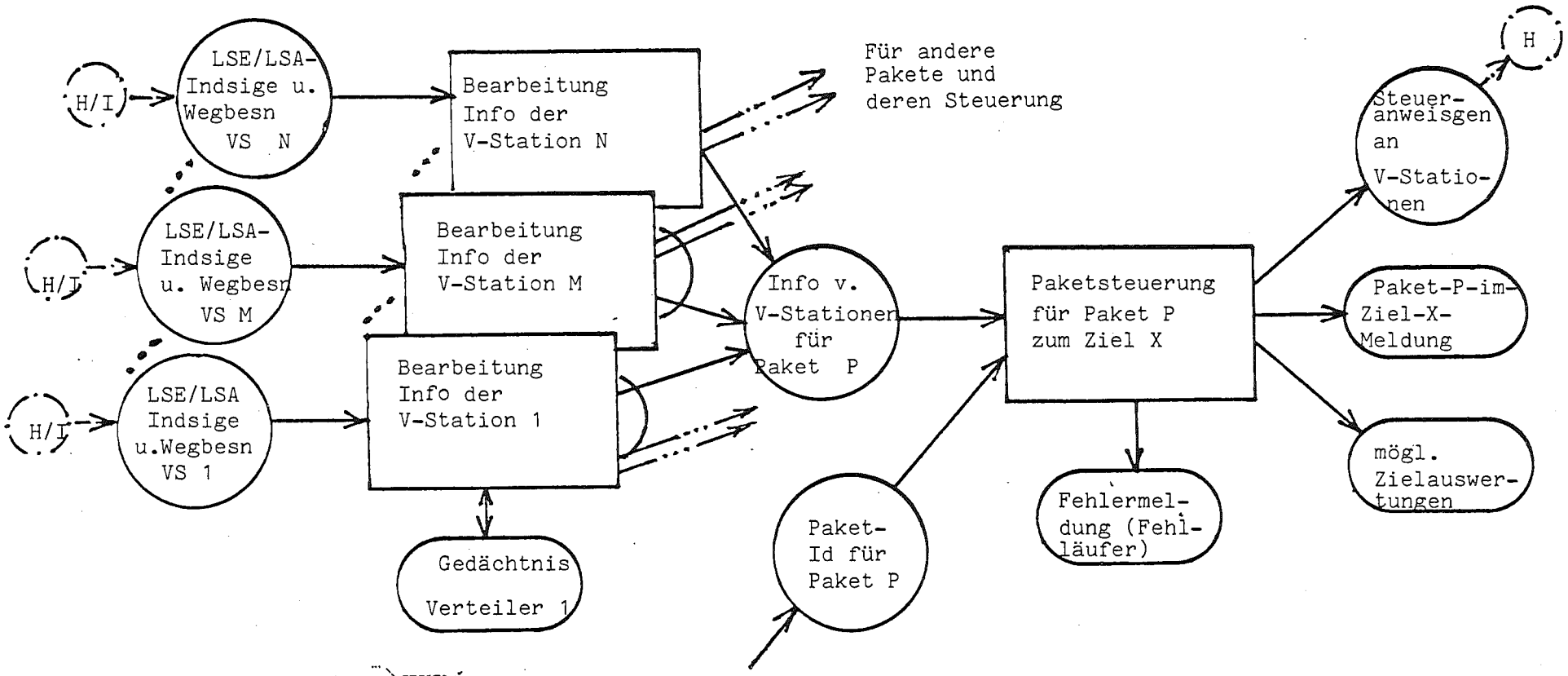
. . .

Anmerkung 1: Realisiert wird das Systembacktracking durch Überschreiben des Ziels und des restlichen Weges in der Paketidentifikation (mit zusätzlichem Vermerk über das Fehlverhalten).

Anmerkung 2:
Die Operationen "lies Information LS-Eingang" und "lies Information LS-Ausgang" müssen im realisierten System koordiniert sein zum Paketdurchlauf der Lichtschranken; sie haben also eine "Warte auf ..."-Funktion bzw. stellen Anstoß-Stellen des Prozesses dar; folglich müssen dann - falls Pakete im System sind - Warteschlangen geführt werden.

Nachfolgend ist der Entwurf für den PAKETPROZESS angegeben.

. . .



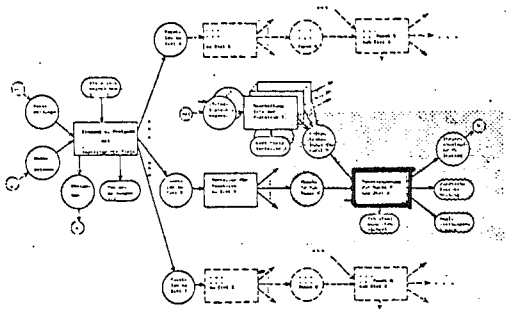
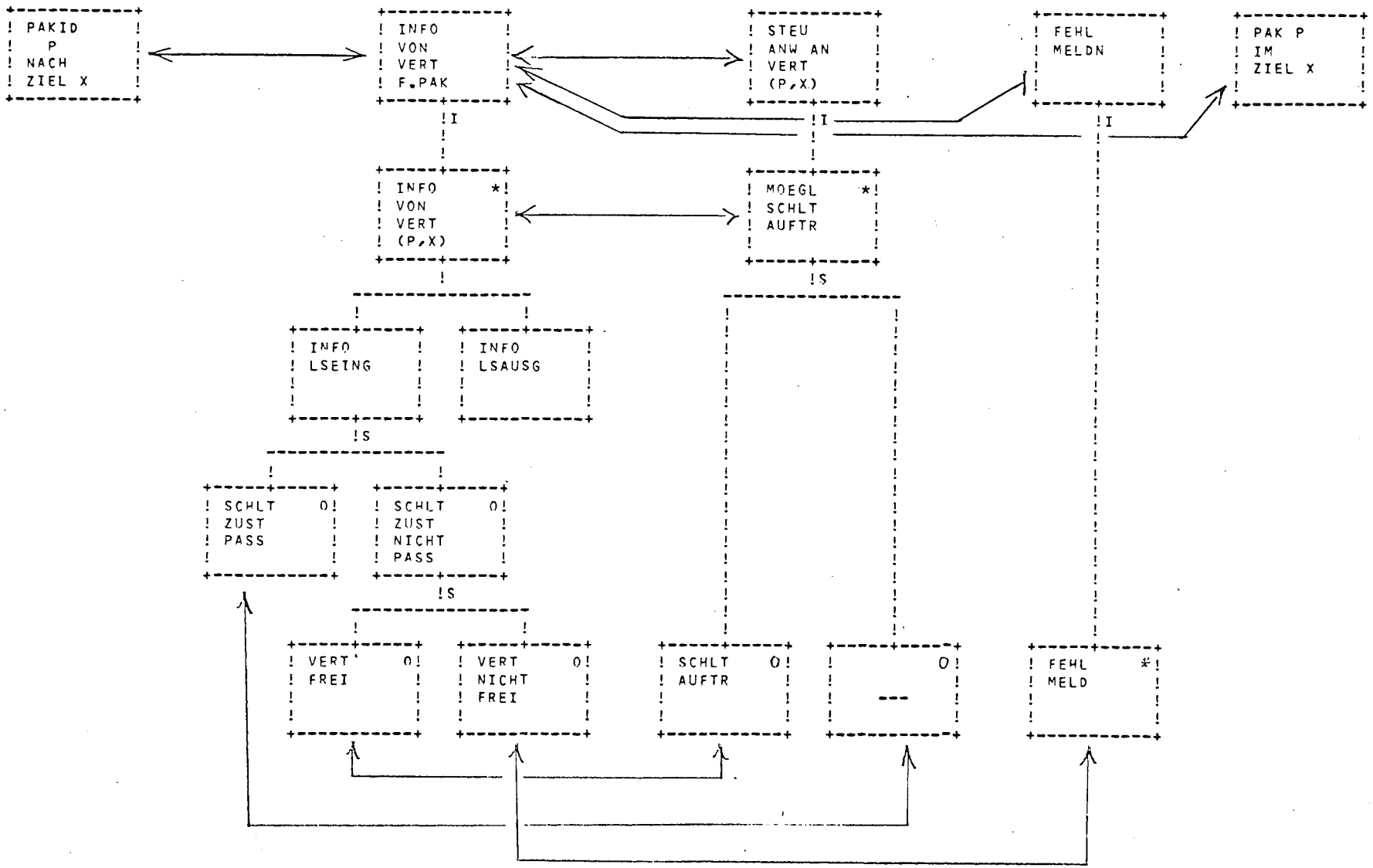
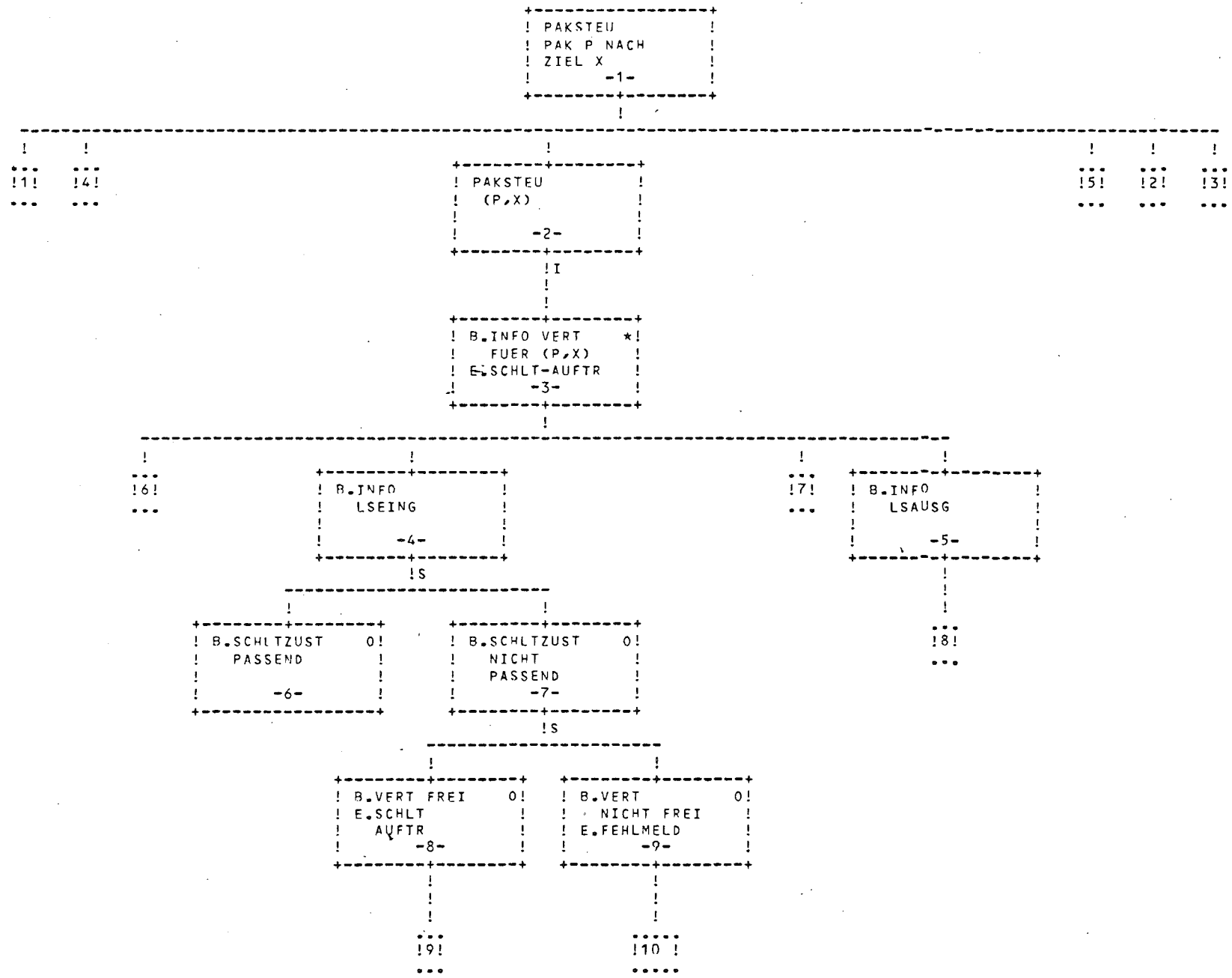


Bild 5 : Übersicht zu PAKETSTEUERUNG FÜR EIN PAKET

PAKETSTEUERUNG FUER EIN PAKET P ZU EINEM ZIEL X [DPAKST]



PROGRAMMSTRUKTUR MIT OPERATIONEN FUER PAKETSTEUERUNG [PPAKST]



```
==========================================================================
OPERATIONENLISTE
 1 anfangsbearbeitung fuer datenobjekte durchfuehren
   ( datenobjekte sind - paketidentifikation zum ziel
                       - informationen vom verteiler
                           ( information ls-eingang, information ls-ausgang )
                       - schaltauftraege
                       - fehlermeldungen
                       - zielankunftsmeldung  )
 2 endebearbeitung fuer datenobjekte ( siehe 1 ) durchfuehren
 3 programm stop
 4 lies paketidentifikation von paket p fuer ziel x
 5 melde paket p im ziel x
   ( gegebenenfalls mit fehlerauswertung: paket war fehllaeufer ( auswertung
     der paket-id ) )
 6 lies information ls-eingang
   ( enthaelt: schaltzustand des verteilers
               verteiler frei / nicht frei
               wegbeschreibung des vorgaengers )
 7 lies information ls-ausgang
 8 bearbeite information ls-ausgang
 9 erteile schaltauftrag
10 fehlermeldung ( fehllaeufer )
==========================================================================
ITERATIONSBEDINGUNGEN
  -2- while Wegbeschreibung nicht zuende
SELEKTIONSBEDINGUNGEN
  -4- Schaltung ( nicht ) richtig
  -7- Verteiler ( nicht ) frei
==========================================================================
KNOTENBESCHREIBUNG
-1- B.PAKETSTEUERUNG FUER PAKET P NACH ZIEL X
-2- B.PAKETSTEUERUNGS-KERN ( PAK P, ZIEL X )
-3- B.INFORMATION VON EINEM VERTEILER FUER ( PAK P, ZIEL X )
    E.SCHALTAUFTRAG
-4- B.INFORMATION VON LS-EINGANG
-5- B.INFORMATION VON LS-AUSGANG
-6- B.SCHALTZUSTAND IST PASSEND
-7- B.SCHALTZUSTAND IST NICHT PASSEND
-8- B.VERTEILER IST FREI & E.SCHALTAUFTRAG
-9- B.VERTEILER IST NICHT FREI & E.FEHLERMELDUNG
==========================================================================
```

SCHEMATIC LOGIC

```
B.PAKSTE PAK P NACH ZIEL X * SEQ *                                  -1-
   anfangsbearbeitung fuer datenobjekte durchfuehren                (1)
   lies paketidentifikation vom paket p fuer ziel x                 (4)
   PAKSTEU (P,X) * ITER * while Wegbeschreibung nicht zuende        -2-
      B.INFO VERT FUER (P,X) * SEQ *                                -3-
         lies information ls-eingang                                (6)
         B.INFO LSEING * SELECT * Schaltung richtig                 -4-
            B.SCHLTZUST PASS * SEQ *                                -6-
            B.SCHLTZUST PASS * END *
         B.INFO LSEING * OR * Schaltung nicht richtig               -4-
            B.SCHLTZUST NICHT PASS * SELECT * Verteiler frei        -7-
               B.VERT FREI, E.SCHLTAUFTR * SEQ *                    -8-
                  erteile schaltauftrag an verteilstation           (9)
               B.VERT FREI, E.SCHLTAUFTR * END *
            B.SCHLTZUST NICHT PASS * OR * Verteiler nicht frei      -7-
               B.VERT NICHT FREI, E.FEHLERMELD * SEQ *              -9-
System-           fehlermeldung ( fehllaeufer )                     (10)
backtracking ==>  B.VERT NICHT FREI, E.FEHLERMELD * END *
            B.SCHLTZUST NICHT PASS * END *
         B.INFO LSEING * END *
         lies information ls-ausgang                                (7)
         B.INFO LSAUSG * SEQ *                                      -5-
            bearbeite information ls-ausgang                        (8)
         B.INFO LSAUSG * END *
      B.INFO VERT * END *
   PAKSTEU (P,X) * END *
   melde paket p im ziel x                                          (5)
   endebearbeitung fuer datenobjekte durchfuehren                   (2)
   programm stop                                                    (3)
B.PAKSTEU PAK P NACH ZIEL X * END *
```

4.5. Bearbeitung einer Verteilstation

Jeder einzelnen Verteilstation (V-Station) ist ein Prozeß zugeordnet, der die Informationen dieser Station bearbeitet. Diese Prozesse bilden eine Klasse "ähnlicher" Prozesse, sie unterscheiden sich "nur" durch die Stationszuordnung. Der Entwurf eines Repräsentanten dieser Klasse, des Prozesses BEARBEITUNG INFO DER V-STATION M, kurz STATIONSPROZESS, wird nachfolgend beschrieben (siehe Bild 6).

Der STATIONSPROZESS liest die Lichtschrankensignale des Eingangs LSE und des Ausgangs LSA der V-Station M. Er muß ein Gedächtnis für seine V-Station führen und zwar über die Paketanzahl innerhalb der Station und über die Wegbeschreibung der jeweils letzten Paketidentifikation. Der STATIONSPROZESS benötigt folglich diese Wegbeschreibungen; sie können o.B.d.A. als an die LSE-Information gekoppelt angesehen werden.

Ein STATIONSPROZESS muß jeweils paarweise die LSE- und LSA-Information für die bei ihm aktuellen Pakete ausgeben (zur Weiterbearbeitung durch deren PAKETPROZESS). Ein solcher Informationsblock enthält als Daten

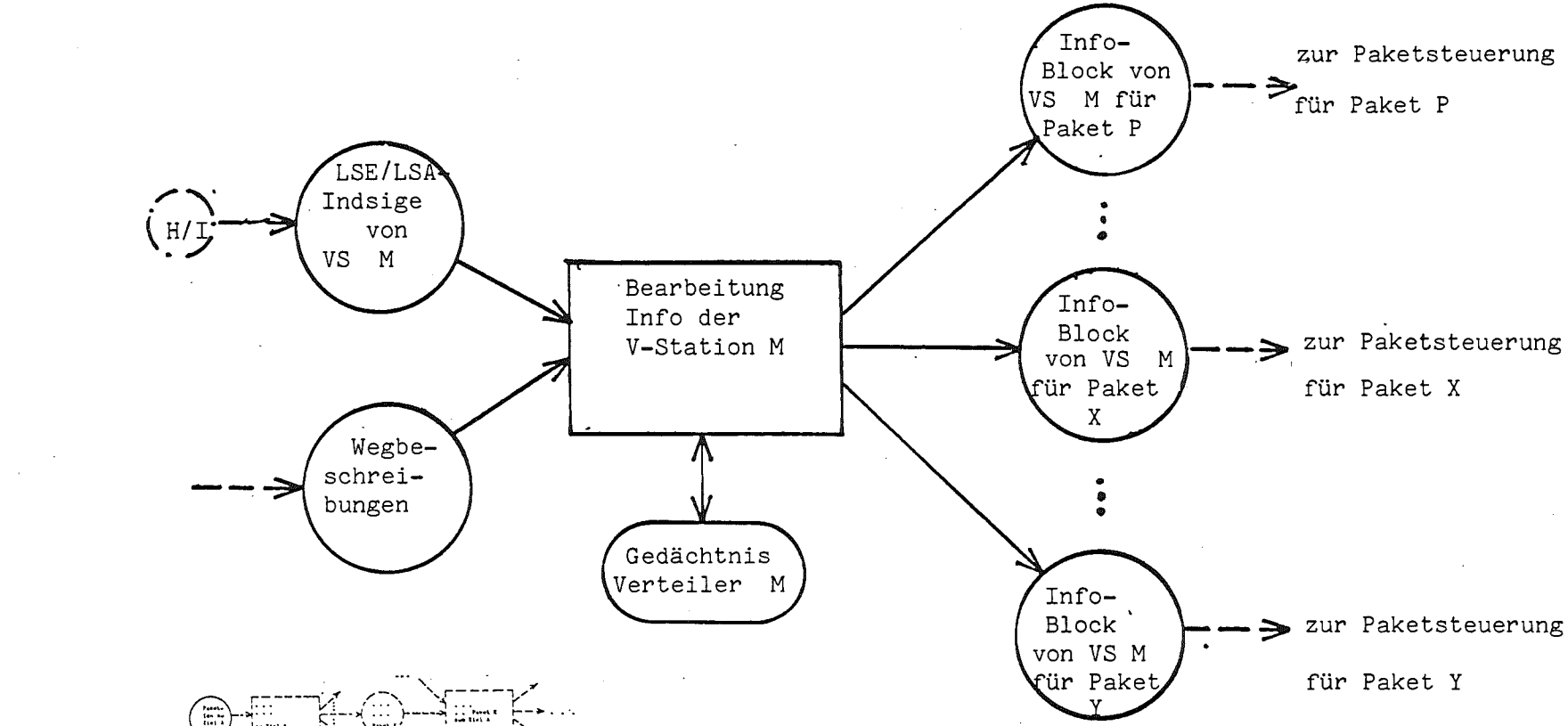
- o LSE-Signal gekommen
  - Freizustand der V-Station (leer/belegt)
  - Schaltzustand der V-Station (linkes/rechtes Teilziel)
  - Vorgängerwegbeschreibung

- o LSA-Signal gekommen
  - (- evtl. Ausgangsseite des Paketes, zur Überprüfung der Steuerung)

Die Korrespondenzanalyse der Datenstrukturen (DVERT2) zeigt eine Nicht-Korrespondenz, da die LS-Signale nicht paarweise für ein Paket eintreffen, falls sich mehrere Pakete innerhalb der Station befinden. Dieser Einschubkonflikt (interleaving-clash) wird aufgelöst durch Verteilung in einzelne Datenmengen, je Paketidentifikation, siehe DVERT3.

Anmerkung: Entsprechend der Anmerkung 2 zu 4.4. (PAKETPROZESS) stellen die Schreib-Operationen für die Info-Blöcke bei Realisierung des Steuermodells die logischen Anstöße der PAKETPROZESSE dar.

. . .



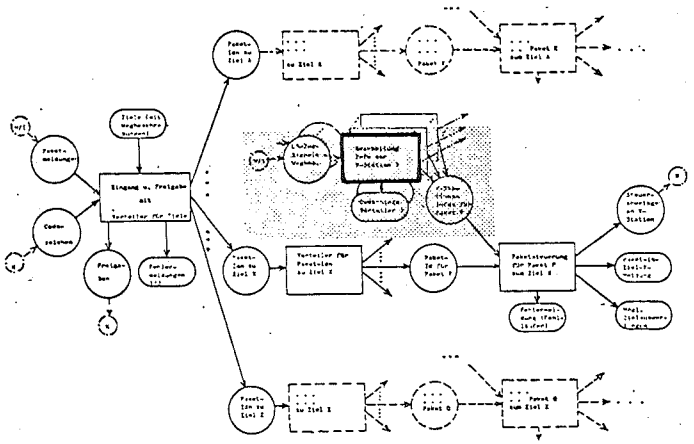
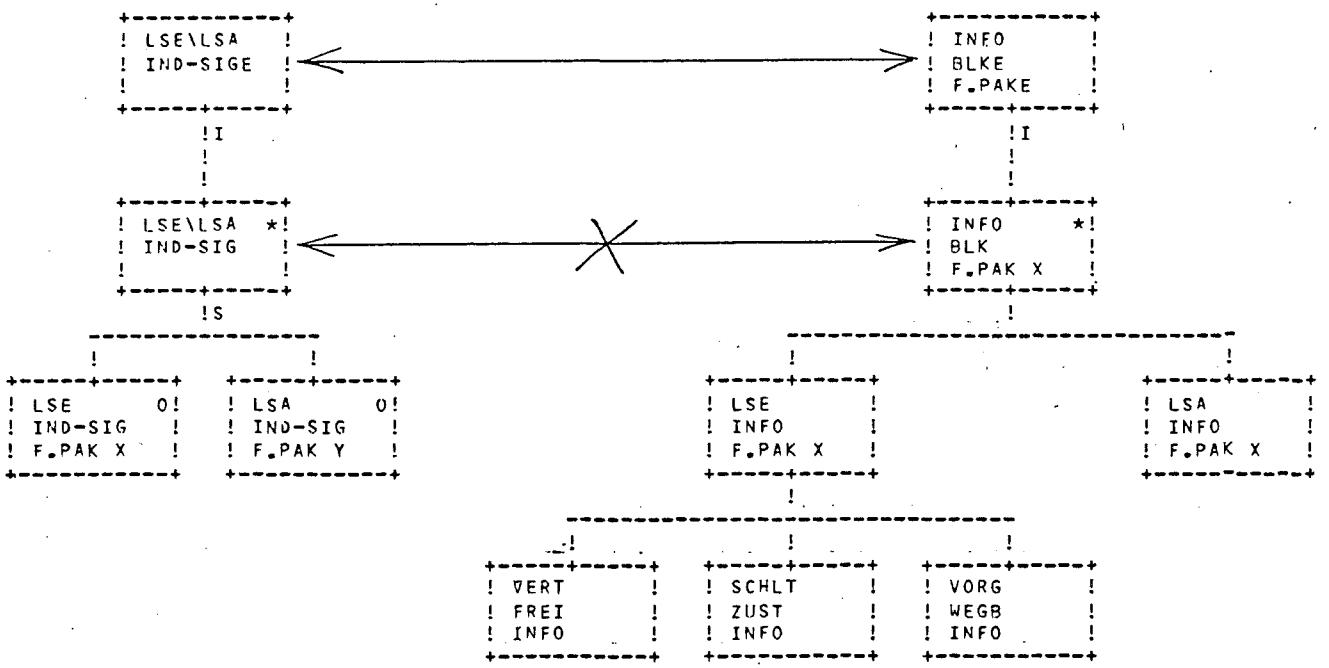
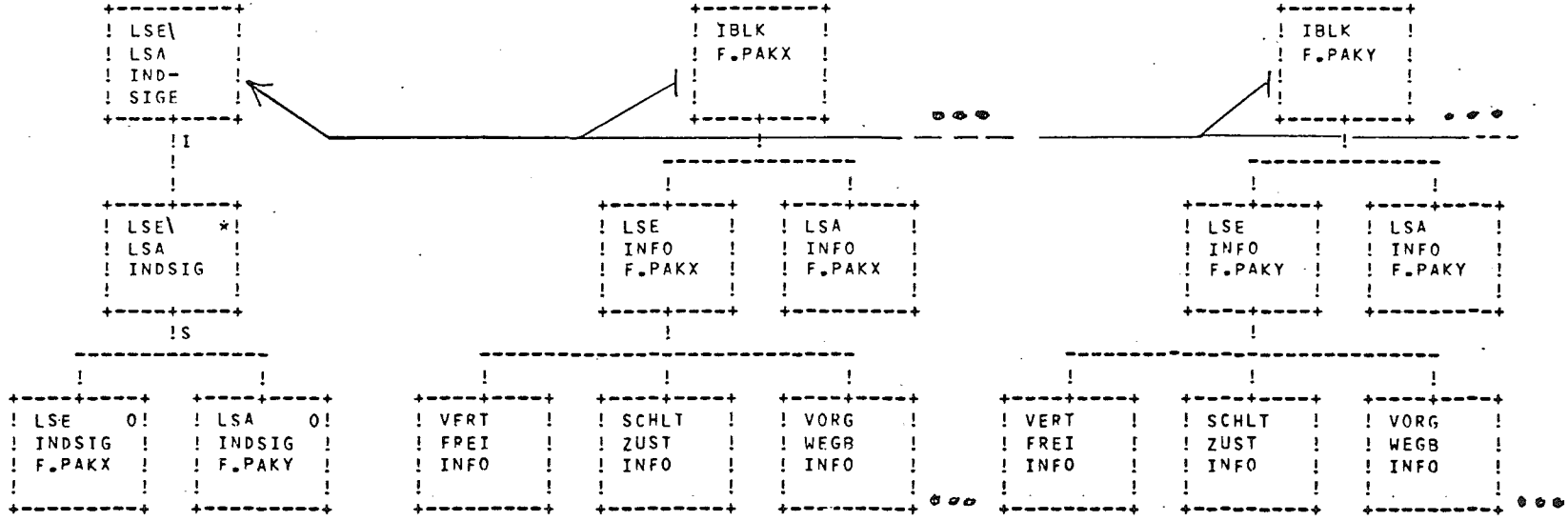


Bild 6: Übersicht zur BEARBEITUNG EINER VERTEIL-STATION

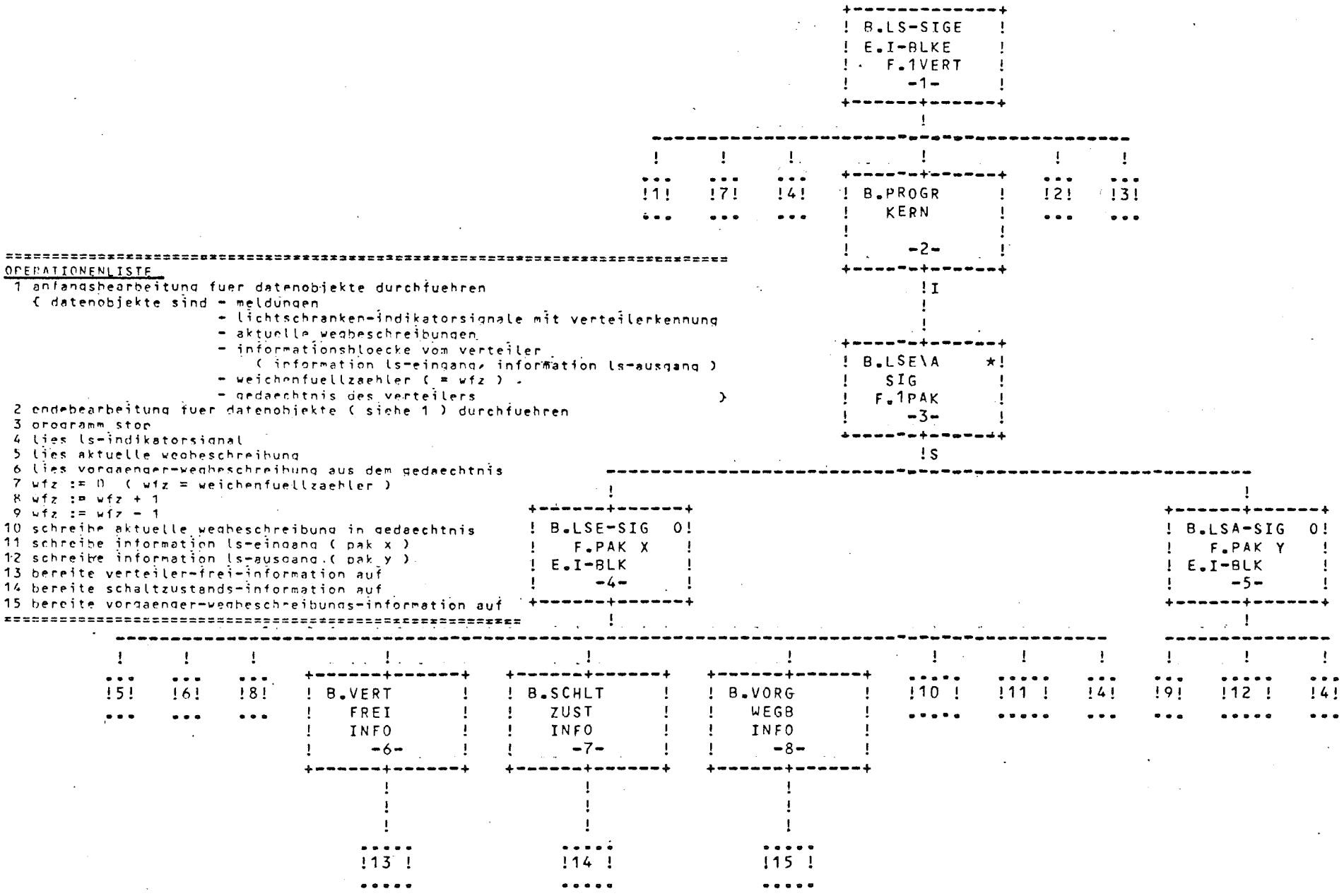
DATENSTRUKTUREN FUER EIN " PROGRAMM FUER EINEN VERTEILER " [DVERT2]



DATENSTRUKTUREN FUER EIN " PROGRAMM FUER EINEN VERTEILER " [DVERT3]



PROGRAMMSTRUKTUR MIT OPERATIONEN FUER VERTEILERPROGRAMM [PVERT2]



OPERATIONENLISTE

1 anfangsbearbeitung fuer datenobjekte durchfuehren
(datenobjekte sind - meldungen
- lichtschranken-indikatorsignale mit verteilerkennung
- aktuelle wegbeschreibungen
- informationsbloecke vom verteiler
(information ls-eingang, information ls-ausgang)
- weichenfuellzaehler (= wfz) .
- gedaechtnis des verteilers)
2 endebearbeitung fuer datenobjekte (siehe 1) durchfuehren
3 programm stop
4 lies ls-indikatorsignal
5 lies aktuelle wegbeschreibung
6 lies vorgaenger-wegbeschreibung aus dem gedaechtnis
7 wfz := 0 (wfz = weichenfuellzaehler)
8 wfz := wfz + 1
9 wfz := wfz - 1
10 schreibe aktuelle wegbeschreibung in gedaechtnis
11 schreibe information ls-eingang (pak x)
12 schreibe information ls-ausgang (pak y)
13 bereite verteiler-frei-information auf
14 bereite schaltzustands-information auf
15 bereite vorgaenger-wegbeschreibungs-information auf

```
================================================================================
OPERATIONENLISTE
 1 anfangsbearbeitung fuer datenobjekte durchfuehren
   ( datenobjekte sind - meldungen
                       - lichtschranken-indikatorsignale mit verteilerkennung
                       - aktuelle wegbeschreibungen
                       - informationsbloecke vom verteiler
                           ( information ls-eingang, information ls-ausgang )
                       - weichenfuellzaehler ( = wfz )
                       - gedaechtnis des verteilers                          )
 2 endebearbeitung fuer datenobjekte ( siehe 1 ) durchfuehren
 3 programm stop
 4 lies ls-indikatorsignal
 5 lies aktuelle wegbeschreibung
 6 lies vorgaenger-wegbeschreibung aus dem gedaechtnis
 7 wfz := 0  ( wfz = weichenfuellzaehler )
 8 wfz := wfz + 1
 9 wfz := wfz - 1
10 schreibe aktuelle wegbeschreibung in gedaechtnis
11 schreibe information ls-eingang ( pak x )
12 schreibe information ls-ausgang ( pak y )
13 bereite verteiler-frei-information auf
14 bereite schaltzustands-information auf
15 bereite vorgaenger-wegbeschreibungs-information auf
================================================================================
ITERATIONSBEDINGUNGEN
  -2- while LS-Indikatorsignal vorhanden
SELEKTIONSBEDINGUNGEN
  -3- LSE- ( LSA- ) Indikatorsignal
================================================================================
KNOTENBESCHREIBUNG
-1- B.LSE- UND LSA-INDIKATORSIGNALE & E.INFORMATIONSBLOECKE FUER PAKETE IN
      EINEM VERTEILER
-2- B.PROGRAMMKERN * ITER * until eof-Meldung
-3- B.LSE- UND LSA-INDIKATORSIGNAL * SELECT * LSE- oder LSA-Indikatorsignal
-4- B.LSE-INDIKATORSIGNAL FUER PAKET X & E.INFORMATIONSBLOCK FUER PAKET X
-5- B.LSE-INDIKATORSIGNAL FUER PAKET Y & E.INFORMATIONSBLOCK FUER PAKET Y
-6- B.VERTEILER-FREI-INFORMATION
-7- B.SCHALTZUSTANDS-INFORMATION
-8- B.VORGAENGER-WEGBESCHREIBUNG-INFORMATION
================================================================================
```

SCHEMATIC LOGIC

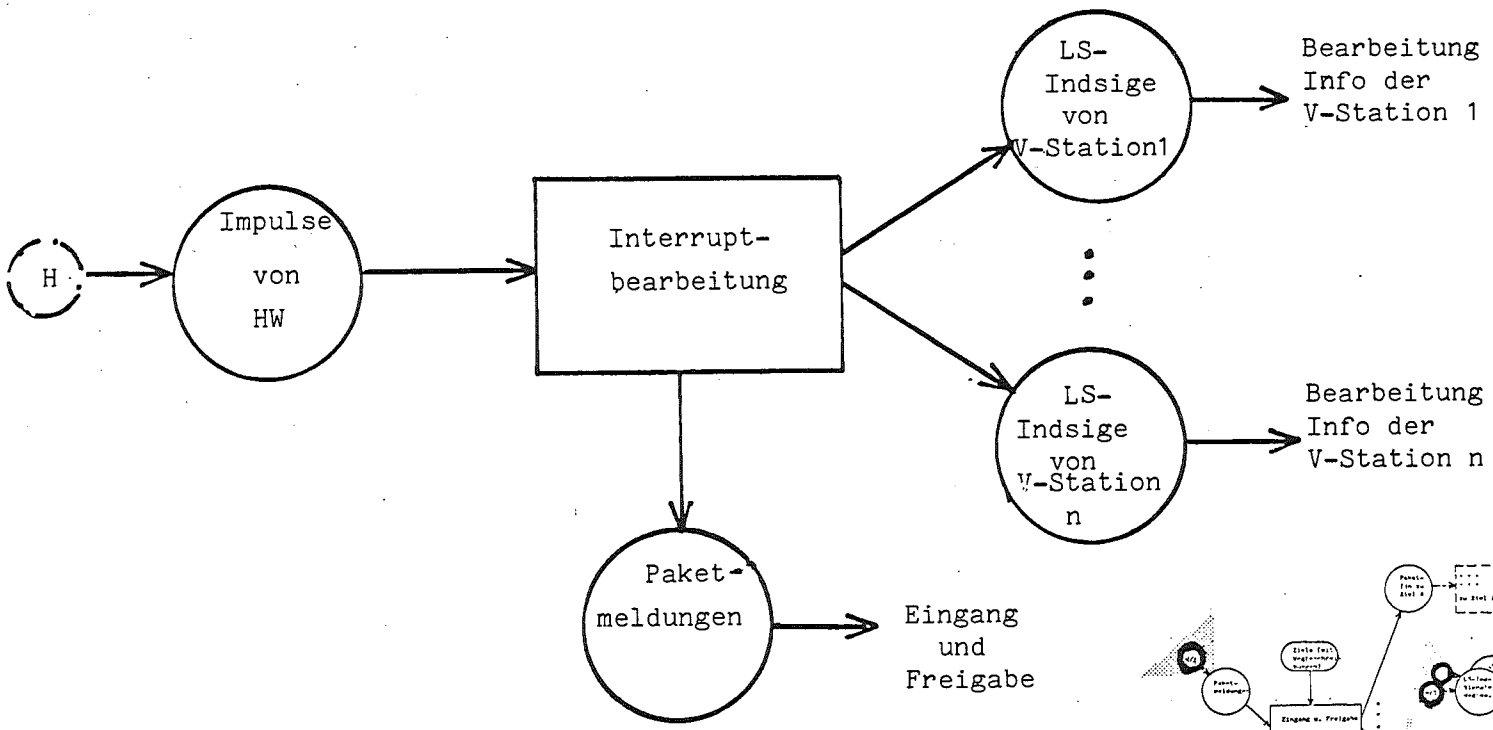
```
B.LSE\A IND-SIGE & E.I-BLKE F.1VERT * SEQ *                          -1-
  anfangsbearbeitung durchfuehren                                    (1)
  wfz := 0                                                           (7)
  lies ls-indikatorsignal                                            (4)
  B.PROGR.KERN * ITER * while LS Ind-Sig                             -2-
    B.LSE\A IND-SIG F.1PAK * SELECT * LSE Ind-Sig                    -3-
      B.LSE IND-SIG F.PAK X & E.I-BLK * SEQ *                        -4-
        lies aktuelle wegbeschreibung                                (5)
        lies vorgaengerwegbeschreibung aus dem gedaechtnis           (6)
        wfz := wfz + 1                                               (8)
        B.VERT-FREI-INFO * SEQ *                                     -6-
          bereite verteiler-frei-info auf                           (13)
        B.VERT-FREI-INFO * END *
        B.SCHLTZUST-INFO * SEQ *                                     -7-
          bereite schaltzustands-info auf                           (14)
        B.SCHLTZUST-INFO * END *
        B.VORG.WEGB-INFO * SEQ *                                     -8-
          bereite vorgaenger-wegbeschreibungs-information auf       (15)
        B.VORG.WEGB-INFO * END *
        schreibe aktuelle wegbeschreibung in gedaechtnis             (10)
        schreibe information ls-eingang ( pak x )                    (11)
        lies ls-indikatorsignal                                      (4)
      B.LSE IND-SIG F.PAK X & E.I-BLK * END *
    B.LSE\A IND-SIG F.1PAK * OR * LSA Ind-Sig                        -3-
      B.LSA IND-SIG F.PAK Y & E.I-BLK                                -5-
        wfz := wfz - 1                                               (9)
        schreibe information ls-ausgang ( pak y )                   (12)
        lies ls-indikatorsignal                                      (4)
      B.LSA IND-SIG F.PAK Y & E.I-BLK * END *
    B.LSA IND-SIG F.1PAK * END *
  B.PROGR.KERN * END *
  endebearbeitung fuer datenobjekte durchfuehren                     (2)
  programm stop                                                      (3)
B.LSE\A IND-SIGE & E.I-BLKE F.1VERT * END *
```

4.6. Interruptbearbeitung

Wie bereits im Systembild (Bild 2) angedeutet, werden die Hardware-Impulse, wie die Paketmeldungen und die Lichtschranksignale der V-Stationen, u.U. sequentialisiert, einem Prozeß INTERRUPTBEARBEITUNG zugeführt, der diese evtl. aufbereitet und an die zuständigen Prozesse weiterleitet (vgl. Bild 7).

Nachfolgend ist der Entwurf dieses Prozesses angegeben.

. . .



