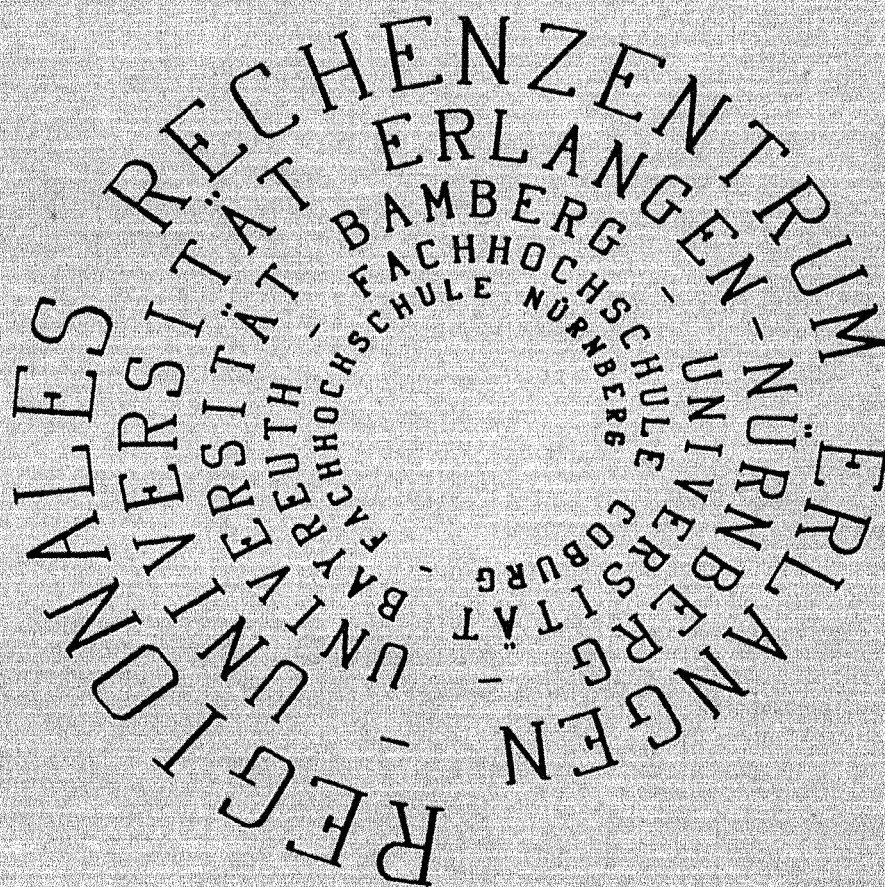


# MITTEILUNGSBLATT

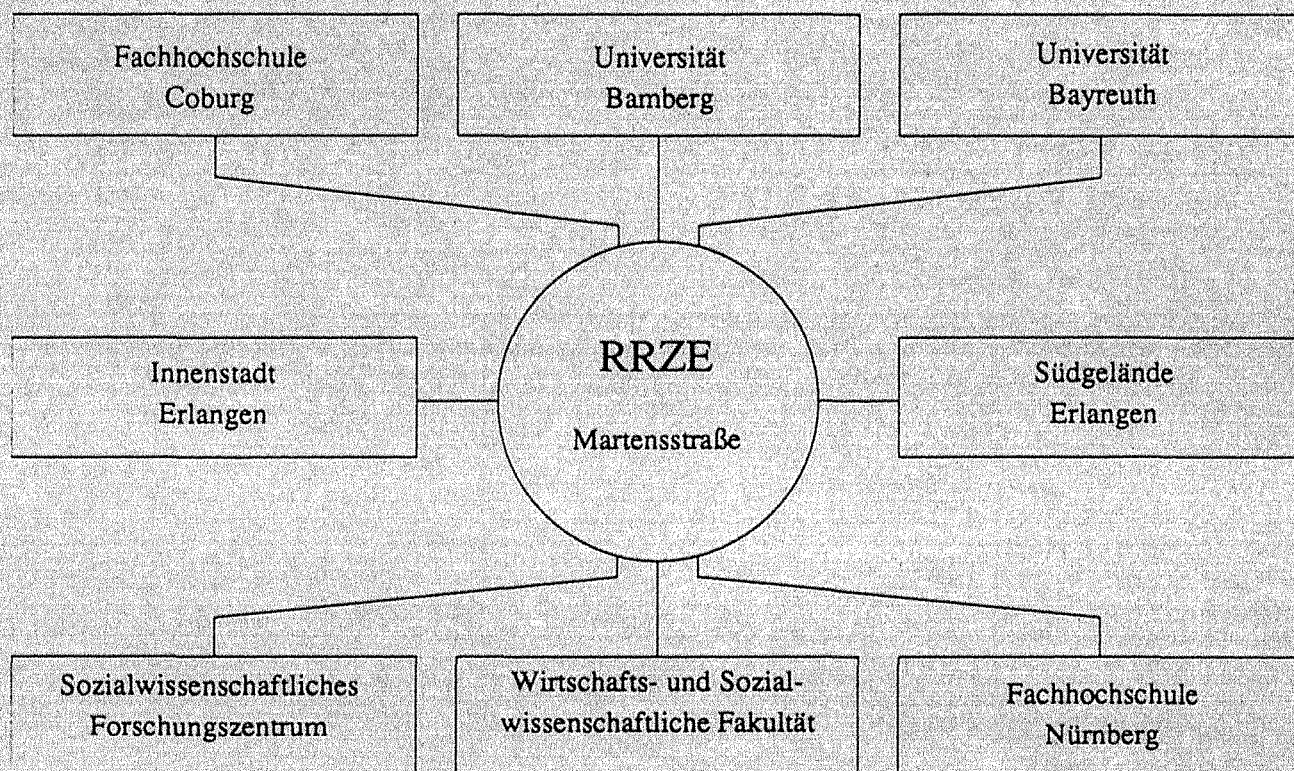
DES REGIONALEN  
RECHENZENTRUMS ERLANGEN

HERAUSGEBER F. WOLF



NR. 51      - ERLANGEN -      Februar 1989





## **REGIONALES RECHENZENTRUM ERLANGEN**

Martensstraße 1, 8520 Erlangen, Tel.: 09131/85-7031

Beteiligte Institutionen:

Universität Erlangen-Nürnberg  
Universität Bamberg  
Universität Bayreuth  
Fachhochschule Coburg  
Fachhochschule Nürnberg

ISSN 0172-2905 (Mitteilungsblatt des Regionalen Rechenzentrums)

# MITTEILUNGSBLATT

DES REGIONALEN  
RECHENZENTRUMS ERLANGEN

HERAUSGEBER F. WOLF

Ein graphentheoretischer Ansatz zur  
rechnergestützten parallelen Komposition von Prozessen  
für verteilte Echtzeitsysteme

Christian Andres

NR. 51      - ERLANGEN -      Februar · 1989

## Inhaltsverzeichnis

0 Zusammenfassung . . . . .	1
1. Einleitung . . . . .	2
11 Die Entwurfsphase im Software-Entwicklungszyklus . . . . .	3
12 Validieren von Spezifikationen . . . . .	5
13 Rapid Prototyping . . . . .	7
14 Beschreibung und Analyse verteilter Systeme mit Hilfe formaler Methoden . . . . .	9
15 Ziele dieser Arbeit . . . . .	10
2 Spezifizieren von Echtzeitsystemen. . . . .	12
21 Architektur von Echtzeitsystemen. . . . .	12
22 Eigenschaften eines Echtzeitsystems . . . . .	14
23 Synchronisation und Kommunikation in Realzeitsystemen . . . . .	17
23.1 Ein Modellprozeß . . . . .	17
23.2 Synchronisations- und Kommunikationsprobleme in der Pro- zeßautomatisierung . . . . .	18
23.3 Das Botschaftskonzept zur Synchronisation und Kommuni- kation verteilter Prozesse . . . . .	21
23.4 Logische Verknüpfung von Botschaftsoperationen . . . . .	22
24 Prozesse in Realzeitsystemen . . . . .	24
24.1 Der Prozeßbegriff. . . . .	24
24.2 Ablaufplan . . . . .	26
25 Zusammenfassung . . . . .	30
3 Beschreibung verteilter Systeme . . . . .	31
31 Petri-Netze . . . . .	31
3.1.1 Grundbegriffe der Netztheorie . . . . .	32
3.1.2 Analyse von Petri-Netzen . . . . .	36
3.1.3 Ausdrucksfähigkeit . . . . .	39
3.1.4 Analysefähigkeit . . . . .	41
32 CCS - ein Kalkül zur Beschreibung paralleler Prozesse . . . . .	42
3.2.1 Syntax und Semantik von CCS-Operatoren . . . . .	43
3.2.2 Äquivalenz von Prozessen . . . . .	46
3.2.3 Ausdrucksfähigkeit . . . . .	48

3.2.4 Analysefähigkeit . . . . .	49
3.3 PASS - Parallel Activities Specification Scheme. . . . .	50
3.3.1 Strukturierungseinheiten in PASS . . . . .	51
3.3.2 Kommunikationsstruktur . . . . .	51
3.3.3 Prozeßbeschreibung . . . . .	52
3.3.4 Ausdrucksfähigkeit . . . . .	56
3.3.5 Analysefähigkeit . . . . .	57
3.4 Zusammenfassung . . . . .	57
4. Die Semantik von PASS . . . . .	60
4.1 Ein Zustandsmodell für PASS-Prozesse . . . . .	60
4.2 Zeit in Realzeitsystemen . . . . .	63
4.2.1 Ein Modell zur Beschreibung von Zeit . . . . .	65
4.2.2 Lokale und globale Uhren . . . . .	66
4.3 Graphersetzungssysteme . . . . .	70
4.3.1 Graphen und Einbettung von Untergraphen . . . . .	70
4.3.2 Graphersetzungen . . . . .	76
4.3.2.1 Sequentielle Graphersetzungen . . . . .	76
4.3.2.2 Programmierte Ersetzungen . . . . .	80
4.4 Eine Graphgrammatik für PASS . . . . .	82
4.4.1 Der Programmgraph . . . . .	82
4.4.2 Attributierung . . . . .	85
4.4.3 Ein Beispiel: Das synchrone Senden einer Botschaft . . . . .	89
4.5 Gemischte Ersetzungen . . . . .	94
4.6 Das Semantikmodell. . . . .	96
5. Simulation von Systemen paralleler Prozesse . . . . .	98
5.1 Darstellung des Modellverhaltens . . . . .	99
5.1.1 Benutzersicht . . . . .	99
5.1.2 Realisierung paralleler Abläufe durch gemischte Ableitungen . . . . .	99
5.2 Beschränkung des Zustandsraums . . . . .	103
5.2.1 Simulation unvollständiger Spezifikationen . . . . .	103
5.2.2 Zyklen in Ablaufsteuerungen . . . . .	104
5.3 Steuerung des Modellablaufs . . . . .	105
5.3.1 Auswahl alternativer Zustandsübergänge . . . . .	105
5.3.2 Einbettung von Steuermechanismen in die Graphproduktionen . . . . .	107
5.4 Analysefähigkeiten des Interpreters . . . . .	109
6. Untersuchungen mit dem Interpreter - ein Beispiel . . . . .	112