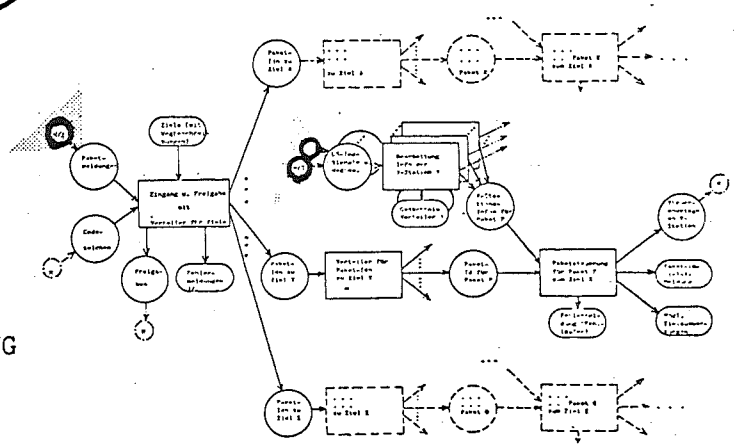
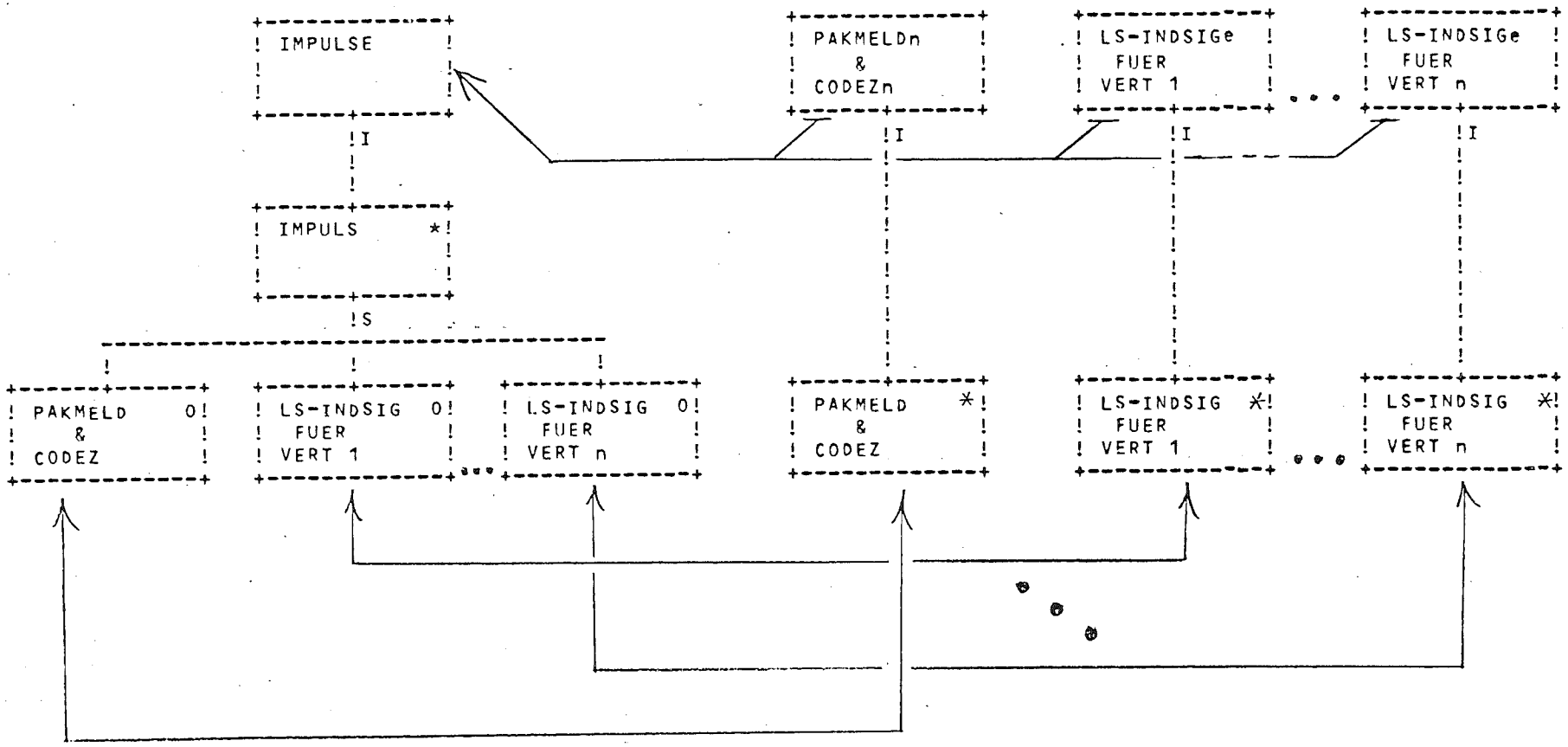


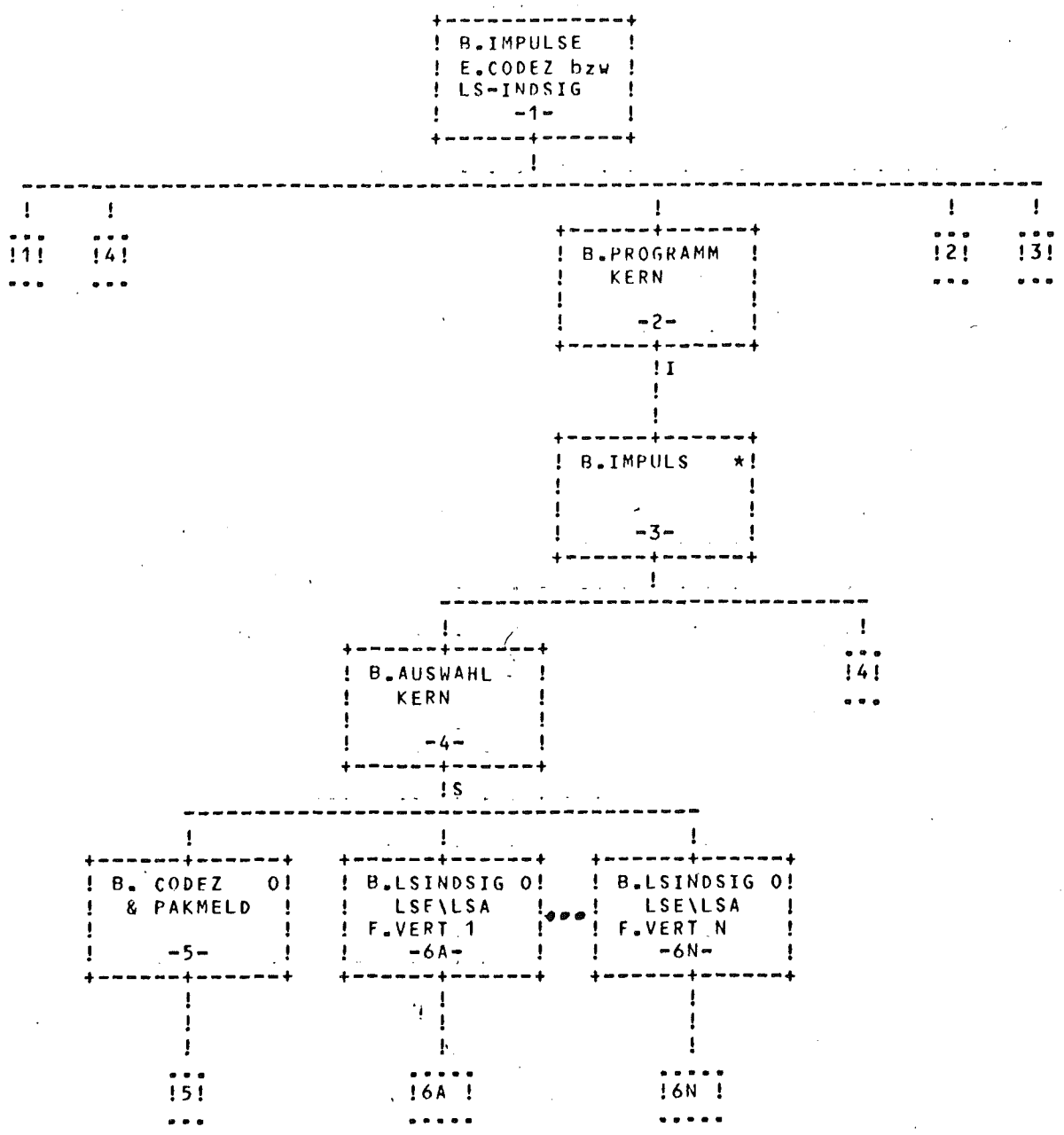
Bild 7 : Übersicht zu INTERRUPTBEARBEITUNG

DATENSTRUKTUREN FUER ERWEITERTEN INTERRUPTHANDLER

[DEINTE]



PROGRAMMSTRUKTUR FUER ERWEITERTEN INTERRUPTHANDLER [PEINTE]



OPERATIONENLISTE

1 anfangsbearbeitung fuer datenobjekte durchfuehren
(datenobjekte sind - impulse
- lichtschranken-indikatorsignale mit verteilerkennung
- paketmeldungen (mit codezeichen) ausgang)
2 endebearbeitung fuer datenobjekte (siehe 1) durchfuehren
3 programm stop
4 lies impuls
5 bereite paketmeldung auf und schreibe sie
6A bereite lichtschranken-indikatorsignal fuer verteiler 1
auf (lse oder lsa) und schreibe lichtschranken-indikatorsignal
. . .
. . .
. . .
6N bereite lichtschranken-indikatorsignal fuer verteiler n
auf (lse oder lsa) und schreibe lichtschranken-indikatorsignal

ITERATIONSBEDINGUNGEN
while not eof-Meldung
SELEKTIONSBEDINGUNGEN
Impuls ist Paketmeldung oder Ls-Indikatorsignal fuer verteiler 1
. . .
. . .
. . .
oder Ls-Indikatorsignal fuer verteiler n

SCHEMATIC LOGIC

```
B.IMPULSE, E.PAKMELDN & CODEZ BZW. LS-INDSIGE F.VERT I  * SEQ *          -1-
  anfangsbearbeitung fuer datenobjekte durchfuehren                       (1)
  lies impuls                                                             (4)
  B.PROGRAMMKERN * ITER * while not eof-Meldung                           -2-
    B.IMPULS * SEQ *                                                      -3-
      B.AUSWAHL-KERN * SELECT * Impuls ist Paketmeldung                   -4-
        B.CODE7 & PAKMELD * SEQ *                                         -5-
          bereite paketmeldung auf und schreibe sie                       (5)
        B.CODE7 & PAKMELD * END *
      B.AUSWAHL-KERN * OR * Impuls ist Ls-Indsig fuer Verteiler 1         -4-
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER 1  * SEQ *                -6A-
          bereite ls-indsig fuer verteiler 1 auf und schreibe dieses      (6a)
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER 1  * END *
            . . .
            . . .
            . . .
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER N  * SEQ *                -6N-
          bereite ls-indsig fuer verteiler n auf und schreibe dieses      (6n)
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER n  * END *
      B.AUSWAHL-KERN * END *
      lies impuls                                                         (4)
    B.IMPULS * END *
  B.PROGRAMM-KERN * END *
  endebearbeitung fuer datenobjekte durchfuehren                          (2)
  programm stop                                                           (3)
B.IMPULSE, E.PAKMELDN & CODEZ BZW. LS-INDSIGE F.VERT I  * END *
```

5. Übergang zur Realisierung

Der vorliegende Bericht beschreibt den Entwurf des Steuersystems; nachfolgend werden einige Gedanken zur Realisierung (Implementierung) des Modells dargestellt.

Bei der Realisierung sind die Randbedingungen bzgl. Laufzeit und Speicherplatz sowie das Koordinierungsverhalten zwischen den Entwurfsprozessen zu berücksichtigen.
Nach der erweiterten Aufgabenstellung gibt es dazu keine einschränkenden Randbedingungen, die z.B. getrennte Prozessoren für die einzelnen Stationsprozesse fordern würden.

Also ist das folgende Vorgehen möglich:

Die Klasse der V-Stationsprozesse (siehe 4.5.) wird zu einem einzigen Prozeß - dem R(ealen) V-S(tations-) Prozeß - zusammengelegt. Dieser RVS-Prozeß bearbeitet die einzelnen V-Stationsprozesse über deren Statusinformation: Stationsnummer und individuelles Gedächtnis.
Die Klasse der potentiell unendlich vielen Paketprozesse (siehe 4.4.) wird zu einem einzigen Prozeß - dem R(realen) P(aket)-Prozeß zusammengelegt. Dieser RP-Prozeß bearbeitet die einzelnen Paketprozesse über deren Statusinformation: Paketidentifikation mit Wegbeschreibung und Ortsangabe in der Wegbeschreibung. Das Beenden eines Paketprozesses bedeutet Löschen und Freigeben seiner Paketidentifikation.

Weiter können Prozesse zerlegt werden in einzelne Prozeduren, die ihren möglichen verschiedenen Prozeßständen entsprechen. Nahtstellen sind hier im allgemeinen Schreib- bzw. Leseoperationen zu gemeinsamen Datenmengen zweier Prozesse.
Ein derart zerlegter Prozeß wird dann im zugeordneten Prozeß als Prozedur-Folge aufgerufen.

. . .

Dabei ist zu beachten:

- o Zerlegung in
  - Anfangsbearbeitung
  - Endebearbeitung
  - Rumpfbearbeitung
- o Bei Auflösung von Iterationen und/oder Selektionen muß der übergeordnete Prozeß die Bedingungen sicherstellen.
- o Der übergeordnete Prozeß muß die Statusinformation des Prozesses verwalten.

In der beschriebenen Weise kann der VERTEILER FÜR ZIELE dem Prozeß EINGANG UND FREIGABE untergeordnet werden. Nahtstelle ist die Operation "schreibe Paketidentifikation mit Code". Der Prozeß heißt nachfolgend EINGAN. Ebenso ist der oben beschriebene RP-Prozeß zerlegbar in LSE- und LSA-Bearbeitung. Er wird an den Nahtstellen "schreibe Information" LS-Eingang" bzw. "schreibe Information LS-Ausgang" dem RVS-Prozeß zugeordnet; dieser heißt nachfolgend VERTPAK (Verteiler- und Paketsteuerung).
Da sich mehrere Pakete vor bzw. in jeder V-Station aufhalten können, müssen Warteschlangen für die paketspezifischen Daten den Lichtschranken zugeordnet geführt werden. Die Warteschlange der 1. Verteilstation wird vom Prozeß EINGAN nach der Zuordnung der Wegbeschreibungen zu den Paketidentifikationen mit diesen Daten (bzw. Verweise darauf) belegt. Die weiteren Warteschlangen werden dann von VERTPAK bei verteilerspezifischer LSE- bzw. LSA-Bearbeitung manipuliert.

Der bis hierher beschriebene Stand entspricht der anliegenden Beschreibung des Steuersystems in erweiterter Schematic Logic.

Wird bei getrennten Prozessen EINGAN und VERTPAK eine INTERRUPTBEARBEITUNG eingesetzt, so sind auch an deren Nahtstellen zu den Steuer-Prozessen Warteschlangen notwendig (falls nicht vom Betriebssystem verwaltet).

. . .

Falls das Zeitverhalten eine weitere Sequentialisierung zuläßt, können die Prozesse EINGAN und VERTPAK zerlegt und direkt einem übergeordneten Zentralprozeß als Prozeduren zugeordnet werden (entsprechend erweiterter Interruptbearbeiter).Dieser Zentralprozeß wickelt dann die Anfangs- und Endebearbeitung für alle Prozesse ab, sowie die gesamte Steuerung der Paketverteilung.

In dieser letztgenannten Form haben wir das Steuersystem als PASCAL-Programm implementiert und die Verteilersteuerung simuliert.

. . .

UEBERGANG ZUR REALISIERUNG
==========================

VEREINBARUNG:
  (1,i) := Operationen von PEINEP, (i = 1...9)
  (2,j) := Operationen von PVERFZ, (j = 1...8)
  (3,k) := Operationen fuer alle PPAKST, (k = 1...10)
           { vom Entwurf her gibt es beliebig viele prozesse }
  (4,l) := Operationen fuer alle PVERT2, (l = 1...15)
           { vom Entwurf her existieren soviele prozesse wie verteiler }
  (5,m) := Operationen von PEINTF, (m = 1...6)
  (6,n) := Operationen des Moduls LIESCZ, (n = 1...4)

```
!=============================================================================!
! KOMMENTAR                                                                   !
! Zeilen, die mit der Kennung --- versehen sind, wurden nachtraeglich in      !
! die Schematic Logic der drei Prozesse eingefuegt.                           !
!   Zeilen, die mit der Kennung +++ versehen sind, sind naeher spezifizierte  !
! Operationen ( fuer Implementierung ), die bereits in der Programmstruktur   !
! vorhanden waren. Sie beschreiben u.a. die Schnittstellen der drei Prozesse. !
!=============================================================================!
```

ERWEITERTE SCHEMATIC LOGIC FUER DEN PROZESS EINGAN
(EINGANG UND FREIGABE MIT VERTEILUNG DER ZIELE)

```
B.PROZESS EINGAN * SEQ *
  DO  a n f a n g - e i n g a n   { entspricht: (1.1) und (2.1) }
  lies paketmeldung { 1.paket }                                        (1.4)
  DO  l i e s c o d e z  { 1.paket / entspricht: (1.5) }
  PAKETBEARBEITUNGS-HAUPTTEIL * ITER * while not eof-meldung
    B.PAKETGRUPPE GLEICHEN ZEICHENS * SEQ *
      vergleichszeichen := codezeichen                                 (1.9)
      B.1.PAKETELEMENT * SEQ *
        erzeuge freigabe mit verzoegerung                              (1.7)
        DO  v e r t f u e r z i e l e  { entspricht:(1.8)u.(2.4) }
        lies paketmeldung { naechstes paket - read ahead }             (1.4)
        DO  l i e s c o d e z  { naechstes paket / entspricht: (1.5) }
      B.1.PAKETELEMENT * END *
      B.GRUPPENKERN * ITER * while not eof-meldung u. vgl-zeichen=codez
        B.PAKETELEMENT * SEQ *
          erzeuge freigabe ohne verzoegerung                           (1.6)
          DO  v e r t f u e r z i e l e  { entspricht:(1.8)u.(2.4) }
          lies paketmeldung { naechstes paket - read ahead }           (1.4)
          DO  l i e s c o d e z  { naechstes paket / entspricht: (1.5) }
        B.PAKETELEMENT * END *
      B.GRUPPENKERN * END *
    B.PAKETGRUPPE GLEICHEN ZEICHENS * END *
  PAKETBEARBEITUNGS-HAUPTTEIL * END *
  DO  e n d e - e i n g a n   { entspricht: (1.2) und (2.2) }
  programm stop                                                        (1.3)
B.PROZESS EINGAN * END *
```

```
PROCEDURE  a n f a n g - e i n g a n
B.ANFANG-EINGAN  * SEQ *
  anfangsbearbeitung fuer datenobjekte durchfuehren ( entspricht: (1.1) )
    aus EINGANG UND FREIGABE :
                  - meldungen
                  - codezeichen
                  - fehlermeldungen
                  - paketidentifikationen mit code
                  - freigaben
  anfangsbearbeitung fuer datenobjekte durchfuehren ( entspricht: (2.1) )
    aus VERTEILER FUER ZIELE :
                  - ziele mit wegbeschreibungen
                  - paketidentifikationen zu zielen
    einrichten der warteschlangen fuer alle verteiler
    initialisieren der warteschlangenelemente fuer die verteilerwarteschlangen
B.ANFANG-EINGAN  * END *


MODUL  l i e s c o d e z
B.CODEZ-EINGANG, E.MOEGL.FEHLMELD 1, E.CODEZ-AUSGANG   * SEQ *
  lies codezeichen eingang                                          (6.1)
  B.MODUL-RUMPF  * SELECT *  richtiges Codezeichen
    B.RICHTIGES CODEZ-EINGANG  * SEQ *
    B.RICHTIGES CODEZ-EINGANG  * END *
  B.MODUL-RUMPF  * OR *  falsches Codezeichen
    B.FALSCHES CODEZ-EINGANG, E.FEHLMELD 1  * SEQ *
      ersetze falsches codezeichen durch codezeichen fa             (6.2)
      fehlermeldung 1                                               (6.3)
    B.FALSCHES CODEZ-EINGANG, E.FEHLMELD 1  * END *
  B.MODUL-RUMPF  * END *
    codezeichen-ausgang ausgeben                                    (6.4)
B.CODEZ-EINGANG, E.MOEGL.FEHLMELD 1, E.CODEZ-AUSGANG   * END *


PROCEDURE  v e r t f u e r z i e l e
B.PAKID MIT CODE * SEQ *
  lies ziel zu paketidentifikation mit code                         (2.6)
  B.CODE-AUSWAHL KERN * SELECT * Ziel zu Codez vorhanden
    B.PAKID MIT CODE U.M.ZIEL * SELECT * Codezeichen A
      B.CODEZ ZU ZIEL A * SEQ *
        schreibe pakid zu ziel a , haenge in lse-ws.1.vert.ein      (2.5a)
      B.CODEZ ZU ZIEL A * END *
          . . .                                                     .
          . . .                                                     .
          . . .                                                     .
    B.PAKID MIT CODE U.M.ZIEL * OR * Codezeichen Z
      B.CODEZ ZU ZIEL Z * SEQ *
        schreibe pakid zu ziel z , haenge in lse-ws.1.vert.ein      (2.5z)
      B.CODEZ ZU ZIEL Z * END *
    B.PAKID MIT CODE U.M.ZIEL * OR * Codezeichen FA
      B.CODEZ ZU ZIEL FA * SEQ *
        schreibe pakid zu ziel fa, haenge in lse-ws.1.vert.ein      (2.5fa)
      B.CODEZ ZU ZIEL FA * END *
    B.PAKID MIT CODE U.M.ZIEL * END *
  B.CODE-AUSWAHL KERN * OR * Ziel zu Codez nicht vorhanden
    B.PAKID MIT CODE O.ZIEL ( FEHLMELD 2 ) * SEQ *
      fehlerbehandlung durchfuehren                                 (2.7)
      fehlermeldung 2                                               (2.8)
      schreibe pakid zu ziel fa, haenge in lse-ws.1.vert.ein        (2.5fa)
    B.PAKID MIT CODE O.ZIEL ( FEHLMELD 2 ) * END *
  B.CODE-AUSWAHL KERN * END *
B.PAKID MIT CODE * END *


PROCEDURE  e n d e - e i n g a n
B.ENDE-EINGAN  * SEQ *
  endebearbeitung fuer datenobjekte durchfuehren
  ( entspricht: (1.2) und (2.2) )
B.ENDE-EINGAN  * END *
```

ERWEITERTE SCHEMATIC LOGIC FUER PROZESS VERTPAK
(VERTEILERSTEUERUNG UND PAKETSTEUERUNG)

```
B.VERTEILERSTEUERUNG FUER PAKETE * SEQ *
  DO  a n f a n g - v e r t p a k  ( entspricht: (4.1) und (3.1) )
  lies ls-indikatorsignal ( lse oder lsa ) mit verteilerkennung        (4.4)
  B.PROGRAMMKERN * ITER * while Ls-Indikatorsignal vorhanden
    B.LSE/LSA-INDIKATORSIGNAL FUER EIN PAKET * SELECT * Lse-Indikatorsignal
      B.LSE-INDIKATORSIGNAL FUER EIN PAKET * SEQ *
------- lies 1.lse-ws.element (akt.verteiler)
        lies paketidentifikation des 1.lse-ws.elements                 (3.4)
        lies wegbeschreibung der akt.pakid                             (4.5)
        lies vorgaengerweg aus gedaechtnis (akt.verteiler)             (4.6)
        wfz := wfz + 1 (akt.verteiler)                                 (4.8)
        B.VERTEILER-FREI-INFORMATION * SEQ *
          bereite vert-frei-info auf                                   (4.13)
        B.VERTEILER-FREI-INFORMATION * END *
        B.SCHALTZUSTANDS-INFO * SEQ *
          bereite schltzust-info auf                                   (4.14)
        B.SCHALTZUSTANDS-INFO * END *
        B.VORGAENGER-WEGBESCHREIBUNG-INFO * SEQ *
          bereite vorgaengerweg-info auf                               (4.15)
        B.VORGAENGER-WEGBESCHREIBUNG-INFO * END *
        schreibe akt. wegb in gedaechtnis (akt.verteiler)              (4.10)
        DO  l s e - p a k e t s t e u e r u n g (entspricht(3.6)u.(4.11))
------- schreibe 1.lse-ws.element in lsa-ws.(desselben verteiler)
------- loesche 1.lse-ws.element (akt.verteiler)
        lies ls-indikatorsignal (lse oder lsa) mit verteilerkennung    (4.4)
      B.LSE-INDIKATORSIGNAL FUER EIN PAKET * END *
    B.LSE/LSA-INDIKATORSIGNAL FUER EIN PAKET * OR * Lsa-Indikatorsignal
      B.LSA-INDIKATORSIGNAL FUER EIN PAKET * SEQ *
        wfz := wfz - 1 (akt.verteiler)                                 (4.9)
------- lies 1.element aus lsa-ws.(akt.verteiler)
        DO  l s a - p a k e t s t e u e r u n g (entspricht(3.7)u.(4.12))
        lies ls-indikatorsignal                                        (4.4)
      B.LSA-INDIKATORSIGNAL FUER EIN PAKET * END *
    B.LSE/LSA-INDIKATORSIGNAL FUER EIN PAKET * END *
  B.PROGRAMMKERN * END *
  DO  e n d e - v e r t p a k  ( entspricht: (4.2) und Teilen von (3.2) )
  programm stop                                                        (4.3)
B.VERTEILERSTEUERUNG FUER PAKETE * END *
```

```
PROCEDURE  a n f a n g - v e r t p a k
B.ANFANG-VERTPAK  * SEQ *
  anfangsbearbeitung fuer datenobjekte durchfuehren ( entspricht: (4.1) )
    aus VERTEILERSTEUERUNG :
                  - lichtschranken-indikatorsignale mit verteilerkennungen
                  - aktuelle wegbeschreibungen
                  - verteiler-informationsbloecke fuer alle verteiler
                    ( lse-/ lsa-informationsbloecke )
                  - gedaechtnisse fuer alle verteiler
                  - weichenfuellzaehler fuer alle verteiler
    wfz := 0   fuer alle verteiler                                    (4.7)
  anfangsbearbeitung fuer datenobjekte durchfuehren ( entspricht: (3.1) )
    aus PAKETSTEUERUNG :
                  - schaltauftraege
                  - fehlermeldungen
                  - zielankunftsmeldungen
B.ANFANG-VERTPAK  * END *


PROCEDURE  l s e - p a k e t s t e u e r u n g
  B.INFO LS-EINGANG * SELECT * Schaltung richtig
    B.SCHALTZUSTAND PASSEND * SEQ *
    B.SCHALTZUSTAND PASSEND * END *
  B.INFO LS-EINGANG * OR * Schaltung nicht richtig
    B.SCHALTZUSTAND NICHT PASSEND * SELECT * Verteiler frei
      B.VERTEILER FREI/E.SCHALTAUFTR. * SEQ *
        erteile schaltauftrag                                         (3.9)
      B.VERTEILER FREI/E.SCHALTAUFTR. * END *
    B.SCHALTZUSTAND NICHT PASSEND * OR * Veteiler nicht frei
      B.VERTEILER NICHT FREI/E.FEHLERMELDUNG * SEQ *
        fehlermeldung                                                 (3.10)
+++++++ ( systembacktracking )
+++++++ gib dem fehllaeufer die vorg.wegbes                           (3.10)
+++++++ schreibe daten des fehllaufs in die pakid                     (3.10)
      B.VERTEILER NICHT FREI/E.FEHLERMELDUNG * END *
    B.SCHALTZUSTAND NICHT PASSEND * END *
  B.INFO LS-EINGANG * END *


PROCEDURE  l s a - p a k e t s t e u e r u n g
  B.INFO.AUSGANG * SEQ *
    bearbeite information ls-ausgang ( leeroperation )                (3.8)
--- pruefe richtigen ausgang ( leeroperation )
  B.INFO.AUSGANG * END *
  B.BEARBEITUNG NACH LSA * SELECT * Verteiler <> letzter Verteiler
    B.NORMALFALL * SEQ *
----- schreibe lsa-ws.element (akt.verteiler) in lse-ws.(naechster verteiler)
----- loesche lsa-ws.element (1.element)
    B.NORMALFALL * END *
  B.BEARBEITUNG NACH LSA * OR * Verteiler = letzter Verteiler
    B.ENDEBEHANDLUNG * SEQ *
+++++ protokolliere (paket p im ziel x & evtl.fehlermeldung)          (3.5)
+++++ loesche lsa-ws.element (akt.verteiler)                          (3.2)
+++++ loesche akt.pakid                                               (3.2)
    B.ENDEBEHANDLUNG * END *
  B.BEARBEITUNG NACH LSA * END *


PROCEDURE  e n d e -  v e r t p a k
B.ENDE-VERTPAK  * SEQ *
  endebearbeitung fuer datenobjekte durchfuehren
  ( entspricht: (4.1) und Teilen von (3.1) )
B.ENDE-VERTPAK  * END *
```

ERWEITERTE SCHEMATIC LOGIC FUER PROZESS INTERR
(ERWEITERTER INTERRUPTHANDLER)

```
B.IMPULSE, E.PAKMELDN & CODEZ BZW. LS-INDSIGE F.VERT I  * SEQ *
  DO  a n f a n g - i n t e r r   ( entspricht: (5.1) )
  lies impuls                                                          (5.4)
  B.PROGRAMMKERN * ITER * while not eof-Meldung
    B.IMPULS * SEQ *
      B.AUSWAHL-KERN * SELECT * Impuls ist Codezeichen
        B.CODEZ & PAKMELD * SEQ *
          bereite paketmeldung mit codezeichen auf                     (5.5)
        B.CODEZ & PAKMELD * END *
      B.AUSWAHL-KERN * OR * Impuls ist Ls-Indsig fuer Verteiler 1
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER 1  * SEQ *
          bereite ls-indsig fuer verteiler 1 auf und schreibe dieses   (5.6a)
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER 1  * END *
            . . .
            . . .
            . . .
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER N  * SEQ *
          bereite ls-indsig fuer verteiler n auf und schreibe dieses   (5.6n)
        B.LSINDSIG ( LSE \ LSA ) FUER VERTEILER n  * END *
      B.AUSWAHL-KERN * END *
      lies impuls                                                      (5.4)
    B.IMPULS * END *
  B.PROGRAMM-KERN * END *
  DO  e n d e - i n t e r r   ( entspricht: (5.2) )
  programm stop                                                        (5.3)
B.IMPULSE, E.PAKMELDN & CODEZ BZW. LS-INDSIGE F.VERT I  * END *


PROCEDURE  a n f a n g - i n t e r r
B.ANFANG-INTERR  * SEQ *
  anfangsbearbeitung fuer datenobjekte durchfuehren ( entspricht: (5.1) )
    aus INTERRUPTHANDLER :
                    - impulse'
                    - paketmeldungen-ausgabe
                    - lichtschranken-indikatorsignale mit verteiler-
                      kennung ausgang
B.ANFANG-INTERR  * END *


PROCEDURE  e n d e - i n t e r r
B.ENDE-INTERR  * SEQ *
  endebearbeitung fuer datenobjekte durchfuehren ( entspricht: (5.2) )
B.ENDE-INTERR  * END *
```

Anhang 1

HÄUFIG BENUTZTE ABKÜRZUNGEN

1. Entwurfsstrukturen:

erster Buchstabe des Elementnamens ist entweder ein D (=Datenstruktur) oder ein P (=Programmstruktur).

Im einzelnen bedeuten

DEPAKV = Datenstrukturen der Eingabe (1. Tiefe)

D/PEINFR = Daten-/Programmstruktur für Eingang und Freigabe

D/PVERFZ = Daten-/Programmstruktur für Verteiler für Ziele

D/PPAKST = Daten-/Programmstruktur für Paketsteuerung

DVERT1/2/3 = Datenstrukturen für Verteilersteuerung

PVERT1/2 = Programmstruktur für Verteilersteuerung

D/PEINTE = Daten-/Programmstruktur erweiterten Interrupthandler

. . .

2. Standardabkürzungen innerhalb der Strukturen

AKT	=	aktuell
AUSG	=	Ausgang
B	=	bearbeiten
BLK	=	Block
CODEZ	=	Codezeichen
E	=	erzeugen
EING	=	Eingang
ELEM	=	Element
FEHL	=	Fehler
FREIGAB	=	Freigabe
GL	=	gleich
GRUP	=	Gruppe
IBLK	=	Informationsblock
INDSIG	=	Indikatorsignal
INFO	=	Information
KOR	=	Korrekt
LS	=	Lichtschranke
LSA	=	Lichtschranke-Ausgang
LSE	=	Lichtschranke-Eingang
M	=	mit
MELD	=	Meldung
O	=	ohne
PAK	=	Paket
PAKID	=	Paketidentifikation
PASS	=	passend
PROGR	=	Programm
SCHLTAUFTR	=	Schaltauftrag
SCHLTZUST	=	Schaltzustand
SORT	=	sortiert
STEU	=	Steuerung
V, VERT	=	Verteiler (Verteilstation)
VERZ	=	Verzögerung
VORG	=	Vorgänger
WEGB	=	Wegbeschreibung
WFZ	=	Weichenfüllzähler
WS	=	Warteschlange

. . .

Literaturangaben

/1/ M.A. JACKSON:
Principles of program design
Academic Press, London, New York,
San Francisco 1975

/2/ M.A. JACKSON:
Grundsätze des Programmentwurfs
S. Toeche-Mittler Verlag Darmstadt, 1979

/3/ J. LUDEWIG, W. STRENG:
Überblick und Vergleich verschiedener Mittel für die
Spezifikation und Entwurf von Software
Kernforschungszentrum Karlsruhe KfK 2506, März 1978

/4/ M.A. JACKSON:
Information Systems: Modelling, Sequencing and Transformations; in
Proceedings of 3rd International Conference on
Software Engineering; ACM/IEEE; 1978

/5/ J.W. HUGHES:
A Formalization and Explication of the Michael Jackson Method
of Program Design; SOFTWARE-PRACTICE AND EXPERIENCE, VOL 9,
191 - 202 (1979)

/6/ B. KÜHNEL:
Erfahrungen mit datenorientierter Software-Entwurfsmethode
in der Prozeßrechnerprogrammierung in: PDV-Berichte:
Verfahren und Hilfsmittel für Spezifikation und Entwurf von
Prozeßautomatisierungssystemen
KfK-PDV 154, Juni 1978, S. 239 - 245

/7/ E. GRÖTSCH, T. SCHLURICK:
Protokollbeschreibung und Protokollmaschinen-Entwurf
für Rechnerkopplungen mit Hilfe einer datenstruktur-
orientierten Entwurfsmethode in:
Proceedings of the GI-Workshop "Computernetworks and Remote Computing",
Schindler/Schröder (ed.) TU Berlin 5./6.10.1978, S. 68 - 82

/8/ M. HERZOG, B. KÜHNEL:
Data-Structured Design of Process Computer Software
Erscheint in Siemens Forschungs- und Entwicklungsberichte 05/80,
Springer Verlag, Berlin, 1980

S T E U E R U N G E I N E R P A K E T V E R T E I L A N L A G E

- - - - EIN ENTWURF IN DER D S L - SOFTWARETECHNIK - - - -

von

Hans-Juergen Ehling
AEG-TELEFUNKEN, Berlin

I N H A L T

Berlin, im Juni 1980

U E B E R B L I C K Z U R D S L - T E C H N I K

Die DSL-Softwaretechnik ist ein Komplex aus:

- einem Systembeschreibungs-Schema,
- einer Entwurfssprache: DSL -- design language -,
- einer Entwicklungs-Methodik,
- sowie Entwicklungs-Hilfsmitteln.

Teile dieser Technik werden nachfolgend am Beispiel einer 'Paket-Verteilanlagen - Steuerung' [1] dargestellt. Zuvor soll jedoch ein Ueberblick zur DSL-Technik gegeben werden.

Die DSL-Technik hat mehrere Wurzeln; als erste ist die Entwurfstechnik von M.A. Jackson [2] zu nennen.Von ihr wurde die Datenstromorientierung beim Prozess-Entwurf uebernommen und - engstens damit verbunden - die Konzeption 'einfacher' Prozesse als Grundbausteine von Sytemen.

Solche 'einfachen Prozesse' stehen in Beziehung zu bestimmten, formal definierbaren sequentiellen Maschinen [3]; diese Beziehung ist jedoch modellhafter Art, da in fast jedem praktischen Fall Abweichungen von dem reinen Schema auftreten. Die Eingrenzung 'einfacher' Prozesse dient aber letztlich nur der Sicherung von 'Verstaendlichkeit' fuer einen menschlichen Leser, sodass anthropomorphe - also nicht-formale Kriterien - durchaus angemessen sind. Wir betrachten Prozesse deshalb als 'verantwortliche' Personen. Dies wird im Vollzug des Beispiels naeher erklaert werden und ist ausfuehrlicher in [4] dargestellt.

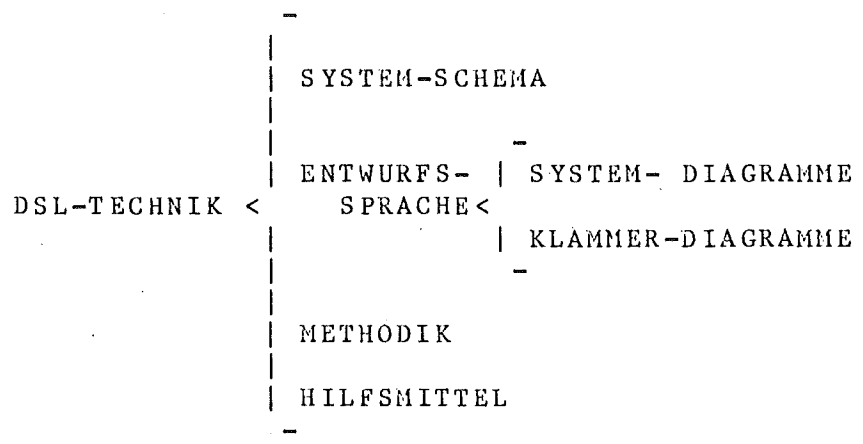
Sei die Erkennbarkeit eines 'einfachen' Prozesses einmal unterstellt, dann schraenkt dies die Willkuer bei der Systemzerlegung in einer aehnlichen Art ein, wie strukturierte Codierung die Ablauefe in Programmen. Dies fuehrt zu einem 'System(beschreibungs-) Schema, in welchem ein System als eine Schichtung von Netzwerken aus kommunizierenden Prozessen betrachtet wird,s. dazu auch [5].

Die Aufteilung der Prozesse einer Schicht wird durch 'Strukturbrueche' bestimmt - das sind, allgemein gesprochen, Inkongruenzen zwischen Input- und Output- Datenstroemen, die ein 'einfacher' Pro-Prozess nicht mehr bewaeltigen kann.

Die Bildung von interpretierenden Schichten wird noetig bei gemeinsamer Benutzung von Betriebsmitteln durch mehrere Prozesse.

Die Entwurfssprache verwendet graphische Elemente, dem Vorgehen jeder anderen Ingenieurdisziplin folgend. Fuer den Systementwurf werden sog. Systemdiagramme benutzt; sie entstehen aus Blockdiagrammen durch Einfuegen von Symbolen fuer die Kommunikationswege, die in Analogie zu den 'Plaetzen' zwischen 'Transitionen' in Petri-Netzen zu sehen sind.

Fuer den Prozess-Entwurf werden Klammerdiagramme aehnlich dem nachstehend gezeigten benutzt:



DSL- Klammerdiagramm, schematisch

Die Grundform wurde von Warnier [6] und Orr [7] angegeben; bei ausreichender optischer Gliederung sind diese horizontalen Baeume wesentlich schneller zu schreiben und zu aendern als Jacksons vertikale Baeume mit Kaesten. In der 'Papier-und-Bleistift-Phase' kann die Produktionsgeschwindigkeit die Qualitaet eines Entwurf erheblich beeinflussen.

Im Inhaltlichen geht die Ausdruckskraft der DSL wesentlich ueber die Vorbilder hinaus. Bei der Behandlung des Beispiels wird das noetige zur Sprache gesagt werden; eine Zusammenstellung befindet sich in der Anlage B.

Hinsichtlich der Methodik wurde die Datenstrom-Orientierung schon erwaehnt; ein tiefer liegendes Prinzip ist jedoch dasjenige der Zielorientierung'. Eine bewaehrte Planungstechnik ist es, vom angestrebten Ziel aus zum Ist-Zustand rueckwaerts zu rechnen. Aehnlich geht die DSL - Methodik

von den Outputs rueckwaerts zu den Inputs.

Dieses Prinzip haben wir von Orr [7] uebernommen. Es hat sich als vorzuegliche Anleitung beim Systementwurf erwiesen; beim Prozessentwurf begruendet es eine Abwandlung der Jackson-Technik, die gerade Aufgaben aus der Prozessrechentechnik besonders angemessen ist und auch in diesem Beispiel benutzt wird.

An Hilfsmitteln gibt es zur Zeit ein System zur Erzeugung von Klammerdiagrammen, mit dem die Mehrzahl der hier gegebenen Diagramme erzeugt wurde.

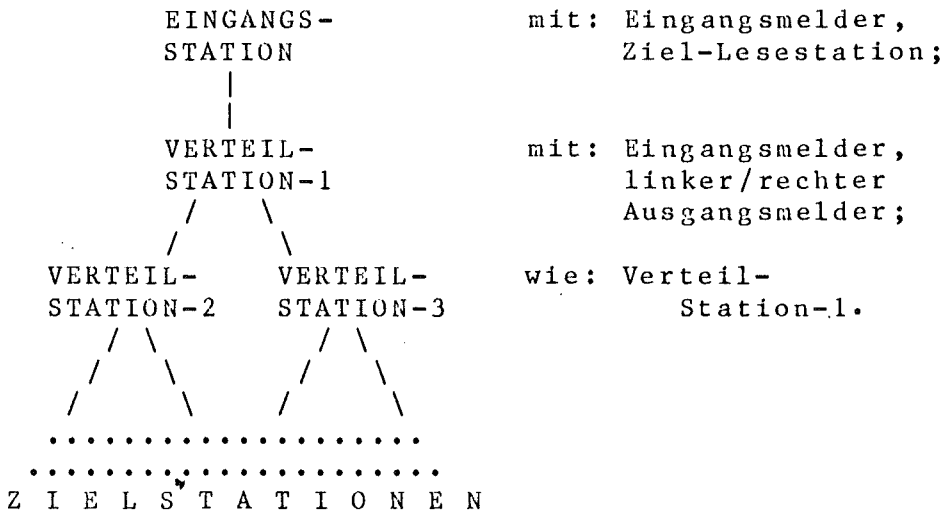
Ein weiteres Mittel ist ein Betriebssystem, das die Prozesskommunikation besser unterstuetzt und das insbesondere die 'invertierte Codierung' (s.[2]) unnoetig macht. Unsere Systemkonzeption scheint uns dadurch aufs Beste bestaetigt, dass in der US-DoD - Programmiersprache ADA genau die benoetigten Mittel vorgesehen sind.- Um keine Zeit zu verlieren, ist ein Betriebssystem fuer Mikroprozessoren auf der Basis der ADA-Rendezvoustechnik in Entwicklung genommen worden.

A U F G A B E N S T E L L U N G

Es ist die Steuerung einer Paketverteil-Anlage zu entwerfen.

Die Paketverteilanlage befoerdert Pakete von einer 'Eingangs-station' zu mehreren 'Zielstationen'; dieser Transport erfolgt auf schraegen Rutschen mit Hilfe der Schwerkraft.

Die Verteilung der Pakete erfolgt mittels Weichen, die zwischen den Rutschen derart angebracht sind, dass die Anlage die Form eines 'binaeren Baums' erhaelt. Die Weichen mit zugehoerigen Steuerungs- und Stell-Einrichtungen werden 'Verteilstationen' genannt. Die Anordnung der Anlage ist schematisch die folgende:



Die Eingangs-Station verfuegt ueber einen 'Eingangs-Melder' und eine 'Lese-Station', die das Paket-Ziel erkennt. Durch Betaetigen eines 'Einlass-Organs' wird das Paket in der Eingangsstation - und nur dieses- in die Verteilung eingelassen.

Die Verteilstationen verfuegen ueber einen Eingangsmelder und je einen Ausgangsmelder fuer den linken und den rechten Abzweig. Die Weiche der Station darf nur dann geschaltet werden, wenn sich zwischen Ein- und Ausgangsmeldern kein Paket befindet. Ist dies der Fall, dann darf bei Eintreffen eines neuen Pakets am Eingangsmelder gerade noch geschaltet werden: der Eingangsmelder ist entsprechend von der Weiche entfernt.

Aus diesem Sachverhalt bestimmt sich die Einschleus-Strategie der Eingangs-Station: ein Paket mit gleichem Ziel wie das vorige laesst sie jederzeit zu; ein Paket mit anderem Ziel wird jedoch nur nach einem gewissen Zeitabstand zum vorigen zugelassen. Die Zeit ist so bemessen, dass in der Regel das vorige Paket eine Weiche bereits verlassen hat, wenn das neue Paket am Eingangsmelder eintrifft.

Ist jedoch ein Paket zu schnell, sodass die Weiche nicht mehr schalten kann, dann entsteht eventuell ein Fehllaeufer. Ein solcher wird von jeder Verteilstation ohne Verstellung ihrer Weiche durchgelassen. In einer 'End-Verteilstation'- d.i. eine Verteilstation vor einem Ziel - wird fuer solche Fehllaeufer jedoch ein Fehlerprotokoll mit der Angabe von Soll- und Ist-Ziel erzeugt.

S Y S T E M E N T W U R F

ALLGEMEINES

DSL betrachtet ein System als eine Schichtung von Netzwerken aus 'Prozessen', die ueber Kanaele und Variable miteinander kommunizieren:

- Ein Prozess ist anzusehen wie eine Person, die permanent fuer bestimmte Outputs verantwortlich ist;

- Ein Kanal ist anzusehen als eine Art Rohrpost, eine Variable gleicht einer Wandtafel.

Beim Systementwurf wird demgemaess unterschieden zwischen `Prozessfindung` und 'Netzwerksfindung`.

Diese Arbeitsschritte werden anschliessend am Beispiel der obersten Schicht der Paketverteilstation - ihrer 'Nutzschicht' - erlaeutert.

PROZESSFINDUNG

Die Prozessfindung orientiert sich zuerst an dem oben formulierten Prinzip der 'Verantwortlichkeit'fuer gewisse Outputs.

Die Outputs des Steuersystems (nicht: der Paketverteil-Anlage!) sind:

- Der Stellimpuls fuer das Freigabeorgan der Eingangsstation ; seine Wertmenge ist der einzelne Impulswert 'da', mit der Bedeutung: 'Paket marsch !'

- Fuer jede Verteilstation V:
  der Weichenstell-Ausgang mit der Wertmenge ('links, rechts '),

- Fuer jede End-Verteilstation:
  Ausgaben an den Protokolldrucker mit der Wertmenge Protokoll - z.B. "Paket fuer Ziel-1 liegt in Ziel-2".

Nun fragt sich: ist fuer jeden der Outputs ein eigener Prozess notwendig? Das Kriterium dafuer ist die 'Einfachheit' von Prozessen:

> Ein einfacher Prozess soll keine 'Fortschritts-Grade' als Variablen verwenden - ausgenommen: Wiederholungszaehler.

Ein 'Fortschrittsgrad' ist ein Wert, der an das Durchlaufen einer gewissen Programmstelle erinnern soll. Fortschrittsgrade sind z.B.: 'Merker', unanschauliche Zustaende ('ich war hier, aber Du warst nicht zu Hause ') oder allgemein Inschriften der Art 'Kilroy was here'.

Die 'Einfachheit' eines Prozesses erweist sich letzlich erst beim Prozess-Entwurf.Zu Anfang hilft die Faustregel:

> Zwei strukturierte, asynchron verlaufende Vorgaenge lassen sich nicht zu einem einfachen Prozess zusammenfassen.

Im vorliegenden Falle bedeutet das, dass man tatsaechlich fuer jeden Output einen Prozess braucht. Denn das Einschleusen eines Paketes in der Eingangsstation ist sowenig wie das Steuern der verschiedenen Weichen zeitlich starr verbunden. *)

NETZWERKSFINDUNG

Zur Netzwerksfindung wird von den Outputs rueckwaerts nach den fuer sie benoetigten Inputs gefragt. Sind darunter Outputs von anderen Prozessen, dann hat man ein Kante des Netzes gefunden. Im Falle des Beispiels:

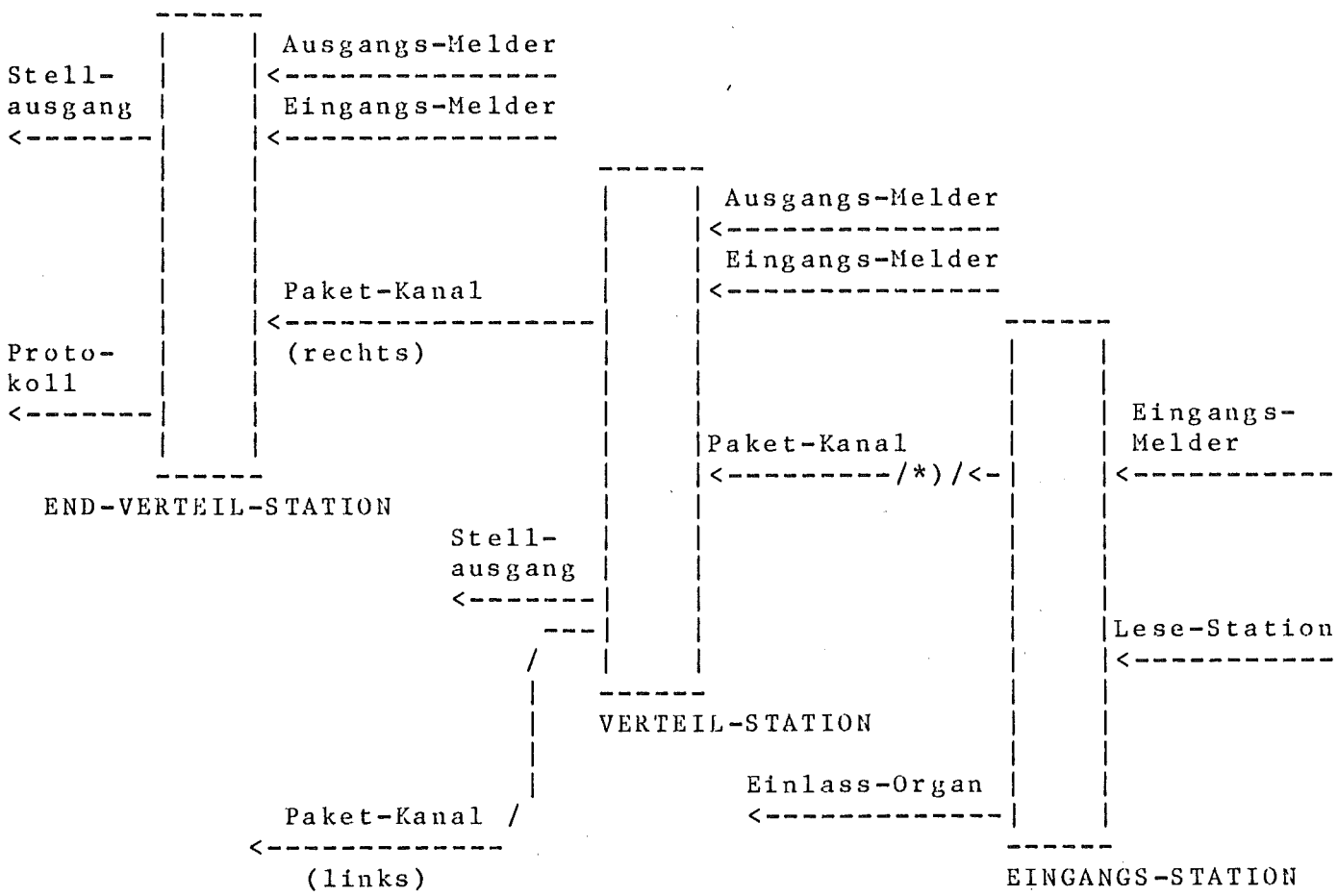
> Jede Verteilstation benoetigt zur Ausgabe der Weichenstellung:
>
> - den Impuls ihres Eingangs-Melders
> - den Fuellstand der Weiche
> - das Ziel fuer das Paket.
>
> Der Eingangsmelder ist ein Systemeingang und deshalb nicht weiter zurueckzuverfolgen.
>
> Der Fuellstand der Weiche bestimmt sich aus
>
> - dem Impuls des Eingangs-Melders und
> - den Impulsen der beiden Ausgangsmelder.
>
> Auch die Ausgangsmelder sind Systemausgaenge und nicht weiter zu verfolgen. - Das Ziel fuer das Paket muss
>
> - von der vorhergehenden Verteilstation oder von der Eingangsstation mitgeteilt werden, und zwar ueber einen gepufferten Kanal (die Rohrpost).

Mit aehnlicher Argumentation fuer die Prozesse von Eingangsstation und Protokolldrucker kommt man zu der umstehenden Prinzip-Struktur:

---

*)Falls etwa die Eingangsstation erst schalten wuerde, wenn die erste Verteilstation 'Paket-Ausgang' meldet, dann koennten Eingangs- und Verteilstation zusammengefasst werden. - Weiter koennte das ganze System als nur ein (rekursiv definerter) Prozess behandelt werden, wenn das naechste Paket erst zugelassen wuerde, wenn das Vorherige sein Ziel erreicht hat.

Prinzipstruktur der Paketverteilanlagen-Steuerung



Die Paketkanaele sind Entsprechungen der Verteiler-Rutschen. Verteilstationen koennen an Verteilstationen oder an End-Verteilstationen liefern. Endverteilstationen haben keine Paket-Kanaele; stattdessen koennen sie Protokolle ueber fehlgelaufene Pakete produzieren.

Bemerkungen zum Ablauf

In diesem Zustand der Analyse kann es vorkommen, dass die ermittelten Prozesse noch nicht 'einfach' sind. Die Grundstruktur bleibt aber meistens erhalten. Im vorliegenden Beispiel gibt es keine Aenderungen mehr; alle Prozesse erweisen sich als einfach.

Die Analyse der funktionellen Zusammenhaenge hat auch keinen weiteren Prozess zutage gebracht, denn die benoetigten Inputs waren System-Inputs oder erwiesen sich als Outputs bereits bekannter Prozesse. Stattdessen findet man oft vermittelnde Prozesse, z.B. Aufbereitungen von Systeminputs fuer mehrere output-naehere Verbraucher.

DARSTELLUNG IM SYSTEMDIAGRAMM

Die Prinzipstruktur wird nun zu einem 'Systemdiagramm' praezisiert. Dieses zeigt neben den Prozessen auch die Kanaele und Variablen.

Benennungen

Zunaechst sind Namen fuer Prozesse, Kanaele und Variable zu vergeben. Dabei werden auch die an den Kanaelen und Variablen auftretenden Wertmengen benannt.

Ein Kanal ist daran erkennbar, dass der Empfaenger den Wert 'verbraucht'; d.h: zweimal Dasselbe fuehrt zu zwei Reaktionen. Die Melder melden jeweils ein neues Paket; hingegen eine Weiche zweimal statt einmal nach rechts zu schalten: das ist kein Unterschied.

Prozesse: Die Stationsnamen werden als Namen fuer Identifikationsmengen benoetigt (die Menge der VERTEILSTATION(en)); die Prozesse heissen:

VERTEILE-PAKET(V:), wobei V eine VERTEILSTATION ist.

ANNEHME-PAKET, fuer den Prozess der Eingangs-Station.

Der ':' hinter V zeigt die Notwendigkeit eigener lokaler Variablen an; 'Verteile-Paket' ist 'vielfach', ist eine 'Prozess-Familie'.

Kanaele: dies sind die Paketkanaele, die Melder und der Drucker (der hier auf seinen Zugriffsweg reduziert ist). Im einzelnen:

Kanal.V: Paket-Ziel ; der Eingangskanal in Verteilstation V. Seine Wertmenge sind Paket-Ziele.

Melder.V: {Rein,Rraus,Lraus}; die Melder an Verteilstation V, zusammengefasst und durch die Werte Rein (fuer Eingang),Rraus und Lraus (fuer linken,rechten Ausgang) unterschieden.

```
Eingangs-Melder: {da};      der Melder an der Eingangs-Station.

Drucker: Protokoll;         der Drucker(kanal) mit Protokoll(en)
                            als Wertmenge.

Einlass-Organ: {go};        das Einlass-Organ mit dem Wert
                            fuer 'Paket marsch'.
```

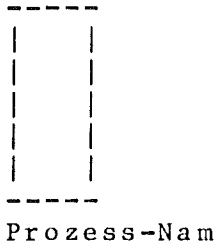
Variable: Die Weichen der Verteilstationen und der Leser an der Eingabestation werden als Variable betrachtet:

```
Weiche.V: {links,rechts};   die Stellung der Weiche an der
                            Verteilstation V.

Leser: Paket-Ziel;          der Leser an der Eingangs-Station.
```

Symbole in Systemdiagrammen

Die Information ueber Prozesse, Kanaele und Variable wird umstehend in einem 'Systemdiagramm' zusammengefasst. Ein Systemdiagramm ist ein erweitertes 'Blockdiagramm', in welchem neben den Prozessen die Kanaele und Variablen - die (Daten- oder Signal-)'Traeger' - dargestellt sind.

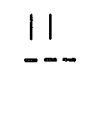
Es bedeuten:



ein Prozess

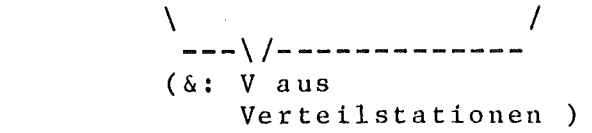
ein ungepufferter Kanal

ein gepufferter Kanal



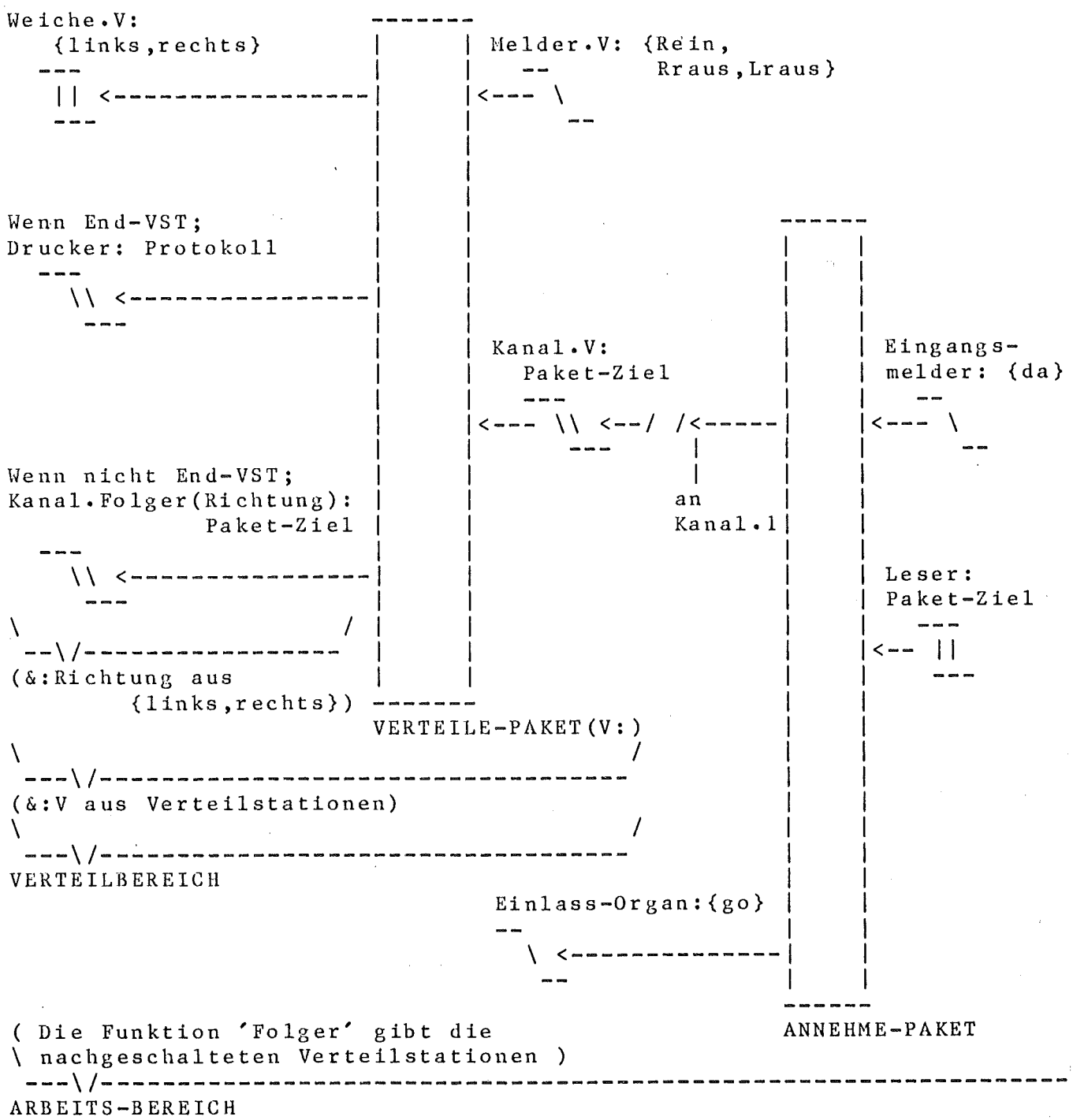
eine Variable

Vervielfachungen von Prozessen und Traegern werden mit Hilfe von Unterklammerungen angezeigt, z.B.:



Aehnlich fasst man System-Teile unterschiedlicher Natur zusammen. - Es ergibt sich das folgende Systemdiagramm:

Systemdiagramm der Paket-Verteilanlagen-Steuerung



Die 'Verteile-Paket' - Prozesse sind zum 'Verteilbereich' zusammengefasst. Die zwei Paketkanaele einer Verteilstation sind durch das Vielfach ueber den Richtungs-Werten -'links,rechts'- dargestellt. An welche Kanaele die Verteilstation V angeschlossen ist, besagt die 'Folger'-Funktion. Als Bedingung ist gesagt, dass Kanaele nur von Nicht-Endstationen versorgt werden; umgekehrt ist es mit den Fehllaeufer- Protokollen auf dem Drucker.

Der Verteilbereich ist mit 'Annehme-Paket', dem Prozess der Eingangsstation, zum 'Arbeits-Bereich' zusammengefasst. Zum Gesamtsystem gehoert noch ein Start-Prozess, evtl. mit Ueberwachungsaufgaben; dieser ist hier nicht dargestellt.

Statt als Systemdiagramm kann die hier entwickelte Aufbaustruktur auch als 'Klammerdiagramm' dargestellt werden; diese Darstellung ist im Anhang A, Seite 1 sowie im Anhang B, Seiten 1 und 2 verwendet. Die Klammerdiagramme zeigen die hierarchische Struktur, die in Systemdiagrammen nur begrenzt ausdrueckbar ist. Sie bieten ausserdem Raum fuer Kurzbeschreibungen - wie in der 'Textstruktur' der Anlage A - oder fuer weitergehende formale Beschreibung von Inputs und Outputs, wie in der 'strikten Form' der Anlage B.

ZUSAMMENFASSUNG ZUM 'SYSTEM-ENTWURF'

Die Grundoperationen der DSL-Systementwurfstechnik sind

- die 'Prozess-Findung', nach dem 'Verantwortlichkeits'-Prinzip und ausgehend von den System-Outputs,
- die 'Netzwerks-Findung', mit der Technik methodischen Rueckverfolgens zu den Inputs,
- die Zusammenfassung im 'Systemdiagramm', nachdem Prozesse, Kanaele und Variable sowie deren Wertmengen bestimmt und benannt wurden.

Die Netzwerksfindung kann wiederum zur Prozessfindung hinleiten, wenn man beim Rueckverfolgen der Outputs die System-Inputs noch nicht erreicht hat.

Zur Prozessfindung sind zunaechst Annahmen ueber die 'Einfachheit' der Prozesse noetig. Diese koennen sich beim anschliessenden Prozess-Entwurf als verbesserungsbeduerftig erweisen. Innerhalb der jeweiligen Anwendungsbereiche entwickelt man jedoch bald ein gutes Einschaetzungsvermoegen, das im Einzelfall durch provisorischen Prozess-Entwurf unterstuetzt werden kann.

Das Ergebnis des Systementwurfs ist eine statische Ansicht des System-Zusammenhangs, seines Aufbaus aus weitgehend autonomen Individuen. Dieses Systemmodell ist der Lebenssituation angepasst und foerdert deshalb die Klarheit und Einfachheit.

P R O Z E S S - E N T W U R F

Beim Prozess-Entwurf wird zunaechst die Ablaufstruktur gesucht, und dann der funktionelle Zusammenhang ausgearbeitet.
Dies wird anschliessend am Beispiel von VERTEILE-PAKET gezeigt.

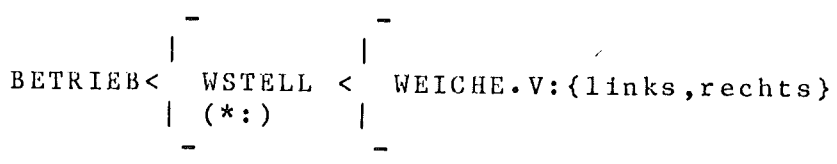
Dabei werden Klammerdiagramme verwendet, meistens in formloser Weise. Bei Gelegenheit werden aber auch einige formale Elemente der DSL-Entwurfssprache eingefuehrt.

FINDEN DER ABLAUF-STRUKTUR

Nachdem Prozesse unter dem Gesichtspunkt der 'Verantwortlichkeit' fuer bestimmte Outputs ausgewaehlt wurden, soll ihre zeitliche Struktur auch zunaechst anhand dieser Outputs gesucht werden. Die Inputs muessen sich dann zuordnen lassen, oder der Prozess ist kein 'einfacher'.

Inputs und Outputs von Prozessrechner-Anwendungen erscheinen auf den ersten Blick oft als unergiebig. Jedoch nimmt der physisch ganz triviale Weichenstell-Output sogleich Gestalt an, wenn nach den naeheren Umstaenden gefragt wird.

1)Der Zweck eines VERTEILE-PAKET- Prozesses ist, seine Weiche zu stellen; deshalb wird dieser Output zuerst betrachtet. Physisch ist er eine endlose Wiederholung von {Links,Rechts}- Einstellungen; als Klammerdiagramm:



Strom des Weichen- Stellens

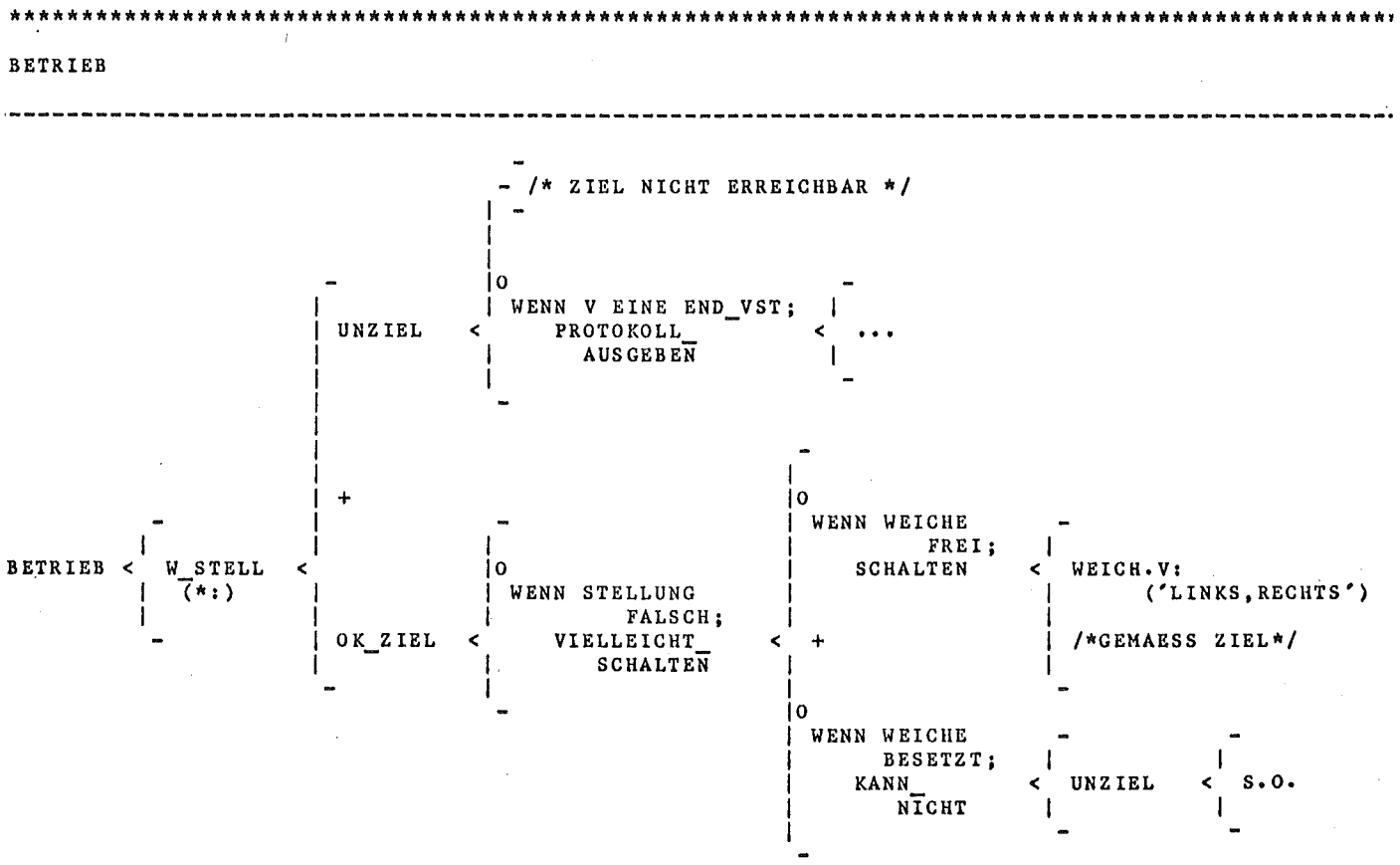
Das bedeutet:

```
BETRIEB ist:
   endlos folgt ( --> '*:' )
   WSTELL ,das ist:
      am Ort WEICHE.V:
         {links,rechts}
   WSTELL -Ende
BETRIEB-Ende
```

2) Nun fragt sich, wann und warum die Weiche gestellt - oder nicht gestellt - wird:

- Wenn sie gestellt wird, dann hat ihr Melder 'Rein' geliefert,
- dennoch wird nicht immer gestellt: Fehllaeufer sind moeglich. D.h: ein nicht erreichbares Ziel, ein UNZIEL liegt vor.
- Das Gegenteil des UNZIELs ist ein gutes, ein OK-ZIEL. Dann aber:
  - ist Stellen vielleicht nicht noetig, wenn naemlich die Richtung stimmt;
  - wenn doch Stellen n o e t i g ist -
    - ist es vielleicht nicht m o e g l i c h , weil noch Pakete in der Weiche sind;
    - oder aber: die Weiche ist leer; nur dann kann 'gestellt' werden.

Dies stellt - als 'Figur' - das folgende Klammerdiagramm dar:



Strom des Weichen -Stellens und -Nicht-Stellens

Man liest in diesen Diagrammen von oben nach unten und geht nach rechts in die Einzelheiten.

Die Klammerbezeichner BETRIEB und WSTELL wurden trotz veraenderter Texte beibehalten. Manche Namen druecken anfaenglich das Ziel der Entwurfsaktivitaet aus und ihr Inhalt gibt nur einen Bruchteil der behaupteten Bedeutung wieder; es waere unsinnig, dann staendig neue Varianten zu bilden. Die formale Bedeutung einer Klammer - soweit sie in fruehem Stadium ueberhaupt schon vorhanden ist - haengt ohnehin nicht vom Namen ab.

' /* ..... */ ' ist ein Kommentar; im Bild steht einer in 'Top-'Position, an der oberen Spitze einer Klammer.

'+',wenn zwischen untereinander befindlichen Komponenten stehend, bedeutet 'oder (ist)'; z.B.:

```
WSTELL  ist:
   UNZIEL...
   oder ist ( --> '+' )
   OK-ZIEL...
WSTELL -Ende
```

'O' mit Bedingung - z.B.:

```
O
 WENN V EINE ENDSTATION IST;
```

bedeutet: IF Bedingung (die bis zum ';' reicht); fehlt eine Bedingung, dann heisst es: 'optional'. - Die Bedingung ist bei strikter Sprachverwendung natuerlich formal; spaeter heisst es z.B.:

```
O
 V @ EVST;     -- 'V ist eine Endverteilstation'.
```

'@' ist - ersatzweise - das Elementzeichen.

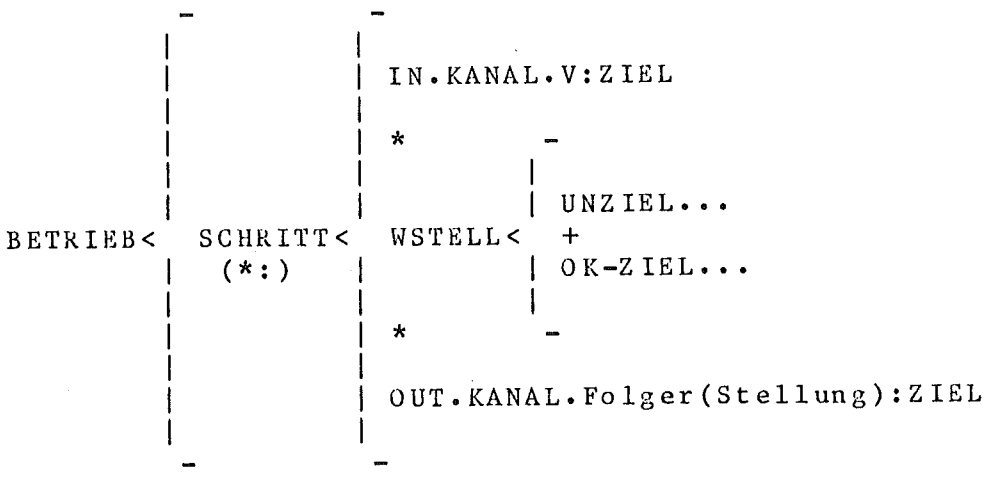
Bedingungen fuer UNZIEL und OK-ZIEL sind noch nicht formulierbar. Auch ohne Bedingungen hat eine Alternative eine klare Bedeutung: die beiden Zweige koennen auftreten, nur kann oder will man die naeheren Umstaende noch nicht angeben.

Hinzugenommen wurde, dass im Falle 'UNZIEL' eventuell ein Fehler-Protokoll auszugeben ist. - '...' verweist auf Unvollstaendigkeit; das 'Bildsystem', welches das Diagramm berechnet hat, markiert diese Stelle intern fuer evtl. spaeteren Verweis-Eintrag. Die Eingabe zu obigem Diagramm war - mit Auslassungen - die folgende:

```
BETRIEB;<WSTELL ` (*);<UNZIEL;<..Auslassung..>+
OK-ZIEL;<..Auslassung..>>
```

Im Vergleich wird sich das Bildungsgesetz bereits abzeichnen.

3)Rueckkehrend zur Verteilstation sind die Paket-Kanaele leicht einzuordnen:



Einordnung der Paket-Kanaele

'*' bedeutet: 'danach folgt',z.B.:

```
SCHRITT ist:
   ...INput aus Kanal V....
   danach folgt ( --> '*' )
   WSTELL...
   danach folgt ( --> '*' )
   ...OUTput an Kanal......
SCHRITT-Ende
```

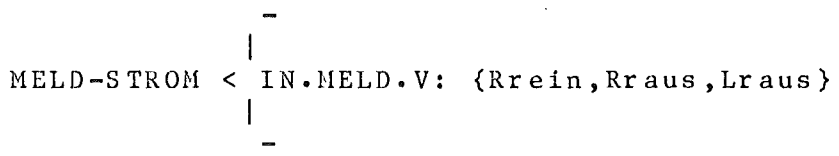
Der 'Konnektor' '*' wurde vorher - in runde Klammern gestellt - schon in seiner 'Vielfach-' Form verwendet:

```
 *       'danach folgt'
(*)      'unbestimmt oft folgt'
(*:)     'endlos folgt'
```

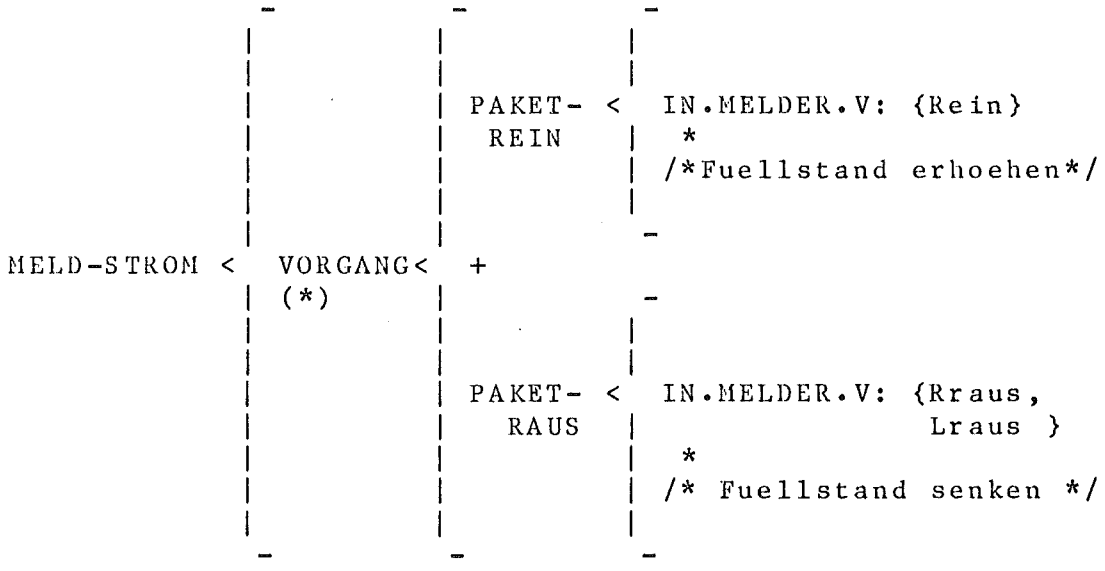
In der Klammer kann die Wiederholung naeher spezifiziert werden, z.B. koennen Anzahl oder Lauf- Bereiche und -Variable angegeben werden; der ':' ist der Rest einer weggelassenen Angabe mit der Bedeutung: '..durchlaufe alle natuerlichen Zahlen..'.

IN bedeutet 'Lies', OUT bedeutet 'Schreib' - und zwar aus bzw. an einen Kanal; bei Variablen fehlen derartige Zugriffs-Anzeiger. Hinter diesen Schluesselwoertern steht die Bezeichnung des Kanals, der im obigen Fall noch - mit '.V' bzw. der Folger-Angabe - 'selektiert' ist; KANAL bezeichnet - wie auch WEICHE - ein 'Vielfach', nun aber eine raum-zeitliche Koexistenz, die in den Systemdiagrammen schon mit dem Konnektor '&' dargestellt wurde. Dort wurde auch bereits die Vielfach-Form benutzt: '(&: V aus Verteilstationen)', was analog mit dem '*'-Konnektor moeglich ist.

4) Dem Auslesen des Ziels aus dem Paket-Kanal muss jedoch ein 'Lesen' des Melderkanals vorausgegangen sein, das den Wert 'Rein' erbracht hatte. - Die Trivialversion des Melderstroms ist:



Die Zuordnung zu den WSTELL-Aktionen verlangt eine Abtrennung des Paket- Einlaufs 'Rein' von den beiden Auslauf-Faellen. Dieser Aufteilung laesst sich dann auch die Fuellstands-Fuehrung der Weiche zuordnen, die ja fuer die Stell-Entscheidung gebraucht wird:



Hier waere nun der Ort fuer Fehlerbetrachtungen. Wenn wegen Fehlfunktionen der Fuellstand zu gross wird oder unter Null absinken will, dann muss die Struktur in Anlehnung an die erwaehnten Faelle entwickelt werden; das fuehrt auf eine ganz andere und kompliziertere Form. Zusaetzlich noetige Signale und weitere Prozesse koennten dabei entdeckt werden, je nach Anlagengroesse und erwarteter Fehlerhaeufigkeit. - Dies bleibt hier aber ausser Betracht.

5) Der SCHRITT aus dem BETRIEB unter 3) ist ersichtlich bei PAKET-REIN einzuordnen. Dies ist im nachstehenden Diagramm erfolgt, mit kleineren Aenderungen in der Zusammenstellung:

```
**************************************************************************************
 BETRIEB
--------------------------------------------------------------------------------------
                                                  -
                                                  |
                                                  | IN.MELD.V: 'REIN'
                                                  | *
                                                  | IN.KANAL.V: ZIEL
                                                  |
                                                  | *            -
                                                  |               |
                                                  | W_STELL   <   | ...
                                                  |               |
                                  -               |              -
                                  |               |
                                  | PAKET_    <   | *            -
                                  |    REIN       |              |
                        -         |               |              | FUELLSTAND ERHOEHEN
                        |         |               |              | *
BETRIEB <   VORGANG     <         |               | WEITER    <  | OUT.KANAL.FOLGER(STELLUNG): ZIEL
                        |         |               |              |
                        -         |               -              |
                                  | +                           -
                                  |
                                  |               -
                                  |               |
                                  |               | IN.MELD.V: ('RRAUS,LRAUS')
                                  | PAKET_    <   | *
                                  |    RAUS       | FUELLSTAND SENKEN
                                  |               |
                                  -               -
```

Integration des Melder-Stroms mit den frueheren Ergebnissen.

Der SCHRITT ist in PAKET-REIN integriert worden; ueber demselben Konnektor (hier: '*') kann man vorhandene Klammerungen aufloesen oder zusaetzliche Klammerungen einfuehren.

5)Nur noch eine formale Umformung unterscheidet die obige Struktur von der in den Anhaengen gegebenen. Der auf die Zweige der Alternative verteilte Melder-Input muss fuer die Codierung herausgezogen werden, wobei die Zweige Bedingungen erhalten; zur Demonstration geben wir nachstehend die formale Fassung:

```
                                  -
                                  |
                                  | IN.MELDER.V: {Rein,Rraus,Lraus} ( =M )
                                  |
                                  |                -
                                  | *              | 0
                                  |                |   M = Rein;
                                  |                |       PAKET-REIN...
                        VORGANG < | VG-RUMPF   <   |   +
                                  |                | 0
                                  |                |   M @ {Rraus,Lraus};
                                  -                |       PAKET-RAUS...
                                                   -
```

....(= M) sagt, dass der aktuell aus dem Meldekanal gelesene Wert 'M' heissen soll; dieser Wert kann nun in den Bedingungen verwendet werden. Die erste Bedingung ist offenkundig; bei der zweiten erinnere: '@' ist der zeichensatzbedingte Ersatz fuer das Elementzeichen. Die Bedingung ist aequivalent mit:

M = Rraus \/ M = Lraus ;

'\/' ist das logische 'oder' (und wird ausserdem als Vereinigungs - Operator benutzt).

Betont sei, dass die frueher gegebene Form mit verteiltem IN.MELD.. exakt dieselbe Bedeutung wie die obige Form hat; letztere ist lediglich der Steuerung durch den Rechner angemessen. Solche 'gesteuerten' Formen sind belastet mit den Definitionen von Testgroessen wie M und mit den Bedingungen; wer die traditionsbedingte Hemmung gegenueber ungesteuerten Formen ueberwunden hat, weiss deren leichtere Handhabung und bessere Lesbarkeit bald zu schaetzen.

AUSARBEITUNG DES FUNKTIONELLEN ZUSAMMENHANGES

Bei Vorliegen der Ablaufstruktur stehen bei Kanal- und Variablenbezeichnern die dort erwarteten Wertmengen, jedoch sind aktuelle Werte noch nicht bezeichnet. Funktionelle Zusammenhaenge sind evtl. durch unkomplette Klammern angedeutet, die zwischen zusammengehoerigen In/Outputs vorsorglich eingeschoben wurden.

Die Ausarbeitung besteht dann:

- aus der Vergabe von Bezeichnern fuer die aktuellen Werte -- von 'Wertbezeichnern',
- mit deren Hilfe dann die Funktionen dargestellt werden bzw. die 'Steuerung' formuliert wird.

Es war bereits ein Wertbezeichner fuer die Melder-Werte angegeben worden; als weiteres Beispiel:

IN.KANAL.V: ZIEL (= Z) ,

das heisst: ' lies aus Kanal V
ein (Element der Menge) ZIEL,
das ab jetzt 'Z' genannt wird '.

Dieses Ziel Z wird spaeter an einen Kanal FOLG weitergereicht. Dafuer steht dann:

OUT.KANAL.FOLG: ZIEL /\ 'Z' ,

das heisst: ' Schreibe an Kanal V
ein (Element der Menge) ZIEL,
u n d z w a r Z '.

Die Lesart ist plausibel genug; die Form hat aber auch eine eine strikte Bedeutung. 'Z' bezeichnet {Z}, die Einermenge zu Z; '/\' bedeutet den Mengendurchschnitt (und auch: logisches 'und'). Der Ausdruck insgesamt hat also den Wert {Z}, -- s o f e r n Z ein ZIEL ist (!), eine ganz erwuenschte zusaetzliche Aussage.

Die Syntax ist so gewaehlt, dass die bis dahin entwickelten Formen der Ablaufstruktur moeglichst nur durch Zusaetze zu ergaenzen sind; die schon vorhandene Information soll natuerlich erhalten bleiben.

Der Wertbezeichner FOLG soll die Identifikation der Verteilstation bedeuten, an die das Paket weiterzuleiten ist. FOLG ist ein Funktionswert, der in einer 'Top-'Komponente (an der oberen Spitze einer Klammer) als solcher definiert worden ist:

(+: FOLG = FOLGER(V, SA))

das heisst: ' W a e h l e FOLG = ... '

(s.Seite 6 der Anlage B; V ist eine Verteilstation, SA eine Weichenstellung.)

Obiges ist die Vielfachform des '+' - Konnektors, die zur Einfuehrung existentiell gebundener Variablen verwendbar ist. Ausfuehrlicher waere auch moeglich:

(+: FOLG = FOLGER(V, SA) @ VST /* Verteilstation */)

woraus gleich der Typ von FOLG ersichtlich waere. Dieser liesse sich aber auch aus einer Definition von FOLGER entnehmen:

(+: FOLGER @ (((VST\EVST)&{Links,Rechts}) /--<> VST)),

was besagt, dass die Abbildung FOLGER jedem Tupel aus einer Verteilstation - die keine Endverteilstation ist - und einer der genannten Richtungen -- eine Verteilstation umkehrbar eindeutig zuordnet: ' /--<> '.

Diese Definition wuerde vorzugsweise in den Definitionen zur Paketverteilanlage stehen, die auf Seite 1 der Anlage B hinter '(+)' erwaehnt werden; sie sind aber hier nicht ausgefuehrt.

Die Definition koennte ausserdem auf Seite 5 der Anlage B anstelle oder zusaetzlich zum dort stehenden Kommentar 'versichert' werden. Die Rundklammer dort fuehrt kein '+:', ihr Inhalt bedeutet dann einen Bool-Ausdruck, in dem aber abkuerzende Schreibweisen verwendbar sind. FOLGER z.B. ist ergaenzt zu denken durch '...ist eine Menge'.

Wertbezeichner koennen bereits in den Datenstroemen zur Beschreibung beliebiger Kontextabhaengigkeit verwendet werden. Im Unterschied zu Variablen sind sie nicht Teil des beschriebenen Objekts; deshalb koennen sie zur Beschreibung von Eigenschaften herangezogen werden, ohne dass dadurch das zu beschreibende Objekt verfaelscht wuerde - ein wesentlicher Grund fuer ihre Einfuehrung.

DSL - die Entwurfssprache - verfuegt ueber sehr starke Ausdrucksmittel. Das Ausmass ihrer Verwendung haengt von den Umstaenden ab. Bis zum Finden der Ablaufstruktur genuegen in der Regel - wie auch in diesem Beispiel - die grundlegenden Sprachelemente.

ZUSAMMENFASSUNG ZUM 'PROZESS-ENTWURF'

Grundlage des Prozss-Entwurfs ist eine Aufbaustruktur des Systems, die als System- oder Klammerdiagramm vorliegen kann. Aus ihr kann der Rand jedes Prozesses entnommen werden - das sind die Namen der Kanaele und Variablen, zusammen mit den zugehoerigen Datenstroemen.

Das erste Ziel ist dann, die Ablaufstruktur zu finden. Ausgangspunkt dafuer ist die Lebensgeschichte der Variablen, fuer die der Prozess verantwortlich ist. Die Datenstroeme dieser Variablen werden gemaess dem Problemverstaendnis 'geformt', d.h.: im Hinblick auf ihre Abhaengigkeit von den Inputs formuliert. Dem so entwickelten Strukturskelett werden weitere Outputs und Inputs eingefuegt, noetigenfalls unter Veraenderung der anfaenglich abgeschaetzten Form.

Gelingt es, alle Inputs und Outputs in einer Struktur unterzubringen, dann kann der funktionelle Zusammenhang ausgearbeitet werden. Gelingt es nicht, dann liegt ein 'Strukturbruch' vor; der Prozess war nicht 'einfach', der Systementwurf noch nicht beendet.