<b>7. Aufbau des Interpreters . . . . .</b>	<b>122</b>
7.1 Der Interpreter aus Benutzersicht . . . . .	123
7.2 Interner Aufbau des Interpreters . . . . .	127
7.2.1 Window-Manager . . . . .	128
7.2.2 Steuerebene . . . . .	128
7.2.3 Funktionsebene . . . . .	129
 <b>8. Erfahrungen . . . . .</b>	 <b>131</b>

## 0. Zusammenfassung

Der Entwurf von verteilten Echtzeitsystemen zur Prozeßautomatisierung erfordert Methoden zur Beschreibung von Kommunikation und Synchronisation zwischen Prozessen, von Ereignissen, denen Prioritäten zugeordnet sind, und von Zeitbedingungen.

Dadurch entstehen besondere Anforderungen an die Ausdrucksfähigkeit und Analysefähigkeit der verwendeten Spezifikationssprache.

In dieser Arbeit soll ein Analysewerkzeug für die Spezifikationsmethode PASS entworfen werden. PASS ist eine graphisch orientierte Spezifikationsmethode, die auf dem Modell des erweiterten Zustandsautomaten basiert. Die Methode hat sich im ingenieurmäßigen Einsatz zur Beschreibung von Echtzeitsystemen bewährt.

Die Semantik von PASS-Sprachelementen wurde bisher nur informell beschrieben. Mit Hilfe eines Graphersetzungssystems soll eine formale, operationelle Semantikbeschreibung erstellt werden. Dies ist Voraussetzung für die Analyse von PASS-Modellen.

Beschreibt man die Semantik der Elemente einer Spezifikationssprache mit Hilfe von Produktionen einer Graphgrammatik, so ergibt sich dadurch ein abstrakter Interpreter, der, ausgehend von einem Startgraphen, alle korrekten Zustände eines Spezifikationsmodells ableitet.

Die Zuordnung von Prozessen zu verschiedenen Prozessoren, die Einbeziehung von Zeitbedingungen sowie die Forderung nach Unteilbarkeit von Befehlen bestimmen die zulässigen parallel oder sequentiell anwendbaren Ersetzungsschritte.

Der verfolgte Ansatz vereint die Beschreibung von Parallelität, ähnlich dem True-Concurrency-Modell der Petri-Netz-Theorie, und dem kompositionellen Ansatz von algebraischen Kalkülen, wie CCS und CSP.

Mit einem auf den vorgestellten Ansatz aufbauenden Interpreter werden Spezifikationsmodelle ausführbar und stellen einen Prototypen des zu entwerfenden Systems dar. Untersuchungen am Prototypen können interaktiv oder automatisch durchgeführt werden.

Typische, untersuchbare Eigenschaften sind zum Beispiel:

- Erreichen eines bestimmten Kommunikationszustands,
- Bestimmen von zeitlichen Pfadlängen,
- Dimensionierung von Wartebereichen.

Das Werkzeug kann in der Entwurfsphase zur Konkretisierung der Anforderungsdefinition und zur Validierung eines Spezifikationsmodells eingesetzt werden.

## 1. Einleitung

In der Software-Entwicklung zeichnet sich der Trend ab, immer "verantwortungsvollere" Aufgaben an Computersysteme zu delegieren. So werden Verkehrssysteme wie Eisenbahn und Luftverkehr von Computersystemen gesteuert und überwacht, werden ganze Fabrikationsanlagen, wie z.B. in der Automobilindustrie, vollständig von Rechensystemen kontrolliert. Letztlich hängt die Sicherheit und Funktionstüchtigkeit von hochtechnisierten Einrichtungen, wie z.B. Kernkraftwerken, von zuverlässig arbeitenden Software-Systemen ab.

Für diese Aufgaben, die unter dem Begriff *Prozeßautomatisierung* eingeordnet werden, sind in der Regel einzelne Rechner nicht mehr leistungsfähig genug. Die Komplexität der Software-Systeme erfordert meist eine Verteilung der Aufgaben auf mehrere Rechner.

Gründe dafür können sein:

- erhöhte Zuverlässigkeitsanforderungen, die eine Mehrfachauslegung der Hard- und Software notwendig machen, um ein bestimmtes Maß an Ausfallsicherheit zu gewährleisten;
- Zeitanforderungen, die die Korrektheit einer Funktion davon abhängig machen, daß sie in einem festgelegten Zeitintervall durchgeführt wird;
- räumliche Verteilung der zu kontrollierenden Objekte, die eine Anpassung des Rechnernetzwerkes an technologische oder ökonomische Gegebenheiten erfordern.

Zur Vernetzung bieten sich verschiedene Hardware-Lösungen an. Verwendet man speichergekoppelte Systeme, die mit Hilfe von Multiportspeichern (z.B. EGPA



/Händ 84/) realisiert werden, spricht man von Multiprozessoren. Sind Rechner über lokale Netze (z.B. Ethernet, Token-Ring) oder Weitverkehrsnetze (z.B. X.25) gekoppelt, bezeichnet man solche Konfigurationen als *verteilte Systeme*.

Die Anwendungsprogramme in der Prozeßautomatisierung zeichnen sich durch den Einsatz des *Prozeßkonzepts* als Strukturierungsmittel aus. Komplexe Aufgabenstellungen werden in Teilaufgaben zerlegt, die durch parallele oder quasi-parallele Prozesse bearbeitet werden.

Als *parallele* Prozesse bezeichnen wir solche Prozesse, die, verteilt auf mehrere Prozessoren gleichzeitig ablauffähig sind. *Quasi-parallele Prozesse* sind prinzipiell auch parallel ablauffähig. Da für solche Prozesse jedoch nur ein Prozessor zur Verfügung steht, kann zu einem Zeitpunkt nur genau ein Prozeß bearbeitet werden. Die Prozesse unterliegen dabei einer Betriebssystemverwaltung, die die Prozessorzuteilung regelt.

Die Prozesse organisieren ihre Zusammenarbeit mit Hilfe von Synchronisations- und Kommunikationsoperationen. In verteilten Systemen werden dazu *Botschaftsmechanismen* verwendet. Sie sind dann unerlässlich, wenn Prozesse auf verschiedenen Rechnern lokalisiert sind und eine Kommunikation über einen gemeinsamen Speicher ausgeschlossen ist. Derartige verteilte Systemarchitekturen sind im Bereich der Prozeßautomatisierung sehr häufig anzutreffen.

Unterliegt die Kommunikation zwischen einem verteilten Prozeßsystem und der Umgebung zeitlichen Anforderungen, so sprechen wir von einem verteilten *Realzeitsystem* oder *Echtzeitsystem*. Zeitliche Anforderungen werden dann als "hart" bezeichnet, wenn das korrekte Arbeiten des Prozeßsystems von der Einhaltung der Zeitbedingungen abhängt.

### 1.1 Die Entwurfsphase Im Software-Entwicklungszyklus

Der bedeutendste qualitative Anspruch an Software-Systeme ist deren Korrektheit. Die Korrektheit von Programmen ist früh zum Gegenstand wissenschaftlicher Forschung geworden; die Forschungsgebiete Semantik von Programmiersprachen und Programmverifikation widmen sich diesem Thema.

Allerdings ist die Vorgehensweise, qualitative Eigenschaften an einem "fertigen" Produkt, nämlich dem Programm, nachzuprüfen, keine optimale Strategie. Man kann erst zu einem späten Zeitpunkt feststellen, daß gewünschte Eigenschaften des zu konstruierenden Software-Systems nicht vorhanden sind. Deshalb wurde, angelehnt an klassische Ingenieurdisziplinen, der Begriff "Software-Engineering" geprägt, als das systematische Entwerfen von Software-Systemen mit dem Anspruch qualitative Eigenschaften in ein System hineinzukonstruieren /Kimm 79/.

Zur Zerlegung der komplexen Aufgabe "Software-Entwicklung" wird dazu ein phasenorientiertes Modell des Entwicklungsvorgangs definiert.

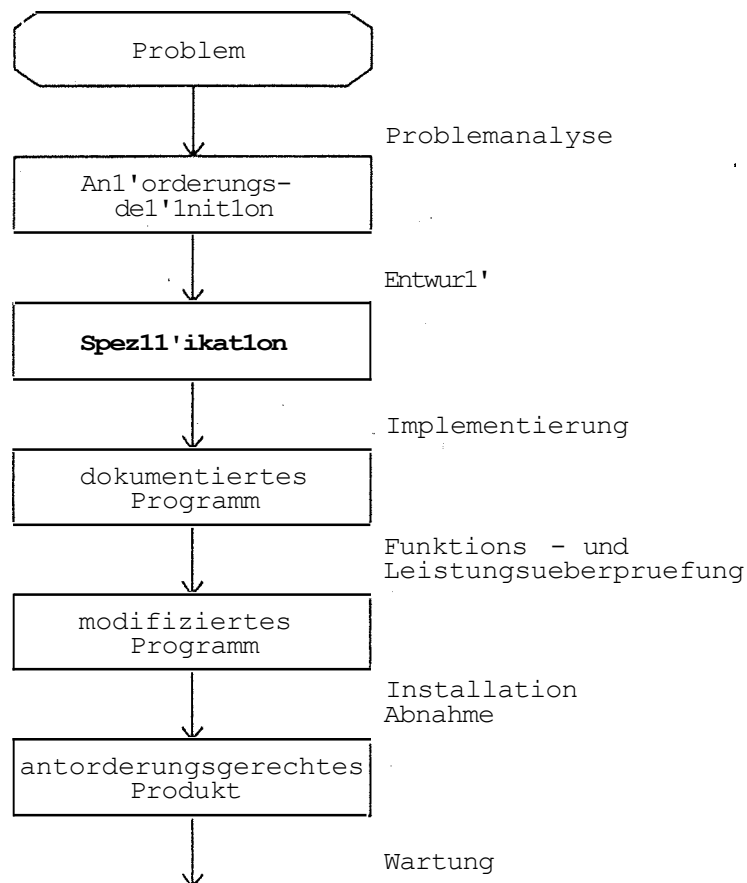


Bild 1-1 : Der Software-Entwicklungszyklus /Kimm 79/

Die systematische Entwicklung von Programmen wird in Phasen aufgeteilt. Jedes in einer Phase erstelltes Produkt muß auf Korrektheit überprüft werden, um das Auftreten von Fehlern möglichst frühzeitig zu erkennen.