DIE URSACHEN DER PROZSSBILDUNG IN DER PAKETVERTEILANLAGE

Ursachen der Zerlegung eines Systems in Prozesse sind Strukturbrueche oder Benutzung von Betriebsmitteln im Zeitmultiplex.

Der auffaelligste Strukturbruch in der Paket-Verteilanlage - diese als eine Einheit betrachtet - ist der sog. 'Mischkonflikt': das ist der asynchrone Ablauf einer Reihe gleichartiger Prozesse, hier: der 'Verteile-Paket'-Prozesse.

Als zweiter Strukturbruch besteht in der Paketverteilanlage ein Grenzkonflikt, und zwar zwischen der Eingangsstation und allen Verteilstationen. Ein Grenzkonflikt liegt vor bei teilerfremden oder phasenversetzten Wiederholungen; der 'klassische' Fall ist Jacksons Telegramm-Analyse, wo Saetze variabler Laenge (die 'Telegramme') auf einem fest gerasterten Traeger (z.B. Lochkarten) angeliefert werden. Telegramme und Traeger sind dann teilerfremd. - Das 'Raster' in der Paketverteilanlage sind die durch Pausen getrennten 'Zuege' von Paketen zu unterschiedlichen Zielen. Dieses Raster wird von der Eingangsstation erzeugt; es wiederholt sich in den Verteilstationen, die ja nur in Pausen zwischen den Paketen schalten koennen. Jedoch tritt das Raster der Pausen in den Verteilstationen phasenversetzt auf und ausserdem zerlaeuft es - wird also auch noch teilerfremd. Grenzkonflikte werden durch Hintereinanderschaltung von Prozessen aufgeloest - hier: die Hintereinanderschaltung der Verteilstationen durch die 'Paket-Kanaele'.

Ein dritter Konflikt - der Ordnungskonflikt, wobei Daten in falscher Reihenfolge angeliefert werden - besteht bei der Paketverteilanlage nicht.

Eine Schichtzerlegung ist bei der Paketverteilanlage wenig ausgepraegt, sie besteht jedoch. Schichtzerlegungen treten auf bei Konkurrenz um Betriebsmittel: in diesem Falle ist es der Protokolldrucker. Nach dem Verantwortlichkeitsprinzip kann ein Kanal oder eine Variable nur von einem Prozess schreibend benutzt werden; d.h., konzeptuell hat jede Endverteilstation ihren eigenen Protokolldrucker. Diese verschiedenen Drucker werden durch den einen physischen Protokolldrucker i n t e r p r e t i e r t ; dazu bedarf es eines Druckerprozesses, den wir hier der Grundsoftware zurechnen koennen; auch die stets bestehende Konkurrenz der Prozesse eines Rechners um die Zentraleinheit findet in Betriebssystemschichten ihren Niederschlag. Bei kleinen Systemen koennen solche Schichten aber auch in das Blickfeld des Anwenders geraten.

L I T E R A T U R

[1] Gottschalk, W.: Beispiel 'Paketverteil-Anlage', mitgeteilt im PDV - Arbeitskreis "Systematische Entwicklung von PDV-Systemen", Karlsruhe, (1978).

[2] Jackson, M.A.,: Principles of Program Design. Academic Press, London (1975).

[3] Hughes, J.W.: A Formalization and Explication of the Michael Jackson Method of Program Design. Software-Practice and Experience, 9(3) 191-202 (1979).

[4] Ehling, H.J.: Evolutionaerer System-Entwurf. Informatik-Fachberichte 21: Formale Modelle fuer Informationssysteme 69-93 Springer Berlin (1979).

[5] Ehling, H.J.: Datenstrukturierter Software-Entwurf fuer Echtzeitsysteme. PDV-Berichte: Verfahren und Hilfsmittel fuer Spezifikation und Entwurf von Prozessautomatisierungssystemen, Kernforschungszentrum Karlsruhe, KfK-PDV 154 6 (1978).

[6] Warnier, J.D.: Logical Construction of Programs. Stenfort Kroese BV, Amsterdam (1974).

[7] Orr,K.: Structured Systems Development. New York, Yourdem Inc. (1977).

[8] Ehling, H.J.: Kurze Einfuehrung in die Entwurfssprache DSL. AEG/Interner Bericht Z5612 SB 020/79.

ANHANG A: P A K E T - V E R T E I L A N L A G E N - S T E U E R U N G

STRUKTUREN IN TEXTFORM

Auf den nachfolgenden Seiten ist die Struktur der Verteilagen-Steuerung in DSL-Klammerdiagrammen mit freier Sprachverwendung gegeben.

Solche Darstellungen heissen 'Text(uelle)strukturen' zur Unterscheidung von Strukturen mit strikter Sprachverwendung. Sie sind eine Art 'Pseudocode' und dienen entweder der Uebersichtsinformation oder als eigenstaendige Grundlage fuer weitere Bearbeitung. Im vorliegenden Fall sind die Textstrukturen aus der im Anhang B gegebenen strikten Form als Uebersichtsdokumentation abgeleitet.

Es enthalten:

- Seite 1: die Aufbaustruktur - die Entsprechung zum Systemdiagramm. Die Endkomponenten des Klammerdiagramms 'typisieren' die Prozesse durch die Angabe von Outputs und Inputs; erstere oberhalb, letztere unterhalb des '&&' -Symbols.

- Seite 2: die Ablaufstruktur des Prozesses 'Annnehme-Paket', der die Eingangsstation steuert.

- Seiten 3/4: die Ablaufstruktur des Prozesses 'Verteile-Paket', der fuer die Verteilstationen zustaendig ist.

Die Diagramme wurden mit der Bild-Funktion des DSL-Entwicklungssystems erstellt. Eingabetexte, die zu grosse Diagramme liefern wuerden, werden vom System automatisch oder nach Benutzer-Angaben aufgeteilt. Seite 4 ist z.B. aus dem Rumpfdiagramm auf Seite 3 durch die Teilungsvorgabe 'W-STELL' abgetrennt worden.

Eine Erklaerung von DSL-Grundsymbolen steht anschliessend.

E R K L A E R U N G V O N D S L - G R U N D S Y M B O L E N

Bei freier Verwendung von Klammerdiagrammen genuegen die nachstehend gegebenen Formen:

GRUNDVERKNUEPFUNGEN (KONNEKTOREN)

```
*   'danach folgt',      zeitliche Bedeutung;
+   'oder ist',          logisch;
&   'und besteht aus',   raeumlich, ohne Anordnung;
#   'dahinter steht'     raeumlich, mit Anordnung ('String').
```

EINIGE VIELFACH-FORMEN

```
(*)      'unbestimmt oft folgt'
(*:)     'endlos (bis QUIT, s. u.) folgt'
(*=10)   'genau 10mal folgt nacheinander'
(*>0)    'mindestens einmal folgt..'
```

Entsprechende Formen sind mit '&' und '#' moeglich.
Hinter ':' kann ein Laufanweisung stehen,z.B.

```
        (*: K aus 1 bis 10)  oder
        (&: V ist Verteilstation)
```

BEDINGTE FORMEN

```
O
 xxxxxx      'xxxxxx ist optional'

O
 Bedingung;
    xxxxxx   'wenn Bedingung, dann xxxxxx'

QUIT kkkkk   'verlasse Klammer kkkkk'

(* WHILE Bedingung )
                'solange Bedingung erfuellt ist, folgt..'
                (Sonderfall von 'endlos folgt' mit QUIT)
```

SCHNITTSTELLENBESCHREIBUNG (PROZESSTYP)

```
       -
      | Output
pppp <  &&        "Prozess 'pppp' liefert 'Output' aus 'Input'"
      | Input
       -
```

PAKET_VERTEIL_STEUERUNG

(SYSTEM ZUR STEUERUNG EINER PAKETVERTEILANLAGE,
BESTEHEND AUS EINER EINGANGSSTATION UND
BAUMFOERMIG NACHFOLGENDEN VERTEILSTATIONEN.)

PAKET_VERTEIL_STEUERUNG

UEBERWACHER

STARTET ALLE PROZESSE -
GIBT (EVTL.) STATUSMELDUNGEN ODER
AUSFALLBERICHTE

&&

NIMMT STEUEREINGRIFFE VOM
BEDIENUNGSPERSONAL ENTGEGEN -
NIMMT (EVTL.) STOERUNGSMELDUNGEN AN.

PAKET_VERTEIL_STEUERUNG

&

ANNAHME_PAKET

GIBT PAKETE FREI, BEI ZIELWECHSEL
(EVTL.) VERZOEGERT -
MELDET PAKETZIEL AN 1. VERTEILSTATION

&&

ANSTOSS VON EINGANGSLICHTSCHRANKE -
LIEST ZIELSTATIONSNR. VOM PAKET

PAKET_VERTEIL_STEUERUNG

&

ARBEITS_SYSTEM

STELLT ZUGEORDNETE WEICHE -
PROTOKOLLIERT , WENN ENDVERTEILER , FEHLLAEUFER -
MELDET , WENN NICHT ENDVERTEILER , PAKETZIEL

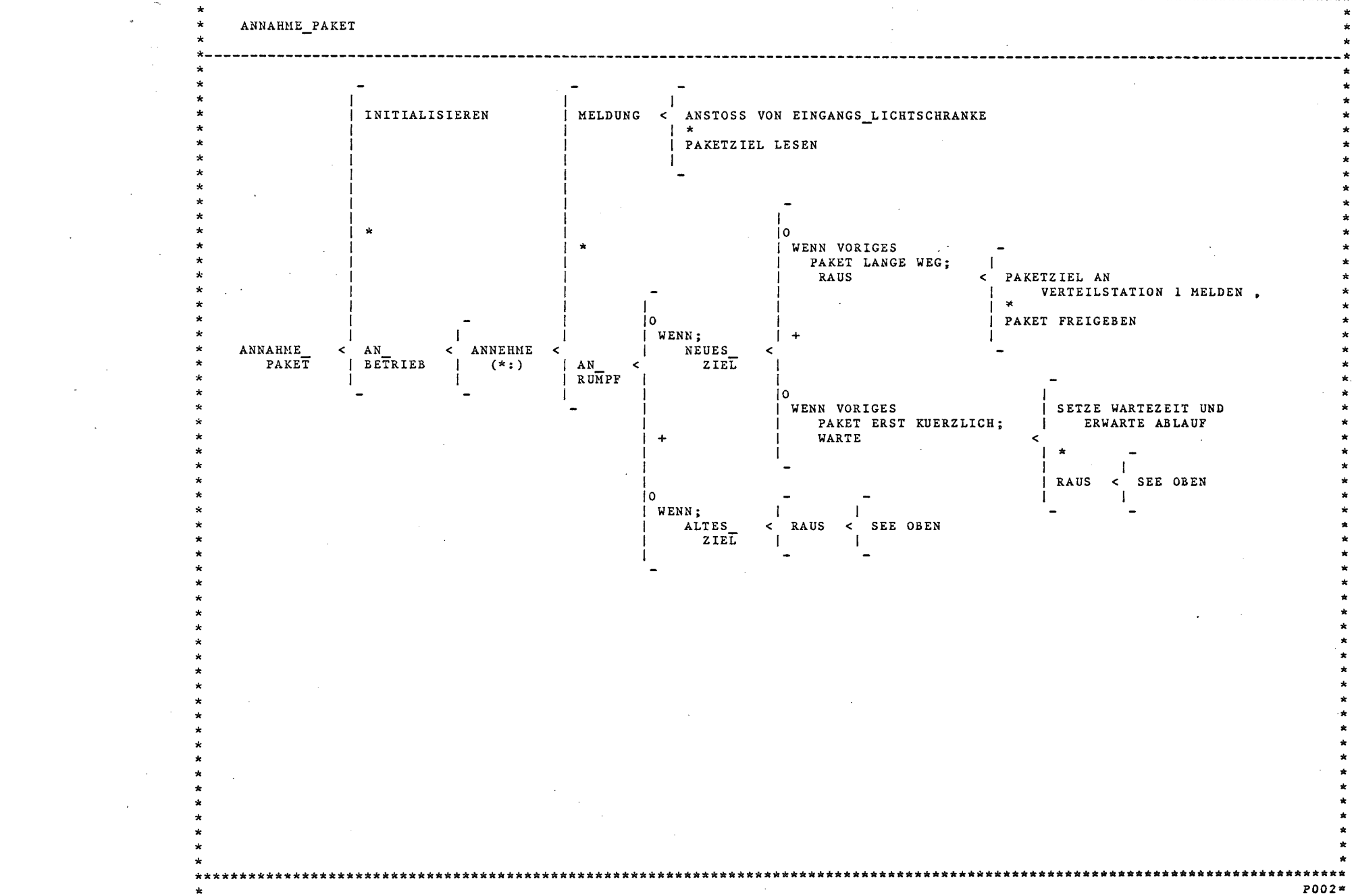
VERTEIL_BEREICH

VERTEILE_PAKET(V:)
(&: V IST VERTEIL_STATION)

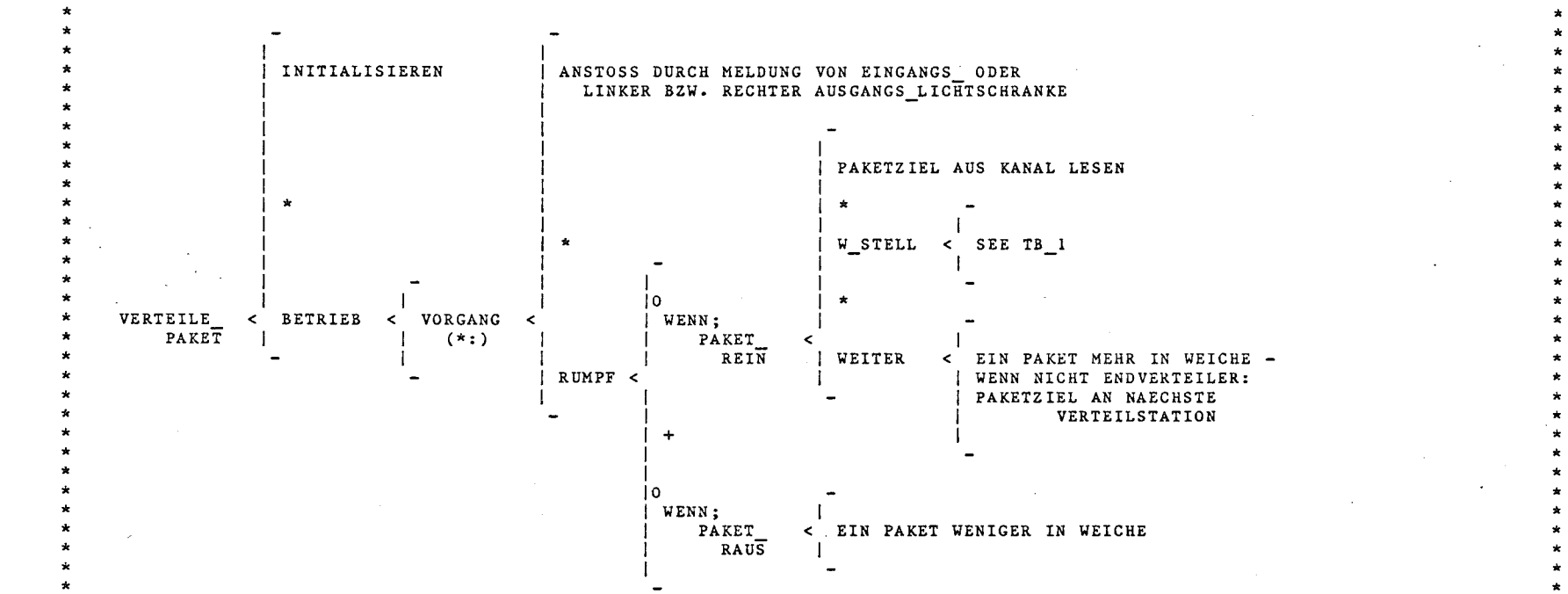
AN NAECHSTE VERTEILSTATION

&&

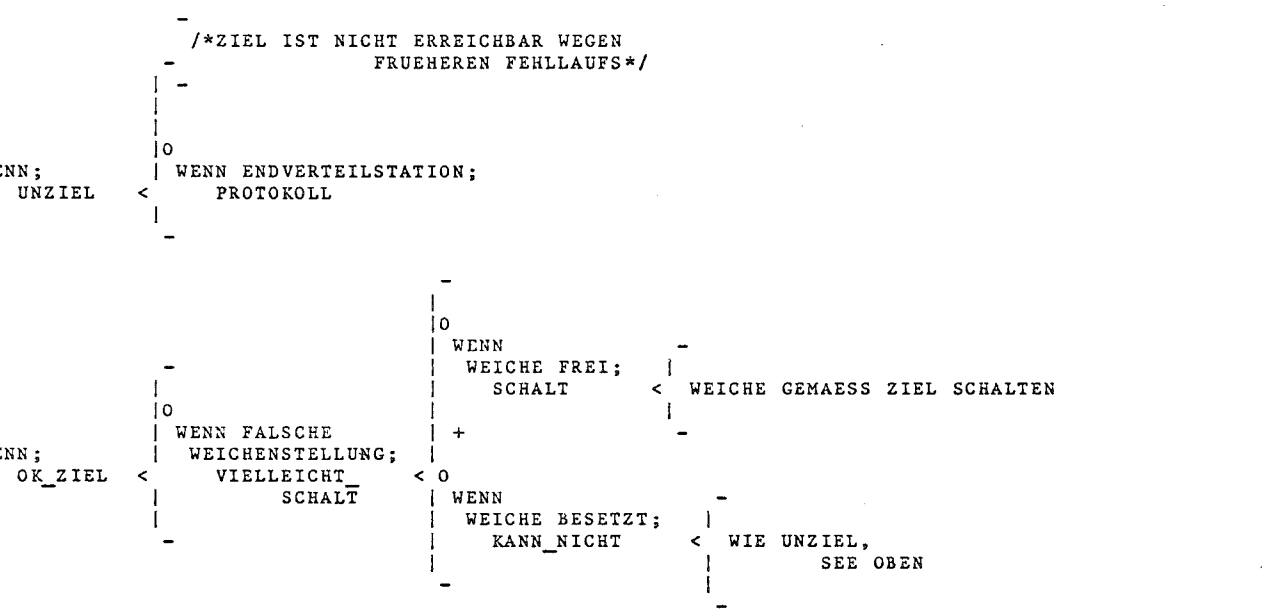
ANSTOSS VON EINGANGS_ SOWIE LINKER UND
RECHTER AUSGANGSLICHTSCHRANKE -
LIEST VON VORSTATION GEMELDETES
PAKETZIEL AUS KANAL


ANNAHME_PAKET
INITIALISIEREN
MELDUNG
ANSTOSS VON EINGANGS_LICHTSCHRANKE
PAKETZIEL LESEN
ANNAHME_ PAKET
AN_ BETRIEB
ANNEHME (*:)
AN_ RUMPF
WENN; NEUES_ ZIEL
WENN VORIGES PAKET LANGE WEG; RAUS
PAKETZIEL AN VERTEILSTATION 1 MELDEN ,
PAKET FREIGEBEN
WENN VORIGES PAKET ERST KUERZLICH; WARTE
SETZE WARTEZEIT UND ERWARTE ABLAUF
RAUS
SEE OBEN
WENN; ALTES_ ZIEL
RAUS
SEE OBEN

VERTEILE_PAKET



TB_1: W_STELL



P003*

ANHANG B: P A K E T - V E R T E I L A N L A G E N - S T E U E R U N G

STRUKTUREN IN STRIKTER FORM

Anschliessend ist die Struktur der Verteilanlagen-Steuerung in strikter DSL - Sprachverwendung gegeben.

Es enthalten:

- Seiten 1/2: die Aufbaustruktur der Paketverteil-anlagensteuerung.
- Seiten 3/4: die Ablaufstruktur des Prozesses 'Annehme-Paket'fuer die Eingangs-station,
- Seiten 5/6: Die Ablaufstruktur des Prozesses 'Verteile-Paket' fuer die Verteil-stationen.

Die Ablaufstrukturen sind durch Einfuegung von Statusmarkierungen auf invertierte Codierung vorbereitet. Der 'Verteile-Paket-' Prozess auf Seite 5 z.B. beginnt mit INIT, nimmt einige Einstellungen vor und endet - vorerst - mit FIN(S1>). Das besgt, dass zum letzten Aufrufer zurueckzukehren ist, nachdem der Prozess einen (hier nicht benannten) Status auf 'S1' gestellt hat. Diese Rueckehr ist eingefuegt, weil anschliessend -bei '(S1>)IN.MELD.V:..'- ein Kanal zu lesen ist, auf den in der Regel gewartet werden muss. Durch die Status-Einstellung kann das Melder-Ereignis an die richtige Stelle gelenkt werden. Nach Erledigung des Ereignisses in VG-RUMPF erfolgt wieder ein FIN, im Beispiel wiederum mit der Einstellung 'S1'. 'Verteile-Paket' kann wegen dieser einfachen Struktur in eine Initialisierungs- und eine Betriebs- Routine aufgeteilt werden.- Schon bei der Beruecksichtigung von Melderstoerungen wird es aber vermutlich nicht mehr bei dem einen Status bleiben. Eine Zerlegung in Routinen ist dann keineswegs mehr angebracht; wir vermeiden sie deshalb auch in den einfacheren Faellen.

Normalerweise wuerden die Traeger, Funktionen (wie: FOLGER) und Wertmengen in Definitionsdiagrammen formal erklaert werden; hier haben wir es jedoch bei kommentierten 'Versicherungen' in den Top-komponenten der Ablaufstrukturen bewenden lassen.

Erklaerungen zu weiteren DSL- Symbolen folgen nachstehend.

E R K L A E R U N G W E I T E R E R D S L - F O R M E N

EINFACHE MENGEN

Besondere Objekte

0	die leere Menge;
NONE	'nichts' (neutral fuer *,#,& -Konnektoren)
ANY	'unbestimmt' (= Klasse aller Mengen);

Standard-Mengen

NAT, INT ,REAL	natuerliche, ganze, rationale Zahlen;
BOOL	= {0,1}, 'Wahrheitswerte';
CHAR	die Druckzeichen; "A" :das Zeichen A , "AB" :der Zeichenstring AB .

Natuerliche Zahlen werden als Mengen ihrer Vorgaenger betrachtet:
0 = leere Menge, 1 = {0}, 2 = {0,1}, n = [0..n-1].

Aufgezaehlte Mengen

('A,B')	= {A,B} (ersatzweise): aufgezaehlte Menge,
'A'	= {A} : Einermenge.
[1..10]	Intervall 1,2,3,....9,10 .

OPERATOREN

Term-Operatoren

+ - * / MOD **	haben die ueblichen Bedeutungen: plus, minus, Multiplikation, Division, Potenzierung; A u s n a h m e: zwischen den Komponenten einer Diagrammklammer; dort bedeuten: + : 'oder' (Vereinigung), * : 'danach folgt' (Verkettung analog '#', aber verschieden)
\/ /\ \	Vereinigung, Durchschnitt, Mengendifferenz.
. #	Verkettung, 'dahinter steht' - elementweise Verkettung. Es ist: A#B = {a.b | a@A /\ b@B }; @ ist die Elementbeziehung. Z.B.: 1#2 = {0.0,0.1}. Bei Zeichenverkettung z.B.: "A"."B" = "AB". '#' kann in Komponenten und dazwischen (als Konnektor) stehen.
&	'und besteht aus', Funktionsmengenbildner, s. bei 'Strukturen'. S. dort auch zur Verwendung in bzw. zwischen Komponenten.
+ - \/ /\ # &	koennen auch monadisch verwendet werden; Beispiele zu # und & siehe bei 'Stringmengen' und 'Strukturen'.
. (monadisch)	verhindert Dereferenzierung.

Relationelle Operatoren

```
= < > <= >=      haben die ueblichen Bedeutungen: gleich, kleiner,
                 groesser, kleiner oder gleich, groesser oder gleich.
@                Element-Beziehung,
<<               Teilmengenbeziehung.
/=  /@ ....      '/' negiert alle relationellen Operatoren.
```

Bool-Operatoren

```
NOT  /\  \/  =>  <=>  /<=>   Negation, Konjunktion, Disjunktion
                 Implikation, Aequivalenz, Antivalenz.
```

MENGEN MIT STRUKTUR

Funktionen, Tupel

```
[A:a,B:b]                = F; aufgezaehlte Funktion, F(A) = a,
                                                     F(B) = b.
[a,b]                    = [0:a,1:b], (2-)Tupel.
[:(X @ 10) X**2 ]        = Q; termdefinierte Funktion, Q(9) = 81,
                              Q(-1) = Q(10) =...= ANY (Vereinbarung).
```

Stringmengen

```
                  -
                 | AGE
AGE#SEX =  --- <  #          Menge der Strings der Laenge 2
                 | SEX       mit Elementen aus AGE an erster
                  -          und Elementen aus SEX an zweiter
                             Stelle.

                   -
                  |
#CHAR   =     ---<  CHAR     die Druckzeichen-Reihen.
                  | (#)
                   -
```

Strukturen

Datenstrukturen werden als Abbildungsmengen zwischen 'Ortsbezeichnern' (abstrakten Adressen) in die dort zulaessigen Wertmengen aufgefasst.

```
                              -
                             | A:NAT     Menge der Abbildungen von
[A:NAT]&[B:BOOL]   =     ---<  &        {A,B} in NAT (fuer A) und
                             | B:BOOL    BOOL (fuer B).
                              -

                              -
                             |
&[:(K@20) REAL]    =     ---<  K: REAL   Multipel: Menge der Abbil-
                             | (&:K@20)   dungen von [0..19] in REAL.
                              -
```

```
                                  -
                                  |
                =       ---<  REAL       Abkuerzung fuer das
                              | (:20)     Multipel.
                                  -
```

INT&BOOL = [INT]&[BOOL] = &[INT,BOOL] ; Abkuerzung fuer Tupelmengen. Im Klammerdiagramm muss '0:INT' bzw. '1:BOOL' stehen.

Abbildungsmengen, allgemein

A/-->B Abbildung von A in B. Die vollstaendige Form ist (mit '[[....]]' fuer Option, '|' fuer Alternative):

A [[/]] -- [[<]] > [[/]] [[(ASC|DES)]], dabei betreffen:

'/' die Vollstaendigkeit, '<' die Umkehrbarkeit, 'ASC' 'DES' die Monotonie (aufsteigend, fallend).

KOMPONENTEN IN DEN KLAMMERN

Die Komponenten der Klammern haben in der Regel folgenden Aufbau:

[[Markendefinition]] [[Zugriffsart.]] Ortsangabe :
Wertmenge [[Wertzusatz]]

(mit '[[...]]' fuer Option)

Markendefinition

(Bezeichner >) Marken werden hauptsaechlich als Vorbereitung fuer invertierte Codierung benutzt.

Zugriffsart

(keine)	bei Variablen, Zugriffsart ist 'schreibend', ausser im Input-Teil von Prozesstypen.
IN	lesender Zugriff auf Kanal
OUT	schreibender Zugriff auf Kanal
PASS	Aufruf: OUT, gefolgt von IN.
FIN	Rueckkehr (nach Aufruf). - FIN(S1>) besagt, dass vor der Rueckkehr ein Status fuer den naechsten Aufruf zu stellen ist.
INIT FINIT	Anfang, Ende eines Prozesses.

Ortsangabe

Bezeichner von Kanaelen oder Variablen. Kann mit '.' angekettete Selektoren haben z.B:

KONTO.KUNDE , MASCHINE.(J+1).

Bei PASS enthaelt ein an letzter Stelle geketteter Klammerausdruck Output-Parameter,z.B:

PASS.TIMER.(WARTEZEIT): 'DA'.

Wertmengen
---
koennen alle vorangehend Beschriebenen sein.

Wertzusatz

(=Z) Z ist ein hier festgelegter Wertbezeichner.
(=Z>0) es koennen Bedingungen angeschlossen werden.

Sprechweise:

INT(=Z) :'ein INT, der ab hier Z heisst'

Dazu komplementaer:

INT/\'Z' :'ein INT, u n d z w a r Z ';
(der bereits bekannt ist).

TOP-KOMPONENTEN

Eine Topkomponente ist eine Komponente an der oberen Spitze einer Klammer. Es gibt mehrere Arten, die aneinander gereiht werden koennen.

Auswahl-Spezifikation

(+:X@100) waehle X aus 0 bis 99
(+:X = 3) 'waehle' X gleich 3
(+:X Y @ 10 ;/\ X+Y <= 15) waehle X und Y aus 0..9 und so, dass X+Y <= 15 ist.

Dies ist die 'Vielfach'- Form des '+'- Konnektors.

Versicherungen

(V@VST, Z@ZIEL) ; V und Z sind ausserhalb der Klammer definiert worden. Es wird 'versichert', dass V ein VST und Z ein ZIEL ist.

Argument-Spezifikation

[:X@NAT] X ist ein formaler Parameter fuer die Klammer, der eine natuerliche Zahl sein darf.
Die Klammer wird in anderen Diagrammen mit 'Klammername(P)', P: der aktuelle Parameter - zitiert.

Unterscheider-Spezifikation

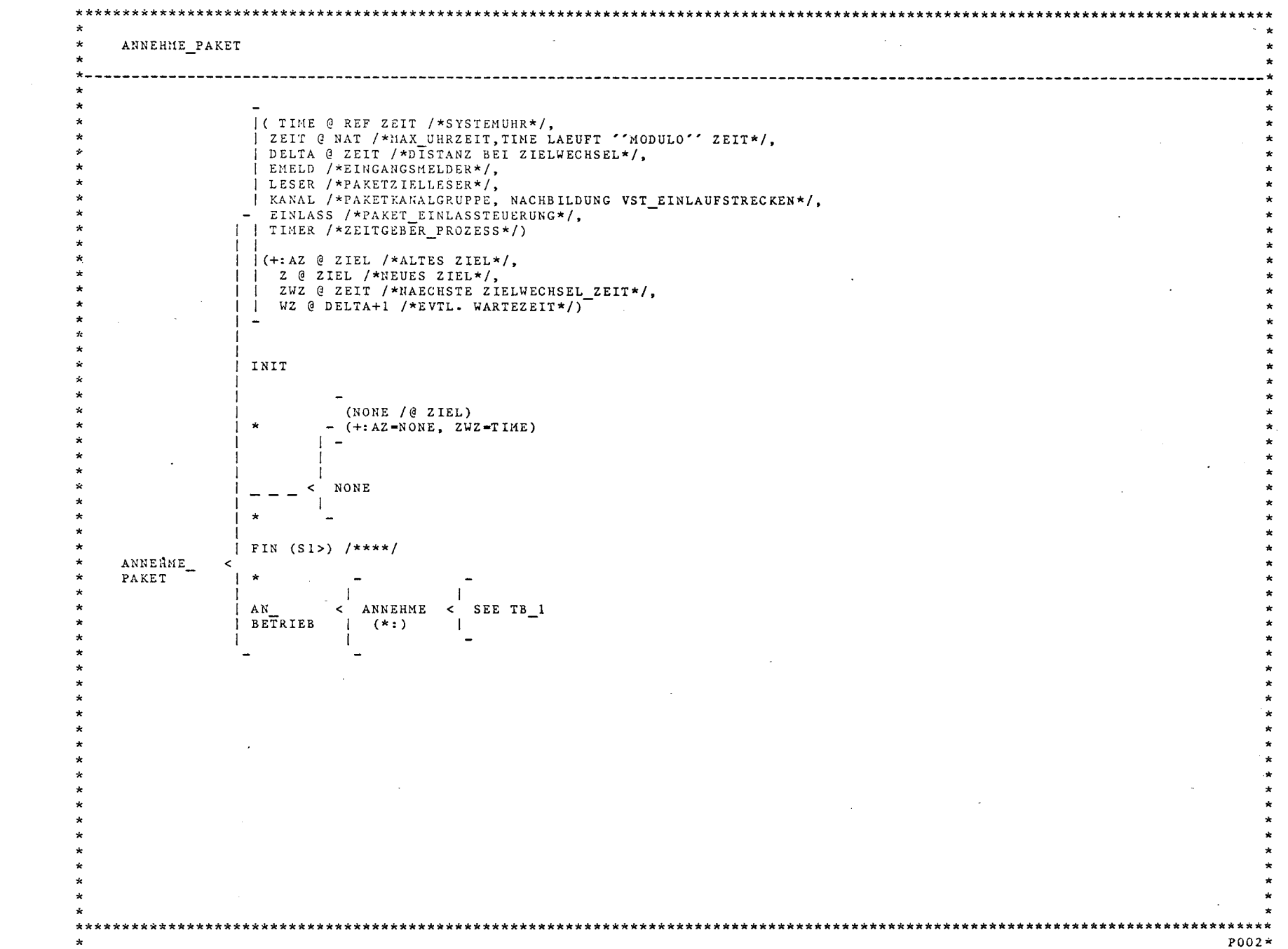
[:: V @ VERTEILSTATION] ; die Klammer beschreibt eine Prozessfamilie, deren Mitglieder durch V unterschieden werden. Ein Mitglied der Familie wird mit 'Klammername(W:)' angesprochen.- Damit ist gesagt, dass Wertbezeichner und Variable der Prozesse verschieden sind, ohne dass dies - durch Selektoren o.ae. - angezeigt sein muss.

```
PAKET_VERTEIL_STEUERUNG

                 /*SYSTEM ZUR STEUERUNG EINER PAKET_VERTEILANLAGE, BESTEHEND
                   AUS EINER EINGANGSSTATION UND BAUMFOERMIG NACHFOLGENDEN
                   VERTEILSTATIONEN, S. AUFGABENSTELLUNG*/

                 (+:)PAKET_VERTEIL_DEFINITIONEN  SEE.../*ENTHAELT TYP_ ODER WERTANGABEN
                    ZU ORTEN, MENGEN, ABBILDUNGEN, U.S.W. MIT KOMMENTAREN.
                    NICHT AUSGEFUEHRT IN VORLIEGENDER FASSUNG.*/

                                          /*STARTS FUER ALLE PROZESSE*/

PAKET_      <   UEBER_     <   /*EVTL_STATUS_BERICHTE, ODER SELEKTIVE AUSSERBETRIEBNAHME VON
 VERTEIL_       WACHER         VERTEILSTATIONEN, LEERFAHREN D. ANLAGE U. AE.MOEGLICHKEITEN*/
 STEUERUNG

                                          &&

                                          /*STEUEREINGRIFFE BEDIENPERSONAL*/

                 &                        /*GGF. VERWERTUNG VON STOERUNGSMELDUNGEN*/

                 ARBEITS_    <   SEE TB_1
                   SYSTEM
```

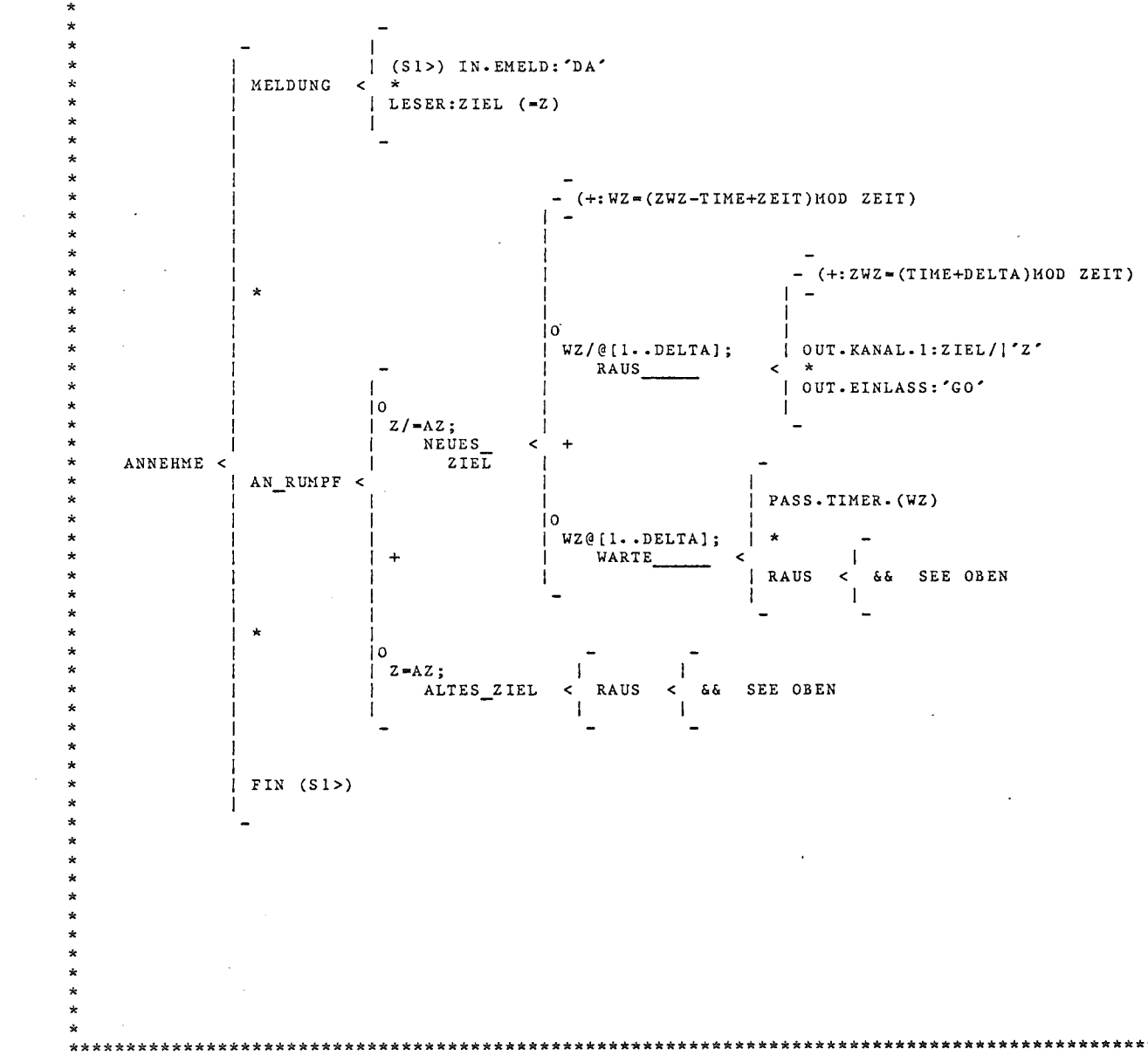
TB_1: ARBEITS_SYSTEM

```
                                  -
                                  |
                                  | (/*FUEHRT EINGANGS_STATION*/)
                                  |
                        -         | /*EINLASS_ORGAN*/  OUT.EINLASS:...GO
                        |         | /*EINGANGS_KANAL*/  OUT.KANAL.1:...ZIEL
                        | ANNEHME_ <  /*WARTEZEITEN*/  PASS.TIMER.[ZEIT]
                        |    PAKET  |
                        |           | &&
                        |           |
                        |           | /*EINGANGS_MELDER*/  IN./*EMELD*/:.../*'DA'*/
                        |           | /*LESE_STATION*/  LESER:...ZIEL
                        |           |
                        |           -
                        |           |
                        |           -
                        |           - (VST /*VERTEILSTATIONEN*/)
                        |           | -
                        |           | |
                        |           | |                  -
                        | &         | |                  |
                        |           | |                  | /*WEICHEN_STELLER*/
                        |           | |                  |    WEICH.V:...('L,R')
                        |           | |                  |
                        |           | |                  |0
                        |           | |                  | V @ EVST /*ENDVERTEILER*/;
                        |           | |                  |    /*PROTOKOLLE*/
ARBEITS_SYSTEM          <           | |                  |       OUT.DRUCKER:...ZIEL#ZIEL
                        |           | |                  |
                        |           | |                  |                                  -
                        |           | |                  |0                                 |
                        |           | |                  | V /@ EVST;                       | FOLGER(V,L):...ZIEL
                        | VERTEIL_ <  VERTEILE_    <     |    /*PAKET_KANAELE,            <  | &
                        |  BEREICH  |  PAKET(V:)         |       AUSLAUF*/  OUT.KANAL:_ _   | FOLGER(V,R):...ZIEL
                        |           | (&:V@VST)          |                                  |
                        -           |                    |                                  -
                                    -                    |
                                                         | &&
                                                         |
                                                         | /*EIN- U. AUSGANGSMELDER*/
                                                         | IN.MELD.V:...'V'#('REIN,LRAUS,RRAUS')
                                                         |
                                                         | /*PAKETKANAL,EINLAUF*/
                                                         |    IN.KANAL.V:...ZIEL
                                                         |
                                                         -
```

P001


ANNEHME_PAKET
(TIME @ REF ZEIT /*SYSTEMUHR*/,
ZEIT @ NAT /*MAX_UHRZEIT,TIME LAEUFT ''MODULO'' ZEIT*/,
DELTA @ ZEIT /*DISTANZ BEI ZIELWECHSEL*/,
EMELD /*EINGANGSMELDER*/,
LESER /*PAKETZIELLESER*/,
KANAL /*PAKETKANALGRUPPE, NACHBILDUNG VST_EINLAUFSTRECKEN*/,
EINLASS /*PAKET_EINLASSTEUERUNG*/,
TIMER /*ZEITGEBER_PROZESS*/)
(+:AZ @ ZIEL /*ALTES ZIEL*/,
Z @ ZIEL /*NEUES ZIEL*/,
ZWZ @ ZEIT /*NAECHSTE ZIELWECHSEL_ZEIT*/,
WZ @ DELTA+1 /*EVTL. WARTEZEIT*/)
INIT
(NONE /@ ZIEL)
(+:AZ=NONE, ZWZ=TIME)
NONE
FIN (S1>) /****/
ANNEHME_ PAKET
AN_ BETRIEB
ANNEHME (*:)
SEE TB_1

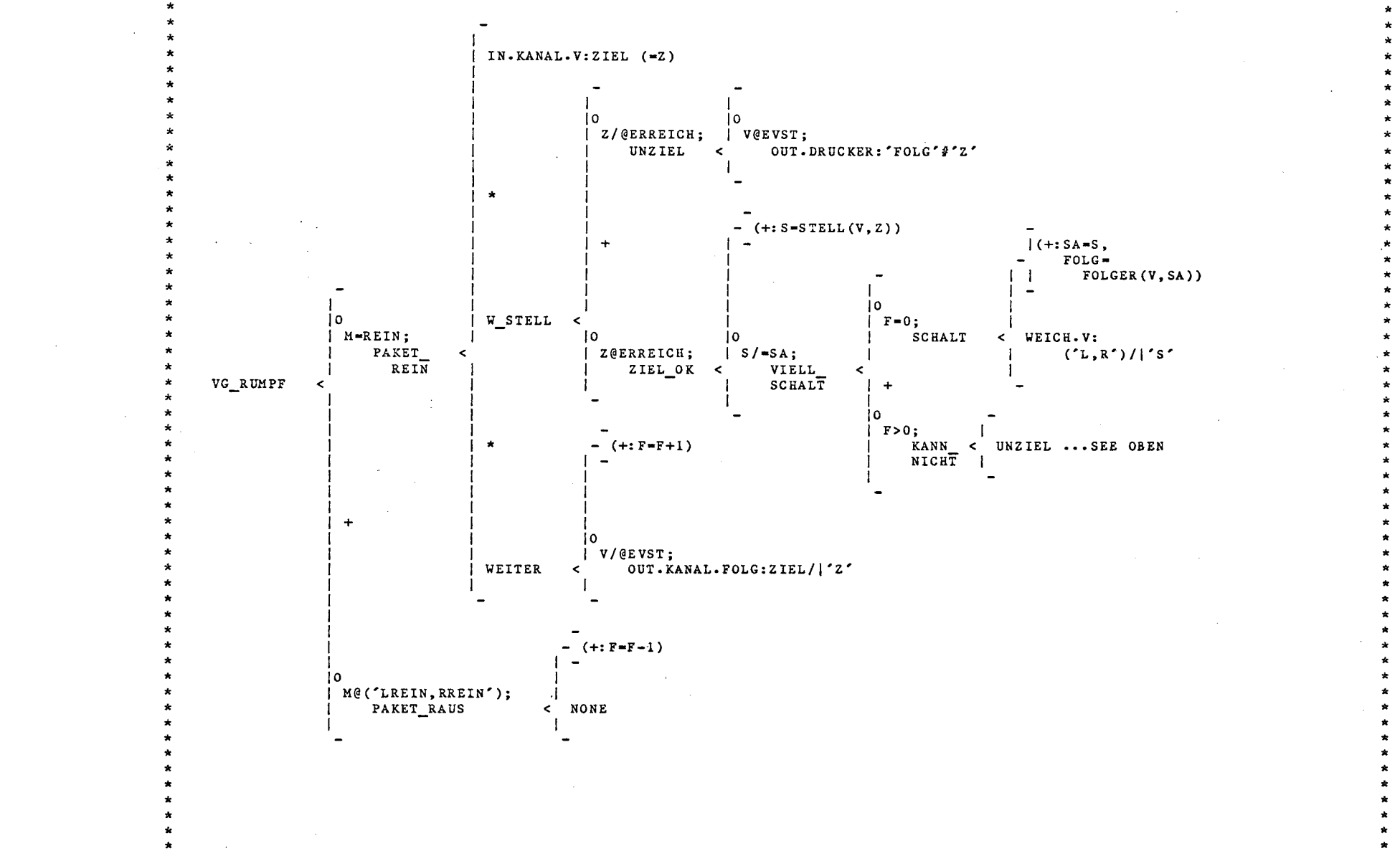
TB_1: ANNEHME


MELDUNG
(S1>) IN.EMELD:'DA'
LESER:ZIEL (=Z)
(+:WZ=(ZWZ-TIME+ZEIT)MOD ZEIT)
(+:ZWZ=(TIME+DELTA)MOD ZEIT)
WZ/@[1..DELTA];
RAUS
OUT.KANAL.1:ZIEL/|'Z'
OUT.EINLASS:'GO'
Z/=AZ;
NEUES_
ZIEL
ANNEHME
AN_RUMPF
PASS.TIMER.(WZ)
WZ@[1..DELTA];
WARTE
RAUS
&& SEE OBEN
Z=AZ;
ALTES_ZIEL
RAUS
&& SEE OBEN
FIN (S1>)

```
*****************************************************************************************************************
*                                                                                                               *
*    VERTEILE_PAKET                                                                                             *
*                                                                                                               *
*---------------------------------------------------------------------------------------------------------------*
*                                                                                                               *
*                     -                                                                                         *
*                     |( VST    /*VERTEILSTATIONEN*/,                                                           *
*                     |  EVST<<VST    /*ENDVERTEILSTATIONEN*/,                                                  *
*                     |  ERR    /*ERREICHBARKEIT ZIEL V.VST*/,                                                  *
*                     |  FOLGER    /*FOLGESTATION, ABHGG.V.STELLUNG*/,                                          *
*                     |  STELL    /*STELLUNG F. ZIEL BEI ERREICHBARKEIT*/,                                      *
*                     |  ('REIN,LRAUS,RRAUS')    /*LICHTSCHRANKENSIGNALE*/,                                     *
*                     |  ('L,R')    /*LINKS,RECHTS: WEICHENSTELLUNGEN*/,                                        *
*                     |  WEICH    /*WEICHEN (_GRUPPE)*/,                                                        *
*                    -   KANAL    /*PAKET_KANAL_GRUPPE,                                                         *
*                   | |           F. NACHBILDUNG VST_EINLAUF_STRECKEN*/,                                        *
*                   | |  MELD    /*EIN_ U. AUSGANGSMELDER,ZUSAMMENGEFASST*/)                                    *
*                   | |                                                                                         *
*                   | |[::V @ VST   /*V IST DIE BETRACHTETE VST*/]                                              *
*                   | |(+:ERREICH=ERR(V))                                                                       *
*                   | |(+:SA /*STELLUNG_ALT*/, F /*FUELLSTAND DER WEICHE*/,                                     *
*                   | |   S /*STELLUNG DER WEICHE*/, Z /*ZIEL DES JEWEILIGEN PAKETS*/)                          *
*                   | -                                                                                         *
*                   |                                                                                           *
*                   |                                                                                           *
*                   | INIT: VST/|'V'                                                                            *
*                   |                                                                                           *
*                   |       -                                                                                   *
*                   |       |(+:SA@('L,R'),                                                                     *
*                   |      -    FOLG=FOLGER(V,SA),                                                              *
*                   | *   | |   F=0)                                                                            *
*                   |     | -                                                                                   *
*                   |     |                                                                                     *
*                   |     |                                                                                     *
*                   | AN <  WEICH.V:'SA'                                                                        *
*                   |     |                                                                                     *
*                   | *   -                                                                                     *
*                   |                                                                                           *
*                   | FIN (S1>) /****/                                                                          *
*                   | /*ENDE BEI                                                                                *
*                   |   INITIALISIERUNG*/                                                 -                     *
*                   |                                                                     |                     *
*                   |                                 -                                   | VST/|'V'            *
*    VERTEILE_  <   | *            -                  |                                   | #                   *
*     PAKET         |             |                   | (S1>)IN.MELD.V:_ _             <  ('REIN,RRAUS,LRAUS') (=M) *
*                   | BETRIEB  <  VORGANG  <          |                                   | ;                   *
*                   |             |  (*:)             | *        -                        -                     *
*                   -             |                   |          |                                              *
*                                 -                   | VG_   <  SEE TB_1                                       *
*                                                     | RUMPF    |                                              *
*                                                     |          -                                              *
*                                                     | *                                                       *
*                                                     |                                                         *
*                                                     | FIN (S1>)                                               *
*                                                     |                                                         *
*                                                     -                                                         *
*                                                                                                               *
*                                                                                                               *
*                                                                                                               *
*****************************************************************************************************************
*                                                                                                          P003*
```

- 13/38 -

TB_1: VG_RUMPF



P003*

Program Design Language (PDL)

Bearbeiter: Hans Keutgen, GEI

PDL wurde als Entwurfswerkzeug von St.A. Caine und E.K. Gordon auf der NCC 1975 vorgestellt. Das Produkt basiert auf den Überlegungen zu Pseudocode, einem Ansatz der IBM im Rahmen der Improved Programming Techniques.