Untersuchungen an größeren Software-Projekten haben ergeben, daß rund ein Drittel aufgetretenen Fehler bei der Implementierung, zwei Drittel jedoch beim Entwurf und bei der Integration entstanden /Boeh 82/. Deshalb ist die Prüfung einer Spezifikation auf Entwurfsfehler unerlässlich. Die Prüfung muß auch, deshalb in diesem Entwicklungsstadium erfolgen, da die Kosten für die Beseitigung steigen, je später Fehler entdeckt werden.

Diese Arbeit befaßt sich deshalb mit der Validierung von Spezifikationen für Echtzeitsystemen. Bei Echtzeitsystemen werden besonders hohe Anforderungen an die Zuverlässigkeit gestellt.

## 1.2 Validieren von Spezifikationen

Auf die Erfüllung des Qualitätsanspruchs beim Entwurf verteilter Systeme zielen Techniken zur formalen Spezifikation und Verifikation. Die Verwendung von Spezifikationstechniken hebt das Problemverständnis, hilft Mißverständnisse zu vermeiden und eröffnet die Möglichkeit, Entwurfsfehler durch Validieren der Spezifikation zu erkennen.

Nach /Timm 82/ umfaßt die Validierung einer Spezifikation zwei Bereiche:

### - Überprüfung der externen Korrektheit

Untersucht wird, ob die spezifizierten Eigenschaften eines Produkts mit den vom Auftraggeber gewünschten Funktionen und Eigenschaften übereinstimmen. Abweichungen sollen lokalisiert und deren Ursachen ermittelt werden. Problematisch dabei ist, daß Anforderungen nur informell gegeben sind und gerade im Bereich der Prozeßautomatisierung oft unvollständig sind /Levi 81/. Daher müssen Anforderungen und Spezifikation oft in einem gegenseitigen Wechselspiel konkretisiert und angepaßt werden.

### - Überprüfung der internen Korrektheit

Die interne Überprüfung umfaßt die Untersuchung auf sachliche Richtigkeit der Spezifikation. Darunter versteht man das Vorhandensein allgemeingültiger Eigenschaften wie z.B.:

- Vollständigkeit

Alle während der Ausführung möglichen Situationen werden behandelt.

- Verklemmungsfreiheit

Außer den definierten Endzuständen gibt es keine terminalen Zustände, aus denen das System nicht mehr weiter fortschreiten kann.

- Fortschritt

Es gibt keine unproduktiven Schleifen, in denen ein Prozeß des Systems aktiv ist; für alle Partner gilt gleichmäßiger Fortschritt.

- Erreichbarkeit

Im Gesamtsystem sind bestimmte Konstellationen, die sich durch Zusammenarbeit der Einzelprozesse ergeben, erreichbar.

Die Validierung von Spezifikationen wird beeinflusst durch die Meßbarkeit von Eigenschaften. Eine Spezifikation besitzt die Eigenschaft der Meßbarkeit, wenn die einzelnen aufgabenbezogenen Aussagen in quantifizierbarer Form beschrieben sind /Timm 82/. Die Möglichkeit der Meßbarkeit spielt besonders bei der Validation von zeitlichen Leistungsanforderungen eine Rolle. Realzeitsysteme sind ja gerade dadurch gekennzeichnet, daß die Korrektheit von Funktionen von der "Rechtzeitigkeit" der Ausführung abhängt.

Formale Spezifikationstechniken und darauf basierende Verifikationstechniken erlauben die Überprüfung der internen Korrektheit. Sie können danach klassifiziert werden, inwieweit sie vom detaillierten Programmablauf abstrahieren.

Im wesentlichen lassen sich die Abstraktionsebenen Synchronisationsverhalten, Kommunikationsverhalten und detaillierter Programmablauf unterscheiden.

Das Synchronisationsverhalten ist über den Prozeßzuständen "blockiert" und "aktiv" definiert. Prozesse werden als Erzeuger und Verbraucher von Signalen behandelt, die zwischen den Prozessen ausgetauscht werden und blockierte Prozesse fortsetzen.

Das Kommunikationsverhalten berücksichtigt zusätzlich die zwischen Prozessen ausgetauschten Daten. Die Behandlung des Programmablaufs befaßt sich mit der Beschreibung von Operationen auf Daten von Prozessen.

Da der zur Analyse von Systemen paralleler Prozesse nötige Zeitaufwand im allgemeinen mit dem Produkt der Zustände wächst, wird in der Regel nur das Synchronisations- bzw. Kommunikationsverhalten unter weitgehenden Abstraktionen von prozeßinternen Abläufen untersucht.

### 1.3 Rapid Prototyping

Die Validierung der externen Korrektheit wird häufig mit Hilfe eines Prototypen des zu entwickelnden Programmsystems durchgeführt. Da die Anforderungsdefinition nur informell vorgegeben ist, eignen sich formale Techniken nur bedingt. Prototyping wird als beste Methode angesehen, um in einem frühen Stadium der Entwicklung zu gewährleisten, daß die Spezifikation mit der Intention des Auftraggebers übereinstimmt /Floy 84/. Ziel des Prototyping ist, ein Programm zu erstellen, das das funktionale Verhalten der Spezifikation wiedergibt. Es kann durch Werkzeuge wie z.B. Debugger, Aufrufverfolger und Meßwerkzeuge zur Leistungsanalyse untersucht werden.

Man unterscheidet verschiedene Prototyping-Ansätze /Floy 84/:

- Exploratives Prototyping

Exploratives Prototyping wird angewendet, um eine erste Umsetzung von Anforderungsdefinitionen in eine Spezifikation zu demonstrieren. Dabei werden Anforderungsdefinitionen präzisiert, vervollständigt und erste Wege zur Problemlösung definiert. Wesentliches Ergebnis ist ein gemeinsames Verständnis von Auftraggeber und Entwickler darüber, wie das endgültige Produkt auszusehen hat.

- Experimentelles Prototyping

Bei diesem Ansatz wird eine vorgeschlagene Lösung des gegebenen Problems auf bestimmte Eigenschaften, wie z.B. Leistungsfähigkeit oder Ressourcenverbrauch, untersucht und mit der Anforderungsdefinition verglichen. Der Prototyp wird entsprechend einer vorliegenden Spezifikation entwickelt. Er kann als Voraussetzung zur Verfeinerung der Spezifikation oder als erster Zwischenschritt von der Spezifikation zur Implementierung dienen.

- Evolutionäres Prototyping

Evolutionäres Prototyping verfolgt den Gedanken der schrittweisen Verfeinerung. Der Übergang von einer Phase zur nächsten erfolgt nicht erst dann, wenn ein vollständiges Produkt, also z.B. eine vollständige Spezifikation, vorliegt; es wird vielmehr mit unvollständigen Produkten der gesamte Entwicklungszyklus durchlaufen. Am vorläufigen Endprodukt werden Untersuchungen und Analysen vorgenommen, um so Hinweise auf noch vorzunehmende Vervollständigungen zu gewinnen. Damit werden solange neue Versionen der Endprodukte erzeugt,



bis das Entwicklungsziel erreicht ist.

Diese Prototyping-Ansätze lassen sich, wie in Bild 1-2 gezeigt, den Phasen des Software-Entwicklungszyklus zuordnen.

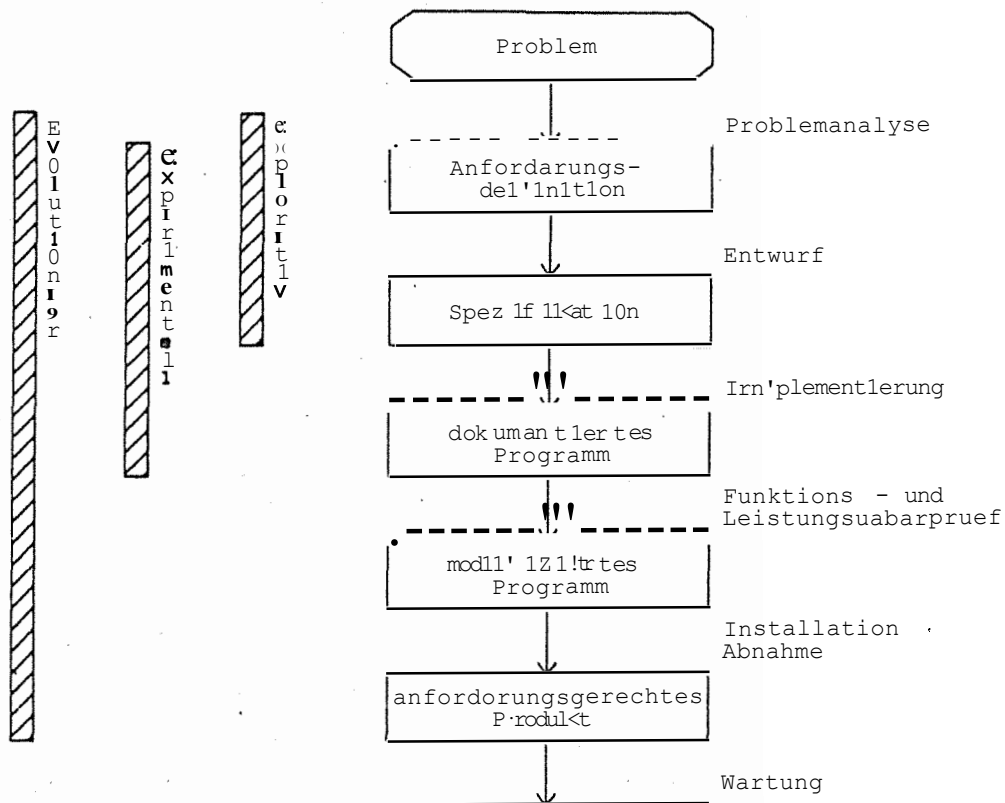


Bild 1-2 : Prototyping im Entwicklungszyklus

Der evolutionäre Ansatz ist eigentlich nur eine Umschreibung der üblichen Software-Entwicklung, die sich nur in geringem Maße formaler Techniken und geeigneter Entwicklungswerkzeuge bedient, und daher zu unvollständigen und fehlerhaften Programmen führt, die ständig verbessert werden.

Exploratives und experimentelles Prototyping, angewandt in den frühen Phasen des Entwicklungszyklus, sind unterstützende Vorgehensweisen, die der Rückkopplung zwischen den Phasen Problemanalyse und Entwurf dienen. Dadurch kann die Entwicklung anforderungsgerechter Spezifikationen gewährleistet werden. Außerdem sind Prototypen häufig unerlässlich, um Leistungsaussagen über eine

gewählte Problemlösung zu machen.

Normalerweise werden hohe Programmiersprachen (z.B. PROLOG, SETL und SMALLTALK) zum Erstellen eines Prototyps benutzt. Da sich solche Sprachen aber nicht zur Implementierung von Realzeitanwendungen eignen, bedeutet dies, daß Prototyp und endgültiges System in verschiedenen Sprachen implementiert werden müssen. Damit werden Prototypen zu Wegwerf-Produkten.

Außerdem müssen Prototypen, wie andere Programme auch, auf Übereinstimmung mit der Spezifikation geprüft werden, um auch verlässliche Aussagen machen zu können.

Daher ist mit Prototyping insgesamt ein hoher Aufwand verbunden, der die Anwendung dieses Ansatzes nur bei besonders hohen Anforderungen an ein zu entwickelndes System wirtschaftlich erscheinen läßt.

#### **1.4 Beschreibung und Analyse verteilter Systeme mit Hilfe formaler Methoden**

Zur Beschreibung und Analyse verteilter Systeme haben in den letzten Jahren sogenannte algebraische Kalküle, wie CCS (Calculus of Communicating Systems) /Miln 80/ und CSP (Communicating Sequential Processes) /Hoar 85/, große Bedeutung erlangt. Grundidee dieser Ansätze ist die modulare Beschreibung von Systemen. Systeme werden in parallel ablauffähige Prozesse strukturiert, die miteinander kommunizieren. Das Verhalten eines Gesamtsystems wird durch das Verhalten seiner Komponenten bestimmt, die mit Hilfe von Operatoren parallel komponiert werden.

Bei diesen kompositionellen Kalkülen wird die Modellierung der Parallelität durch nichtdeterministische Sequentialisierung vorgenommen (Interleaving-Modell).

Im Gegensatz dazu bietet die Petrinetz-Theorie /Reis 82/ eine direkte Beschreibung der Parallelität von Abläufen. Hier bleibt die kausale Unabhängigkeit, d.h. das parallele Eintreten von Ereignissen, im Modell erhalten ("True-Concurrency"-Modell).

Allerdings werden in der Netz-Theorie kompositionelle Aspekte kaum betrachtet. Dies beeinträchtigt die Anwendbarkeit der Netze zur Spezifikation größerer Systeme.

CCS und Petrinetze stellen formale Methoden dar, die eine Untersuchung von Spezifikationen auf interne Korrektheit erlauben. Zur Überprüfung der externen Korrektheit existieren seit kurzem Simulationswerkzeuge, die Spezifikationsmodelle ausführbar machen [Prot 88/, /CCS 88/]. Diesen Werkzeugen liegt der Formalismus der jeweiligen Beschreibungsmethode zugrunde. Durch die Ausführbarkeit sind Spezifikationen zugleich Prototypen des zu erstellenden Systems. An den Prototypen werden Untersuchungen über das Verhalten eines Spezifikationsmodells vorgenommen.

## 1.5 Ziele dieser Arbeit

Diese Arbeit befaßt sich mit der Untersuchung von Spezifikationen für verteilte Echtzeitsysteme zur Prozeßautomatisierung. Dazu werden in Kapitel 2 die typischen Eigenschaften solcher Systeme herausgearbeitet. Es wird untersucht, welche Beschreibungsmittel in einer Spezifikationssprache vorhanden sein müssen, um diese Eigenschaften darstellen zu können.

Werden Prozesse quasi-parallel implementiert, ist dies mit einer Einschränkung der möglichen Abläufe in einem System paralleler Prozesse verbunden. Deshalb muß zur Untersuchung des Synchronisations- und Zeitverhaltens einer Spezifikation das Betriebssystem der Implementierung modelliert werden.

In Kapitel 3 wird die Beschreibung von Echtzeitsystemen durch Petri-Netze, CCS und PASS kritisch betrachtet. Dabei stellt sich heraus, daß die ersten beiden Methoden mit ihrer Ausdrucksfähigkeit bzw. Analysefähigkeit nur bedingt geeignet sind.

PASS erfüllt die Voraussetzungen hinsichtlich der Ausdrucksfähigkeit; eine rechnergestützte Analyse von PASS-Modellen ist aber bisher nicht möglich, da die Semantik der Spezifikationssprache nur informell festgelegt ist.

In Kapitel 4 wird ein formales Semantik-Modell für PASS entwickelt. Die Semantik der Sprachkonstrukte wird durch Produktionen eines Graphersetzungssystems beschrieben. Ein System paralleler Prozesse wird durch einen Programmgraphen repräsentiert. Die Menge aller zulässigen gemischten Ableitungsfolgen definiert operationell die Semantik eines Prozeßsystems.

Damit liegt ein Modell vor, das die Beschreibung von Parallelität entsprechend dem True-Concurrency-Modell der Petri-Netz-Theorie und dem kompositionellen Ansatz von Methoden wie CCS vereint.

Allerdings ist mit PASS eine wesentlich größere Ausdrucksfähigkeit gegeben, da

- parallele und Quasi-parallele Prozesse beschreibbar sind;
- die Kommunikation zwischen Prozessen synchron und asynchron erfolgen kann;
- nichtdeterministische Kommunikationsanweisungen durch Prioritäten beschränkt werden können;
- Kommunikationsanweisungen konjunktiv verknüpfbar sind;
- Zeitabläufe modellierbar sind.

Kapitel 5 beschreibt einen Interpreter, der aufbauend auf den Graphproduktionen, das Verhalten eines Spezifikationsmodells durch parallele Komposition der Einzelprozesse sichtbar macht. Die parallele Komposition erfolgt unter Berücksichtigung des Betriebssystemverhaltens bei quasi-parallelen Prozessen. Modelliert werden verdrängende und nichtverdrängende prioritätengesteuerte Prozessorvergabe-strategien. Durch Zuordnung von Zeitdauern zu Aktionen der Prozesse können Zeitabläufe im Modell untersucht werden. Die zeitlichen Einflüsse des Betriebssystems durch Prozeßwechselzeiten und Aufruf des Kommunikationsdienstes werden dabei berücksichtigt.

Der Ablauf des Spezifikationsmodells wird dem Benutzer graphisch unter Verwendung von PASS-Symbolen angezeigt.

Kapitel 6 zeigt in einem Beispiel die Anwendungsmöglichkeiten des Interpreters. Dabei wird deutlich, wie unterschiedliche Realisierungen das Verhalten eines Systems paralleler Prozesse verändern.

In Kapitel 7 wird der strukturelle Aufbau des Interpreters gezeigt. Es wird die funktionelle Gliederung in Module und deren Beziehungen in Form einer Benutzt-Relation dargestellt. Die Bedienung des Interpreters durch den Benutzer wird erläutert.

In Kapitel 8 schließen Anmerkungen zur Leistungsfähigkeit und Ausblicke auf weitere Entwicklungsmöglichkeiten die Arbeit ab.

## 2. Spezifiziere ■ YO ■ Echtzeitsysteme ■

Spezifikationsmethoden sollen die Beschreibung eines zu realisierenden Systems auf abstraktem Niveau ermöglichen. Um formale Techniken zur Analyse von Spezifikationsmodellen einsetzen zu können, werden in der Regel starke Abstraktionen von Gegebenheiten einer Realisierung eingeführt (z. B. Zeitverhalten, Art der Kommunikationsmechanismen). Diese Vorgehensweise kann aber den ingenieurmäßigen Einsatz einer Spezifikationsmethode erschweren.

Deshalb sollen in diesem Kapitel die wesentlichen Anforderungen der Prozeßautomatisierung durch Echtzeitsysteme untersucht werden. In einer zu verwendenden Spezifikationsmethode und den dazugehörigen Analysemethoden müssen solche Eigenschaften einfach und unkompliziert zu beschreiben bzw. zu untersuchen sein.

### 2.1 Architektur YOD Echtzeitsystemen

Prozeßdatenverarbeitung oder Echtzeitdatenverarbeitung beschäftigt sich mit der Regelung und Steuerung von technischen Prozessen mit Hilfe eines Echtzeitsystems. Ein technischer Prozeß ist laut /DIN 66201/:

"ein Prozeß, dessen Zustandsgrößen mit technischen Mitteln gemessen, gesteuert und/oder geregelt werden können. Technische Prozesse dienen der Umformung und/oder dem Transport von Materie, Energie und/oder Information."

Technische Prozesse sind dadurch gekennzeichnet, daß gleichzeitig und nichtdeterministisch Ereignisse auftreten können, auf die das steuernde Echtzeitsystem reagieren muß. Zum Anschluß des Rechensystems an das technische System, bestehend aus mehreren Teilprozessen, sind geeignete Schnittstellen erforderlich, die die auftretenden Ereignisse erfassen bzw. Steuerbefehle an das technische System weitergeben; man spricht hier von der Instrumentierung zwischen Rechner und technischem System.