PDL-Kurzbeschreibung

PDL beinhaltet sowohl eine Sprache als auch einen Prozessor. Die Sprache entnimmt ihr Vokabular der Umgangssprache und unterwirft sie den syntaktischen Regeln einer formalen Sprache (wie etwa einer Programmiersprache). Der Prozessor bringt die mit Hilfe eines üblichen Texteditors erstellte formatfreie Eingabe für PDL in ein übersichtliches Layout und generiert Verweislisten des Kontroll- und Datenflusses.

In PDL besteht jeder Systementwurf aus einer Menge von Entwurfssegmenten, die in einer Baumstruktur verknüpft sind. Die Detaillierung des Systementwurfs erfolgt durch die Methode der schrittweisen Verfeinerung dieser Struktur.

Das Entwicklungsdokument, das der Prozessor erstellt, besteht aus:

- o einer Titelseite, die den Namen und das Datum des Entwurfs sowie den Installationsort des Prozessors enthält,
- o einem Inhaltsverzeichnis,
- o dem eigentlichen Entwurfsprogramm aus:
  - Datensegmenten,
  - Textsegmenten,
  - Ablaufsegmenten,
  - Externsegmenten
- o einem Datenindex, aus dem ersichtlich ist, wo ein Datenelement definiert und angesprochen wird,
- o einem Segmentindex, dem die Aufrufe der Ablaufsegmente und Externobjekte entnommen werden können,
- o einem Segmentaufrufbaum, mit dem die Hierarchie des Systems dargestellt wird.

Die unterschiedlichen Segmenttypen werden durch spezielle Umrandungen gekennzeichnet. Ein Textsegment enthält ergänzende Kommentare, ein Datensegment Definitionen von Datenobjekten.

Ein Ablaufsegment enthält Kontrollflußinformationen und entspricht etwa einer Prozedur in der Implementierung. Falls in einem Ausdruck eines Ablaufsegmentes ein Verweis auf ein anderes Segment auftritt, wird die Seitenzahl, auf der dieses Segment beschrieben wird, am linken Rand dieses Ausdrucks angegeben. Die formatfrei eingegebenen Ablaufsegmentbeschreibungen werden durch den PDL-Prozessor in eine leicht lesbare Form (z.B. Einrückungen, Unterstreichungen von Schlüsselwörtern usw.) gebracht.

Die Beschreibung des Kontrollflussses in einem Ablaufsegment lehnt sich in seinen Grundelementen an die Konzepte der strukturierten Programmierung an. Die beiden elementaren Sprachelemente sind das IF- und das DO-Konstrukt zur Beschreibung einer bedingten bzw. wiederholten Ausführung.

Durch Externsegmente kann der Bezug zu Ablaufsegmenten in anderen Entwurfsteilen hergestellt werden.

Literatur

S.H. Caine, E.K. Gordon
PDL - A Tool for Software Design; Proc of the NCC 1975, pp271-276

PROGRAM DESIGN LANGUAGE PROCESSOR

26 JUN 80

PDL 03.06

GGGG EEEE II
G   E   II
G GG EEEE II
G G E   II
GGGG EEEE II

ALBERT-EINSTEIN-STR. 61
5100 AACHEN-WALHEIM

TEL.: 02408/7024
TLX.: 8329745 GEI D


```
*****************************
*                           *
* DIE PAKETVERTEILANLAGE    *
*                           *
*         26 JUN 80         *
*                           *
*         PDL 03.06         *
*                           *
*****************************
```

TABLE OF CONTENTS

```
**********************************
*                                *
* ALLGEMEINE ERLAEUTERUNGEN      *
*                                *
**********************************
```

VERWENDUNGSZWECK

```
##########################################################################################
#                                                                                        #
#  1    EIN IN DIE EINGANGSSTATION EINLAUFENDES PAKET IST DURCH EIN CODEZEICHEN          #
#  2    MARKIERT, DAS DIE ZIELSTATION ANGIBT. DAS STEUERSYSTEM LIEST DAS                #
#  3    CODEZEICHEN UND STEUERT DANACH DIE EINZELNEN VERTEILSTATIONEN.                   #
#                                                                                        #
##########################################################################################
```

SYSTEMBEGRIFFE

1	CODEZEICHEN	IST EINE MARKIERUNG AUF JEDEM PAKET, AUS DER DIE ZIELSTATION
2		BESTIMMT WERDEN KANN
3	ZIEL	GIBT DIE ZIELSTATION EINES PAKETES AN
4	RICHTUNG	GIBT FUER EIN PAKET AN, OB ES IN DER VERTEILSTATION NACH
5		LINKS ODER RECHTS GELENKT WERDEN MUSS
6	FEHLERZEICHEN	KENNZEICHNET EINEN FALSCHLAEUFER ODER EIN PAKET MIT NICHT
7		LESBAREM ODER SINNLOSEM CODEZEICHEN
8	VORGAENGER	EIN PAKET, DAS EINE STATION UNMITTELBAR VOR DEM AKTUELLEN
9		PAKET PASSIERT
10	WEICHE	LENKORGAN IN EINER VERTEILSTATION MIT ZWEI EINSTELLUNGEN
11	WARTESCHLANGE	ZEITLICH GEORDNETE FOLGE VON ZIELEN BZW ZIELEN MIT
12		FEHLERZEICHEN, DIE AN EINEM MELDEPUNKT ERWARTET WERDEN
13	PAKETZAEHLER	GIBT DIE ANZAHL DER PAKETE IN EINER VERTEILSTATION AN

```
**************************
*                        *
* DIE ABLAUFSTEUERUNG    *
*                        *
**************************
```

PROGRAMMVERTEILER DER STEUERUNG

```
REF
PAGE ***************************************************************************************************
     *                                                                                                   *
   7 *  1    INITIALISIERUNG                                                                            *
     *  2    DO FUER JEDE MELDUNG DER EINGANGSSTATION ... PARALLEL ABLAUFFAEHIG                         *
   9 *  3        BEHANDLE DIE EINGANGSSTATION                                                           *
     *  4    ENDDO                                                                                      *
     *  5    DO FUER JEDES EINGANGSSIGNAL EINER VERTEILSTATION ... PARALLEL ABLAUFFAEHIG                *
     *  6        STELLE NR DER VERTEILSTATION FEST                                                      *
  12 *  7        BEHANDLE EINGANGSSIGNAL DER VERTEILSTATION                                             *
     *  8    ENDDO                                                                                      *
     *  9    DO FUER JEDES AUSGANGSSIGNAL EINER VERTEILSTATION ... PARALLEL ABLAUFFAEHIG                *
     * 10        STELLE NR DER VERTEILSTATION UND RICHTUNG DES AUSGANGES FEST                           *
  13 * 11        BEHANDLE AUSGANGSSIGNAL DER VERTEILSTATION                                             *
     * 12    ENDDO                                                                                      *
     *                                                                                                   *
     ***************************************************************************************************
```

INITIALISIERUNG

```
REF
PAGE ********************************************************************************************************************
     *                                                                                                                  *
     *  1    SETZE EINGANGSSTATION IN LESESTELLUNG                                                                      *
     *  2    SETZE VERTEILSTATION1 FREI                                                                                 *
     *  3    DO FUER ALLE VERTEILSTATIONEN                                                                              *
     *  4        INITIALISIERE WARTESCHLANGE VERTEILSTATION EINGANG                                                      *
     *  5        INITIALISIERE WARTESCHLANGE VERTEILSTATION AUSGANG RECHTS                                               *
     *  6        INITIALISIERE WARTESCHLANGE VERTEILSTATION AUSGANG LINKS                                                *
     *  7    ENDDO                                                                                                      *
     *                                                                                                                  *
     ********************************************************************************************************************
```

VERBALE BESCHREIBUNG DER EINGANGSSTATION

```
#  1    DIE EINGANGSSTATION BESTEHT AUS EINEM EINGABEORGAN MIT DEN TEILEN
#  2    F1 UND F2. F2 HAELT DAS EINGELAUFENE PAKET SOLANGE FEST, BIS DAS
#  3    MELDEORGAN DIE ANKUNFT AN DAS STEUERSYSTEM GEMELDET HAT UND DIESES
#  4    MIT HILFE DES LESEORGANS DAS CODEZEICHEN AUFGENOMMEN HAT. DANACH GIBT
#  5    DAS STEUERSYSTEM EINEN AUFTRAG AN DAS FREIGABEORGAN, DIE SPERRE F2
#  6    GIBT DEN WEITERLAUF FUER DAS PAKET FREI UND DAS BESCHLEUNIGUNGSTEIL
#  7    F1 NEIGT SICH. DADURCH GLEITET DAS PAKET WEITER, GLEICHZEITIG WIRD
#  8    DAS NACHFOLGENDE PAKET SOLANGE AM EINLAUFEN GEHINDERT, BIS DIE
#  9    SPERRE F2 WIEDER EINGETRETEN IST.
# 10    .
# 11    BEI DER BEHANDLUNG DES NACHFOLGENDEN PAKETES IST ZU PRUEFEN, OB SICH
# 12    DAS ZIEL VON DEM DES VORLAEUFERS UNTERSCHEIDET. IN DIESEM FALL IST DIE
# 13    FREIGABE SEINES WEITERLAUFES SOLANGE ZU VERZOEGERN, DASS IN DIESER
# 14    ZEITSPANNE, DIE DADURCH ZWISCHEN DEM PASSIEREN DER EINZELNEN VER-
# 15    TEILSTATIONEN LIEGT, DIE UMSTELLUNG DER LENKORGANE DURCHGEFUEHRT
# 16    WERDEN KANN. IM ANDEREN FALL KOENNEN DIE PAKETE UNMITTELBAR AUFEIN-
# 17    ANDER FOLGEN.
```

```
     BEHANDLE DIE EINGANGSSTATION

REF
PAGE ********************************************************************************************************************
     *                                                                                                                  *
  10 *  1     LIES CODEZEICHEN                                                                                          *
     *  2     IF ZIEL UNGLEICH VORGAENGER-ZIEL UND FEHLERZEICHEN NICHT GESETZT                                          *
     *  3         WARTE AUF VERTEILSTATION-1 FREI                                                                       *
  16 *  4         SETZE WEICHE                                                                                          *
     *  5     ENDIF                                                                                                     *
     *  6     BELEGE WARTESCHLANGE DER ERSTEN VERTEILSTATION EINGANG                                                    *
     *  7     TRAGE ZIEL BZW FEHLERZEICHEN EIN                                                                          *
     *  8     GIB WARTESCHLANGE DER ERSTEN VERTEILSTATION EINGANG FREI                                                   *
     *  9     LOESE SPERRE                                                                                              *
     * 10     SETZE VORGAENGER-ZIEL AUF ZIEL                                                                            *
     *                                                                                                                  *
     ********************************************************************************************************************
```

```
     LIES CODEZEICHEN

REF
PAGE ********************************************************************************************************
     *                                                                                                      *
     *  1     IF CODEZEICHEN NICHT LESBAR ODER SINNLOS                                                    *
     *  2         SETZE FEHLERZEICHEN                                                                     *
     *  3     ELSE BESTIMME ZIEL                                                                          *
     *  4     ENDIF                                                                                       *
     *                                                                                                      *
     ********************************************************************************************************
```

VERBALE BESCHREIBUNG DER VERTEILSTATION

```
#########################################################################################
#                                                                                       #
#  1    EINGANGS- UND AUSGANGSPUNKTE JEDER VERTEILSTATION SIND MIT LICHTSCHRANKEN       #
#  2    VERSEHEN. DIESE KOENNEN DAS PASSIEREN DER EINZELNEN PAKETE MIT SICHERHEIT       #
#  3    ERKENNEN, AUCH WENN DIESE DICHT AUFEINANDER FOLGEN. DIE MELDUNGEN WERDEN IM     #
#  4    STEUERSYSTEM ZUR LAUFWEGERKENNUNG JEDES PAKETES AUSGEWERTET. DADURCH KANN IN    #
#  5    VERBINDUNG MIT DEM BEREITS MARKIERTEN ZIEL DER STEUERAUFTRAG FUER DAS NAECHSTE  #
#  6    LENKORGAN ERMITTELT WERDEN.                                                     #
#  7                                                                                    #
#  8    BEIM AUSGEBEN DES STEUERAUFTRAGES IST DARAUF ZU ACHTEN, DASS ALLE VORLAEUFER    #
#  9    DIE BETREFFENDE VERTEILSTATION PASSIERT HABEN. DIE VERTEILSTATION SELBST MUSS   #
# 10    FREI SEIN, D.H. ZWISCHEN EINGANGS- UND AUSGANGSLICHTSCHRANKE DARF SICH KEIN     #
# 11    PAKET BEFINDEN. ES KOENNTEN U.U. MEHRERE SEIN, WENN ES SICH UM PAKETE MIT       #
# 12    GLEICHER ZIELSTATION HANDELT.                                                   #
# 13                                                                                    #
# 14    TRITT AUFGRUND UNTERSCHIEDLICHER GESCHWINDIGKEITEN DER FALL EIN, DASS EIN       #
# 15    PAKET, FUER DAS EINE VERTEILSTATION UMZUSTELLEN IST, DEREN EINGANGSMELDE-       #
# 16    PUNKT ERREICHT, BEVOR DER VORLAEUFER DIESE VERLASSEN HAT, SO MUSS DIE UM-       #
# 17    STELLUNG UNTERBLEIBEN, DER FALSCHLAEUFER BEKOMMT DAS ZIEL SEINES VORLAEUFERS    #
# 18    UND DER VORGANG WIRD AUSGEDRUCKT.                                               #
#                                                                                       #
#########################################################################################
```

BEHANDLE EINGANGSSIGNAL DER VERTEILSTATION

REF
PAGE

```
*****************************************************************************************************
*                                                                                                   *
*  1    BELEGE WARTESCHLANGE DER VERTEILSTATION EINGANG                                             *
*  2    ENTNIMM NAECHSTES ZIEL.                                                                     *
*  3    GIB WARTESCHLANGE DER VERTEILSTATION EINGANG FREI                                           *
*  4    BELEGE PAKETZAEHLER                                                                         *
*  5    ERHOEHE PAKETZAEHLER                                                                        *
*  6    GIB PAKETZAEHLER FREI                                                                       *
*  7    BELEGE WARTESCHLANGE DER VERTEILSTATION AUSGANG GEMAESS WEICHENSTELLUNG                      *
*  8    TRAGE ZIEL EIN                                                                              *
*  9    GIB WARTESCHLANGE DER VERTEILSTATION AUSGANG GEMAESS WEICHENSTELLUNG FREI                    *
*                                                                                                   *
*****************************************************************************************************
```

BEHANDLE AUSGANGSSIGNAL DER VERTEILSTATION

```
REF
PAGE ********************************************************************************************************
     *                                                                                                      *
     *  1     BELEGE WARTESCHLANGE DER VERTEILSTATION AUSGANG GEMAESS WEICHENSTELLUNG                        *
     *  2     ENTNIMM NAECHSTES ZIEL                                                                        *
     *  3     GIB WARTESCHLANGE DER VERTEILSTATION AUSGANG FREI GEMAESS WEICHENSTELLUNG                      *
     *  4     IF WARTESCHLANGE LEER                                                                         *
     *  5         BELEGE WARTESCHLANGE DER VERTEILSTATION AUSGANG ENTGEGEN WEICHENSTELLUNG                   *
     *  6         ENTNIMM NAECHSTES ZIEL                                                                    *
     *  7         GIB WARTESCHLANGE DER VERTEILSTATION AUSGANG ENTGEGEN WEICHENSTELLUNG FREI                 *
     *  8         SETZE FEHLERZEICHEN                                                                       *
     *  9     ENDIF                                                                                         *
     * 10     BELEGE PAKETZAEHLER                                                                           *
     * 11     ERNIEDRIGE PAKETZAEHLER                                                                       *
     * 12     LIES PAKETZAEHLER AB                                                                          *
     * 13     GIB PAKETZAEHLER FREI                                                                         *
     * 14     IF KEIN PAKET MEHR IN VERTEILSTATION                                                          *
     * 15         BELEGE WARTESCHLANGE DER VERTEILSTATION EINGANG                                           *
     * 16         LIES NAECHSTES ZIEL                                                                       *
     * 17         GIB WARTESCHLANGE DER VERTEILSTATION EINGANG FREI                                         *
     * 18         IF WARTESCHLANGE LEER UND VERTEILSTATION IST ERSTE VERTEILSTATION                         *
     * 19             SETZE VERTEILSTATION-1 FREI                                                           *
  16 * 20         ELSE SETZE WEICHE                                                                         *
     * 21         ENDIF                                                                                     *
     * 22     ENDIF                                                                                         *
     * 23     IF PAKET BISHER KEIN FALSCHLAEUFER                                                            *
  14 * 24         FALSCHLAEUFERPRUEFUNG FUER AKTUELLES PAKET                                                *
     * 25     ENDIF                                                                                         *
     * 26     IF VERTEILSTATION IST LETZTE VERTEILSTATION VOR ZIELSTATION                                   *
  17 * 27         ZIELBEHANDLUNG                                                                            *
     * 28     ELSE BELEGE WARTESCHLANGE DER FOLGENDEN VERTEILSTATION EINGANG                                *
     * 29         TRAGE ZIEL EIN                                                                            *
     * 30         GIB WARTESCHLANGE DER FOLGENDEN VERTEILSTATION EINGANG FREI                               *
     * 31     ENDIF                                                                                         *
     * 32     SETZE VORGAENGER-ZIEL AUF ZIEL                                                                *
     *                                                                                                      *
     ********************************************************************************************************
```

FALSCHLAEUFERPRUEFUNG FUER AKTUELLES PAKET

```
REF
PAGE ********************************************************************************************************************
     *                                                                                                                  *
     *  1    IF MEHRERE PAKETE IN VERTEILSTATION                                                                       *
     *  2        IF RICHTUNG UNGLEICH VORGAENGER-RICHTUNG ODER VORGAENGER FALSCHLAEUFER                                *
     *  3            SETZE FEHLERZEICHEN                                                                               *
     *  4        ENDIF                                                                                                 *
     *  5    ELSEIF VERTEILSTATION NICHT DIREKT VOR ZIELSTATION                                                        *
  15 *  6        STEUERUNG DER NACHFOLGESTATION                                                                        *
     *  7    ENDIF                                                                                                     *
     *                                                                                                                  *
     ********************************************************************************************************************
```

STEUERUNG DER NACHFOLGESTATION

```
REF
PAGE ********************************************************************************************************************
     *                                                                                                                  *
     *  1    BELEGE WARTESCHLANGE DER FOLGENDEN VERTEILSTATION EINGANG                                                  *
     *  2    LIES ZAHL DER EINTRAEGE                                                                                    *
     *  3    GIB WARTESCHLANGE DER FOLGENDEN VERTEILSTATION EINGANG FREI                                                 *
     *  4    IF KEINE EINTRAEGE IN WARTESCHLANGE                                                                        *
     *  5        BELEGE PAKETZAEHLER DER FOLGENDEN VERTEILSTATION                                                        *
     *  6        LIES PAKETZAEHLER AB                                                                                   *
     *  7        GIB PAKETZAEHLER DER FOLGENDEN VERTEILSTATION FREI                                                     *
     *  8        IF KEIN PAKET IN DER FOLGENDEN VERTEILSTATION                                                          *
  16 *  9            SETZE WEICHE                                                                                       *
     * 10        ENDIF                                                                                                  *
     * 11    ENDIF                                                                                                      *
     *                                                                                                                  *
     ********************************************************************************************************************
```

```
     SETZE WEICHE

REF
PAGE ********************************************************************************************************************
     *                                                                                                                  *
     *  1     BESTIMME NACHFOLGENDE VERTEILSTATION                                                                      *
     *  2     IF WEICHE LIEGT FALSCH                                                                                    *
     *  3         LEGE WEICHE UM                                                                                        *
     *  4     ENDIF                                                                                                     *
     *                                                                                                                  *
     ********************************************************************************************************************
```

ZIELBEHANDLUNG

```
REF
PAGE ********************************************************************************************************************
     *                                                                                                                  *
     *  1     IF FEHLERZEICHEN GESETZT                                                                                  *
     *  2         DRUCKE ZIELSTATION UND FALLS BEKANNT RICHTIGES ZIEL AUS                                               *
     *  3     ENDIF                                                                                                     *
     *                                                                                                                  *
     ********************************************************************************************************************
```

INDEX TO DATA ITEMS

INDEX TO DATA ITEMS

INDEX TO DATA ITEMS

INDEX TO DATA ITEMS

PAGE	LINE	TYPE	NAME AND REFERENCES
			10 LIES CODEZEICHEN
			3
			12 BEHANDLE EINGANGSSIGNAL DER VERTEILSTATION
			2 8
			13 BEHANDLE AUSGANGSSIGNAL DER VERTEILSTATION
			2 6 16 29 32 32
			17 ZIELBEHANDLUNG
			2

```
*****************************
*                           *
* INDEX TO FLOW SEGMENTS    *
*                           *
*****************************
```

INDEX TO FLOW SEGMENTS

INDEX TO FLOW SEGMENTS

PAGE	LINE	TYPE	NAME AND REFERENCES
	15	FS	STEUERUNG DER NACHFOLGESTATION
			14 FALSCHLAEUFERPRUEFUNG FUER AKTUELLES PAKET
			6
	17	FS	ZIELBEHANDLUNG
			13 BEHANDLE AUSGANGSSIGNAL DER VERTEILSTATION
			27

```
*****************************
*                           *
* SEGMENT REFERENCE TREES *
*                           *
*****************************
```

PROGRAMMVERTEILER DER STEUERUNG

LN	DEF	SEGMENT
1	6	PROGRAMMVERTEILER DER STEUERUNG
2	7	INITIALISIERUNG
3	9	BEHANDLE DIE EINGANGSSTATION
4	10	LIES CODEZEICHEN
5	16	SETZE WEICHE
6	12	BEHANDLE EINGANGSSIGNAL DER VERTEILSTATION
7	13	BEHANDLE AUSGANGSSIGNAL DER VERTEILSTATION
8	16	SETZE WEICHE
9	14	FALSCHLAEUFERPRUEFUNG FUER AKTUELLES PAKET
10	15	STEUERUNG DER NACHFOLGESTATION
11	16	SETZE WEICHE
12	17	ZIELBEHANDLUNG

```
****************************
*                          *
* END OF DESIGN DOCUMENT   *
*                          *
****************************
```

```
PDL RUN TIME STATISTICS:
       170 INPUT CARDS PROCESSED
        10 FLOW SEGMENTS
      5000 WORDS OF CORE AVAILABLE,   16.7 PERCENT USED
```

PLASMA / D - Eine Sprache für den Systementwurf

Helmut Balzert, Joachim Christ, Bruno Hübner
TRIUMPH-ADLER-GRUPPE, Nürnberg

Zusammenfassung

PLASMA / D ist eine Sprache für den Systementwurf. Es werden ein Hierarchie-, ein Modul-, ein System-, ein Schnittstellen-, ein Zugriffs-, ein Projekt- und ein Dokumentationskonzept zur Verfügung gestellt, die eine praxisorientierte Reihenfolge von Entwurfsentscheidungen ermöglichen. In PLASMA / D wird das Sprachentwurfsproblem, eine geeignete Symbiose zwischen einem Hierarchie- und einem Modulkonzept zu realisieren, gelöst. PLASMA / D unterstützt die Methoden des Systementwurfs insbesondere durch eine ebenenweise Systembeschreibung. Durch die Sprachkonzepte sind ausführliche Überprüfungen und Analysen möglich, die die Zuverlässigkeit des Systementwurfs erhöhen.

Stichworte

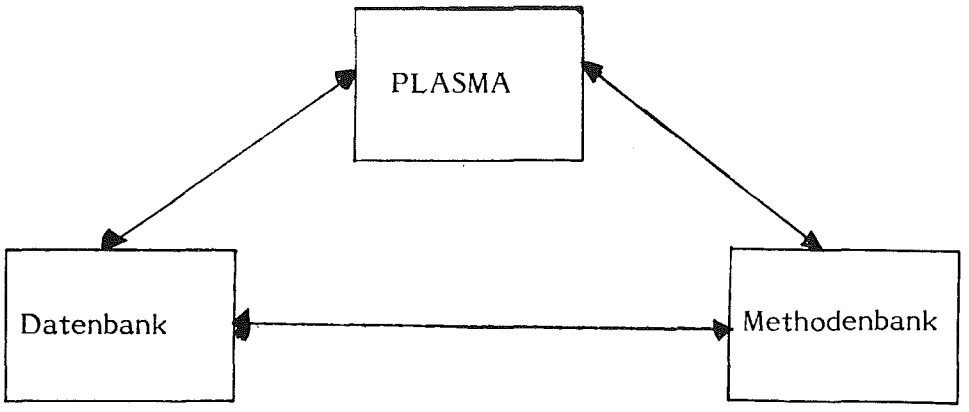
Systementwurf, Software Engineering, Hierarchiekonzept, Modulkonzept, Spezifikation, Sprachentwurf, PLASMA, Software-Entwicklungssystem.

Einleitung

Die Sprache PLASMA und das die Sprache umgebende PLASMA-System wurde im Rahmen der Dissertation von / Balzert 79 / entwickelt und in einer Pilotversion an der Universität Kaiserslautern implementiert. In einer Reihe von Projekt- und Diplomarbeiten sowie durch eine Förderung der DFG wurde die Implementierung verbessert und erweitert. Aufbauend auf diesen Arbeiten wurde eine Konzeption für ein umfassendes Software-Entwicklungssystem entworfen, das sich zur Zeit bei der Firma TRIUMPH-ADLER in der Realisierungsphase befindet. Das gesamte System wird Ende 1983 einsatzfähig sein.

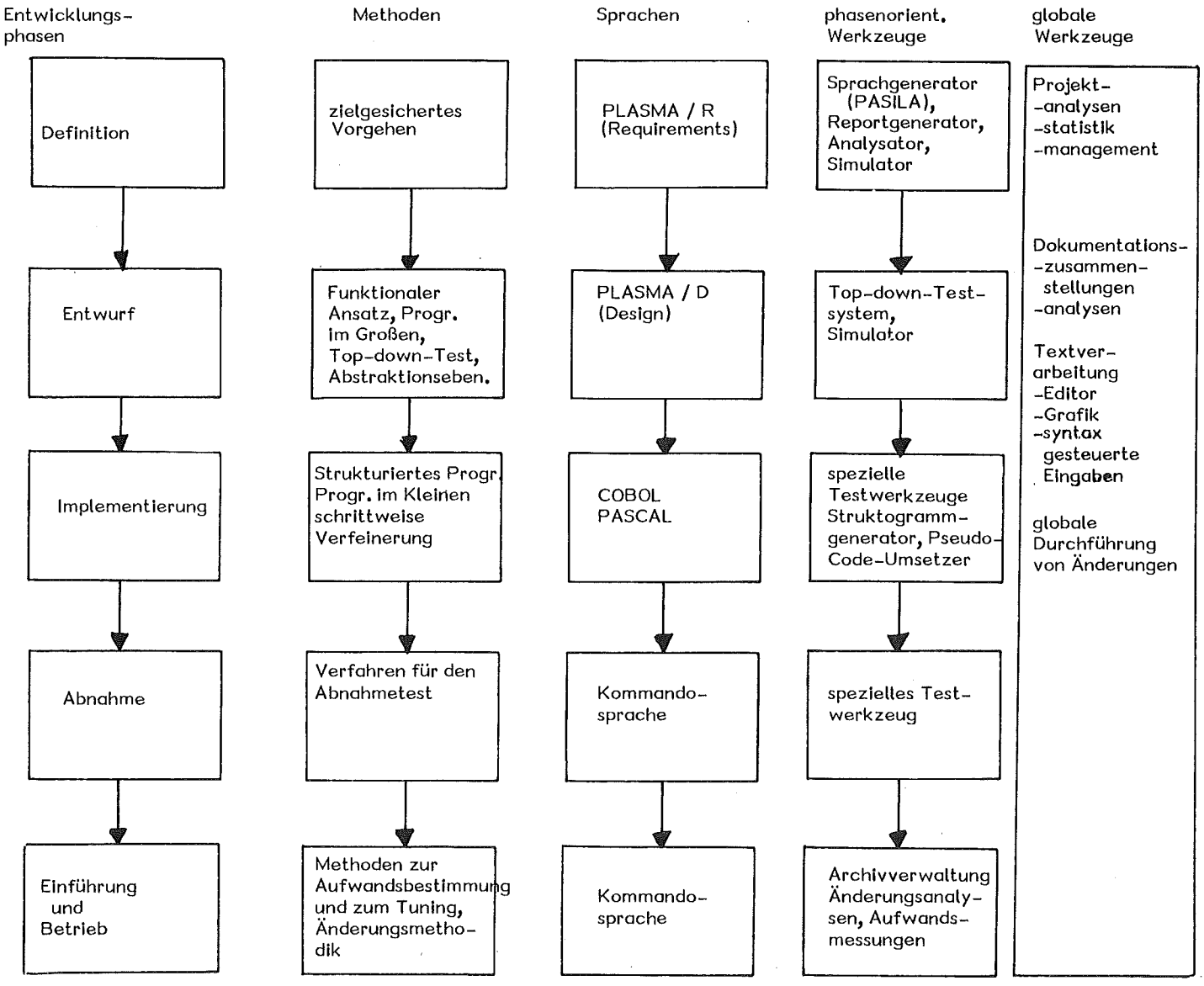
Das Software-Entwicklungssystem PLASMA

Das Gesamtkonzept zeigt folgende Abbildung:



Das Software-Entwicklungssystem PLASMA legt alle Daten in einer Datenbank ab. Neben der Datenbank greift das Software-Entwicklungssystem auf eine Methodenbank zu, die dazu dient, für besondere Anwendungsbereiche und Entwicklungsphasen auf Wunsch des Anwenders automatische Hilfsmittel und Werkzeuge zur Verfügung zu stellen. Die Methodenbank selbst hat Zugriff auf die Datenbank, um methodenbankspezifische Daten sowohl abzulegen als auch auf sie zuzugreifen.

Den Software-Entwicklungsprozeß kann man grob in folgende Phasen gliedern, die durch entsprechende Komponenten des PLASMA-Systems unterstützt werden:



Die Vorgehensweise bei der Software-Entwicklung wird durch ein organisatorisches Projektmodell festgelegt und durch das PLASMA-System erzwungen.

Die Systementwurfssprache PLASMA / D

Innerhalb des Software-Entwicklungssystems PLASMA wird für den Systementwurf die Sprache PLASMA / D verwendet. PLASMA / D (Programming Language for System Development, Modularization and Data Abstraction / Design) ist eine Sprache, die Konzepte und Beschreibungsmittel für den Systementwurf zur Verfügung stellt. Generelles Ziel ist es, den Systementwurf unabhängig von der Programmiersprache zu halten, in der die Modulimplementierung erfolgt.

PLASMA / D stellt für den Systementwurf folgende Sprachkonzepte zur Verfügung:

- Hierarchiekonzept zur Beschreibung der statischen Systemstruktur.

- Modulkonzept, das eine Moduldefinition (beschreibt, welche Ressourcen der Modul dem Benutzer bereitstellt und welche Ressourcen zur Implementierung verwendet werden können) und eine Modulimplementierung beinhaltet.

- Systemkonzept, das es ermöglicht, unterschiedliches Systemverhalten pro Systemkomponente zu modellieren:

  EXECUTIVE SYSTEM:

  Ermöglicht die Zusammenfassung von logisch zusammengehörenden Prozedur- und/oder Funktionsschnittstellen. Kennzeichnend für ein EXECUTIVE SYSTEM ist, daß die Prozeduren und/oder Funktionen Eingabegrößen in Ausgabegrößen transformieren und keine inneren Speicherzustände besitzen. Außer durch die Eingaben werden die Ausgaben durch keine anderen gespeicherten Größen beeinflußt, d. h. es ist kein internes Gedächtnis vorhanden.
  Beispiel: Komplexarithmetik

  DATA SYSTEM:
  Ermöglicht die Zusammenfassung von logisch zusammengehörenden Prozedur- und/oder Funktionsschnittstellen, die ein gemeinsames, internes Gedächtnis (Daten) besitzen und Operationen darauf ausführen.
  Das interne Gedächtnis besitzt eine Lebensdauer, die über das Aufrufende der Zugriffsoperatoren hinausgehen.
  Beispiel: Keller, Zufallszahlengenerator

  ABSTRACT SYSTEM:
  Ermöglicht die Beschreibung abstrakter Datentypen, d. h. Typen können parametrisiert werden und das System kann in anderen Systemkomponenten mit aktuellen Parametern vereinbart werden und mehrere Inkarnationen besitzen.
  Beispiel: typparametrisierte Keller

  VIRTUAL SYSTEM:
  Dient zur logischen Strukturierung. Ein VIRTUAL SYSTEM stellt alle oder eine Teilmenge seiner Kinder als Ressourcen zur Verfügung. Eine solche Systemkomponente besitzt keine zugehörige Implementierung.

  - Schnittstellenkonzept, das es ermöglicht, in eindeutiger Weise die bereitgestellten Ressourcen einer Systemkomponente bzw. eines Moduls an einer lokalen Stelle zu spezifizieren.

  - Zugriffskonzept, das es erlaubt, partielle Zugriffsrechte zwischen Systemkomponenten zu vergeben.

- Projektkonzept: PLASMA / D ist in ein umfassendes Software-Entwicklungssystem eingebettet, zu dem ein definierter Projektablauf gehört. In PLASMA / D werden dazu folgende Angaben gemacht: Autoren, Zeitangaben, Status, Version.
- Dokumentationskonzept: Für die automatische Zusammenstellung von Dokumentationen dienen folgende Angaben:

Purpose	: Zweck des Systems
User Interface	: Angaben, die die Benutzerschnittstelle betreffen.
Keywords	: Namen, die für Zwecke der Selektion, Analyse und für Geheimhaltungsaspekte ausgewertet werden.
Memos	: Namen, die auf Dokumente verweisen, die für mehrere Systemkomponenten relevant sind.
Pre- und Postcondition	: Bedingungen, die für die Systemkomponente gelten.

  Ein Textaufbereitungssystem kann direkt angesprochen werden.
- DATA POOL - Konzept, das es ermöglicht, Konstanten und Typen, die standardmäßig in einem Gesamtsystem oder in Systemteilen verwendet werden sollen, zu beschreiben.

PLASMA / D unterstützt die Methodik des Systementwurfs in folgender Form:

- Die Beschreibung erfolgt ebenenweise in Form von Ein-Ebenen-Systembeschreibungen. Dadurch wird eine Vorgehensweise in Abstraktionsebenen sowie eine arbeitsteilige Vorgehensweise ermöglicht. Die Problemkomplexität wird durch die Konzentration auf die Aspekte jeweils einer Ebene reduziert.
- Durch die für den Systementwurf bereitgestellten Sprachkonzepte ist sichergestellt, daß nur das "was" , nicht aber das "wie" in PLASMA / D beschrieben wird.
- Die Dokumentation ist integraler Bestandteil des Systementwurfs.
- Durch die PLASMA / D-Konzepte (insbesondere ebenenweise Beschreibung, Zuordnung von Modulimpl. zu jedem Baumknoten und -blatt) sind ein top-down-Test sowie die Formulierung von "dummies" möglich. Erst dadurch ist ein vollständiges top-down-Vorgehen möglich, das einen Integrationstest überflüssig macht.
- Eine entwurfskontrollierte Informationsverteilung ist durch Sprachkonzepte, die das Verbergen der Feinstruktur von Typen ermöglichen, sowie durch die Zugriffsregeln möglich.
- Für jede Ein-Ebenen-Systembeschreibung sowie für jede Modulimplementierung ist eine Qualitätskontrolle möglich.

Durch die Sprachkonzepte können eine Reihe von Überprüfungen (Vollständigkeits-, Konsistenzprüfungen) und Analysen (z. B. bei geplanten Modifikationen) vorgenommen werden, die wesentlich zur Erhöhung der Zuverlässigkeit des Systementwurfs beitragen.

Das gegenwärtig vorhandene Sprachkonzept von PLASMA / D beschränkt sich
- auf die Beschreibung sequentieller Systeme und
- auf die Beschreibung von Software-Systemen.

Das bedeutet, daß die vorhandenen Konzepte keine adäquate Beschreibung von Echtzeit-Problemen und von Hardware-Elementen erlauben. An entsprechenden Spracherweiterungen wird jedoch gearbeitet.

Überblick über die Sprachkonzepte

PLASMA / D besteht aus Ein-Ebenen-Systembeschreibungen einschl. Ressourcenbeschreibungen, Moduldefinitionen und Modulimplementationen.

Ein-Ebenen-Systembeschreibung

In PLASMA / D wird ein System ebenenweise entwickelt und beschrieben (Ein-Ebenen-Systembeschreibung).
Die einzelnen Ebenen werden nacheinander beschrieben.