In der Regel verfügen Echtzeitsysteme ebenso über eine Schnittstelle zu einem Bediener, über die Protokolle des Prozeßablaufs oder -Zustands ausgegeben werden bzw. Bedieneingaben zur Beeinflussung des Steuervorgangs gemacht werden können.

Damit ergibt sich die in Bild 2-1 gezeigte Einbettung von Realzeitsystemen in die Umgebung.



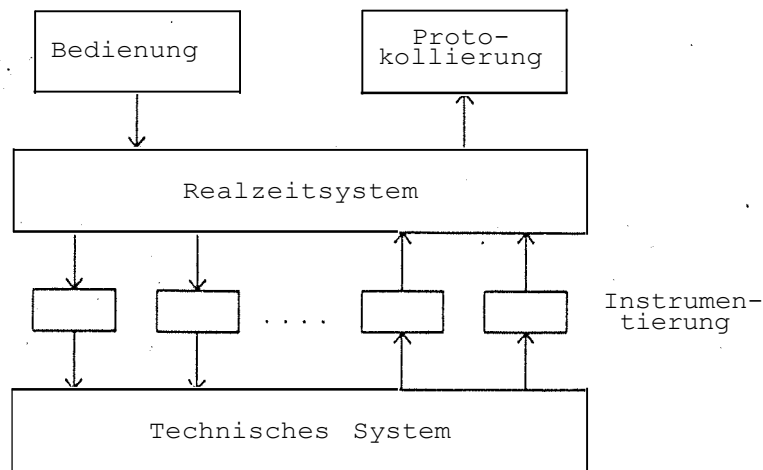


Bild 2-1 : Einbettung von Echtzeitsystemen

Verteilte Echtzeitsysteme besitzen typische Strukturen, die sich durch die Grundstruktur des technischen Systems und durch die Aufgliederung in einzelne Aufgabenbereiche ergeben. Zwischen technischem System und steuerndem Echtzeitsystem wird in /Syrb 78/ folgende Beziehung konstatiert

"Ein technisches System besteht aus verteilten, parallel arbeitenden, gekoppelten Teilsystemen. Das duale Automatisierungssystem besteht aus wenigstens der gleichen Anzahl entsprechender, paralleler, gleichartig gekoppelter Rechenprozesse. Der Rechner-und/oder Programmaufbau ist hierdurch bestimmt."

Damit wird eine Minimalgliederung des Programmsystems durch die Art und Anzahl der Teilsysteme eines technischen Systems gefordert.

Betrachtet man die verschiedenen Aufgaben eines Automatisierungssystems, ergibt sich eine Struktur gemäß den folgenden Kontrollaufgaben /Färb 84/:

- Kontrollebene 1 : direkte Kontrolle der zu einem Teilsystem gehörenden Maschinen.

Hier besteht die Anforderung, daß mehrere Ein-/Ausgabeschnittstellen mit dem technischen Prozeß gleichzeitig bedient bzw. überwacht werden müssen, da auf spontane Ereignisse im Prozeßgeschehen reagiert werden muß. Zwischen den Maschinen eines Teilsystems besteht eine zeitliche Abhängigkeit (Taktzeit, Reaktionszeit), die durch den Arbeitsablauf vorgegeben ist. Dies macht eine zeitliche und logische Synchronisation der Aktionen des Realzeitsystems notwendig.

- **Kontrollebene 2 : Kontrolle von Teilsystemen**

Auf dieser Ebene werden die Prozeßdaten, die den augenblicklichen Zustand des technischen Prozesses beschreiben, in vorgegebenen Zeitintervallen von allen Stationen eines Teilsystems abgefragt. Alarmmeldungen, die Bedeutung für ein ganzes Teilsystem haben, müssen vorrangig behandelt werden. Aktionen der Ebene 1 werden zu festgesetzten Zeitpunkten initialisiert oder synchronisiert (z. B. Schichtwechsel).

- **Kontrollebene 3 : Koordination von Teilsystemen**

Hier werden komplexe Berechnungsaufgaben durchgeführt und umfangreiche Kommunikationsprotokolle mit den Prozessen der darunterliegenden Kontrollebenen abgewickelt. Außerdem wird hier eine Anbindung an Informations- und Kontrollsysteme realisiert. Entsprechend dem Fertigungsablauf werden die einzelnen Teilsysteme synchronisiert und Fertigungswege umkonfiguriert, wenn einzelne Systeme ausfallen. Die Aufgaben sind ebenfalls zeitkritisch, da die Prozesse der Ebenen 1 und 2 rechtzeitig mit Information versorgt oder abgefragt werden müssen.

## **2.2 Eigenschaften eines Echtzeitsystems**

Nach [Nehm 84] sind Echtzeitsysteme in ihrem Aufbau durch folgende Eigenschaften geprägt

- a) **Parallelität der Schnittstelle Rechner - technisches System**

Wie bereits gesagt, können in den verschiedenen Funktionskomponenten des technischen Systems Ereignisse gleichzeitig auftreten, die vom steuernden System erfaßt werden müssen. Dies zwingt zu einer simultanen Bedienung von mehreren unabhängigen Ein-/Ausgabeeinheiten. Diese Bedienung muß auf der Ebene des Anwendungsprogramms erfolgen. Tritt zum Beispiel ein Alarm (Interrupt) ein, weil ein Instrument das Überschreiten eines Grenzwertes meldet, kann in Gegensatz zu konventionellen Rechnersystemen der Interrupt nicht vom Betriebssystem selbst bearbeitet werden. Es ist eine Programmunterbrechung und Reaktion im Anwenderprogramm erforderlich.

Daraus resultiert eine Strukturierung des Anwenderprogramms in Prozesse, die unabhängig voneinander auf Ereignisse warten bzw. Aufträge abgeben können. Diese konkurrierenden Prozesse müssen jedoch miteinander kooperieren, um verschiedene Ereignisse logisch verknüpfen zu können.

#### b) Anwendungsabhängige Rechnerkonfiguration

Im Gegensatz zum konventionellen Rechenzentrumsbetrieb wechselt mit jeder Anwendung gewöhnlich auch Umfang und Art der Rechnerkonfiguration. Die Rechnerkonfiguration kann durch folgende Merkmale bestimmt sein:

- Die räumliche Verteilung des technischen Prozesses zwingt zur entsprechenden Anpassung der Rechnerkonfiguration, um die Kommunikation mit dem technischen Prozeß kostengünstig und einfach zu gestalten. So kann z.B. durch sogenannte "Front-End"-Rechner die Meßdatenerfassung direkt vor Ort erfolgen. Lange Übertragungsleitungen von jedem einzelnen Meß- oder Stellglied entfallen.
- Zuverlässigkeitsanforderungen machen eine redundante Auslegung des Rechnersystems notwendig. Dabei können sowohl bestimmte Prozesse als auch Prozessoren mehrfach vorhanden sein. Dies ist z. B. bei besonderen Umgebungsbedingungen (Temperatur, mechanische Beanspruchung) oder besonders kritischen Anlagenteilen, bei denen das Meß- und Regelsystem wegen der Gefahr für die Umwelt nicht ausfallen darf, erforderlich.
- Realzeitanforderungen bedingen die Verteilung von Prozessen auf verschiedene Prozessoren, um innerhalb vorgegebener Intervalle die geforderte Rechenleistung erbringen zu können.

Die gewählte Rechnerkonfiguration beeinflusst damit die Architektur des Anwendungsprogramms. Damit kann eine Aufteilung des Programms in parallele Prozesse erzwungen werden, die nicht durch eine funktionale Gliederung, sondern durch technische Gegebenheiten diktiert wird.

### c) Zeitabhängigkeit der Programmabläufe

Sequentielle Programme können dann als korrekt betrachtet werden, wenn sie die gewünschte Abbildung von der Menge der zulässigen Eingabewerte in die Menge der Ausgabewerte realisieren.

Auf parallele Programme läßt sich diese Vorstellung nicht ohne weiteres übertragen. Hier spielt die Reihenfolge von Bearbeitungsschritten und die davon abhängige gegenseitige Beeinträchtigung von Schritten in parallelen Programmteilen eine Rolle. Bei Echtzeitsystemen kommt noch hinzu, daß das System nur dann korrekt arbeitet, wenn die Bearbeitungsschritte innerhalb eines vorgegeben Zeitintervalls oder zu bestimmten absoluten Zeitpunkten ausgeführt werden.

Hier müssen in der Regel folgende Bedingungen erfüllt werden, damit eine festgelegte Aufgabe korrekt durchgeführt wird:

- Zwischen zwei Bearbeitungsschritten dürfen höchstens  $x$  Zeiteinheiten liegen (z.B. zwischen Erfassen und Verarbeiten eines Prozeßinterrupts).
- Zwischen zwei Bearbeitungsschritten müssen mindestens  $y$  Zeiteinheiten liegen (z.B. zyklisches Abfragen von Zustandsgrößen des technischen Prozesses).

Die Formulierung solcher Bedingungen ist stark mit dem Begriff des Zustands gekoppelt. Durch die Zeitbedingungen wird genau die Tatsache ausgedrückt, daß ein Rechenprozeß innerhalb eines vorgegebenen Zeitintervalls einen Zustandswechsel vornehmen muß.

Die Bearbeitung von Prozeßinterrupts erfolgt beispielsweise dadurch, daß ein Prozeß im Empfangszustand auf die Meldung "Interrupt" wartet, die entsprechende Bearbeitung durchführt und dann wieder in seinen Empfangszustand zurückkehrt. Für diesen Zyklus wird bei Realzeitsystemen ein genau definiertes Zeitintervall vorgegeben, dessen Größe von der Art des technischen Prozesses abhängt. Bei zeitkritischen Prozessen kann es im Bereich von Millisekunden liegen.

Zur Erfüllung solcher harten Zeitbedingungen muß beim Entwurf von Realzeitsystemen auch das Eigenzeitverhalten des Betriebssystems, auf dem das System realisiert werden soll, in Betracht gezogen werden. Durch das Betriebssystem können starke Verfälschungen im Zeitverhalten des Anwendungsprogramm auftreten.

#### d) Feste Programmausrüstung

Betriebssysteme für Prozeßrechner unterscheiden sich wesentlich von solchen für den klassischen Rechenzentrumsbetrieb, wo eine im vorneherein unbekannte Zahl von Benutzern gleichzeitig bedient werden muß. Bei Realzeitsystemen ist die Programmausrüstung, das heißt die Menge der Prozesse, die vom Betriebssystem verwaltet werden müssen, über die Laufzeit einer Anwendung konstant.

Dies bedeutet, daß beim Entwurf von Realzeitsystemen das Betriebssystemverhalten bezüglich zeitlicher Abläufe und Vergabe von Betriebsmitteln mit eingeplant werden kann, und auch mit eingeplant werden muß, um die Anforderungen des Realzeitbetriebs zu erfüllen. Dies betrifft vor allem zeitliche Aspekte. Die feste Programmausrüstung erleichtert dabei in der Entwurfsphase von Realzeitsystemen die Einbeziehung von Bearbeitungszeiten für Betriebssystemaufrufe, wie z.B. Prozeßwechselzeiten, da diese Werte als konstant betrachtet werden können.

### 2.3 Synchronisation und Kommunikation in Realzeitsystemen

Die Automatisierungsaufgaben der beschriebenen Kontrollebenen sind meist untereinander gekoppelt, sowohl innerhalb einer Kontrollebene, als auch über Kontrollebenen hinweg. Daher können die einzelnen Rechenprozesse, die zur Realisierung von Aufgaben verwendet werden, nicht vollkommen unabhängig voneinander ablaufen. Es gibt zwischen ihnen logische und zeitliche Zusammenhänge, die eine Kooperation notwendig machen. Dazu tauschen die beteiligten Prozesse Information, d. h. Daten, sowohl untereinander als auch mit der Umwelt (Benutzer, technischer Prozeß) aus und stellen Beziehungen zwischen ihren ausführbaren Aktionen durch Synchronisationsbedingungen her. Dadurch wird der Grad an Parallelarbeit in einem verteilten System von Rechenprozessen eingeschränkt.

In /Göhn 81/ werden die Synchronisations- und Kommunikationsprobleme untersucht, die bei der Realisierung von Prozeßautomatisierungssystemen mit Hilfe paralleler Prozesse auftreten. Anhand eines Modellprozesses aus der Verfahrenstechnik sollen die Probleme erläutert werden.

#### 2.3.1 Ein Modellprozeß

Das in Bild 2-2 dargestellte technische System zeigt ein typisches Problem, wie es in der Verfahrenstechnik häufig anzutreffen ist. In einem Vorratsbehälter



sind die Stoffe Reingas und Rohgas, getrennt durch Sauerstoff (Luft) gelagert. Über Rohrleitungen werden sie in eine Mischkammer geleitet, wo zunächst beide Komponenten mit Sauerstoff verdünnt und anschließend aufgeheizt werden. Wenn in beiden Kammern ein bestimmter Druck erreicht ist, öffnet ein Stellmotor die Trennwand zwischen beiden Kammern, so daß sich die Komponenten vermischen.

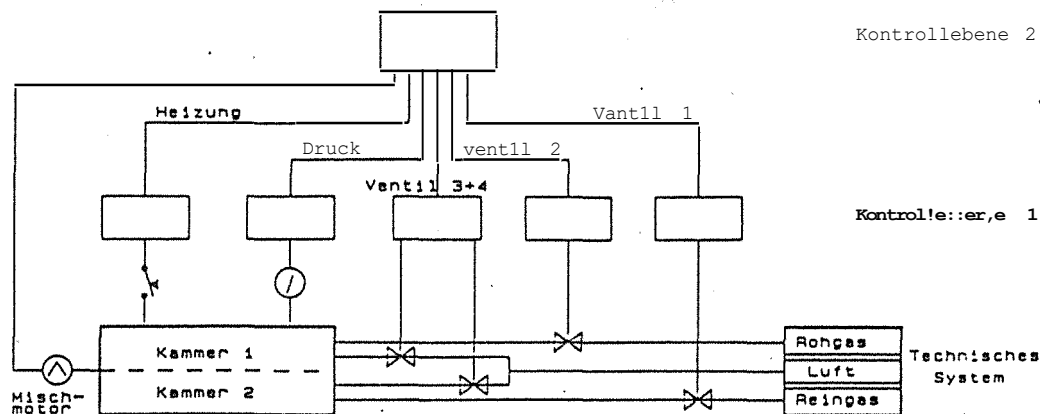


Bild 2-2 : Der technische Modell-Prozeß

Auf diesen Modellprozeß wird im folgenden Bezug genommen.

### 2.3.2 Synchronisations- und Kommunikationsprobleme in der Prozeßautomatisierung

Die auftretenden Probleme können 5 Klassen zugeordnet werden:

- Pufferung und Synchronisierung von Protokollierungsaufträgen

Bei der Protokollierung von Betriebszuständen und Störmeldungen überlagern sich Protokollaufträge aus den verschiedenen Anlagenteilen. Da sie auf einem Gerät protokolliert werden sollen, müssen sie zeitlich koordiniert werden, um Anforderungen an eine logisch zusammenhängende Darstellung zu erfüllen.

Die protokollierenden Ausgabegeräte sind relativ langsam. Deswegen muß es möglich sein, Aufträge an den Protokollprozeß zwischenzuspeichern. So können die übergabenden Prozesse unabhängig von der Bearbeitungsgeschwindigkeit und augenblicklichen Auslastung des Protokollprozesses ihre Meß- und Steuerungsaufgaben durchführen, ohne an der Ausgabeeinheit blockiert zu werden.

- konjunktiv verknüpfte Bedingungen zur Fortsetzung eines Rechenprozesses

Aktionen eines Rechenprozesses dürfen erst dann ausgeführt werden, wenn alle Vorbedingungen dazu erfüllt sind. Die Gültigkeit einer Vorbedingung ist durch das Auftreten eines Ereignisses im technischen System oder durch Erreichen eines bestimmten Zustands in einem kooperierenden Rechenprozeß gekennzeichnet.

Im Modellprozeß soll sichergestellt sein, daß die Ventile für Gas nur dann geöffnet werden, wenn auch das Ventil für Sauerstoff geöffnet wird, und umgekehrt. Die E/A-Prozesse erhalten die entsprechenden Aufträge von der Kontrollebene 2. Nur wenn beide Prozesse empfangsbereit für einen Steuerauftrag sind, gilt der Auftrag als übermittelt. Es darf nicht der Fall eintreten, daß einer der Prozesse allein beginnt den Auftrag "Ventil auf" zu bearbeiten.

Genauso kann das Voranschreiten des Rechenprozesses vom konjunktiv verknüpften Empfangen von Meldungen aus dem technischen Prozeß abhängig sein. Die Steuerung des Aufheizvorgangs in der Mischkammer beginnt dann, wenn in beiden Kammern ein bestimmter Druck erreicht ist.

- Disjunktiv verknüpfte Bedingungen zum Fortsetzen von Rechenprozessen

Zur Überwachung werden Meßeinheiten eingesetzt, die den Verlauf des technischen Prozesses kontrollieren. Die überwachten Teile können oft in Klassen eingeteilt werden, die die gleichen Aufgaben haben. Deswegen wird die Meßwerterfassung und -reaktion in einem Rechenprozeß zusammengefaßt. Der Rechenprozeß muß zur Bearbeitung nur wissen, von welcher Stelle die Meldung kommt; die Art der Meldungsbearbeitung ist immer gleich.

In obigem Beispiel seien in jedem Zulaufrohr Meßinstrumente angebracht, die den Druck in der Leitung überwachen. Bei Überschreiten eines Grenzwertes wird der Zufluß gedrosselt. Der steuernde Rechenprozeß wartet auf Eintreffen einer Meldung von einer Leitung und gibt dann an das zuständige Ventil den Befehl zum Drosseln. Er befindet sich also in einem Wartezustand, in dem er auf eine Meldung von einer vordefinierten Menge von Absendern wartet.

- Vorrangige Behandlung von Ereignissen.

Wartet ein Rechenprozeß auf das Eintreten eines Ereignisses aus einer Menge vorgegebener Alternativen und können mehrere Ereignisse gleichzeitig ein-

treffen, muß es möglich sein, eine Auswahl aus diesen Ereignissen zu treffen. Dazu wird alternativen Ereignissen eine Priorität zugeordnet. Die Bearbeitung erfolgt dann prioritätengesteuert. Mit diesem Mechanismus können wir in unserem Beispiel die Leitung "Reingas", die eine besonders gefährliche Komponente transportiert, bevorzugt behandeln, wenn die Drucküberschreitung gemeldet wird.

#### - Zeitliche Überwachung des Eintritts von Ereignissen

Durch die zeitliche Überwachung des Eintritts von Ereignissen wird bei der Automatisierung von technischen Prozessen das Auftreten von Störungen erfaßt. Nach der Ausgabe eines Stellsignals wird auf das Ende einer angestoßenen Operation des technischen Prozesses, angezeigt durch eine Rückmeldung, gewartet. Wenn die Rückmeldung nach einer bestimmten Zeitspanne nicht eintrifft, wird dies als Störung im technischen Prozeß interpretiert.

Der Stellmotor zum Öffnen der Mischkammer muß nach einer bestimmten Zeit eine Endeposition erreichen. Erhält der überwachende Prozeß keine Rückmeldung, wird dies als Störmeldung "Defekt im Antrieb" protokolliert.

Aus diesen Beispielen kann man für die Zusammenarbeit von technischen Prozessen und Rechenprozessen folgende Forderungen ableiten:

- 1) Ein Partner muß Meldungen auch dann senden können, wenn der Kommunikationspartner gerade aktiv ist und nicht auf das Eintreffen der Meldung wartet. Meldungen müssen bis zum Empfang gespeichert werden können.
- 2) Zwischen Meldungen müssen sowohl konjunktive als auch disjunktive Verknüpfungen erlaubt sein. Die Fortsetzung eines Rechenprozesses ist davon abhängig, daß alle gewünschten Meldungen gesendet oder empfangen werden können, bzw. daß eine bestimmte, nach Absender identifizierbare, Meldung aus einer vorgegebenen Menge empfangen werden kann.
- 3) Zwischen alternativen Ereignissen muß die Auswahl prioritätengesteuert erfolgen können, wenn mehrere Ereignisse gleichzeitig eintreten.
- 4) Das Warten auf Eintreten eines Ereignisses muß zeitüberwacht erfolgen können. Dies entspricht im Prinzip dem alternativen Warten auf das Eintreten des Ereignisses bzw. auf Eintreffen der Meldung "Zeit abgelaufen".