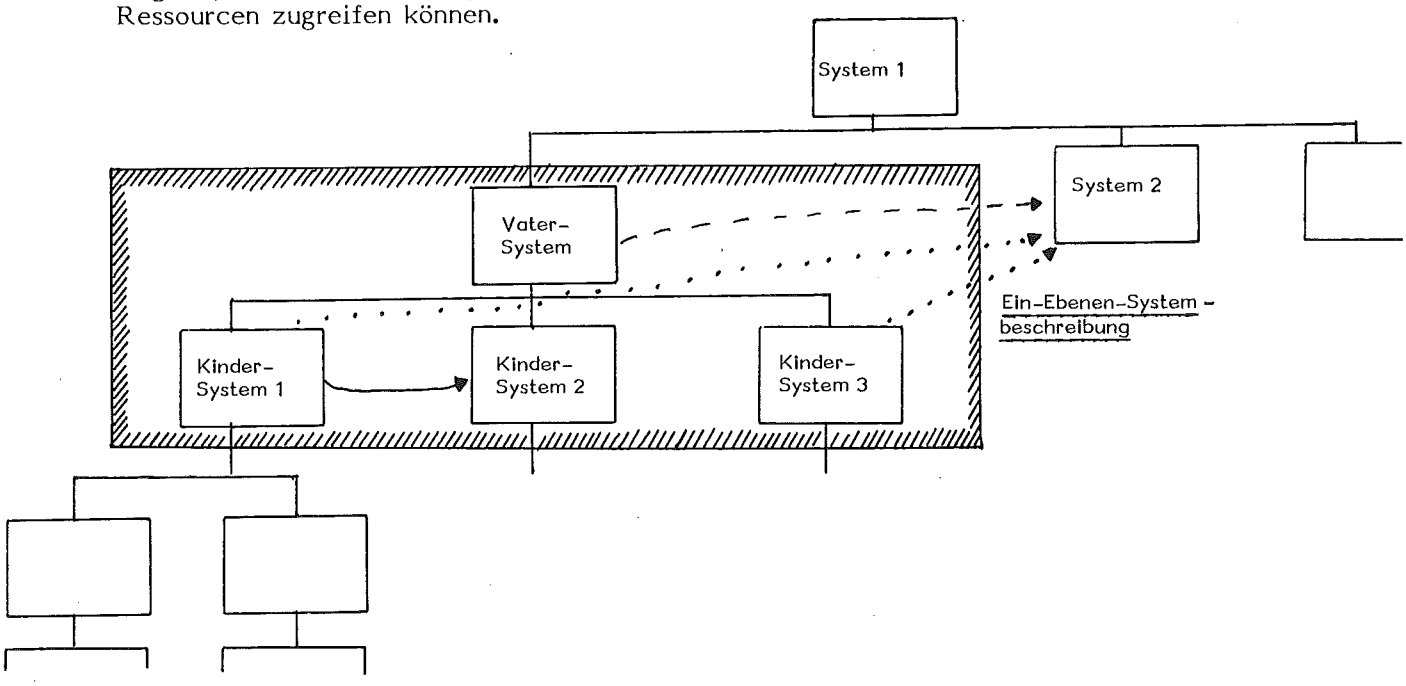
In einer Ein-Ebenen-Systembeschreibung sind sämtliche Informationen über eine Vater-Systemkomponente sowie sämtliche Informationen über alle seine Kinder enthalten (Informationen über die Enkel nicht).

Innerhalb eines Systems gelten in PLASMA folgende Sichtbarkeits- bzw. Zugriffsregeln:

a. Ein Vater-System kennt bzw. sieht alle seine Kinder (implizite Sichtbarkeit). Die Nachkommen der Kinder sind jedoch unsichtbar.

b. Ein Kinder-System kennt nur die Geschwister-Systeme, deren Sichtbarkeit das Vater-System explizit gestattet hat. Zugriffsrechte zwischen Geschwistern sind nicht transitiv.

c. Ein Kind kann Zugriffsrechte seines Vater-Systems erben. Das Vater-System muß die vererbten Zugriffsrechte explizit auflisten. Durch das Vererben des Zugriffs ist es möglich, daß tief unten liegende Untersysteme auf weit oben im Systembaum stehende Ressourcen zugreifen können.



explizit vom Vater-System vergebenes Zugriffsrecht zwischen den Geschwistern Kinder-System 1 und Kinder-System 2.

explizit vom System 1 vergebenes Zugriffsrecht zwischen den Geschwistern Vater-System und System 2.

Vererbt das Vater-System sein eigenes Zugriffsrecht auf das System 2 an seine Kinder 1 und 3, dann können diese Kinder ebenfalls auf das System 2 zugreifen (•••→).

Bei der Vergabe der expliziten Zugriffsrechte kann das Zugriffsrecht auf einzelne Prozeduren/Funktionen eingeschränkt werden. Ebenfalls ist es möglich, Zugriffsrechte weiterzugeben, ohne daß sie selbst verwendet werden können.

Aus jedem Kinder-System wird auf der nächst niedrigeren Systemebene ein Vater-System, das wiederum in Kinder-Systeme aufgeteilt werden kann.

Moduln

Die in den Ein-Ebenen-Systembeschreibungen aufgeführten Ressourcen müssen durch Moduln realisiert werden. Von jeder Ein-Ebenen-Systembeschreibung erzeugt das PLASMA-System unter Beachtung der Zugriffsrechte automatisch eine Moduldefinition.

Eine Moduldefinition besteht aus zwei Teilen. Der erste Teil beschreibt, welche Ressourcen der Modul dem Benutzer zur Verfügung stellt. Der zweite Teil zeigt dem Modulimplementator, welche Ressourcen zur Implementation benutzt werden können.

Im Gegensatz zur Ein-Ebenen-Systembeschreibung werden die benutzbaren Ressourcen in der Moduldefinition linear aufgelistet, d. h. man kann nicht mehr feststellen woher diese Ressourcen stammen. Diese Information benötigt der Modulimplementator aber auch nicht. Der Modul selbst - ebenso wie der Modulbenutzer und der Modulimplementator - werden von unnötiger Information befreit.

Die Moduldefinition beschreibt die Modulschnittstelle sowohl für den Implementator als auch für den Benutzer. Eine Moduldefinition ist vollständig in dem Sinne, daß der Modul anhand der Definition sowohl benutzt als auch implementiert werden kann.

Eine Moduldefinition muß durch eine Modulimplementation realisiert werden.
Moduln können prinzipiell in einer beliebigen Sprache implementiert werden.

Die Sprachkonzepte werden detailliert anhand des folgenden Beispiels erläutert.

Beispiel: Paketverteilanlage

Im folgenden wird das Beispiel "Paketverteilanlage" in PLASMA / D beschrieben. Aus Platzgründen wird sich auf die Beschreibung der oberen Ebenen beschränkt. Anhand des Beispiels werden wichtige Sprachkonzepte von PLASMA / D durch Anmerkungen erläutert.

Ein-Ebenen-Systembeschreibung, oberste Ebene:

```
SYSTEMBESCHREIBUNG : PAKET_VERTEIL_SYSTEM                    SEITE    1
-------------------------------------------------------------------------

EXECUTIVE SYSTEM PAKET_VERTEIL_SYSTEM

AUTHORS          DR. BALZERT, CHRIST, HUEBNER

DATES            JUNI/JULI 1980

1 STATUS         PRERELEASED

VERSION          1.00

PURPOSE
        SIEHE "AUFGABENSTELLUNG"

KEYWORDS         OEFFENTLICH,
                 TECHNISCHE_DOKUMENTATION

MEMOS            AUFGABENSTELLUNG,
                 BEDIENUNGSUNTERLAGEN,
                 WARTUNGSUNTERLAGEN

PRE-CONDITION    ANLAGE IST BETRIEBSBEREIT

POST-CONDITION   ES BEFINDET SICH KEIN PAKET MEHR
                 IN DER VERTEILANLAGE

2 INTERFACE

     PROCEDURE PAKETVERTEILSYSTEM;
     3 WITH CONST ZIELMAX = 8;
          TYPE  ZIELSTATION = 1 .. ZIELMAX FROM PDV_DATAPOOL;
          TYPE  ZIELCODE    = 0 .. 9999    FROM PDV_DATAPOOL;
```

---

1 Status-Information :

Für eine Ein-Ebenen-Systembeschreibung EES werden folgende Status-Arten unterschieden:

- dummy:	Bei Einrichtung einer EES z. B. bei Vergabe des Systemnamens
- incomplete:	Während der Bearbeitung durch den Systementwerfer
- complete:	Der Systementwerfer teilt dem PLASMA-System mit, daß aus seiner Sicht die EES vollständig ist. Sind die vom PLASMA-System dann vorgenommenen Semantiküberprüfungen korrekt, dann wird der Status auf "complete" gesetzt.
- prereleased	Nach Freigabe durch die Qualitätskontrolle
- released	Nach Abnahmetest des gesamten Systems durch die Qualitätskontrolle.

Die Verwaltung des Status geschieht durch das PLASMA-System.

2 Interface-Beschreibung :

In der Schnittstellen-Beschreibung werden alle diejenigen Prozedur- und/oder Funktionsschnittstellen beschrieben, die dieses System nach außen hin, d. h. für die nächst höhere Systemebene oder den Benutzer, zur Verfügung stellt.

4 SUBSYSTEMS ARE

DATA SYSTEM PAKET_EINGANG_VERARBEITEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_EINGANG_VERARBEITEN HAT ZWEI, SICH GEGENSEITIG AUSSCHLIESSENDE AUFGABEN:

1. FESTSTELLEN, OB EIN NEUES PAKET EINGEGANGEN IST. IN DIESEM FALL MUSS DER ZIELORT DES PAKETES ERMITTELT WERDEN.
   MOEGLICHE FEHLERFAELLE SIND: FEHLENDE ODER FALSCHE ZIELANGABE.

2. ZURUECKHALTEN EINES AUF FREIGABE WARTENDEN PAKETES. DIE WARTEZEIT WIRD IN EINHEITEN AUFGETEILT; DABEI IST EINE EINHEIT DIE ZEIT, DIE ZWISCHEN ZWEI AUFRUFEN DES SYSTEMS PAKET_EINGANG_VERARBEITEN VERSTREICHT. DIE WARTEZEIT ENTFAELLT, WENN ZWEI AUFEINANDERFOLGENDE PAKETE DAS GLEICHE ZIEL HABEN. ANSONSTEN WIRD EINE KONSTANTE ANZAHL VON WARTE- EINHEITEN GROESSER ALS NULL GEWAEHLT.

INTERFACE

PROCEDURE EINGANG;

PROCEDURE INIT_EINGANG;
PURPOSE
DIE EINGANGS- STATION WIRD INITIALISIERT : SPERRE ARRETIEREN, BESCHLEUNIGER SENKEN, LICHTSCHRANKE UND CODELESER INITIALISIEREN.

5 HAS ACCESS TO
DATA SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN VIA
PROCEDURE PAKET_AM_EINGANG (IN CODE : ZIELCODE);
3 WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;

END PAKET_EINGANG_VERARBEITEN

---

3 with - Teil :

Im with - Teil wird angegeben, welche Konstanten- und Typdefinitionen für das beschriebene System gültig sind. I. a. werden hier die Parametertypen spezifiziert. Entweder werden die Konstanten und Typen direkt spezifiert oder es erfolgt ein Verweis auf einen DATA POOL. Die Spezifikationen sollten dann in einen DATA POOL gelegt werden, wenn sie standardmäßig in einem Gesamtsystem oder in Systemteilen verwendet werden sollen. Bei der Ausgabe der Ein-Ebenen-Systembeschreibungen können die DATA POOL-Angaben in jede EES expandiert werden oder gesondert ausgedruckt werden (im Beispiel der Fall).

4 Aufteilung in Untersysteme :

Nach "subsystems are" wird die funktionale Aufteilung des Systems "Paket-Verteil-System" in Untersysteme beschrieben.

5 Geschwister-Zugriffe :

Durch "has access to" werden Zugriffsrechte zwischen Geschwistern durch den Systementwerfer des Systems "Paket_Verteil_System" vergeben.
Das Untersystem "Paket_Eingang_Verarbeiten" hat also Zugriffsrechte auf das Geschwister-System "Paket_Fluß_Steuern_und_Ueberwachen" und dort auf die Prozedur "Paket_am_Eingang", d. h. das Zugriffsrecht ist auf diese Prozedur eingeschränkt.

DATA SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN FUEHRT UEBER JEDES IN DER VERTEIL- ANLAGE BEFINDLICHE PAKET BUCH UND SORGT FUER DIE RICHTIGE STELLUNG DER VERTEILER- WEICHEN.

INTERFACE

```
PROCEDURE PAKETE_STEUERN;
   PURPOSE
    DIE PROZEDUR PAKETE_STEUERN KONTROLLIERT DER REIHE NACH
    SAEMTLICHE VERTEIL- STATIONEN UND VERBUCHT ALLE DURCHLAEUFE
    DER PAKETE.

PROCEDURE PAKET_AM_EINGANG ( IN CODE : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
   PURPOSE
    IN DIE VERBUCHUNG DER PAKETE WIRD EIN NEUES PAKET
    AUFGENOMMEN.

PROCEDURE PAKET_AM_AUSGANG ( IN ZIEL : ZIELSTATION );
   WITH CONST ZIELMAX = 8;
        TYPE  ZIELSTATION = 1 .. ZIELMAX FROM PDV_DATAPOOL;
   PURPOSE
    EIN PAKET WIRD AUS DER BUCHFUEHRUNG GESTRICHEN. EVENTUELL
    WIRD EINE FEHLERMELDUNG AUSGESCHRIEBEN, DIE DIE GEWUENSCHTE
    UND DIE TATSAECHLICHE ZIELANGABE ENTHAELT.

PROCEDURE INIT_PAKETE_STEUERN;
   PURPOSE
    DIE EINZELNEN VERTEIL- STATIONEN WERDEN INITIALISIERT, D.H.
    DIE WEICHEN IN EINE VORDEFINIERTE POSITION GEBRACHT UND DIE
    LICHTSCHRANKEN INITIALISIERT.

FUNCTION PAKET_ANZAHL : INTEGER;
   PURPOSE
    ES WIRD DIE ANZAHL DER MOMENTAN IN DER VERTEIL- STATION
    BEFINDLICHEN PAKETE ERMITTELT.

END PAKET_FLUSS_STEUERN_UND_UEBERWACHEN
```

6

6

6

6 Parameterliste :

Parameterlisten sind syntaktisch analog wie in PASCAL aufgebaut. In PLASMA / D werden jedoch drei Parameterarten unterschieden:

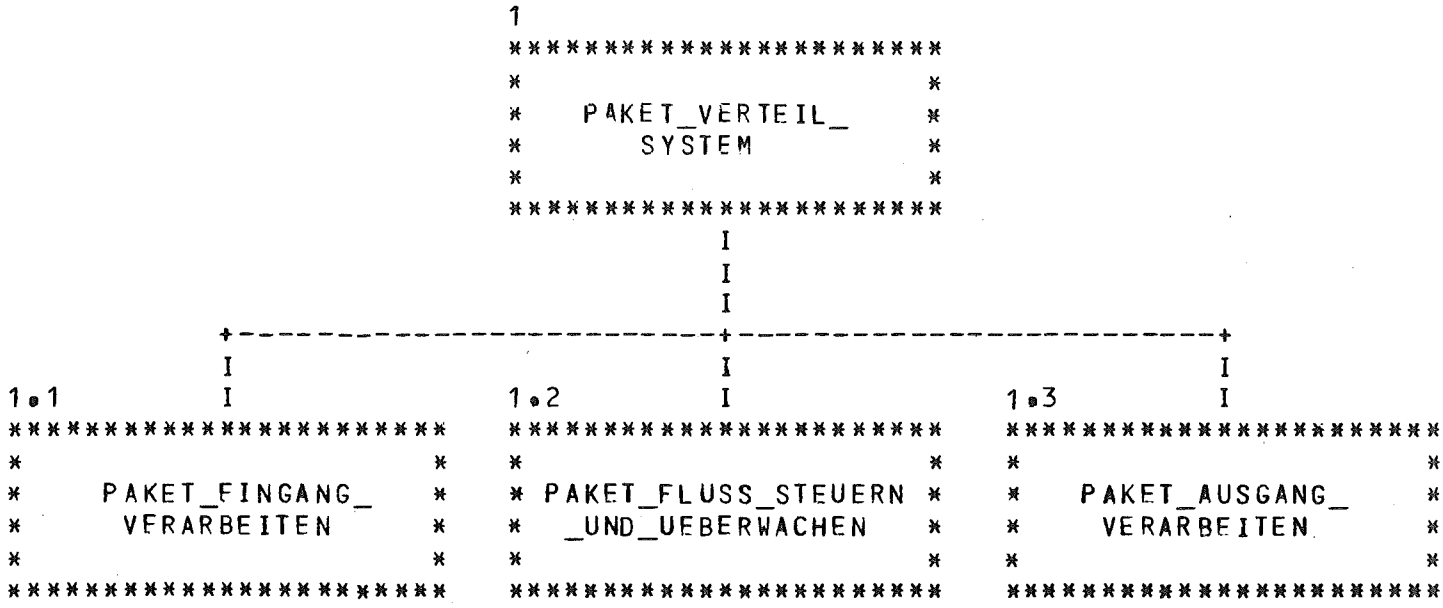
a) in: Eingabegrößen

b) out: Ausgabegrößen

c) inout: Ein- und Ausgabegrößen (transiente Parameter)

```
DATA SYSTEM       PAKET_AUSGANG_VERARBEITEN

AUTHORS           DR. BALZERT, CHRIST, HUEBNER

DATES             JUNI/JULI 1980

PURPOSE
          ES WIRD ERMITTELT, OB EIN PAKET EINE DER ZIELSTATIONEN
          PASSIERT HAT. IN DIESEM FALL WIRD DAS SYSTEM
          PAKET_FLUSS_STEUERN_UND_UEBERWACHEN DAVON IN KENNTNIS
          GESETZT.

INTERFACE

     PROCEDURE AUSGANG;

     PROCEDURE INIT_AUSGANG;
        PURPOSE
          DIE LICHTSCHRANKEN DER ZIELSTATIONEN WERDEN INITIALISIERT.

HAS ACCESS TO
      DATA SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN VIA
        PROCEDURE PAKET_AM_AUSGANG ( IN ZIEL : ZIELSTATION );
          WITH CONST ZIELMAX = 8;
               TYPE  ZIELSTATION = 1 .. ZIELMAX FROM PDV_DATAPOOL;


END PAKET_AUSGANG_VERARBEITEN


END PAKET_VERTEIL_SYSTEM
```

SYSTEMUEBERSICHT : PAKET_VERTEIL_SYSTEM 7



7 System-Übersichten :

Auf Wunsch erstellt das PLASMA-System von jeder EES eine grafische Übersicht über die funktionale Aufteilung in Untersysteme.

```
PROCEDURE PAKETVERTEILSYSTEM

   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
        CONST ZIELMAX = 8;
        TYPE  ZIELSTATION = 1 .. ZIELMAX FROM PDV_DATAPOOL;

PURPOSE
        SIEHE "AUFGABENSTELLUNG"

USES

PROCEDURE EINGANG;

   PURPOSE
        DAS SYSTEM PAKET_EINGANG_VERARBEITEN HAT ZWEI, SICH GEGEN-
        SEITIG AUSSCHLIESSENDE AUFGABEN:

        1. FESTSTELLEN, OB EIN NEUES PAKET EINGEGANGEN IST. IN
           DIESEM FALL MUSS DER ZIELORT DES PAKETES ERMITTELT
           WERDEN.
           MOEGLICHE FEHLERFAELLE SIND: FEHLENDE ODER FALSCHE ZIEL-
           ANGABE.

        2. ZURUECKHALTEN EINES AUF FREIGABE WARTENDEN PAKETES. DIE
           WARTEZEIT WIRD IN EINHEITEN AUFGETEILT; DABEI IST EINE
           EINHEIT DIE ZEIT, DIE ZWISCHEN ZWEI AUFRUFEN DES SYSTEMS
           PAKET_EINGANG_VERARBEITEN VERSTREICHT. DIE WARTEZEIT
           ENTFAELLT, WENN ZWEI AUFEINANDERFOLGENDE PAKETE DAS
           GLEICHE ZIEL HABEN. ANSONSTEN WIRD EINE KONSTANTE ANZAHL
           VON WARTE- EINHEITEN GROESSER ALS NULL GEWAEHLT.

PROCEDURE INIT_EINGANG;

   PURPOSE
        DIE EINGANGS- STATION WIRD INITIALISIERT : SPERRE ARRE-
        TIEREN, BESCHLEUNIGER SENKEN, LICHTSCHRANKE UND CODE- LESER
        INITIALISIEREN.

PROCEDURE PAKETE_STEUERN;

   PURPOSE
        DAS SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN FUEHRT UEBER
        JEDES IN DER VERTEIL- ANLAGE BEFINDLICHE PAKET BUCH UND
        SORGT FUER DIE RICHTIGE STELLUNG DER VERTEILER- WEICHEN.
        DIE PROZEDUR PAKETE_STEUERN KONTROLLIERT DER REIHE NACH
        SAEMTLICHE VERTEIL- STATIONEN UND VERBUCHT ALLE DURCHLAEUFE
        DER PAKETE.

PROCEDURE PAKET_AM_EINGANG ( IN CODE : ZIELCODE );

   PURPOSE
        IN DIE VERBUCHUNG DER PAKETE WIRD EIN NEUES PAKET AUFGE-
        NOMMEN.
```

PROCEDURE PAKET_AM_AUSGANG (IN ZIEL : ZIELSTATION);

PURPOSE
EIN PAKET WIRD AUS DER BUCHFUEHRUNG GESTRICHEN. EVENTUELL WIRD EINE FEHLERMELDUNG AUSGESCHRIEBEN, DIE DIE GEWUENSCHTE UND DIE TATSAECHLICHE ZIELANGABE ENTHAELT.

PROCEDURE INIT_PAKETE_STEUERN;

PURPOSE
DIE EINZELNEN VERTEIL- STATIONEN WERDEN INITIALISIERT, D.H. DIE WEICHEN IN EINE VORDEFINIERTE POSITION GEBRACHT UND DIE LICHTSCHRANKEN INITIALISIERT.

FUNCTION PAKET_ANZAHL : INTEGER;

PURPOSE
ES WIRD DIE ANZAHL DER MOMENTAN IN DER VERTEIL- STATION BEFINDLICHEN PAKETE ERMITTELT.

PROCEDURE AUSGANG;

PURPOSE
ES WIRD ERMITTELT, OB EIN PAKET EINE DER ZIELSTATIONEN PASSIERT HAT. IN DIESEM FALL WIRD DAS SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN DAVON IN KENNTNIS GESETZT.

PROCEDURE INIT_AUSGANG;

PURPOSE
DIE LICHTSCHRANKEN DER ZIELSTATIONEN WERDEN INITIALISIERT.

END PAKETVERTEILSYSTEM

---

8 Modul-Definition :

Die in den Ein-Ebenen-Systembeschreibungen aufgeführten Ressourcen (Interface-Teil) müssen durch Moduln realisiert werden. Von der Ein-Ebenen-Systembeschreibung erzeugt das PLASMA-System unter Beachtung der Zugriffsrechte automatisch die Moduldefinition.

Eine Moduldefinition besteht aus zwei Teilen. Der erste Teil beschreibt, welche Ressourcen (hier: procedure PAKETVERTEILSYSTEM) der Modul dem Benutzer zur Verfügung stellt. Der zweite Teil (nach uses) zeigt dem Modulimplementator, welche Ressourcen zur Implementation benutzt werden können (jeweils alle Kinder und die Ressourcen, auf die entsprechend den expliziten Zugriffsrechten zugegriffen werden darf). Kann ein Modul auf keine fremden Ressourcen zugreifen (wenn EES ein Systemblatt ist), dann fehlt der uses - Teil.

Moduldefinitionen dienen als Programmiervorgaben für die Modulimplementierer.

```
IMPLEMENTATION PAKETVERTEILSYSTEM;
   VAR AUS      : BOOLEAN;
       SCHLUSS  : BOOLEAN;

BEGIN
   INIT_EINGANG;
   INIT_PAKETE_STEUERN;
   INIT_AUSGANG;
   LESE_AUS_SCHALTER ( AUS );
   SCHLUSS := AUS;

   WHILE NOT SCHLUSS DO
     BEGIN
       IF NOT AUS THEN      (* PAKETE WERDEN NUR DANN            *)
         EINGANG;           (* IN DIE VERTEILSTATION AUFGENOM-   *)
                            (* MEN, WENN SIE EINGESCHALTET IST   *)

       PAKETE_STEUERN;      (* DIE EINZELNEN VERTEILER MUESSEN   *)
                            (* UEBERPRUEFT WERDEN, OB EIN PAKET  *)
                            (* DURCHGELAUFEN IST                 *)

       AUSGANG;             (* DIE EINZELNEN ZIELSTATIONEN       *)
                            (* WERDEN AUF DAS EINTREFFEN EINES   *)
                            (* PAKETES UEBERPRUEFT               *)

       LESE_AUS_SCHALTER ( AUS );
       IF PAKET_ANZAHL <> 0 THEN
         SCHLUSS := FALSE
       ELSE
         SCHLUSS := AUS;
     END;
END;
END PAKETVERTEILSYSTEM
```

---

9 Modul-Implementation :

Eine Moduldefinition muß durch eine Modulimplementation realisiert werden.
Moduln können prinzipiell in einer beliebigen Sprache implementiert werden. Im Beispiel wird PASCAL als Implementierungssprache verwendet.

Im Gegensatz zu anderen Entwurfssprachen ist in PLASMA / D mit jedem Knoten (außer bei VIRTUAL SYSTEM) eine Implementierung verbunden, nicht nur mit jedem Blatt.

Dadurch ist es möglich, bereits nach Beschreibung einer Ebene eine Systemsimulation vorzunehmen. Noch nicht vorhandene Untersysteme werden automatisch durch "dummies" ersetzt.

SYSTEMBESCHREIBUNG : PAKET_EINGANG_VERARBEITEN 10 SEITE 1

DATA SYSTEM PAKET_EINGANG_VERARBEITEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_EINGANG_VERARBEITEN HAT ZWEI, SICH GEGENSEITIG AUSSCHLIESSENDE AUFGABEN:

1. FESTSTELLEN, OB EIN NEUES PAKET EINGEGANGEN IST. IN DIESEM FALL MUSS DER ZIELORT DES PAKETES ERMITTELT WERDEN.
   MOEGLICHE FEHLERFAELLE SIND: FEHLENDE ODER FALSCHE ZIELANGABE.

2. ZURUECKHALTEN EINES AUF FREIGABE WARTENDEN PAKETES. DIE WARTEZEIT WIRD IN EINHEITEN AUFGETEILT; DABEI IST EINE EINHEIT DIE ZEIT, DIE ZWISCHEN ZWEI AUFRUFEN DES SYSTEMS PAKET_EINGANG_VERARBEITEN VERSTREICHT. DIE WARTEZEIT ENTFAELLT, WENN ZWEI AUFEINANDERFOLGENDE PAKETE DAS GLEICHE ZIEL HABEN. ANSONSTEN WIRD EINE KONSTANTE ANZAHL VON WARTE_ EINHEITEN GROESSER ALS NULL GEWAEHLT.

INTERFACE

```
PROCEDURE EINGANG;

PROCEDURE INIT_EINGANG;
   PURPOSE
    DIE EINGANGS- STATION WIRD INITIALISIERT : SPERRE ARRETIEREN, BESCHLEUNIGER SENKEN, LICHTSCHRANKE UND CODE- LESER INITIALISEREN.
```

HAS ACCESS TO

```
DATA SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN VIA
   PROCEDURE PAKET_AM_EINGANG ( IN CODE : ZIELCODE );
      WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
```

---

10 Aus jedem Untersystem wird auf der nächst niedrigeren Systemebene ein System, das wiederum in Untersysteme aufgeteilt werden kann. Das PLASMA-System erzeugt automatisch den ersten Teil der Ein-Ebenen-Systembeschreibung für die nächsttiefere Ebene. Dieser erste Teil kann vom Systementwerfer dieser Ebene nicht verändert werden. Aufgabe dieses Systementwerfers ist es vielmehr, die Aufteilung in Untersysteme vorzunehmen.

SUBSYSTEMS ARE

DATA SYSTEM PAKET_ANKUNFT_ERMITTELN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_ANKUNFT_ERMITTELN STELLT FEST, OB SEIT DEM LETZTEN AUFRUF EIN NEUES PAKET EINGETROFFEN IST.

INTERFACE

```
FUNCTION ANKUNFT_ERMITTELN : BOOLEAN;
```

HAS ACCESS TO

```
DATA SYSTEM EINGANGS_STATION VIA
   FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
```

END PAKET_ANKUNFT_ERMITTELN

EXECUTIVE SYSTEM CODE_ERMITTELN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM CODE_ERMITTELN LIEST MITTELS DES LESE ORGANS DEN CODE-ZETTEL, DER SICH AUF DEM EINGETROFFENEN PAKET BEFINDET. IST KEIN CODE-ZETTEL VORHANDEN ODER DER CODE NICHT ZU ENTZIFFERN, SO WIRD EIN FEHLERCODE ZURUECKGELIEFERT. ANDERNFALLS WIRD DER CODE IN EINE INTERNE DARSTELLUNG UMGEWANDELT.

INTERFACE

```
PROCEDURE CODE ( OUT CODE : ZIELORT );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
```

HAS ACCESS TO

```
DATA SYSTEM EINGANGS_STATION VIA
   PROCEDURE CODE_LESEN ( OUT CODE_WERT : CODE_STRING );
      WITH TYPE CODE_STRING = ARRAY [ 1..4 ] OF CHAR
                FROM PDV_DATAPOOL;
```

END CODE_ERMITTELN

EXECUTIVE SYSTEM PAKET_FREIGEBEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_FREIGABE FUEHRT DIE FREIGABE DES VOR DER SPERRE LIEGENDEN PAKETES DURCH.

INTERFACE

PROCEDURE FREIGEBEN;

HAS ACCESS TO
DATA SYSTEM EINGANGS_STATION VIA
PROCEDURE SPERRE_LOESEN;

END PAKET_FREIGEBEN

```
DATA SYSTEM         EINGANGS_STATION

AUTHORS             DR. BALZERT, CHRIST, HUEBNER

DATES               JUNI/JULI 1980

PURPOSE
        DIESE SYSTEM VERWALTET DIE FUER DIE EINGANGS- STATION BENOE-
        TIGTE HARDWARE.

INTERFACE

     FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
        PURPOSE
         REGISTRIERT EINEN EIN/AUS- WECHSEL AN DER LICHTSCHRANKE.

     PROCEDURE CODE_LESEN ( OUT CODE_WERT : CODE_STRING );
        WITH TYPE CODE_STRING = ARRAY [ 1..4 ] OF CHAR
                  FROM PDV_DATAPOOL;
        PURPOSE
         DER VOM ANLIEGENDEN PAKET OPTISCH ERMITTELTE WERT DES  ZIEL-
         ORTES WIRD ERKANNT UND IN EINEN INTERNEN WERT CODIERT.

     PROCEDURE SPERRE_LOESEN;
        PURPOSE
         DIE SPERRE F2 WIRD GELOEST UND DER  BESCHLEUNIGER  F1  ANGE-
         HOBEN. DANACH WIRD DIE SPERRE F2 WIEDER GESCHLOSSEN UND  DER
         BESCHLEUNIGER F1 GESENKT.
END EINGANGS_STATION


END PAKET_EINGANG_VERARBEITEN
```

---

11 Aus Platzgründen werden keine Moduldefinitionen und Modulimplementationen aufgeführt. Es wird sich auf den Systementwurf in PLASMA / D beschränkt.

SYSTEMBESCHREIBUNG : PAKET_ANKUNFT_ERMITTELN SEITE 1

DATA SYSTEM PAKET_ANKUNFT_ERMITTELN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_ANKUNFT_ERMITTELN STELLT FEST, OB SEIT DEM LETZTEN AUFRUF EIN NEUES PAKET EINGETROFFEN IST.

INTERFACE

```
FUNCTION ANKUNFT_ERMITTELN : BOOLEAN;
```

HAS ACCESS TO

```
DATA SYSTEM EINGANGS_STATION VIA
   FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
```

NO SUBSYSTEMS

END PAKET_ANKUNFT_ERMITTELN

EXECUTIVE SYSTEM CODE_ERMITTELN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM CODE_ERMITTELN LIEST MITTELS DES LESE ORGANS DEN CODE-ZETTEL, DER SICH AUF DEM EINGETROFFENEN PAKET BEFINDET. IST KEIN CODE-ZETTEL VORHANDEN ODER DER CODE NICHT ZU ENTZIFFERN, SO WIRD EIN FEHLERCODE ZURUECKGELIEFERT. ANDERNFALLS WIRD DER CODE IN EINE INTERNE DARSTELLUNG UMGEWANDELT.

INTERFACE

```
PROCEDURE CODE ( OUT CODE : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
```

HAS ACCESS TO

```
DATA SYSTEM EINGANGS_STATION VIA
   PROCEDURE CODE_LESEN ( OUT CODE_WERT : CODE_STRING );
      WITH TYPE CODE_STRING = ARRAY [ 1..4 ] OF CHAR
                FROM PDV_DATAPOOL;
```

NO SUBSYSTEMS

END CODE_ERMITTELN

SYSTEMBESCHREIBUNG : PAKET_FREIGEBEN SEITE 1

EXECUTIVE SYSTEM PAKET_FREIGEBEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_FREIGABE FUEHRT DIE FREIGABE DES VOR DER SPERRE LIEGENDEN PAKETES DURCH.

INTERFACE

PROCEDURE FREIGEBEN;

HAS ACCESS TO
DATA SYSTEM EINGANGS_STATION VIA PROCEDURE SPERRE_LOESEN;

NO SUBSYSTEMS

END PAKET_FREIGEBEN

DATA SYSTEM EINGANGS_STATION

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DIESE SYSTEM VERWALTET DIE FUER DIE EINGANGS- STATION BENOETIGTE HARDWARE.

INTERFACE

```
FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
   PURPOSE
    REGISTRIERT EINEN EIN/AUS- WECHSEL AN DER LICHTSCHRANKE.

PROCEDURE CODE_LESEN ( OUT CODE_WERT : CODE_STRING );
   WITH TYPE CODE_STRING = ARRAY [ 1..4 ] OF CHAR
             FROM PDV_DATAPOOL;
   PURPOSE
    DER VOM ANLIEGENDEN PAKET OPTISCH ERMITTELTE WERT DES ZIEL-
    ORTES WIRD ERKANNT UND IN EINEN INTERNEN WERT CODIERT.

PROCEDURE SPERRE_LOESEN;
   PURPOSE
    DIE SPERRE F2 WIRD GELOEST UND DER BESCHLEUNIGER F1 ANGE-
    HOBEN. DANACH WIRD DIE SPERRE F2 WIEDER GESCHLOSSEN UND DER
    BESCHLEUNIGER F1 GESENKT.
```

NO SUBSYSTEMS

END EINGANGS_STATION

DATA SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN FUEHRT UEBER JEDES IN DER VERTEIL- ANLAGE BEFINDLICHE PAKET BUCH UND SORGT FUER DIE RICHTIGE STELLUNG DER VERTEILER- WEICHEN.

INTERFACE

```
PROCEDURE PAKETE_STEUERN;
   PURPOSE
    DIE PROZEDUR PAKETE_STEUERN KONTROLLIERT DER REIHE NACH
    SAEMTLICHE VERTEIL- STATIONEN UND VERBUCHT ALLE DURCHLAEUFE
    DER PAKETE.

PROCEDURE PAKET_AM_EINGANG ( IN CODE : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
   PURPOSE
    IN DIE VERBUCHUNG DER PAKETE WIRD EIN NEUES PAKET AUFGE-
    NOMMEN.

PROCEDURE PAKET_AM_AUSGANG ( IN ZIEL : ZIELSTATION );
   WITH CONST ZIELMAX = 8;
        TYPE  ZIELSTATION = 1 .. ZIELMAX FROM PDV_DATAPOOL;
   PURPOSE
    EIN PAKET WIRD AUS DER BUCHFUEHRUNG GESTRICHEN. EVENTUELL
    WIRD EINE FEHLERMELDUNG AUSGESCHRIEBEN, DIE DIE GEWUENSCHTE
    UND DIE TATSAECHLICHE ZIELANGABE ENTHAELT.

PROCEDURE INIT_PAKETE_STEUERN;
   PURPOSE
    DIE EINZELNEN VERTEIL- STATIONEN WERDEN INITIALISIERT, D.H.
    DIE WEICHEN IN EINE VORDEFINIERTE POSITION GEBRACHT UND DIE
    LICHTSCHRANKEN INITIALISIERT.

FUNCTION PAKET_ANZAHL : INTEGER;
   PURPOSE
    ES WIRD DIE ANZAHL DER MOMENTAN IN DER VERTEIL- STATION
    BEFINDLICHEN PAKETE ERMITTELT.
```

SUBSYSTEMS ARE

DATA SYSTEM PAKETE_VERWALTEN_UND_VERBUCHEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DIESES SYSTEM FUEHRT UEBER DEN GESAMTEN DURCHLAUF DER PAKETE BUCH. DABEI WERDEN 3 VERSCHIEDENE BEWEGUNGEN REGISTRIERT:

1. DIE EINGANGS- STATION MELDET DIE FREIGABE EINES PAKETES. DAS PAKET WIRD IN DIE WARTESCHLANGEN ALLER BETROFFENEN VERTEIL- STATIONEN EINGETRAGEN.

2. PAKETE, DIE EINE VERTEIL- STATION PASSIERT HABEN WERDEN DORT AUSGETRAGEN. EVENTUELLE FEHLLAEUFER KOENNEN UMGEBUCHT WERDEN.

3. EINE ZIEL- STATION MELDET DIE ANKUNFT EINES PAKETES. ES WIRD AUS DER BUCHHALTUNG GESTRICHEN.

INTERFACE

```
PROCEDURE PAKET_EINTRAGEN ( IN CODE : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;

PROCEDURE PAKET_AM_VERTEILER_EINGANG ( IN VNUMMER : VERTEILER );
   WITH CONST VERTEILMAX = 7;
        TYPE  VERTEILER = 1 .. VERTEILMAX FROM PDV_DATAPOOL;

PROCEDURE PAKET_AM_VERTEILER_AUSGANG ( IN VNUMMER : VERTEILER );
   WITH CONST VERTEILMAX = 7;
        TYPE  VERTEILER = 1 .. VERTEILMAX FROM PDV_DATAPOOL;

PROCEDURE PAKET_AUSTRAGEN ( IN  IST  : ZIELCODE ;
                            OUT SOLL : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
```

END PAKETE_VERWALTEN_UND_VERBUCHEN

ABSTRACT SYSTEM VERTEILER_LESEN_UND_STEUERN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM VERTEILER_LESEN_UND_STEUERN VERWALTET DIE FUER EINE VERTEILER- STATION BENOETIGTE HARDWARE.
DABEI MUESSEN ALS HARDWARE- OPERATIONEN DAS LESEN DER EINGANGS- UND AUSGANGS- LICHTSCHRANKEN UND DAS STELLEN DER WEICHE REALISIERT WERDEN.

INTERFACE

```
PROCEDURE WEICHE_STELLEN ( IN WEG : LINKS_RECHTS );
   WITH TYPE LINKS_RECHTS = ( LINKS , RECHTS )
             FROM PDV_DATAPOOL;

FUNCTION EINGANG_LESEN : BOOLEAN;
FUNCTION AUSGANG_LESEN : BOOLEAN;
```

END VERTEILER_LESEN_UND_STEUERN

END PAKET_FLUSS_STEUERN_UND_UEBERWACHEN

DATA SYSTEM PAKETE_VERWALTEN_UND_VERBUCHEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DIESES SYSTEM FUEHRT UEBER DEN GESAMTEN DURCHLAUF DER PAKETE BUCH. DABEI WERDEN 3 VERSCHIEDENE BEWEGUNGEN REGISTRIERT:

1. DIE EINGANGS- STATION MELDET DIE FREIGABE EINES PAKETES. DAS PAKET WIRD IN DIE WARTESCHLANGEN ALLER BETROFFENEN VERTEIL- STATIONEN EINGETRAGEN.

2. PAKETE, DIE EINE VERTEIL- STATION PASSIERT HABEN WERDEN DORT AUSGETRAGEN. EVENTUELLE FEHLLAEUFER KOENNEN UMGEBUCHT WERDEN.

3. EINE ZIEL- STATION MELDET DIE ANKUNFT EINES PAKETES. ES WIRD AUS DER BUCHHALTUNG GESTRICHEN.

INTERFACE

```
PROCEDURE PAKET_EINTRAGEN ( IN CODE : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;

PROCEDURE PAKET_AM_VERTEILER_EINGANG ( IN VNUMMER : VERTEILER );
   WITH CONST VERTEILMAX = 7;
        TYPE  VERTEILER = 1 .. VERTEILMAX FROM PDV_DATAPOOL;

PROCEDURE PAKET_AM_VERTEILER_AUSGANG ( IN VNUMMER : VERTEILER );
   WITH CONST VERTEILMAX = 7;
        TYPE  VERTEILER = 1 .. VERTEILMAX FROM PDV_DATAPOOL;

PROCEDURE PAKET_AUSTRAGEN ( IN  IST  : ZIELCODE ;
                            OUT SOLL : ZIELCODE );
   WITH TYPE ZIELCODE = 0 .. 9999 FROM PDV_DATAPOOL;
```

```
SUBSYSTEMS ARE

ABSTRACT SYSTEM  SCHLANGEN_OPERATIONEN

AUTHORS          DP. BALZERT, CHRIST, HUEBNER

DATES            JUNI/JULI 1980

PURPOSE
        REALISIERT DIE OPERATIONEN AUF EINER SCHLANGE.

INTERFACE

    ABSTRACT MODULE SCHLANGE ( TYPE  ITEM;
                               CONST ANZAHL : INTEGER );

        CONSISTS OF

        PROCEDURE ENQUEUE ( INOUT SCHL : SCHLANGE;
                            IN    DESC : ITEM     );

        PROCEDURE DEQUEUE ( INOUT SCHL : SCHLANGE;
                            OUT   DESC : ITEM     );

        PROCEDURE INIT    (   OUT SCHL : SCHLANGE );

END SCHLANGEN_OPERATIONEN

END PAKETE_VERWALTEN_UND_VERBUCHEN
```

SYSTEMBESCHREIBUNG : SCHLANGEN_OPERATIONEN SEITE 1

```
ABSTRACT SYSTEM  SCHLANGEN_OPERATIONEN

AUTHORS          DR. BALZERT, CHRIST, HUEBNER

DATES            JUNI/JULI 1980

PURPOSE
        REALISIERT DIE OPERATIONEN AUF EINER SCHLANGE.

INTERFACE

    ABSTRACT MODULE SCHLANGE ( TYPE  ITEM;
                               CONST ANZAHL : INTEGER );

        CONSISTS OF

        PROCEDURE ENQUEUE ( INOUT SCHL : SCHLANGE;
                            IN    DESC : ITEM     );

        PROCEDURE DEQUEUE ( INOUT SCHL : SCHLANGE;
                            OUT   DESC : ITEM     );

        PROCEDURE INIT    (   OUT SCHL : SCHLANGE );

NO SUBSYSTEMS


END SCHLANGEN_OPERATIONEN
```

ABSTRACT SYSTEM VERTEILER_LESEN_UND_STEUERN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM VERTEILER_LESEN_UND_STEUERN VERWALTET DIE FUER EINE VERTEILER- STATION BENOETIGTE HARDWARE.
DABEI MUESSEN ALS HARDWARE- OPERATIONEN DAS LESEN DER EINGANGS- UND AUSGANGS- LICHTSCHRANKEN UND DAS STELLEN DER WEICHE REALISIERT WERDEN.

INTERFACE

```
PROCEDURE WEICHE_STELLEN ( IN WEG : LINKS_RECHTS );
   WITH TYPE LINKS_RECHTS = (LINKS,RECHTS)
             FROM PDV_DATAPOOL;

FUNCTION EINGANG_LESEN : BOOLEAN;
FUNCTION AUSGANG_LESEN : BOOLEAN;
```

SUBSYSTEMS ARE

ABSTRACT SYSTEM LICHT_SCHRANKE

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM LICHT_SCHRANKE MERKT SICH, OB SEIT DEM LETZTEN AUFRUF DER LICHTSTRAHL UNTERBROCHEN WURDE.

INTERFACE

```
FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
```

END LICHT_SCHRANKE

DATA SYSTEM WEICHE

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS HARDWARE- TEIL "WEICHE" WIRD GESTELLT.

INTERFACE

```
PROCEDURE STELLE_WEICHE_LINKS;
PROCEDURE STELLE_WEICHE_RECHTS;
```

END WEICHE

END VERTEILER_LESEN_UND_STEUERN

SYSTEMBESCHREIBUNG : LICHT_SCHRANKE SEITE 1

ABSTRACT SYSTEM LICHT_SCHRANKE

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS SYSTEM LICHT_SCHRANKE MERKT SICH, OB SEIT DEM LETZTEN AUFRUF DER LICHTSTRAHL UNTERBROCHEN WURDE.

INTERFACE

```
FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
```

NO SUBSYSTEMS

END LICHT_SCHRANKE

DATA SYSTEM WEICHE

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DAS HARDWARE- TEIL "WEICHE" WIRD GESTELLT.