Auf Spezifikationsebene ist ein entsprechender Synchronisations- und Kommunikationsmechanismus notwendig, um die Kooperation von Prozessen in Realzeitsystemen gemäß diesen Anforderungen darstellen zu können.

### **2.3.J Das Botschaftskonzept zur Synchronisation und Kommunikation verteilter Prozesse**

Die Synchronisation in einem System von Prozessen ist dafür verantwortlich, daß eine von einem Prozeß auszuführende Berechnung erst dann ausgeführt wird, wenn bestimmte Vorbedingungen erfüllt sind.

Botschaftsorientierte Kommunikationsmechanismen sind dann unerlässlich, wenn Prozesse in verschiedenen Rechnern lokalisiert sind und eine Kommunikation über gemeinsamen Speicher ausgeschlossen ist. Für Anwendungen zur Prozeßautomatisierung ist das Botschaftskonzept besonders geeignet. Damit kann nicht nur die Kommunikation zwischen Rechenprozessen, zwischen Rechenprozessen und Benutzer bzw. Rechenprozessen und technischem System einheitlich dargestellt werden /Flei 84/.

Wegen der Bedeutung für die Spezifikation und Implementation von verteilten Echtzeitsystemen soll im folgenden ein Basismodell für botschaftsorientierte Kommunikation zwischen verteilten Prozessen skizziert werden.

Man unterscheidet Botschaftskonzepte nach:

- Art der Adressierung von Absender und Empfänger einer Botschaft;
- Speichermöglichkeit für Botschaften .

Dies soll anhand des Portmodells zur Kommunikation (Bild 2-3) erläutert werden.

Wir unterscheiden zwischen dem Transportsystem und Instanzen, die über Sendebzw. Empfangsports verfügen. Kommunikationsstrukturen werden unabhängig von den Kommunikationspartnern durch logische Verbindungen zwischen Ports beschrieben. Eine Instanz kann mehrere Prozesse enthalten. Ein Prozeß kommuniziert nicht direkt mit seinem Partner, sondern gibt Aufträge, das heißt Botschaften, an einen Port. Für das Transportsystem werden Ports damit zu Quellen und Senken von Botschaften. Mehrere Prozesse können Botschaften an einen Port

geben bzw. von einem Port erhalten. Für einen sendenden Prozeß ist grundsätzlich nicht bestimmbar, welcher Prozeß eine Botschaft erhalten soll. Er adressiert nur den gewünschten Empfangsport.

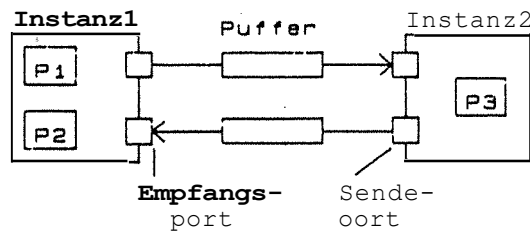


Bild 2-3 : Das Portmodell zur Prozeßkommunikation /Nehm 85/

Das Transportmedium wird in Form von gerichteten Kanälen beschrieben, die von einem Sendeport zum Empfangsport führen. Haben Kanäle speichernde Wirkung, d. h. können Botschaften im Kanal gepuffert werden, spricht man von asynchroner Kommunikation. Bei synchroner Kommunikation besitzt der Kanal keine puffernde Wirkung. Der sendende Prozeß wird so lange blockiert, bis eine Botschaft am Empfangsport entgegengenommen wird (Rendezvous).

Da für viele Anwendungen in der Prozeßautomatisierung aus Sicherheitsgründen eine direkte Beziehung zwischen Sender und Empfänger hergestellt werden muß, ist der Spezialfall der direkten Adressierung von Bedeutung. Dabei werden Kommunikationspartner, also Prozesse, explizit über einen Namen angesprochen.

Dies kann so modelliert werden, daß jeder Prozeß über einen eigenen Sende- bzw. Empfangsport verfügt. Damit werden Port und Prozeß identisch. Die puffernde Wirkung des Kanals kann damit als Eigenschaft des Prozesses gesehen werden. Den Puffer bezeichnen wir dann als Wartebereich des Prozesses. Hat der Wartebereich die Größe Null, ist synchrone Kommunikation gefordert.

#### 2.3.4 Logische Verknüpfung von Botschaftsoperationen

Zur Beschreibung von Problemen, wie alternatives Warten auf Botschaften oder zeitüberwachte Kommunikation werden in /Dijk 75/ nichtdeterministische Kontrollanweisungen eingeführt. Dazu wird der Begriff der "bewachten Anweisung" (guarded region) definiert. Sie enthält mehrere sogenannte "Wächter" (guards). Ein Wächter formuliert eine Vorbedingung, die gültig sein muß, damit die zugehörige Anweisungsfolge ausgeführt werden kann. Mehrere Wächter können durch

eine "Oder"-Bedingung verknüpft werden. Ein Prozeß, der eine bewachte Anweisung bearbeitet, prüft, ob eine der Vorbedingungen wahr ist. Ist keine Bedingung erfüllt, wird der Prozeß blockiert; ansonsten wird die zugehörige Anweisungsfolge bearbeitet.

Während /Dijk 75/ nur boolesche Ausdrücke als Vorbedingungen zuläßt, sind in /Hoar 78/ auch Eingabeanweisungen zulässig. Damit kann das alternative Warten auf Botschaften modelliert werden.

Dieses Konzept wurde ausgeweitet, so daß sowohl Eingabe-, als auch Ausgabeanweisungen in der Vorbedingung enthalten sein können. Vorbedingungen können hier auch konjunktiv verknüpft sein /FIHo 83/, /Kumm 83/.

Damit steht ein Konstrukt zur Verfügung, mit dem beliebige logische Verknüpfungen von Botschaftsoperationen beschrieben werden können.

Mit Hilfe einer Zeitüberwachungs-Bedingung wird ein Prozeß, der keine der Vorbedingungen einer nichtdeterministischen Kontrollanweisung erfüllt findet, nach Ende der Zeitüberwachung deblockiert. Ohne Timeout-Angabe bleibt der Prozeß blockiert, bis alle Bedingungen eines Wächters wahr werden.

Einem Wächter kann eine Priorität zugeordnet werden. Damit ist es möglich, aus mehreren Wächtern, deren Bedingungen erfüllt sind, zur Weiterbearbeitung genau einen auszusuchen. Besitzen mehrere Wächter die gleiche Priorität, erfolgt die Auswahl nichtdeterministisch.

Mit diesen Ausdrucksmöglichkeiten können alle in Kapitel 2.3.2 formulierten Anforderungen an die Mächtigkeit von Synchronisations- und Kommunikationskonzepten für parallele Prozesse zur Prozeßautomatisierung erfüllt werden.

Beispiel:

Die Steuerung des Modellprozesses aus Kap 2.3.1 läßt sich mit Hilfe von Guards einfach formulieren:

GUARDED REGION

GUARD TRANSMIT ventil\_auf TO Ventil\_\_1

& TRANSMIT ventil\_auf TO ventil\_2; REACT gas\_aus;

TIMEOUT AFTER 1 sec REACT "Alarm";

GUARDEND;

In dieser Guarded-Region (syntaktisch angelehnt an die Realisierung in "verteilten PEARL" /FIHo 83/) wird gleichzeitig die Botschaft "Ventil\_\_auf" an die Prozesse

Yentill und Yentil2 gesendet. Nur wenn beide Prozesse die Botschaft entgegennehmen, ist die Wächterbedingung erfüllt. Kann die Kommunikation nicht innerhalb der Überwachungszeit von 1 sec abgeschlossen werden, erfolgt eine Alarmmeldung.

## 2.4 Prozesse In Realzeitsystemen

Prozesse stellen das wichtigste Strukturierungsmittel in Realzeitsystemen dar /Levi 81/. Das Gesamtsystem wird in mehrere selbständige Prozesse zerlegt, die einzelne Teilaufgaben bearbeiten.

Ein Prozeß stellt eine logische Einheit dar, deren Funktion ohne Berücksichtigung der Umgebung, d. h. unabhängig von anderen Prozessen, definiert ist. Damit wird eine Abstraktion vom restlichen Geschehen im System erreicht. Dies wird als horizontale Abstraktion bezeichnet /Kera 82/.

### 2.4.1 Der Prozeßbegriff

Zur Beschreibung eines Prozesses werden die Begriffe Zustand und Ereignis verwendet. Prozesse ändern sich in diskreten Schritten, das heißt sie gehen von einem Zustand in den nächsten über; die Zustandsänderung wird durch ein Ereignis bewirkt.

Damit können wir einen Prozeß folgendermaßen als Transitionssystem definieren.

#### Definition 2-1 : Prozeß

*Ein Prozeß ist ein Transitionssystem  $P = (Q, q_0, Q_f, E, \delta)$  mit*

- $Q$  Menge von Zuständen,*
- $q_0 \in Q$  Anfangszustand,*
- $Q_f \subseteq Q$  Menge der Endzustände mit,*
- $E$  Menge von Ereignissen und*
- $\delta: Q \times E \rightarrow Q$  Zustandsübergangsfunktion, die den Folgezustand nach Eintreten eines Ereignisses bestimmt.*

In der Spezifikation eines Rechenprozesses werden die möglichen Ereignisse partiell geordnet. Ereignisse können kausal oder zeitlich geordnet sein: ein Ereignis  $b$  darf erst eintreten, wenn ein Ereignis  $a$  zuvor eingetreten ist;

Ereignisse können aber auch ungeordnet sein: die Ereignisse a und b dürfen alternativ eintreten oder sind gleichzeitig möglich.

Das Verhalten, das ein sequentieller Prozeß beim Ablauf zeigt, kann man mit Hilfe der Grundelemente Ereignis und Zustand auf zwei Arten definieren: zustandsorientiert oder ereignisorientiert.