INTERFACE

```
PROCEDURE STELLE_WEICHE_LINKS;
PROCEDURE STELLE_WEICHE_RECHTS;
```

NO SUBSYSTEMS

END WEICHE

SYSTEMBESCHREIBUNG : PAKET_AUSGANG_VERARBEITEN SEITE 1

DATA SYSTEM PAKET_AUSGANG_VERARBEITEN

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

ES WIRD ERMITTELT, OB EIN PAKET EINE DER ZIELSTATIONEN PASSIERT HAT. IN DIESEM FALL WIRD DAS SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN DAVON IN KENNTNIS GESETZT.

INTERFACE

```
PROCEDURE AUSGANG;

PROCEDURE INIT_AUSGANG;
   PURPOSE
    DIE LICHTSCHRANKEN DER ZIELSTATIONEN WERDEN INITIALISIERT.
```

HAS ACCESS TO

```
DATA SYSTEM PAKET_FLUSS_STEUERN_UND_UEBERWACHEN VIA
   PROCEDURE PAKET_AM_AUSGANG ( IN ZIEL : ZIELSTATION );
      WITH CONST ZIELMAX = 8;
           TYPE  ZIELSTATION = 1 .. ZIELMAX FROM PDV_DATAPOOL;
```

SUBSYSTEMS ARE

DATA SYSTEM ZIEL_STATION

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DIESES SYSTEM VERWALTET DIE FUER DIE ZIEL- STATIONEN BENOETIGTE HARDWARE.
DIE HARDWARE BESTEHT IN DIESEM FALL NUR AUS EINER LICHTSCHRANKE.

INTERFACE

FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
PURPOSE
REGISTRIERT EINEN EIN/AUS- WECHSEL AN DER LICHTSCHRANKE.

END ZIEL_STATION

END PAKET_AUSGANG_VERARBEITEN

DATA SYSTEM ZIEL_STATION

AUTHORS DR. BALZERT, CHRIST, HUEBNER

DATES JUNI/JULI 1980

PURPOSE

DIESES SYSTEM VERWALTET DIE FUER DIE ZIEL- STATIONEN BENOETIGTE HARDWARE.
DIE HARDWARE BESTEHT IN DIESEM FALL NUR AUS EINER LICHTSCHRANKE.

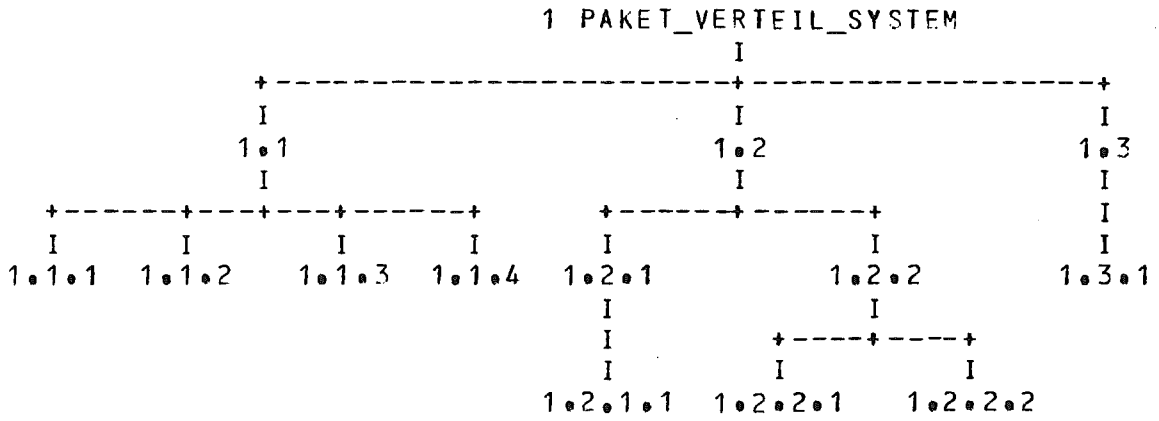
INTERFACE

FUNCTION LICHTSCHRANKE_LESEN : BOOLEAN;
PURPOSE
REGISTRIERT EINEN EIN/AUS- WECHSEL AN DER LICHTSCHRANKE.

NO SUBSYSTEMS

END ZIEL_STATION

SYSTEMBAUM : PAKET_VERTEIL_SYSTEM 12




```
1       PAKET_VERTEIL_SYSTEM
1.1         PAKET_EINGANG_VERARBEITEN
1.1.1           PAKET_ANKUNFT_ERMITTELN
1.1.2           CODE_ERMITTELN
1.1.3           PAKET_FREIGEBEN
1.1.4           EINGANGS_STATION
1.2         PAKET_FLUSS_STEUERN_UND_UEBERWACHEN
1.2.1           PAKETE_VERWALTEN_UND_VERBUCHEN
1.2.1.1             SCHLANGEN_OPERATIONEN
1.2.2           VERTEILER_LESEN_UND_STEUERN
1.2.2.1             LICHT_SCHRANKE
1.2.2.2             WEICHE
1.3         PAKET_AUSGANG_VERARBEITEN
1.3.1           ZIEL_STATION
```

---

12 Es können als Systemübersichten vertikale, horizontale, punktuelle und schlüsselwortgesteuerte Systemdokumentationen unterschiedlicher Detaillierung ausgegeben werden.

Zusammenfassung

In PLASMA / D werden bereits vorhandene Konzepte der Systementwicklung, Hierarchisierung, Modularisierung und Datenabstraktion in modifizierter oder präzisierter Form zu einem neuen, integrierten und praktikablen Konzept einer Systementwurfssprache zusammengefügt. PLASMA / D stellt Konzepte für das "Programmieren im Großen" zur Verfügung und integriert Sprachen für das "Programmieren im Kleinen", um Moduln zu implementieren.

PLASMA / D berücksichtigt Belange der Projektorganisation und der adressatengerechten Dokumentation. Da ein PLASMA / D-Programm nicht als fertiges, komplettes Programm eingegeben und übersetzt, sondern schrittweise während der Systementwicklung von i. a. mehreren Programmierern erstellt wird, gehört zu PLASMA / D ein PLASMA-System, das die dafür notwendigen Dienstleistungen zur Verfügung stellt.

Ein Systementwurf erfordert eine Reihe von Entwurfsentscheidungen. In PLASMA / D können alle frühzeitig möglichen Entscheidungen wie Aufteilung in Untersysteme, Festlegung der Zugriffsrechte und Ressourcen in den Ein-Ebenen-Systembeschreibungen beschrieben werden.

Die in den Sprachen der ALGOL-Familie bewährte Blockstruktur wird als Hierarchieart in PLASMA / D übernommen. Durch die Ein-Ebenen-Systembeschreibungen wird in PLASMA / D jedoch die Verwendung von vielen Systemebenen und der vererbte Zugriff zu höherliegenden Systemebenen erleichtert. Die in ALGOL-Sprachen weitgehenden impliziten Zugriffsregeln werden aus Gründen der Zuverlässigkeit, Verständlichkeit und Wartbarkeit drastisch eingeschränkt.

In PLASMA / D wird ein Datenabstraktionskonzept verwirklicht.

Wesentlich am PLASMA / D-Konzept ist, daß es der Hierarchisierung die gleiche Bedeutung wie der Modularisierung zumißt, daß es an der Blockstruktur als dem Träger der Systemstruktur festhält, dessen Nachteile jedoch vermeidet.

Es wird zwischen der Moduldefinition und der Modulimplementation unterschieden.

In PLASMA / D wird das Entwurfsproblem, eine geeignete Symbiose zwischen Hierarchie- und Modulkonzept zu finden, auf neuartige Weise gelöst.

In PLASMA / D entsteht durch die Ein-Ebenen-Systembeschreibung ein Systembaum, der alle funktionalen Aspekte des Systems sowie die Zugriffshierarchie wiedergibt. Durch die Ein-Ebenen-Systembeschreibungen wird also die Hierarchie eines Systems festgelegt.

Die im Systembaum vorhandenen Systemkomponenten müssen Ressourcen für die jeweils nächsthöhere Systemebene bereitstellen. Diese Ressourcen werden in PLASMA / D durch Moduln realisiert. Die Moduln werden sozusagen in den Systembaum "eingehängt", um die von den Systemkomponenten geforderten Leistungen zur Verfügung zu stellen. Die Moduln selbst sind jedoch von der durch den Systembaum repräsentierten Systemhierarchie unabhängig; Modul- und Hierarchiekonzept sind entkoppelt. Die vom PLASMA-System anhand des Systembaums für jede Systemkomponente erstellte Moduldefinition enthält linear aufgelistet, welche Ressourcen der Modulimplementator zur Implementation des Moduls verwenden kann; die Hierachie spiegelt sich in der Moduldefinition nicht wieder. Durch dieses Konzept entfällt auch die Notwendigkeit, geschachtelte Moduln zu verwenden.

Die in PLASMA / D gefundene Symbiose zwischen Hierarchie- und Modulkonzept ist neu, ein System kann geeignet hierarchisiert werden (unter Beibehaltung der Lokalität durch die Beschreibungsform der Ein-Ebenen-Systembeschreibung); Moduln realisieren die Systemleistungen, sind aber kontextunabhängig, insbesondere auch hierarchieunabhängig.

In PLASMA / D wird zum erstenmal ein Sprachkonzept realisiert, das eine klare Unterscheidung zwischen Konzepten zum "Programmieren im Großen" (Hierarchisierung, Ressourcenfestlegung) und Konzepten zum "Programmieren im Kleinen" (Modulimplementierung) trifft. Das in PLASMA / D gewählte Konzept erlaubt auch eine Erhöhung der Wirtschaftlichkeit, da sowohl Moduln als auch Teilbäume jeweils in anderen Kontexten eingesetzt werden können. Der Entwerfer eines neuen Systems kann also sowohl Teilbäume von vorhandenen Systemen übernehmen als auch Einzelleistungen in Form von Moduln übernehmen.

Literatur

/ Balzert 79 / Balzert H. , Die Programmiersprache PLASMA 78, Dissertation, FB Informatik, Universität Kaiserslautern, 1979

/ Balzert 79 a / Balzert H. , Die Programmiersprache PLASMA 78, Eine kurze Einführung, Interner Bericht, FB Informatik, Universität Kaiserslautern, Februar 1979

/ Balzert 79 b / Balzert H. , PLASMA , A short introduction, Report, FB Informatik, Universität Kaiserslautern, March 1979

/ Balzert 79 c / Balzert H. , Die Programmiersprache PLASMA 78, in : PASCAL, Berichte German Chapter of the ACM, S. 47 - 63, Teubner Verlag, Stuttgart 1979

Die Erweiterungen und Modifikationen an PLASMA / D wurden in enger Zusammenarbeit mit den Herren Dr. D. Weber, W. Fuchs und E. Kurz vorgenommen.
Ihnen gilt unser besonderer Dank.

Lehrstuhl für Programmiersprachen | Friedrich-Alexander-Universität Erlangen-Nürnberg | FAU

Title: Axiomatische Spezifikation der Software zur Steuerung einer Paketverteilungsanlage

Author(s): H. J. Schneider

CR Categories: 4.6; 5.24; 4.34; 3.82

Bibliographical Note:

Separate print from / To appear in / Submitted to:

Ausgearbeitet für den PDV-Arbeitskreis "Systematische Entwicklung von PDV-Systemen"

Address of the Author(s): Lehrstuhl für Programmiersprachen
Martens-Str. 3, D-8520 Erlangen, Germany

Abstract:

Die axiomatische Methode der Spezifikation von Software, die sich bei der Anwendung auf nahe an der Mathematik liegende Probleme bewährt hat, wird dargestellt und anschließend auf die Software zur Steuerung einer Paketverteilungsanlage angewandt. Es wird eine Möglichkeit aufgezeigt, wie die bei dieser Aufgabenstellung auftretenden Synchronisationsprobleme mit Hilfe von Axiomen gehört werden können.

Technical Report

1. VORBEMERKUNG

Dijkstra hat, wie auch andere, darauf hingewiesen, daß der Grad an Komplexität, den der menschliche Verstand zu überblicken vermag, wesentlich kleiner ist als der Komplexitätsgrad moderner Softwaresysteme [Structured Programming, 1972]. Aus diesem Grund muß bei Entwurf und Implementierung derartiger Systeme der Komplexitätsgrad durch Zerlegung des Problems in mehrere kleinere soweit reduziert werden, daß die Teilaufgabe zu jedem Zeitpunkt für den Bearbeiter überschaubar bleibt. Gegebenenfalls muß diese Zerlegung über mehrere Ebenen fortgesetzt werden, bis handhabbare Einheiten entstanden sind.

Oft kann dies durch Abstraktion erreicht werden: Man beschränkt sich auf jeder Entwicklungsstufe ausschließlich auf diejenigen Eigenschaften der betrachteten Objekte und Operationen, die auf dieser Stufe relevant sind, und vernachlässigt alle anderen, insbesondere die sich aus einer konkreten Implementierung ergebenden. Dies widerspricht der klassischen Vorgehensweise, die Repräsentation der einzelnen Objekte zuerst festzulegen, und zwingt dazu, jeden Zugriff auf ein Objekt ausschließlich über die dieser Objektart eigenen Operationen abzuwickeln. Eine solche Denkweise lag bereits dem Klassenkonzept von SIMULA zugrunde und wurde später von *Liskov* und *Zilles* ausführlich untersucht [SIGPLAN April 1974].

Die Lösung eines Problems läuft dann folgendermaßen ab: Wir entwerfen Daten und Operationen, also eine abstrakte Maschine im Sinne von *Parnas* [CACM Mai 1972], die in einfacher Weise das Problem zu lösen gestatten. Ausschließlich unter Verwendung der spezifizierten Eigenschaften dieser Operationen muß die Korrektheit des Problems formal beweisbar oder wenigstens einsichtig sein. Dann wird für jede Operation ein abstraktes Programm geschrieben, das aus Operationen einer tieferen Stufe zusammengesetzt ist. Die Korrektheit jedes dieser Programme wird wiederum gezeigt unter ausschließlicher Verwendung der Spezifikationen für die Operationen der niedrigeren Stufe. So wird fortgefahren, bis die Ebene der gewünschten Programmiersprache erreicht ist.

2. SPEZIFIKATION VON SYNTAX UND SEMANTIK

2.1 SYNTAX

Die Spezifikation eines Datentyps muß Syntax und Semantik der zur Verfügung gestellten Operationen umfassen. Die syntaktische Spezifikation gibt die Bezeichnungen, die Definitions- und die Wertebereiche der Operationen an. Diese Angabe kann sowohl in der bekannten mathematischen Notation als auch in einer programmiersprachlichen Notation erfolgen. Wir betrachten das in der Literatur weit verbreitete Beispiel der Schlange und die mathematische Notation:

NEUE_SCHLANGE: → schlange

ANFUEGE: schlange x element → schlange

ERSTES: schlange → element

ENTFERNE: schlange → schlange

LEER: schlange → boolean

Dieser mathematischen Notation entspricht unmittelbar eine programmiersprachliche, wobei wir hier die Sprache ADA [SIGPLAN Juni 1979] zugrunde legen:

```
function NEUE_SCHLANGE return SCHLANGE;
function ANFUEGE(S : SCHLANGE, E : ELEMENT) return SCHLANGE;
function ERSTES(S : SCHLANGE) return ELEMENT;
function ENTFERNE(S : SCHLANGE) return SCHLANGE;
function LEER(S : SCHLANGE) return BOOLEAN;
```

2.2 VERWENDUNG VON PROZEDUREN

Die ausschließliche Verwendung von Funktionen, deren Parameter unveränderliche Eingabeparameter sind, erleichtert die mathematische Formulierung wesentlich, führt aber an einigen Stellen zu einer unbequemen Programmierweise. *Guttag, Horowitz* und *Musser* haben daher in einer Weiterentwicklung [1977] gezeigt, wie in der syntaktischen Schnittstellenbeschreibung Prozeduren zugelassen werden können, die einen oder mehrere ihrer Parameter verändern, ohne daß auf die bequemere Handhabung reiner Funktionen verzichtet werden muß*).

Der Trick liegt darin, die programmtechnisch benötigten Operationen um "verborgene" Operationen zu erweitern, die außerhalb des Moduls nicht verfügbar sind, modulintern zur Definition der beiden zuletzt genannten Operationen dienen und die ursprünglichen Eigenschaften erfüllen. In unserer Syntaxschnittstelle sind diese verborgenen Operationen durch einen Stern gekennzeichnet:

NEUE_SCHLANGE	→ schlange
LEER (schlange)	→ boolean
*ANFUEGE (schlange, element)	→ schlange
*ERSTES (schlange)	→ element
*ENTFERNE (schlange)	→ schlange
ANFUEGE_AN (var schlange, element)	→
ENTNIMM_ERSTES (var schlange)	→ element

ANFUEGE_AN ist nun eine "echte" Prozedur, die keinen Funktionswert liefert, aber einen ihrer Parameter verändert, und ENTNIMM_ERSTES eine Funktion mit Seiteneffekt, was üblicherweise als schlechter Programmierstil gilt.

---

*) Ferner wird dort die Fehlerbehandlung untersucht.

Die Formulierung in ADA macht den Unterschied zwischen außerhalb verfügbaren und verborgenen Operationen noch deutlicher:*)

```
package SCHLANGENVERWALTUNG is
   type SCHLANGE is ...;
   function NEUE_SCHLANGE return SCHLANGE;
   procedure ANFUEGE_AN(S: in out SCHLANGE, E: in ELEMENT);
   procedure ENTNIMM_ERSTES(S: in out SCHLANGE, E: out ELEMENT);**)
   function LEER(S: in SCHLANGE) return BOOLEAN;
end;
```

ist die Spezifikation der Schnittstelle, während die verborgenen Operationen nur in der Spezifikation des Modulrumpfes auftreten:

```
package body SCHLANGENVERWALTUNG is
   function ERSTES(S: in SCHLANGE) return ELEMENT  is
   begin ... end;
   function ENTFERNE(S: in SCHLANGE) return SCHLANGE is
   begin ... end;
   function NEUE_SCHLANGE return SCHLANGE is
   begin ... end;
   function LEER(S: in SCHLANGE) return BOOLEAN is
   begin ... end;
   function ANFUEGE(S: in SCHLANGE, in ELEMENT) return SCHLANGE;
   procedure ENTNIMM_ERSTES(S: in out SCHLANGE, E: out ELEMENT);
   begin E := ERSTES(S);
         S := ENTFERNE(S);
   end;
   procedure ANFUEGE_AN(S: in out SCHLANGE, E: in ELEMENT) is
   begin S := ANFUEGE(S,E); end;
end SCHLANGENVERWALTUNG;
```

Es soll uns hier nicht stören, daß wir die Schlangenverwaltung als Funktionsmodul beschrieben haben, bei dem Daten von einem Aufruf zum anderen nicht aufbewahrt werden.

---

*) Es wird vorausgesetzt, daß die Schlangenverwaltung im Gültigkeitsbereich der Artdeklaration ELEMENT liegt.

**) Funktionen mit Seiteneffekt sind in ADA nicht zulässig.

2.3 SEMANTIK

Bisher haben wir nur die syntaktischen Regeln für die Verwendung dieser Operationen festgelegt, nicht jedoch deren Bedeutung. Auch wenn dieses einfache Beispiel den Eindruck erweckt, es genüge, hinreichend suggestive Identifikatoren zu wählen, so kann dies selbst bei vertrauten Datentypen zu Mißverständnissen führen und bei ungebräuchlichen Datentypen reicht das Verfahren überhaupt nicht aus.

In der Literatur finden sich verschiedene Methoden zur Spezifikation der Semantik. Die in unserem Zusammenhang interessierenden lassen sich in zwei Gruppen einordnen, die wir als die operationellen und die axiomatischen Methoden bezeichnen wollen. Bei *operationellen* Methoden wird eine Menge von Maschinenzuständen einer abstrakten Maschine vorgegeben, und es wird beschrieben, welche Zustandsänderungen durch die Operationen bewirkt werden. Da es sich in gewisser Weise hierbei bereits um ein Programmieren handelt, wird z.B. von *Guttag* [CACM Jun. 1977] als wichtigster Vorteil angesehen, daß ausgebildete Programmierer damit leichter zurecht kommen. Andererseits verweist aber *Guttag* zu Recht darauf, daß diese Methode fast immer zur Berücksichtigung unnötiger Details verführt. Diese präjudizieren einerseits die Implementierung, indem sie den Freiheitsgrad einschränken, andererseits erschweren sie den Korrektheitsbeweis, weil auch die unnötigen Eigenschaften verifiziert werden müssen.

Diese Schwierigkeiten vermeidet die *axiomatische* Methode. Wie in der Algebra üblich, wird hierbei eine Menge von Relationen angegeben, die zwischen den Operationen bestehen. Diese Menge von Axiomen muß vollständig sein im Sinne der Problemspezifikation, nicht aber im Sinne der mathematischen Logik.*) Sie soll also in dem Sinn vollständig sein, daß alle Eigenschaften der Operationen wiedergegeben werden, die auf der gerade betrachteten Programmierebene benutzt werden. In dem von uns gewählten Beispiel der Schlange kommt es also ausschließlich darauf an, daß die "first-in-first-out"-Eigenschaft gilt. Hierfür gibt *Guttag* die folgenden Axiome an [CACM Jun. 1977]:

*) Die Vollständigkeit im mathematischen Sinne kann insofern hilfreich sein, weil dann die Möglichkeit besteht, nichtinteressierende Eigenschaften ganz bewußt wegzulassen.

```
LEER(NEUE_SCHLANGE)      =  true
LEER(ANFUEGE(s,e))       =  false
ERSTES(ANFUEGE(s,e))     =  if LEER(s) then e
                            else ERSTES(s) end if
ENTFERNE(ANFUEGE(s,e)) =  if LEER(s) then NEUE_SCHLANGE
                            else ANFUEGE(ENTFERNE(s),e) end if
```

Ferner muß festgehalten werden, daß

ERSTES(NEU) und ENTFERNE(NEU)

zu Fehlern führen. (Dies entspricht etwa der Division durch 0 beim Datentyp integer.).

Wenn man sich das Beispiel genauer anschaut, stellt man fest, daß hier nicht ein einzelner Datentyp definiert wurde, sondern eine Schar von gleichstrukturierten Datentypen. Der Scharparameter ist die Art der Elemente der Schlange. Dieser Gesichtspunkt spielt eine große Rolle bei der Erstellung wiederverwendbarer Software.

Kompliziertere Beispiele sind in der Literatur angegeben, so von *Guttag* die Behandlung einer blockstrukturierten Symbolliste in einem Kompilierer [CACM Jun. 1977] oder Zeichenketten, Bäume, allgemeine Graphen, Dateien und Polynomalgebra von *Guttag, Horowitz* und *Musser* [2nd ICSE 1976].

Von *Guttag* [CACM Jun. 1977] werden zwei wichtige Argumente für die axiomatische Spezifikation angegeben. Einmal ist es - wie bereits erläutert - auf diese Weise möglich, die Kommunikation zwischen verschiedenen Komponenten eines Softwaresystems zu einem sehr frühen Zeitpunkt festzulegen, ohne die Details der Speicherung vorwegzunehmen. Diese können vielmehr sehr viel später auf Grund der Kenntnis von Häufigkeit und Art des Zugriffs zu den einzelnen Daten optimal geplant werden. Das zweite Argument ist der Einsatz bei der formalen Programmverifikation. Die *Implementierung* der Operationen stimmt dann mit ihrer Spezifikation überein, wenn für die Implementierung die als Axiome angegebenen Relationen gelten, die Implementierung also - algebraisch formuliert - ein homomorphes Bild der durch die Axiome gegebenen heterogenen Algebra

ist. Durch die gegebene Spezifikation wird die Verifikation des Gesamtsystems nicht nur vertikal, sondern durch die einzelnen Relationen auch horizontal gegliedert. Für den Fall einer blockstrukturierten Symbolliste ist dies von *Guttag, Horowitz* und *Musser* vollständig durchgeführt worden [CACM Dez. 1978]. Eine Untersuchung der formalen Eigenschaften dieser Technik findet sich bei *Guttag* und *Horning* [Acta Informatica 78].

Spitzen und *Wegbreit* geben drei Gründe für die axiomatische Spezifikation von Datentypen an [Acta Informatica 75]:

> "Zunächst sind sie deklarativ und vermeiden so Programmiereinzelheiten und Sprachabhängigkeiten. Zum zweiten handelt es sich um intuitiv einsichtige Beschreibungen des Verhaltens verschiedener Strukturen. Schließlich sind sie hinreichend streng, um einen Beweis dafür zu ermöglichen, daß eine spezielle Realisierung der Datenstruktur den Spezifikationen entspricht."

Diesen Argumenten fügen *Guttag* und *Horning* ein viertes hinzu [Acta Informatica 78]:

> "Sie sind leicht zu lesen und zu verstehen und erleichtern so den informellen Nachweis der Tatsache, daß sie mit der Intention ihres Autors übereinstimmen."

2.4 AUFFINDEN DER AXIOME

Um herauszufinden, welche Beziehungen zwischen den einzelnen Operationen überhaupt als Axiome in Frage kommen, muß man sich noch einmal vergegenwärtigen, daß wir es mit zwei verschiedenen Klassen von Funktionen zu tun haben. Nach *Parnas* (1972) könne wir sie als V-Funktionen und O-Funktionen bezeichnen:

V-Funktionen (value returning) liefern Informationen über die Objekte, verändern sie aber nicht.

O-Funktionen (operate) verändern die Objekte.

Wie wir gesehen haben, können OV-Funktionen zur bequemeren Handhabung eingeführt, in der Regel aber durch eine Kombination von O- und V-Funktionen definiert werden, so daß sie hier nicht untersucht werden müssen.

Da die V-Funktion die einzige Möglichkeit ist, etwas über die Objekte zu erfahren, ist die Semantik dann vollständig beschrieben, wenn man zu jeder O-Funktion angibt, welche Auswirkungen sie auf den nachfolgenden Aufruf einer V-Funktion hat. Die Methode ist u.a. von *Keramidis* und *Mackert* [1979] auf Probleme aus dem Bereich der Systemsoftware angewandt worden. Im Beispiel der Schlangenverwaltung sind ERSTES und LEER die V-Funktionen, NEUE_SCHLANGE, ANFUEGE und ENTFERNE die O-Funktionen. Wir müssen also der Reihe nach die drei O-Funktionen betrachten und festlegen, welchen Wert anschließend die beiden V-Funktionen liefern sollen.

Am einfachsten ist dies für die O-Funktion NEUE_SCHLANGE:

LEER(NEUE_SCHLANGE) = true
ERSTES(NEUE_SCHLANGE) = undefiniert

Wird an eine bestehende Schlange ein Element angefügt, so ist es das erste Element, wenn die Schlange zuvor leer war, andernfalls ändert sich das erste Element nicht:

$$\text{ERSTES(ANFUEGE(s,e))} = \begin{cases} e & \text{falls LEER(s)} \\ \text{ERSTES(s)} & \text{sonst} \end{cases}$$

LEER(ANFUEGE(s,e)) = false

Komplizierter wird die Auswirkung von ENTFERNE, weil sich hier ein früheres ANFUEGE bemerkbar macht. Man kann seine Auswirkung auf die V-Funktionen auf die schon behandelten Fälle zurückführen, indem man feststellt, daß ENTFERNE und ANFUEGE in gewisser Weise vertauscht werden können:

$$\text{ENTFERNE(ANFUEGE(s,e))} = \begin{cases} \text{NEUE\_SCHLANGE} & \text{falls LEER(s)} \\ \text{ANFUEGE(ENTFERENE(s),e)} & \text{sonst} \end{cases}$$

Gilt LEER(s) nicht, so bedeutet dieses Axiom für die V-Funktion ERSTES

ERSTES(ENTFERNE(ANFUEGE(s,e)))

= ERSTES(ANFUEGE(ENTFERNE(s),e))

$$= \begin{cases} e & \text{falls LEER(ENTFERNE(s))} \\ \text{ERSTES(ENTFERNE(s))} & \text{sonst} \end{cases}$$

Auf der rechten Seite wird die O-Funktion ENTFERENE auf eine um ein Element kürzere Schlange angewandt. Zur Vollständigkeit der Definition muß noch aufgenommen werden:

ENTFERNE(NEUE_SCHLANGE) = undefiniert

3. BEISPIEL: PAKETVERTEILUNGSANLAGE

3.1 AUFGABENSTELLUNG

Wir betrachten als Beispiel die Steuerung einer Paketverteilungsanlage. Die Anordnung der Anlage ist in Abb. 1 angegeben. Die in die Eingangsstation einlaufenden Pakete sind durch ein Codezeichen markiert, das die Zielstation angibt. Das System liest das Codezeichen und steuert danach die einzelnen Verteilstationen, welche das Durchlaufen des Paketes melden.

Man beachte, daß die einzelnen Verteilstationen keine Möglichkeit haben, das Codezeichen erneut zu lesen oder das Paket auf andere Weise nachträglich noch einmal zu identifizieren.

Wir gehen bei der Erstellung des Programmsystems davon aus, daß jeder Station ein Prozeß zugeordnet ist und die Verbindung durch Warteschlangen hergestellt wird, in die alle Pakete eingetragen sind, die bereits softwaremäßig von der vorhergehenden Station, nicht jedoch von der nächsten bearbeitet wurden.

Abb. 2 zeigt den Aufbau der Verteilstationen, deren Programmierung wir exemplarisch herausgreifen wollen.

Eingangs- und Ausgangspunkte jeder Verteilstation sind mit Lichtschranken versehen. Diese können das Passieren der einzelnen Pakete mit Sicherheit erkennen, auch wenn diese dicht aufeinander folgen. Die Meldungen werden im Programmsystem zur Laufwegverfolgung jedes einzelnen Paketes ausgewertet. Dadurch kann in Verbindung mit dem bereits markierten Ziel der Steuerauftrag für das nächste Lenkorgan ermittelt und ausgegeben werden.

Beim Ausgeben des Steuerauftrages ist darauf zu achten, daß alle Vorläufer die betreffende Verteilstation passiert haben. Die Verteilstation selbst muß frei sein, d.h. zwischen Eingangs- und Ausgangslichtschranke darf sich kein Paket befinden.

Da die Eingangsstation Pakete mit gleicher Zielstation dicht hintereinander abschicken darf und einen Sicherheitsabstand zur Umstellung der Lenkorgane nur bei unterschiedlichen Zielen einhalten

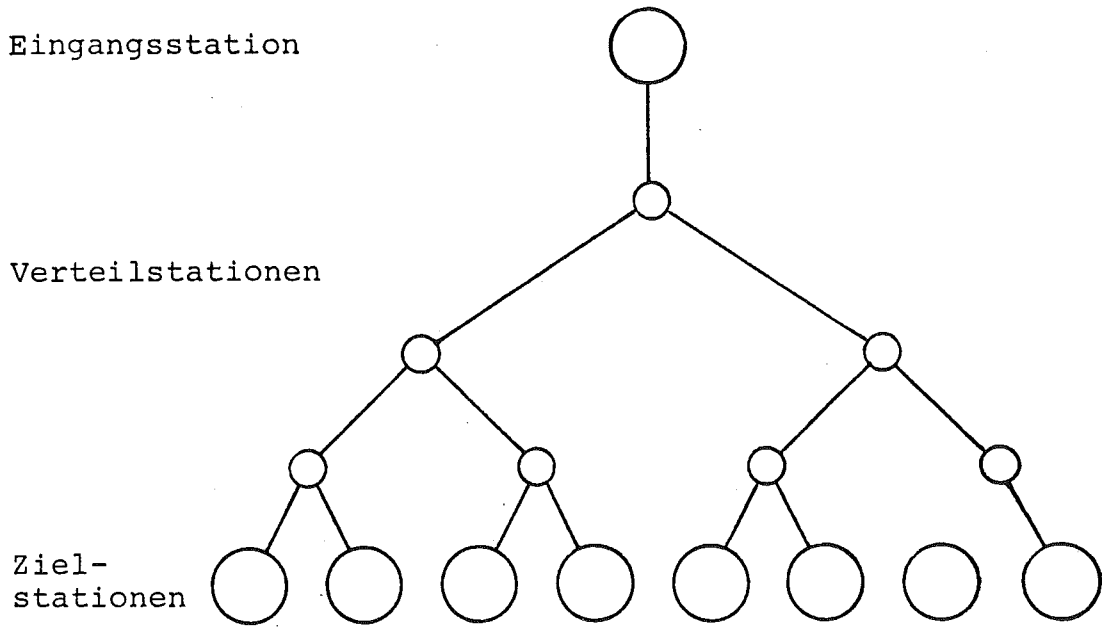

Eingangsstation
Verteilstationen
Ziel-
stationen

Abb. 1

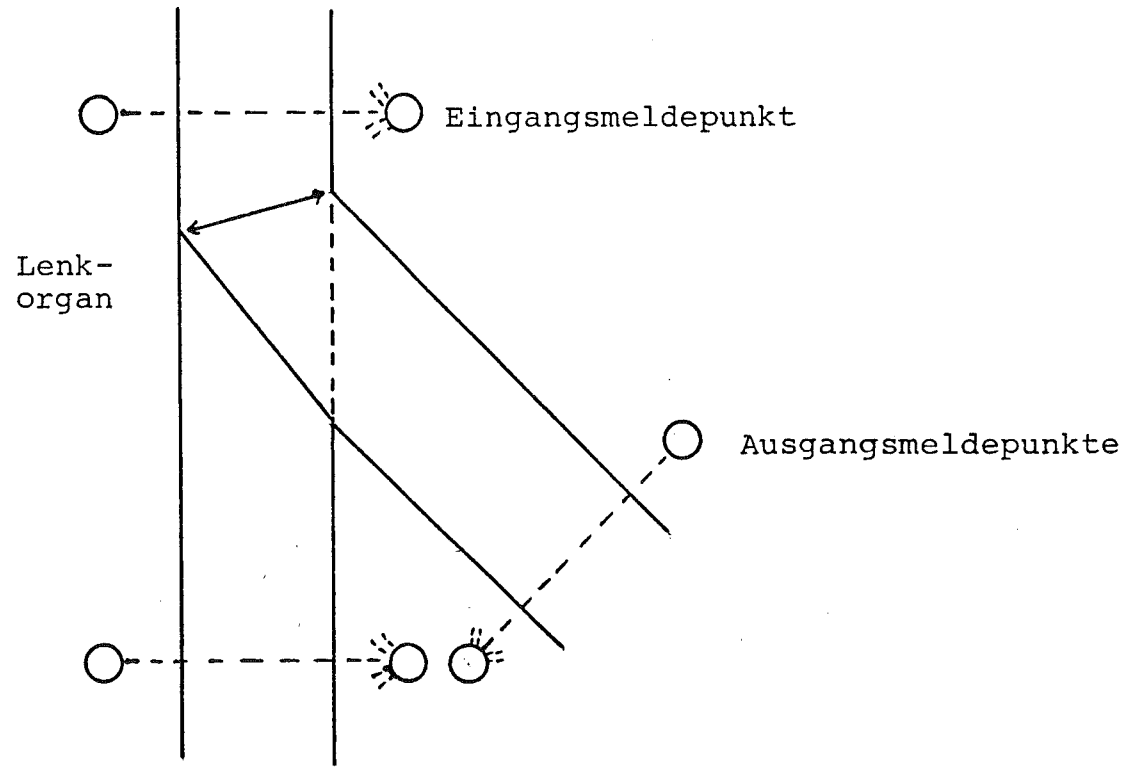

Eingangsmeldepunkt
Lenk-
organ
Ausgangsmeldepunkte

Abb. 2

muß, können sich u.U. mehrere Pakete in der Zielstation aufhalten.

Tritt auf Grund unterschiedlicher Geschwindigkeiten der Fall ein, daß ein Paket, für das eine Verteilstation umzustellen ist, deren Eingangsmeldepunkt erreicht, bevor der Vorläufer diese verlassen hat, muß die Umstellung unterbleiben, der Falschläufer bekommt das Ziel seines Vorläufers, und der Vorgang wird ausgedruckt.

Da die softwaremäßige Behandlung jeden Paketes somit abgeschlossen sein muß, wenn das Paket den Eingangsmeldepunkt erreicht, bietet sich dieser Punkt an, um die Software mit der Hardware zu synchronisieren.

3.2 DER DATENTYP PAKET

Der Datentyp paket soll hier nur kurz angerissen werden. Wir benötigen drei O-Funktionen:

a) Kreiere Paket: Diese Funktion kreiert ein Objekt des Typs paket und weist ihm als Wert eine Folge von Teilzielen (links, rechts) zu, die sich aus dem hardwaremäßig gelesenen Codezeichen bestimmen läßt.

b) Teilziel streichen: Diese Funktion kennzeichnet, daß das als nächstes anstehende Teilziel bearbeitet ist.

c) Als Fehlläufer markieren: Diese Funktion kennzeichnet das Paket als Fehlläufer. Es kann bei Erreichen einer Zielstation dann eine entsprechende Fehlermeldung erfolgen.*)

Ferner benötigen wir innerhalb der Verteilstation nur eine V-Funktion:

d) Bestimme Teilziel: Diese Funktion liefert, in Abhängigkeit von der vorgegebenen Teilzielfolge und den bereits abgearbeiteten Teilzielen, das nächste anstehende Teilziel (links, rechts) oder den Wert "Fehlläufer".

---

*) Da nicht auszuschließen ist, daß ein Paket mehrfach falsch läuft, erfolgt die Fehlerausgabe erst, wenn endgültig feststeht, in welcher Zielstation das Paket ankommt.

3.3 DER DATENTYP VERTEILSTATION

Stellen wir die Funktionen zusammen, so fallen uns zunächst die Funktionen ein, die wir auf Grund der Aufgabenstellung im Programm benötigen:

NEUE_VSTATION:		teilziel → vstation
SCHALTE:	vstation x	teilziel → vstation
VSTATION_LEER:		vstation → boolean
STELLUNG:		vstation → teilziel

Bei den beiden ersten Funktionen handelt es sich um O-Funktionen, bei den beiden letzten um V-Funktionen.

Fragen wir nun danach, auf welchem Wege eine Verteilstation den Zustand "leer" in "nichtleer" und umgekehrt ändert, so kommen wir auf zwei weitere O-Funktionen, die jedoch an keiner Stelle des der Verteilstation zugeordneten Prozesses aufgerufen werden, sondern durch die Unterbrechungen an den Eingangs- und Ausgangsmeldepunkten:

EIN:	vstation → vstation
AUS:	vstation → vstation

Für die soweit zusammengestellten Funktionen können wir nun sehr schnell die Semantik zusammenstellen, indem wir zu jeder O-Funktion angeben, welche Wirkung sie auf einen nachfolgenden Aufruf einer der beiden V-Funktionen hat:

a) Wirkungen auf VSTATION_LEER:

VSTATION_LEER(NEUE_VSTATION(t)) = true
VSTATION_LEER(SCHALTE(v,t)) = VSTATION_LEER(v)
VSTATION_LEER(EIN(v)) = false
VSTATION_LEER(AUS(EIN(v))) = VSTATION_LEER(t)

Man beachte die folgenden Punkte:

- Wir initialisieren die Verteilstation mit einer bestimmten Stellung, die wir aber nicht in der Definition festlegen.
- Daß SCHALTE nur auf eine leere Verteilstation angewandt wer-

den kann, kann dadurch berücksichtigt werden, daß LEER und SCHALTE verborgene Prozeduren werden und TESTE_SCHALTE die Aufgabe übernimmt.

b) Wirkungen auf STELLUNG:

STELLUNG(NEUE_VSTATION(t)) = t

$$\text{STELLUNG(SCHALTE(v,t))} = \begin{cases} t & \text{falls VSTATION\_LEER(v)} \\ \text{STELLUNG(v)} & \text{sonst} \end{cases}$$

STELLUNG(EIN(v)) = STELLUNG(v)

STELLUNG(AUS(v)) = STELLUNG(v)

An zwei Stellen des zu formulierenden Prozesses muß die Software mit der Hardware synchronisiert werden:

- Ist das Lenkorgan umzustellen, so darf dies nur geschehen, wenn der Bereich der Verteilstation leer ist. Andernfalls muß der Prozeß darauf warten, daß vorhergehende Pakete die Station verlassen. Gleichzeitig ist aber darauf zu achten, daß das angekündigte Paket nicht in die Station einläuft. Wir benötigen also eine Operation

  WARTE: vstation → vstation x {ein, aus, schon_da},

  die zurückmeldet, ob die Eingangs- oder Ausgangslichtschranke unterbrochen wurde oder das Paket die Eingangslichtschranke bereits vor Aufruf der Operation passiert hat.

- Ist das Lenkorgan nicht umzustellen oder die Umstellung erfolgt, so muß abgewartet werden, bis das Paket den Eingangsmeldepunkt passiert, um ein Auseinanderlaufen von Hardware und Software zu verhindern. Wir benötigen hierzu eine Operation

  WARTE_EIN: vstation → vstation x {ein, schon_da}

  Diese kann jedoch entweder mit Hilfe von WARTE beschrieben werden:

  WARTE_EIN = repeat S := WARTE;
  until S ≠ aus endrepeat

  oder selbständig in Analogie zu WARTE definiert werden. Die zweite

Möglichkeit ist der Zurückführung auf WARTE vorzuziehen, weil die Zurückführung bereits Implementierungsdetails enthält.

WARTE ist eine OV-Funktion, die im Gegensatz zu den OV-Funktionen im sequentiellen Fall nicht auf zwei voneinander vollständig unabhängige Funktionen

$$WARTE(v) = (WARTE_1(v), WARTE_2(v))$$

mit einer O-Funktion

$$WARTE_1: \quad vstation \rightarrow vstation$$

und einer V-Funktion

$$WARTE_2: \quad vstation \rightarrow \{ein, aus, schon\_da\}$$

zurückgeführt werden kann. Die beiden Komponenten ergeben jede für sich keinen Sinn, da $WARTE_1$ eine O-Funktion ist, die aus der Sicht des aufrufenden Prozesses erst durch $WARTE_2$ beendet wird. Aus dieser Sicht erscheinen also beide stets unmittelbar aufeinanderfolgend, datentypintern können aber die Operationen EIN und AUS zwischenzeitlich auftreten. Dies sind aber auch die einzigen Operationen, die dazwischen auftreten dürfen, was die Zahl der erforderlichen Axiome reduziert.

Wir müssen nun noch festlegen, welche Auwirkungen der Aufruf der O-Funktion $WARTE_1$ auf den späteren Aufruf der V-Funktionen hat:

$$WARTE_2(EIN(WARTE_1(NEUE\_VSTATION(t)))) = ein$$

$$WARTE_2(WARTE_1(EIN(NEUE\_VSTATION(t)))) = schon\_da$$

$$WARTE_2(AUS(WARTE_1(NEUE\_VSTATION(t)))) = aus$$

Auf die beiden anderen V-Funktionen hat WARTE keine unmittelbare Wirkung:

$$STELLUNG(WARTE_1(v)) = STELLUNG(v)$$

$$VSTATION\_LEER(WARTE_1(v) = LEER(v)$$

$$VSTATION\_LEER(AUS(WARTE_1(v)) = VSTATION\_LEER(AUS(v))$$

Wir haben nun die Aussage noch nicht berücksichtigt, daß jedem EIN ein WARTE entsprechen muß. Falsch wäre eine Formulierung wie

$$EIN(WARTE_1(NEUE\_VSTATION(t))) = NEUE\_VSTATION(t),$$

weil dies die Nebenwirkung auf VSTATION-LEER nicht berücksichtigt. Wir müssen also das Wegkürzen von EIN gegen $WARTE_1$ ausdrücklich auf die Warte-Operationen beschränken:

$$F(EIN(WARTE_1(NEUE\_VSTATION(t)))) = F(NEUE\_VSTATION(t))$$

mit

$$F(v) = \begin{cases} WARTE_2(EIN(WARTE_1(v))) & \text{oder} \\ WARTE_2(AUS(WARTE_1(v))) & \text{oder} \\ WARTE_2(WARTE_1(EIN(v))) \end{cases}$$

4. SCHLUSSBEMERKUNG

Diese Untersuchung ist etwas anders entstanden, als es auf dem Gebiet der abstrakten Datentypen üblich zu sein scheint. Dem Autor waren nämlich sowohl die Aufgabenstellung (Paketverteilungsanlage) als auch die Spezifikations-Methode (axiomatische Spezifikation) vorgegeben. Dementsprechend mißtrauisch wurde die Arbeit in Angriff genommen, in der Erwartung, daß die Methode auf ein derartiges Problem nicht anwendbar sei. Der erste Ansatz schien diese Erwartung auch zu bestätigen, da die Synchronisation und die Unterbrechbarkeit von Operationen der mathematischen Vorstellung einer in sich abgeschlossenen, nicht unterbrechbaren Operation widersprechen. Erst eine genauere Untersuchung förderte die in 3.3 beschriebene Lösung zu Tage. Es ist für den Leser vielleicht von Interesse zu erfahren, daß die nun beschriebene Lösung eine nur geringfügige Variante des dritten Ansatzes ist.

Abschließend kann festgestellt werden, daß die Methode, die Auswirkungen von O-Funktionen auf V-Funktionen zu untersuchen, eine sehr gute Prüfung der Vollständigkeit in dem Sinne bietet, ob man alle von der Aufgabenstellung her relevanten Axiome berücksichtigt hat. Einen ähnlichen Effekt könnte man mit den Pfadausdrücken [*Flon/Habermann, 1976*] erreichen. Ein verwandter Ansatz, der jedoch stärker operationell vorgeht, ist von *Keramidis* und *Mackert* [1979] beschrieben worden, die Anwendung auf ein sequentielles Anwenderproblem von *Denert* [1979].

LITERATUR

E. Denert (1979): "Software Modularisierung", Informatik-Spektrum 2, 4, p. 204 - 218

E.W. Dijkstra (1972): "Notes on structured programming", in: Structured Programming, Academic Press, New York

L. Flon/A.M. Habermann (1976): "Towards the construction of verifiable software systems", ACM SIGPLAN Notices 11, Spec. Issue (Conference on Data), p. 141 - 148

J.V. Guttag (1977): "Abstract data types and the development of data structures", Communications Association Computing Machinery 20, 6, p. 397 - 404

J.V. Guttag/J.J. Horning (1978): "The algebraic specification of abstract data types", Acta Informatica 10, p. 27 - 52

J.V. Guttag/E. Horowitz/D.R. Musser (1974): "The design of data type specifications", Proceedings 2nd Internat. Conference Software Engineering, San Francisco, p. 414 - 420
IEEE Conference Proceedings, IEEE, Long Beach

J.V. Guttag/E. Horowitz/D.R. Musser (1977): "Some extensions to algebraic specifications", ACM SIGPLAN Notices 12, 3, p. 63 - 67

J.V. Guttag/E. Horowitz/D.R. Musser (1978): "Abstract data types and software validation", Communications Association Computing Machinery 21, 12, p. 1048 - 1064

S. Keramidis/L. Mackert (1979): "Specification and implementation of parallel activities on abstract objects", Proceedings 4th Internat. Conference Software Engineering, München, p. 203 - 211,
IEEE Conference Proceedings, IEEE, Long Beach

B.H. Liskov/S.N. Zilles (1974): "Programming with abstract data types", ACM SIGPLAN Notices 9, 4, p. 50 - 59

D.L. Parnas (1972): "A technique for the specification of software modules with examples", Communications Association Computing Machinery 15, 5, p. 330 - 336

J.M. Spitzen/K.N. Levitt/L. Robinson (1978): "An example of hierarchical design and proof", Communications Association Computing Machinery 21, 12, p. 1064 - 1075

J.M. Spitzen/B. Wegbreit (1975): "The verification and synthesis of data structures", Acta Informatica 4, p. 127 - 144

N. Wirth (1977):"Towards a discipline of real-time programming", Communications Association Computing Machinery 20, 8, p. 577 - 583

I.D. Ichbiah et al. (1979): "Preliminary ADA reference manual", ACM SIGPLAN Notices 14, 6

SPEZIFIKATION DER PAKETVERTEILANLAGE MIT SPEZI

Wilfried Koch, Technische Universität Berlin
Institut für Angewandte Informatik, Softwaretechnik
Ernst-Reuter-Platz 7, TEL 18, 1000 Berlin 10

1. EINORDNUNG VON SPEZI

Im Entwicklungsprozeß eines Software- (oder Software/Hardware-) Systems werden deutlich mehrere Entwurfsschritte unterschieden. Weder für diese Tätigkeiten noch für die entsprechenden entstehenden Dokumente gibt es im Augenblick eine allgemein akzeptierte Begriffsvereinbarung. Wir grenzen hier die folgenden Entwicklungsphasen (und ihre Dokumente) voneinander ab:

- Funktionsentwurf als Phase der Erarbeitung eines problembezogenen und benutzerorientierten Funktionsmodells des künftigen Systems. Das entstehende, für weitere Phasen verbindliche Dokument, ist die Anforderungsdefinition (requirements specification, Pflichtenheft)

- Strukturentwurf mit Strukturdefinition (functional specification, "Spezifikation")

- weitere Phasen, die sich mit Implementierung und Wartung ungefähr beschreiben lassen.

SPEZI ist eine Sprache zur Formulierung von Strukturdefinitionen des zweiten Schrittes. Während des Strukturentwurfs wird die gestellte und im Modell festliegende Aufgabe in Teilaufgaben zerlegt. Diese Zerlegung erfolgt so, daß die Teilaufgaben später den zu realisierenden Software- oder Hardware-Komponenten entsprechen. Demzufolge sind Definitionen von Komponenten und deren gegenseitige Beziehungen Ziel der Untersuchung. Es interessiert dabei das äußere Verhalten von Komponenten, welches in Schnittstellenbeschreibungen niedergelegt wird. Voraussetzung für sinnvolle Schnittstellenbeschreibungen ist die Angabe von

definierenden Operationen (Funktionen, Prozeduren, Aktionen) jeder Komponente durch sinnvolle Namen und Parameterangaben. Eine Schnittstellenbeschreibung, die ausschließlich in der Gruppierung der definierenden Operationen in Export- und Importlisten besteht, reicht jedoch im allgemeinen nicht aus, um das äußere Verhalten von Komponenten (der Anforderungsdefinition) angemessen zu beschreiben. Auch die definierenden Operationen einer Komponente müssen in ihrem Zusammenspiel beschrieben werden, jedoch ohne die Freiheit der Implementierung in zu starkem Maße einzuschränken. Hier sind die Grenzen des Wünschbaren und Zulässigen oft schwer zu ziehen. Die Strukturdefinition soll die Durchführbarkeit der Implementierung zeigen und ist zugleich Prüfstein für deren Angemessenheit und Korrektheit.

Sprachen zur Formulierung von Strukturdefinitionen sollten die konsistente Überprüfung des Strukturentwurfs in verschiedenen Detaillierungsgraden konzeptionell und durch Realisierung entsprechender rechnergestützter Hilfsmittel möglich machen.

2. KONZEPTE VON SPEZI

2.1 Methodische Gesichtspunkte

Wie bereits erwähnt, ist SPEZI eine Sprache zur Formulierung der Strukturdefinition. Eine vollständige Spezifikation mit SPEZI spiegelt nicht den Entwurfsvorgang wider, sondern ist ihr Ergebnis.

Andererseits ist SPEZI idealerweise nur mit Hilfe eines rechnerunterstützten (Software-) Entwicklungssystems verwendbar, in dem Versionen (oder auch Varianten) der Strukturdefinition unterschiedlichen Detaillierungsgrades gehalten, geprüft und miteinander verschmolzen werden können. Die Sprache unterstützt diese Vorgehensweise durch den Verzicht auf "Vollständigkeit" der Spezifikation. Die meisten Sprachkonstrukte sind optional. Und eine Konsistenzüberprüfung ist nur soweit möglich, wie Formulierungen formalsprachlich vorliegen.

Erzwungen wird lediglich die Verbalisierung auf jeder Detailebene, bei der Modulbenennung ebenso wie bei der mindestens umgangssprachlich zu formulierenden Prozeduraufgabe (purpose-Konstrukt).

Die Komponentenzerlegung des Strukturentwurfs wird unterstützt durch ein Modulkonzept, mit dessen Hilfe sich Operationen, Typmengenspezifikationen und Objekte gruppieren lassen. Das Grundkonzept der Gruppierung ist der (abstrakte) Datentyp, auf den sich idealerweise eine Teilaufgabe des zu strukturierenden Systems abbilden läßt.

Eine Systemspezifikation mit SPEZI besteht aus der Formulierung von Modulschnittstellen. Zur Unterstützung der Übersichtlichkeit und der schrittweisen Vorgehensweise wird zwischen obligatorischen Export-, Import-Listen und weitergehenden Detailspezifikationen (Effektspezifikationen von Prozeduren, Typmengenspezifikationen, Objektspezifikationen, Pfaden und Axiomen) unterschieden. Eine gewisse Redundanz der entstehenden Dokumente ist unvermeidlich und erwünscht. Der erhöhte Schreibaufwand und die damit verbundene Fehleranfälligkeit soll durch geeignete rechnerunterstützte Hilfsmittel minimiert werden.

Zentrales Beschreibungselement der Modulaufgabe ist die Prozedur, deren Parameter (einschließlich eines möglichen Funktionsresultats) jeweils Werte genau eines Datentyps bezeichnen. Der Prozedureffekt kann unter anderem auch durch die direkte Attribut- und Zustandsänderung von Parametern und von anderen prozedurglobalen Objekten beschrieben werden. Prozedurglobale Objekte werden ausschließlich als Mittel zur Effektbeschreibung zusammengehöriger Prozeduren eingeführt. Sie unterliegen sowohl strikten Einschränkungen hinsichtlich des Typs der Werte, welche ihnen zugeordnet werden können (strong typing), als auch des Bereichs ihrer Verwendbarkeit. Moduln sind in SPEZI Konstrukte des "information hiding", um die unerwünschte Benutzung von spezifizierten Objekten und verborgenen Funktionen (hidden procs) außerhalb von Moduln zu unterbinden.

Das äußere Verhalten von Moduln wird in vielen Fällen entscheidend von zulässigen Ablauf-Pfaden ihrer definierenden Prozeduren geprägt. Dies trifft besonders zu auf die Synchronisation des parallelen Ablaufs mehrerer Instanzen gleicher oder verschiedener Prozeduren. SPEZI verzichtet bewußt auf ein Darstellungsmittel für Prozesse.

2.2 Modularten

Moduln dienen vornehmlich der Modellierung von Datentypen oder Datenstrukturen. Demzufolge ist die Kommunikation zwischen Moduln nur durch die definierenden Funktionen von (abstrakten) Datentypen - einschließlich der nullstelligen Konstanten - möglich. Der Modul ist jedoch nicht ausschließlich zur Modellierung jeweils genau eines Datentyps konzipiert. Vielmehr kann und soll zweckmäßigerweise ein Modul auch zur Gruppierung der definierenden Funktionen mehrerer verschiedener Datentypen dienen, wenn ein innerer Zusammenhang zwischen ihnen postuliert oder spezifiziert werden soll. Modul und modellierter Datentyp sind demzufolge nicht in jedem Falle identisch. Deshalb kann eine von der Modulbezeichnung verschiedene Typmengenbezeichnung eingeführt werden, die bei Bedarf ebenfalls exportiert bzw. importiert werden kann.

Andererseits kann der Modul genausogut zur Beschreibung von Systemteilen mit reinem Kontrollcharakter dienen. Neben den normalen Moduln gibt es in SPEZI Hauptmoduln (main modules), welche die Wurzel eines sequentiellen Kontrollbaumes enthalten.

Schließlich gibt es Schnittstellenmoduln (interface modules), welche die Beziehungen zwischen Teilen des technischen Systems und solchen des Steuersystems zu spezifizieren gestatten. Dazu werden Signale des technischen Systems und ihre Korrespondenz (correspondence-Konstrukt) zu parametrisierbaren Prozeduranwendungen des Steuersystems angegeben.

In den defines-Listen (Export) und applies-Listen (Import) werden die Bezeichnungen, Prozeduren, Konstanten und Typmengenbezeichnungen mit ihren syntaktisch unmittelbar formulierbaren Attributen aufgelistet. Durch die applies-Listen aller Moduln, die jeweils nach den zum Import beitragenden Moduln gegliedert sind, wird die "Benutzt-Struktur" des Gesamtsystems beschrieben.

In SPEZI können Moduln als ganzes parametrisiert werden. Ein derart parametrisierter Modul ist als Schema für nebeneinanderliegende Modulinstanzen aufzufassen, wobei jede Instanz über genau denselben Satz definierender Funktionen verfügt. Diese Art der Modulspezifikation kann überall dort angewendet werden,

wo in der Hardware- oder Software-Realisierung des zu spezifizierenden Systems Funktionen (oder Signale) mehrfach realisiert sind oder realisiert werden sollen, wobei ihre funktionellen Eigenschaften jeweils identisch sind. Im Beispiel unter Abschnitt 4 spezifizieren wir für jeden Verteiler (mit Ausnahme der Zielstation) jeweils einen Schnittstellenmodul. Damit soll angedeutet werden, daß es pro Verteiler gleichartige Signale gibt, die Paketeingang, Paketausgang melden und das Lenkorgan umstellen.

2.3 Objekte, Typen, Wertattribute und Zustände

Ein Objekt ist Träger von Werten eines (abstrakten) Typs. In SPEZI treten Objekte explizit auf als vereinbarte Größen auf Modulebene (spezifiziert durch Typbezeichner, Namen und das Zugriffsrecht var oder const), oder als formale Parameter von Prozeduren.

Effekte von Prozeduren werden aufgefaßt als Veränderungen des Wertattributes oder des Zustandes von Objekten. "Wertattribut" ist dabei eine Abstraktion von Wert, "Zustand" ist die Abstraktion von anderen Objekteigenschaften, die nicht durch den Objektwert beschrieben werden sollen oder können. So können wertmäßig gleiche Objekte durchaus in ihrem Zustand verschieden sein. Entsprechend besteht in SPEZI die vollständige explizite Beschreibung einer Typmenge aus der Angabe des Typbezeichners, eines Wertattributbereichs und eines Zustandsbereichs. Der Wertattributbereich ist in benannte Komponenten gegliedert, für die jeweils eine Liste von Attributbezeichnern angegeben wird. Ein Wertattribut ist dann ein Vektor von Attributbezeichnern der einzelnen Komponenten. Der Zustandsbereich wird durch eine Liste von Zustandsbezeichnern angegeben.

Zur Beschreibung von Effekten stellt SPEZI die Selektion von Wertattributkomponenten von Objekten (z.B. station . pos), die Selektion des Objektzustands (z.B. state of object), sowie die Gleichheitsrelation getrennt für Objekte, Wertattributkomponenten und Zustände zur Verfügung. Die Gleichheit von Objekten bedeutet dabei die Gleichheit ihrer Wertattribute.

Neben der expliziten Beschreibung gibt es in SPEZI die aus der algebraischen Spezifikation stammende Möglichkeit, Typen, Objekte, ihre Wertattribute und Zustände implizit zu beschreiben, d.h. nur durch die darauf anwendbaren Operationen.

2.4 Effektspezifikationen

Der Effekt einer Prozedur besteht in der Wertattribut- oder Zustandsveränderung von Objekten. Verändert werden entweder Parameter oder vereinbarte Objekte des definierenden Moduls (unmittelbarer Effekt, immediate effect), oder aber Objekte in anderen Moduln. Effekte der letzten Art lassen sich nur mittelbar mit Hilfe importierter Prozeduren beschreiben (mediate effect) und zwar durch zeitlich geordnete Prozedurapplikationen. Die Syntaxregeln für diese Spezifikation entsprechen denen der Pfadausdrücke. So bedeutet z.B.
(melde an (erste v-station, paketziel), (verzögere; gib eingangssperre frei))
daß die Aktionen

melde an (erste v-station, paketziel) und
(verzögere; gib eingangssperre frei)

zeitlich parallel ablaufen können, während *gib eingangssperre frei* erst abläuft, wenn *verzögere* beendet ist.

Unmittelbare Effekte können explizit und implizit angegeben werden. Die explizite Effektbeschreibung gibt den oder die Wertattribute bzw. Zustände des Objektes nach Ausführung der Prozedur an (angedeutet durch den Apostroph hinter dem Objektnamen, z.B. *vorziel' = paketziel*).

In impliziten Effektbeschreibungen wird dagegen der Effekt mehrerer (im allgemeinen zum gleichen Typ gehörenden) Prozeduren in ihrer gegenseitigen Wirkung aufeinander beschrieben. Implizite Effektbeschreibungen haben die Form einer logischen Implikation.

Das Beispiel

[eingangsteil leer (s); (eintrag (s,)) n; eintrag (s, x); (eintrag (s,)); (übertrag (s)) n]*
=> erwartetes paket (s) = x.

ist folgendermaßen zu interpretieren: Wenn die Prozeduren in der Klammer in angegebener Reihenfolge ausgeführt worden sind (wobei der Exponent n n-malige Ausführung, * null- oder mehrmalige, + ein- oder mehrmalige Ausführung bedeutet), dann gilt die nach dem Pfeil gemachte Aussage. Die einzelnen Terme in der Klammer dürfen auch boolesche Ausdrücke sein, die Parametrisierung applizierter Prozeduren darf fehlen, wenn sie irrelevant für das Verständnis ist. Gleiche Bezeichner einer impliziten Spezifikation bezeichnen dieselben Objekte oder Größen.

Durch implizite Effektbeschreibungen werden im allgemeinen globale Aussagen über Prozeduren eines Typs gemacht. Solche Beschreibungen werden daher nicht einzelnen Prozeduren zugeordnet, sondern sind auf Modulebene zu Axiomen zusammengefaßt.

Ein sehr wichtiges Mittel der Effektbeschreibung ist die Fallunterscheidung nach Wertattributen bzw. Zuständen von Objekten beim Eingang in die Prozedur. Von einer Fallunterscheidung zur Beschreibung des unmittelbaren Effektes einer Prozedur wird verlangt, daß alle Fälle sich gegenseitig ausschließen. Zusammen mit den als illegal gekennzeichneten Fällen muß die Fallunterscheidung des unmittelbaren Effektes außerdem vollständig sein (bzw. die spezifizierten Fälle müssen die als legal gekennzeichnete Bedingung voll überdecken). Für Fallunterscheidungen mittelbarer Effekte wird nur die Überschneidungsfreiheit gefordert.

Einige Fälle in der unmittelbaren Effektbeschreibung sind als Ausnahmefälle (exceptions) gekennzeichnet und benannt. Im Gegensatz zu den illegal-Fällen, welche das System ausschließen muß (und die daher Handlungsanweisungen an den Programmierer sind), können Exceptions vorkommen. Zu ihrer Behandlung wird der normale Programmablauf, der durch die anderen Fälle beschrieben ist, durchbrochen. Wo und wie solche Ausnahmesituationen behandelt werden, ist nicht Gegenstand der Spezifikation. Interessant ist lediglich die Spezifikation der Bedingungen, unter denen nach Ausnahmebehandlung der Normalablauf der Prozedur wiederaufnehmbar ist (resuming possible with).

2.5 Pfadspezifikationen

Eine Pfadspezifikation beschreibt die zulässigen Ablauffolgen von Prozedurinstanzen der in ihr vorkommenden Prozeduren. Jeder Prozedurbezeichner darf höchstens in einer Pfadspezifikation seines definierenden Moduls vorkommen. Sind in einem Modul keine Pfadspezifikationen angegeben, so bedeutet dies, daß beliebig viele Prozedurinstanzen in beliebiger Reihenfolge, insbesondere überlappt ablaufen können. Ist aber eine Pfadspezifikation vorhanden, so können beliebig viele Prozedurinstanzen derjenigen Prozeduren, die nicht in der Pfadspezifikation vorkommen, nebenläufig zu den spezifizierten Pfaden ablaufen.

Eine Pfadspezifikation definiert also Einschränkungen gegenüber beliebiger Nebenläufigkeit. Spezifizieren läßt sich die endliche Sequenz ((a; b; c)), die sequentielle Wiederholung ((a)*, (a) + oder (a) n), die endliche Auswahl (one of (a, b, c)), die kollaterale Folge ((a, b, c)) zur Beschreibung der Nebenläufigkeit und entsprechend die kollaterale Wiederholung ((a)⊛, (a) ⊕ oder (a) ⊕ n). Die elementaren Bestandteile einer Pfadspezifikation sind Prozeduranwendungen mit vollständiger, unvollständiger oder fehlender Parameterliste. Textuell gleiche Parameterbezeichner derselben Pfadspezifikation stehen für textuell gleiche Aktualparameter der spezifizierten Prozedurinstanzen.

Von jeder spezifizierten Prozedurinstanz, Sequenz, sequentiellen Wiederholung, Auswahl, kollateralen Folge und kollateralen Wiederholung wird die Termination postuliert und auf die Termination der Bestandteile zurückgeführt. So terminiert die endliche Sequenz mit der Termination des textuell letzten Elementes, die kollaterale Folge bei Termination aller Elemente. Zur bequemen Spezifikation von Ablauffolgen von Prozedurinstanzen bei beliebigem aber stets gleichem Parameter eines Typs, wird die Schreibweise

for (type s): p (s)

als Abkürzung eingeführt für

$(p\ (s_1),\ p\ (s_2)\ \ldots\ ,\ p\ (s_n))$

mit s_i vom Typ type und $n \geq 0$.

Um die Übersichtlichkeit zu erhöhen werden als elementare Ausdrücke innerhalb von Pfadspezifikationen außerdem Flußbezeichner zugelassen, die in vorangehenden oder nachfolgenden Flußdefinitionen verfeinert werden können.

Die Pfadspezifikation

path for (verteiler s):
(one of (((<eingangsaktion>), (<durchgangsaktion>)*), übertrag (s,)))**
endpath

bedeutet, daß beliebige Sequenzen von Eingangs- und Durchgangsaktionen unabhängig voneinander nebenläufig ablaufen können, daß jedoch eine jede Prozedurinstanz *übertrag (s,)* (deren zweiter Parameter nicht interessiert), niemals parallel zu einer Eingangsaktion oder Durchgangsaktion ablaufen darf. Diese Aussage gilt für jeden Wert aus der Wertemenge *verteiler*. Eingangsaktion und Durchgangsaktion werden im Beispiel durch Flußdefinitionen weiterverfeinert. Die Pfadspezifikation besagt zum Beispiel, daß an jeder Verteilstation unsynchronisiert Pakete einlaufen und auslaufen können, daß jedoch der Übertrag eines Paketes aus der Eingangs-Warteschlange vor der Station in die Durchgangs-Warteschlange in der Station den gleichzeitigen Eintrag eines neuen Paketes in die Eingangs-Warteschlange und den gleichzeitigen Austrag eines Paketes aus der Durchgangs-Warteschlange ausschließt.

3. ERLÄUTERUNGEN DER ENTWURFSENTSCHEIDUNGEN

Aus der Aufgabenstellung wird entnommen, daß ein Steuersystem für die physisch existierende Paketverteilanlage zu spezifizieren ist. Das Steuersystem wird insofern als passiv angesehen, als es lediglich auf Signale der Außenwelt reagiert. Die Reaktion auf Signale erfolgt durch Prozedurapplikationen, die umgekehrt in Signale umgesetzt werden. Entsprechend dem physikalischen Aufbau des technischen Systems wird das Steuersystem in ein solches zur Steuerung der Eingangsstation und der Verteilstationen zerlegt. Entsprechend gibt es den Schnittstellenmodul "Eingangsschnittstelle" und den mit den Verteilstationen parametrisierten Modul "Verteilerschnittstelle".

Die Steuerung der Eingangsstation wird zu einer Datenstruktur mit der definierenden Funktion *melde paketankunft* , welche durch den Modul "Eingangssteuerung" beschrieben wird.

Die Steuerung aller Verteilstationen wird als Datentyp aufgefaßt, der durch den Modul "Verteilersteuerung" modelliert wird.

Der nächste und entscheidende Schritt besteht in der Einführung des abstrakten Datentyps "Verteiler", dessen Elemente nicht mehr die gesamte Steuerung einer Verteilstation abbilden, sondern nur noch die "Topologie" der Verteilstationen und Zielstationen des technischen Systems. Entsprechend dieser Vorstellung gibt es die diesen Typ definierenden Funktionen *nachfolger*, *soll-lenkung*, *lenkerstand*, *ändere lenkerstand* etc. Die Auslagerung des Typs "Verteiler" in den Modul "Systemkonfiguration" hat zu Folge, daß ein Typmengenbezeichner *verteiler* aus "Systemkonfiguration" exportiert werden muß, um an anderer Stelle (z.B. im Modul "Verteilersteuerung") Parameter dieses Typs spezifizieren zu können.

Um das Durchlaufen der Pakete durch eine physische Verteilstation zu simulieren, wird jedes Element der "Verteilersteuerung" mit zwei Paketwarteschlangen - einer Kaskade - dekoriert. Der Eingangsteil der Kaskade simuliert die Pakete vor der Eingangslichtschranke der Station, der Durchgangsteil diejenigen Pakete, die sich in der Station befinden und noch keine der beiden Ausgangslichtschranken passiert haben. Auch "Kaskade" wird zu einem Datentyp, dessen definierende Funktionen in den Modul "Kaskaden" ausgelagert werden. Dieser Modul importiert insbesondere den Typmengenbezeichner *verteiler* (wie übrigens auch der Modul "Verteilersteuerung"), weil natürlich durch die mit demselben Verteiler-Wert parametrisierten Funktionen genau eine Kaskade dargestellt wird.

Pakete werden nur an einer Stelle des technischen Systems durch ihre Zielstation identifiziert. Weitere Paketattribute werden für das Steuersystem nicht benötigt und demzufolge nicht spezifiziert. Pakete sind also vom Typ "Verteiler" mit der besonderen Bedingung, daß für sie die Funktion *ist zielstation* stets wahr ist. Für Pakete wird überdies unterschieden, ob ihr Ziel eine echte Zielstation des Systems ist oder ob sie als Aufschrift ein Ziel tragen, das keiner echten Zielstation zuzuordnen ist (Pseudozielstation). Solche Pakete müssen neben denjenigen,

die echte Ziele tragen, aber den falschen Weg gelaufen sind, ebenfalls als Falschläufer erkannt werden.

Zur sinnvollen Darstellung der Topologie der Verteilstationen wird neben dem Typ "Verteiler" der Datentyp "Richtung" eingeführt, dessen Typmenge genau drei Werte hat, was durch eine Aufzählungskonstruktion der Typspezifikation angegeben wird. Die Konstante *egal* wird benötigt, um nicht existierende Verbindungen zwischen zwei Verteilstationen anzugeben. "Systemkonfiguration" ist das Beispiel für einen Modul, der zwei Datentypen modelliert.

4. DIE PAKETVERTEILANLAGE IN SPEZI

```
interface module eingangsschnittstelle.
  defines: proc gib eingangssperre frei.

  applies  from eingangssteuerung:
             proc melde paketankunft (in verteiler paketziel).
           from systemkonfiguration:
             verteiler, proc konvertiere nach zielcode (in int code) result verteiler.

  path     ((paketankunft, codezeichen); freigabe)* endpath.

  signal   in  paketankunft, codezeichen,
           out freigabe.

  correspondence:
           (paketankunft, codezeichen) causes
             melde paketankunft (konvertiere nach zielcode
                                 (nach int konvertiertes codezeichen)).
            gib eingangssperre frei causes freigabe.

  proc     gib eingangssperre frei using (out freigabe):
           purpose: die vom Steuersystem aufrufbare Prozedur bewirkt die Ausgabe
                    des Freigabesignals end
  endproc.

  hidden proc nach int konvertiertes codezeichen result int using (in codezeichen):
           purpose: das physikalische Codezeichen wird in einen Wert vom Typ int
                    transformiert end
  endproc.

endmod eingangsschnittstelle.
```

```
interface module verteilerschnittstelle [verteiler station].
  illegal: ist zielstation (station).

  defines: proc verstelle lenkorgan.

  applies  from verteilersteuerung:
             proc melde paketeingang (in verteiler station),
             proc melde paketausgang (in verteiler station, in richtung r).
           from systemkonfiguration:
             verteiler, proc ist zielstation (in verteiler v) result bool.
             richtung, richtung const links, rechts.

  signal   in  eingangssignal, ausgangssignal links, ausgangssignal rechts,
           out umstellungssignal.

  correspondence:
           eingangssignal causes melde paketeingang (station).
           ausgangssignal links causes melde paketausgang (station, links).
           ausgangssignal rechts causes melde paketausgang (station, rechts).
           verstelle lenkorgan causes umstellungssignal.

  proc     verstelle lenkorgan using (out umstellungssignal):
           purpose: wird von Verteilersteuerung gerufen und bewirkt Umstellen
                    des Lenkorgans end
  endproc.

endmod verteilerschnittstelle.
```

```
module eingangssteuerung.
  defines: proc melde paketankunft (in verteiler paketziel).

  applies  from eingangsschnittstelle:
             proc gib eingangssperre frei.
           from verteilersteuerung:
             proc melde an (in verteiler station, in verteiler paketziel).
           from systemkonfiguration:
             verteiler, verteiler const erste v-station,
             proc ist zielstation (in verteiler v) result bool,
             proc ist pseudozielstation (in verteiler v) result bool.

  verteiler var vorziel.

  axioms:  ist zielstation (vorziel) and not ist pseudozielstation (vorziel).

  proc     melde paketankunft (in verteiler paketziel) using (trans verteiler vorziel):
           purpose: ein Paket, ausschließlich durch sein Ziel identifiziert,
                    wird dem Steuersystem gemeldet end
           legal:   ist zielstation (paketziel).
           immediate effect:
                    case ist pseudozielstation (paketziel):.
                    otherwise: vorziel' = paketziel.
           mediate effect:
                    case ist pseudozielstation (paketziel) or paketziel = vorziel:
                      (melde an (erste v-station, paketziel), gib eingangssperre frei).
                    otherwise: (melde an (erste v-station, paketziel),
                                (verzögere; gib eingangssperre frei)).
  endproc.

  hidden proc verzögere:
           purpose: es wird eine maximale Durchlaufzeit durch eine leere
                    Verteilstation gewartet end
  endproc.

endmod eingangssteuerung.
```

```
module verteilersteuerung.
  defines: proc melde an (in verteiler station, in verteiler paketziel),
           proc melde paketeingang (in verteiler station),
           proc melde paketausgang (in verteiler station, in richtung r).

  applies from verteilerschnittstelle:
             proc verstelle lenkorgan.
           from kaskaden:
             proc eintrag (in verteiler station, in verteiler paketziel),
             proc übertrag (in verteiler station),
             proc entnahme (in verteiler station) result verteiler,
             proc durchgang frei (in verteiler station) result bool,
             proc erwartetes paket (in verteiler station) result verteiler.
           from protokoll:
             proc vermerke falschläufer (in verteiler ist-ziel, in verteiler soll-ziel).
           from systemkonfiguration:
             verteiler, proc nachfolger (in verteiler station, in richtung r)
                                result verteiler,
             proc soll-lenkung (in verteiler station, in verteiler paketziel)
                                result richtung,
             proc ist zielstation (in verteiler station) result bool,
             proc ist pseudozielstation (in verteiler station) result bool,
             proc lenkerstand (in verteiler station) result richtung,
             proc ändere lenkerstand (in verteiler station).
             richtung, richtung const links, rechts, egal.

  proc     melde an (in verteiler station, paketziel):
           purpose: ein Paket, repräsentiert durch das Paketziel, wird vom
                    Steuersystem der Steuerung der nächsten Station gemeldet end
           legal:   ist zielstation (paketziel) and not ist pseudozielstation (station).
           mediate effect:
                    case ist zielstation (station) and paketziel ≠ station:
                      vermerke falschläufer (station, paketziel).
                    case ist zielstation (station) and paketziel = station:.
                    otherwise: eintrag (station, paketziel).
  endproc.
```

```
proc    melde paketeingang (in verteiler station):
        purpose: der Eingang eines Paketes an einer physikalischen Verteil-
                 station wird der Steuerung der Station gemeldet; die Station
                 darf keine Zielstation sein, weil Zielstationen keine Melde-
                 organge besitzen end
        illegal: ist zielstation (station).
        mediate effect:
                 case not durchgang frei (station)
                      or soll-lenkung (station, erwartetes paket (station)) =
                         egal
                      or soll-lenkung (station, erwartetes paket (Station)) =
                         lenkerstand (station):
                   übertrag (station).
                 otherwise: (verstelle lenkorgan (station),
                             ändere lenkerstand (station),
                             übertrag (station)).
endproc.

proc    paketausgang (in verteiler station, in richtung r):
        purpose: der Ausgang eines Paketes aus einer physikalischen Verteil-
                 station nach rechts oder links wird der Steuerung der Station
                 gemeldet; die Station darf keine Zielstation sein end
        illegal: ist zielstation (station) or r = egal.
        immediate effect:
                 case r ≠ lenkerstand (station):
                      exception lenkerstand nicht ok
                      resuming possible with r = lenkerstand (station).
                 case r = lenkerstand:.
        mediate effect:
                 melde an (nachfolger (station, r), entnahme (station)).
endproc.

endmod verteilersteuerung.
```

```
module kaskaden.
  defines: proc eintrag (in verteiler station, in verteiler paketziel),
           proc erwartetes paket (in verteiler station) result verteiler,
           proc übertrag (in verteiler station),
           proc durchgang frei (in verteiler station) result bool,
           proc entnahme (in verteiler station) result verteiler.

  applies from systemkonfiguration:
             verteiler, proc ist zielstation (in verteiler station) result bool.

  path     for (verteiler s):
             (one of (((<eingangsaktion>)*, (<durchgangsaktion>)*), übertrag (s,)))*
           endpath.

  flow     <eingangsaktion>:
             one of (eintrag (s,), erwartetes paket (s)).
  flow     <durchgangsaktion>:
             one of (entnahme (s), durchgang frei (s)).

  proc     eintrag (in verteiler station, in verteiler paketziel):
           purpose: ein Paket, repräsentiert durch sein Paketziel wird als vor
                    der Station wartend vermerkt end
           illegal: ist zielstation (station).
  endproc.

  proc     erwartetes paket (in verteiler station) result verteiler:
           purpose: das Paketziel des nächsten vor der Station wartenden Paketes
                    wird geliefert end
           illegal: ist zielstation (station).
           immediate effect:
                    case eingangsteil leer:
                         exception kein paket angemeldet
                         resuming possible with not eingangsteil leer.
                    case not eingangsteil leer:.
  endproc.
```

```
proc     übertrag (in verteiler station):
         purpose: das nächste vor der Station als wartend registrierte Paket
                  wird dort entfernt und als in der Station befindlich
                  vermerkt end
         illegal: ist zielstation (station).
         immediate effect:
                  case eingangsteil leer:
                       exception kein paket übertragbar
                       resuming possible with not eingangsteil leer.
                  case not eingangsteil leer:.
endproc.

proc     durchgang frei (in verteiler station) result bool:
         purpose: Abfrage auf leere Verteilstation end
         illegal: ist zielstation (station).
endproc.

proc     entnahme (in verteiler station) result verteiler:
         purpose: das Paketziel des ersten in der Verteilstation befindlichen
                  Pakets wird geliefert, das Paket wird aus der Station
                  entfernt end
         illegal: ist zielstation (station).
         immediate effect:
                  case durchgang frei:
                       exception kein paket zu entnehmen
                       resuming possible with not durchgang frei.
                  case not durchgang frei:.
endproc.

hidden proc eingangsteil leer (in verteiler station) result bool:
         purpose: Abfrage auf leere Warteschlange vor der Station end
endproc.
```

axioms: [*(eintrag (s,))+*]

⟹ *not eingangsteil leer (s).*

[*(übertrag (s))+*]

⟹ *not durchgang frei (s).*

[*eingangsteil leer (s); (eintrag (s,)) m; (übertrag (s)) p;*
(eintrag (s,)) n; (übertrag (s)) q; m+n=p+q]

⟹ *eingangsteil leer (s).*

[*eingangsteil leer (s); (eintrag (s,)) n; eintrag (s, x);*
(eintrag (s,)); (übertrag (s)) n*]

⟹ *erwartetes paket (s) = x.*

[*durchgang frei (s); (übertrag (s)) m; (entnahme (s)) p; (übertrag (s)) n;*
(entnahme (s)) q; m+n=p+q]

⟹ *durchgang frei (s).*

[*durchgang frei (s); (übertrag (s)) n; erwartetes paket (s) = x;*
(übertrag (s)) +; (entnahme (s)) n; entnahme (s) = y]

⟹ *x = y.*

endmod kaskaden.

```
module systemkonfiguration.
  defines: verteiler, verteiler const erste v-station,
           proc nachfolger (in verteiler station, in richtung r) result verteiler,
           proc soll-lenkung (in verteiler station, in verteiler paketziel)
                result richtung,
           proc ist zielstation (in verteiler station) result bool,
           proc ist pseudozielstation (in verteiler station) result bool,
           proc lenkerstand (in verteiler station) result richtung,
           proc ändere lenkerstand (in verteiler station).
           richtung, richtung const rechts, links, egal.
           proc konvertiere nach ziel (in int code) result verteiler.

  path     for (verteiler s):
             (one of (lenkerstand (s), ändere lenkerstand (s)))* endpath.

  type     verteiler is
             value set attributes:
             pos (echte zielstation, pseudozielstation, zwischenstation).
  endtype.

  verteiler const erste v-station in (zwischenstation).

  proc     nachfolger (in verteiler station, in richtung r) result verteiler:
           purpose: der rechte oder linke Nachfolger einer Zwischenstation wird
                    geliefert end
           illegal: ist zielstation (station) or r = egal.
           immediate effect:
                    result . pos ≠ pseudozielstation.
  endproc.

  proc     soll-lenkung (in verteiler station, in verteiler paketziel) result richtung:
           purpose: berechnet die zulässige Richtung von einer Verteilstation
                    zum Paketziel end
           legal:   station . pos = zwischenstation and
                    paketziel . pos ≠ zwischenstation.
  endproc.
```

```
proc    ist zielstation (in verteiler station) result bool:
        purpose: Abfrage auf echte Zielstation oder Pseudozielstation end
        immediate effect:
                result = (station . pos ≠ zwischenstation).
endproc.

proc    ist pseudozielstation (in verteiler station) result bool:
        purpose: Abfrage auf Pseuodzielstation end
        immediate effect:
                result = (station . pos = pseudozielstation).
endproc.

proc    lenkerstand (in verteiler station) result richtung:
        purpose: Abfrage auf aktuellen Lenkerstand der Station im
                 Steuersystem end
        legal:   station . pos = zwischenstation.
        immediate effect:
                result ≠ egal.
endproc.

proc    ändere lenkerstand (in verteiler station):
        purpose: Änderung des aktuellen Lenkerstandes der Station im
                 Steuersystem end
        legal:   station . pos = zwischenstation.
endproc.

proc    konvertiere nach ziel (in int code):
        purpose: Konvertiert einen int-Wert in den Wert einer echten
                 Zielstation oder Pseudozielstation end
        immediate effect:
                result . pos ≠ zwischenstation.
endproc.
```

axioms: [z . pos = echte zielstation; r ≠ egal; nachfolger (s, r) = z]
⟹ soll-lenkung (s, z) = r.
[z . pos = echte zielstation; r ≠ egal;
nachfolger (s, r) . pos = zwischenstation;
soll-lenkung (nachfolger (s, r), z) ≠ egal]
⟹ soll-lenkung (s, z) = r.
[z . pos = echte zielstation;
nachfolger (s, rechts).pos = nachfolger (s, links).pos = echte zielstation;
nachfolger (s, links) ≠ z; nachfolger (s, rechts) ≠ z]
⟹ soll-lenkung (s, z) = egal.
[z . pos = echte zielstation;
nachfolger (s, rechts).pos = nachfolger (s, links).pos = zwischenstation;
soll-lenkung (nachfolger (s, links), z) =
soll-lenkung (nachfolger (s, rechts), z) = egal]
⟹ soll-lenkung (s, z) = egal.
[z . pos = pseudozielstation]
⟹ soll-lenkung (s, z) = egal.
[lenkerstand (s) = rechts; ändere lenkerstand (s)]
⟹ lenkerstand (s) = links.
[lenkerstand (s) = links; ändere lenkerstand (s)]
⟹ lenkerstand (s) = rechts.

type richtung *is*
value set constr: [links, rechts egal].
endtype.

richtung *const* links, rechts, egal.

endmod systemkonfiguration.

```
module protokoll.
  defines: proc vermerke falschläufer (in verteiler ist-ziel, soll-ziel).

  applies from systemkonfiguration:
             verteiler,
             proc ist zielstation (in verteiler ziel) result bool,
             proc ist pseudozielstation (in verteiler ziel) result bool.

  proc     vermerke falschläufer (in verteiler ist-ziel, soll-ziel):
           purpose: notiert ein Paket bei Erreichen einer echten Zielstation
                    durch Angabe dieser Zielstation (ist-Ziel) und des
                    Paketziels (soll-Ziel) end
           legal:   ist zielstation (ist-ziel) and ist zielstation (soll-ziel) and
                    not ist pseudozielstation (ist-ziel).
  endproc.

endmod protokoll.
```

LITERATUR:

Brinch Hansen, P.:
The Architecture of Concurrent Programs,
Prentice Hall, 1977

Brinch Hansen, P.; Staunstrup, J.:
Specification and Implementation of Mutual Exclusion,
IEEE Trans. on Softw. Eng., Vol SE-4, No 5, Sept. 1978

Brinch Hansen, P.:
Distributed Processes: A Concurrent Programming Concept,
CACM 21 (11), pp. 934-941, 1978

Campbell, R.H.; Habermann, A.N.:
The Specification of Process Synchronization by Path Expressions,
Lecture Notes in Computer Science 16, Springer Verlag, 1974

Campbell, R.H.; Kolstad, R.B.:
Path Expressions in PASCAL,
Proc. 4th Intern. Conf. on Software Engineering, München, Sept. 1979,
IEEE Catalog No 79CH1479-5C, pp. 212

Goos, G.; Kastens, K.:
Programming Languages and the Design of Modular Programs,
in: Hibbard, P.G.; Schuman, S. (eds.): Constructing Qualitiy Software,
Proc., IFIP-TC2 Conf. Novosibirsk 1977, North Holland Publ. Comp., 1978

Guttag, J.V.:
Abstract Data Types and the Development of Data Structures,
Conf. on Data, Salt Lake City 1976, CACM 20 6(77), pp. 396-404

Hoare, C.A.R.:
Monitors: An Operating System Structuring Concept,
CACM 17 (10/74), pp. 549-557

Jähnichen, S.:
"Exception Handling" in sequentiellen Programmen,
Dissertation, Fachbereich Informatik, TU Berlin, 1979

Kimm, R.; Koch, W.; Simonsmeier, W.; Tontsch, F.:
Einführung in Software Engineering,
Walter de Gruyter, 1979

Koch, W.; Schmiedecke, I.:
Spezifikation einer Prozess-Steuerung,
TU Berlin, Fachbereich Informatik, Softwaretechnik, April 1979

Koch, W.:
Syntax von SPEZI - Elemente zur Beschreibung sequentieller Programmstrukturen,
TU Berlin, Fachbereich Informatik, Softwaretechnik, Dezember 1979

Kolstad, R.B.; Campbell, R.H.:
Path PASCAL User Manual,
Dept. of Computer Science, Univ. of Illinois at Champain-Urbana, Sept. 1979

Kopetz, H.; Lohnert, F.; Merker, W.:
An Outline of Project MARS Maintainable Realtime System,
TU Berlin, Fachbereich Informatik, Bericht 79-09, Juli 1979

Parnas, D.L.:
A Technique for Software Module Specification with Examples,
CACM 15 (72), pp. 330-336

Parnas, D.L.:
On the Criteria to be Used in Decomposing Systems into Modules,
CACM 15 (72), pp. 1053-1058

Schmiedecke, I.:
Vorschlag zur Begriffsvereinbarung,
PDV-Arbeitskreis

Shaw, A.C.:
Software Specification Languages Based on Regular Expressions,
ETH Zürich, Inst. für Informatik, Juni 1979

Unterlagen zur Lehrveranstaltung "Software Engineering" 1979/80 der TU Berlin:
Beschreibungsmittel für die Spezifikation - Die Sprache SPEZI