Das Verhalten eines Prozesses ergibt sich aus der Menge aller möglichen Zustands- bzw. Ereignisfolgen.

Je nach Beschreibungsziel ist eine der beiden Möglichkeiten von Interesse:

- Will man die Synchronisation kooperierender Prozesse beschreiben, ist die ereignisorientierte Sichtweise angebracht: Synchronisation soll dann erfolgen, wenn zwei zusammengehörige Ereignisse, a in Prozeß P1 und ä in Prozeß P2, auftreten.
- Zur Beschreibung zeitlicher Vorgänge wird die zustandsorientierte Sichtweise verwendet: ein Prozeß wartet mindestens x Zeiteinheiten im Zustand q auf das Eintreffen eines Ereignisses (Timeout-Bedingung) oder die Bearbeitung eines Ereignisses, d. h. der Wechsel von Zustand  $q_i$  zum Zustand  $q_{i+1}$  benötigt maximal x Zeiteinheiten.

Weiten wir diese Definition auf ein System paralleler Prozesse aus, so erhalten wir als Semantik eines Prozeßsystems:

Das Verhalten eines Prozeßsystems ist gegeben durch die parallele Ausführung aller zulässigen Ereignis- bzw. Zustandsfolgen der beteiligten Prozesse.

Bei der Analyse von Spezifikationen eines Prozeßsystems interessieren wir uns für einen Ausschnitt aus der Menge der parallel ausführbaren Ereignisfolgen:

- durch Synchronisationsbeziehungen ist nur ein Teil der Kombinationen zulässig;
- durch Zeitbedingungen werden vom Ablauf her unabhängige Ereignisse in eine lineare Ordnung gezwungen;
- durch den Ablaufplan des Betriebssystems wird die Menge der möglichen Zustands- bzw. Ereignisfolgen eingeschränkt (s. Kap. 2.4.2).



Vom Entwerfer eines Prozeßsystems werden nur die Synchronisationsbedingungen explizit angegeben. Damit sollen gerade Abhängigkeiten zwischen Ereignissen verschiedener Prozesse ausgedrückt werden.

Die Zeitbedingungen werden zum einen statisch für einen bestimmten Zustand in der Spezifikation angegeben (Überwachungszeiten für Kommunikation). zum anderen gelten sie dynamisch erst bei der Ausführung von Aktionsfolgen eines implementierten Prozeßsystems. Die Korrektheit einer Spezifikation eines Realzeitsystems ist aber davon abhängig, daß spezifizierte Aktionen innerhalb vorgegebener Zeiten ausführbar sind. Deshalb muß eine Abschätzung des Prozeßverhaltens bereits auf Spezifikationsebene möglich sein.

Durch die Art der Realisierung einer Spezifikation entstehen sowohl zusätzliche Zeitabhängigkeiten als auch logische Abhängigkeiten zwischen Ereignissen paralleler Prozesse. Eine unabhängige Realisierung der Zustandsüberführung eines jeden Prozesses ist nur dann möglich, wenn die Prozesse jeweils über einen eigenen Prozessor verfügen. Wird ein Prozeßsystem mit Hilfe eines Monoprozessors implementiert, führt das Betriebssystem des Prozessors eine zusätzliche Ordnung auf unabhängigen Ereignissen ein.

Diese Ordnung interessiert bei der Analyse einer Spezifikation dann, wenn wir Aussagen über die Erreichbarkeit von Zuständen machen wollen. In einer Implementation ist gegenüber der Spezifikation nur ein Teil des möglichen Zustandsraumes realisierbar. Im nächsten Kapitel soll gezeigt werden, welche Bedingungen gegeben sein müssen, damit das Verhalten einer Implementation auf Spezifikationsebene vorhersagbar ist.

#### **2.4.2 Ablaufplan**

In einer Monoprozessorimplementierung wird durch das Betriebssystem eine Entscheidung getroffen, welcher Prozeß als nächster bearbeitet werden kann. Damit wird implizit auch eine Auswahl aus der Menge der aktuell ausführbaren Berechnungen durchgeführt und somit eine totale Ordnung zwischen parallelen, ungeordneten Berechnungen hergestellt. Das Betriebssystem führt mit Hilfe eines deterministischen Algorithmus die Auswahl durch. Dieser Algorithmus wird Ablaufplan genannt [Hofm 84].

Typisch bei Echtzeitanwendungen ist ein Ablaufplan, der die Entscheidung, welchem Prozeß als nächstem der Prozessor zugeteilt wird, nach statischen

Prioritäten trifft. Prozesse besitzen über ihre Laufzeit eine feste Priorität. Ausgewählt wird aus der Menge der Prozesse, die ein Ereignis bearbeiten können, also lauffähig sind, derjenige, der die höchste Priorität besitzt. Damit können Prozess! bevorzugt werden, die wichtige Aufgaben wahrnehmen, wenn gleichzeitig Berechnungen bei mehreren Prozessen möglich sind.

Die Art der linearen Ordnung wird von der Strategie bestimmt, die angibt, wann der Auswahlalgorithmus aufgerufen wird. Zur Bestimmung des Zeitpunktes der Prozessorvergabe werden Prozesse in einzelne Teilaufträge zerlegt, die höchstens am Anfang und am Ende synchronisierende Aktionen enthalten. Diese Teilaufträge, die als selbständige Prozesse betrachtet werden können, haben die Eigenschaft, daß sie, falls ihre Bearbeitung begonnen wird, auch vollständig bearbeitet werden. Da nur das spezifizierte Kommunikationsverhalten berücksichtigt wird, seien Unterbrechungen (Interrupts) ausgeschlossen.

Zur Prozessorvergabe werden bei Betriebssystemen für Prozeßrechner im wesentlichen zwei unterschiedliche Strategien angewandt:

- **verdrängende Strategie**

Nach Erledigung jeden Teilauftrags, also dann, wenn wieder eine Synchronisationsoperation ansteht, wird der aktuelle Prozeß in eine Warteschlange von lauffähigen Prozessen eingekettet und muß mit allen anderen Prozessen in der Warteschlange um den Prozessor konkurrieren.

- **nichtverdrängende Strategie**

Der Prozeß bearbeitet wie oben einen Teilauftrag. An dessen Ende wird vom Betriebssystem die Nichtblockierungsbedingung der Synchronisationsoperation am Ende geprüft. Im Unterschied zu verdrängenden Strategien behält der Prozeß jedoch den Prozessor, wenn die Prüfung ergibt, daß die Nichtblockierungsbedingung erfüllt ist. Er muß also nicht mit anderen Prozessen konkurrieren. Erst wenn der Prozeß blockiert werden muß, wird die Prozessorvergabe neu geregelt.

Realisiert wird die Prozessorvergabe durch den sogenannten Prozeßumschalter. Er übt Einfluß auf das zeitliche Verhalten von Prozessen aus. Die Prozeßwechselzeiten können das Zeitverhalten der implementierten Prozesse wesentlich verändern.

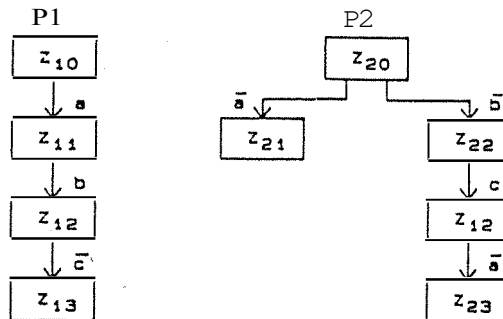
Die Art der Prozessorvergabestrategie beschränkt die realisierbare Folge von Berechnungsschritten der einzelnen Prozesse dadurch, daß sie nur bestimmte total geordnete Folgen zuläßt. Dies soll an folgendem Beispiel demonstriert werden:

Gegeben seien zwei Prozesse P1 und P2:

P1 sei definiert durch  $O_1 = (z_{10}, z_{11}, z_{12}, z_{13})$  mit  $Q_{f0} = z_{10}$ ,

P2 sei definiert durch  $O_2 = (z_{20}, z_{21}, z_{22}, z_{23})$ , mit  $Q_{20} = z_{20}$ ,

die Zustandsüberföhrungsfunktion ist gegeben durch folgendes Bild:



Es gibt zwei Sorten von Ereignissen: Sendeereignisse, die eine Botschaft an den Partnerprozeß weitergeben, z.B. {a, b}, und Empfangsereignisse, z.B. {ā, b̄}, die eine Botschaft von einem Partner entgegennehmen.

Die Kommunikation zwischen beiden Prozessen sei asynchron, d. h. jeder Prozeß kann eine Botschaft senden, solange Platz im zugehörigen Kanal des Empfangsprozesses ist.

Laufen beide Prozesse unabhängig voneinander auf getrennten Prozessoren ab, so sind die Ereignisse nur durch eine Kausalitätsbedingung geordnet:

Ein Ereignis kann erst empfangen werden, nachdem es gesendet wurde. Damit können wir für dieses Beispiel folgende Ereignisfolgen angeben:

{a, ā, b, c}, {a, b, b, c, c, ā}, {a, b, ā, c}, {a, b, c, ā}.

Der Prozeß P2 hat also im Zustand  $z_{20}$  nur dann die Möglichkeit, zwischen den Ereignissen ā und b auszuwählen, wenn Prozeß P1 beide Botschaften angeboten hat. Wird die Auswahl schon dann getroffen, wenn P1 nur a anbietet, kann der Zweig mit b, c nie durchlaufen werden.

Genau diese Situation tritt ein, wenn beide Prozesse auf einem Monoprozessor implementiert sind und dessen Ablaufplan nach einer verdrängenden Strategie arbeitet. Aufgrund der Kausalitätsbedingungen wird zunächst P2 immer im Zustand  $z_{20}$  zunächst blockiert. Prozeß P1 kann a senden; bei einer verdrängenden Strategie wird nach der Sendeoperation die Prozessorvergabe neu geregelt. P2

wird nach Überprüfen seiner Nichtblockierungsbedingung lauffähig gesetzt, da eine der erwarteten Nachrichten vorliegt. Somit konkurrieren jetzt beide Prozesse um den Prozessor. Bei gleicher Priorität und gerechter Prozessorvergabe-strategie oder bei höherer Priorität wird Prozeß P2 fortgesetzt und läuft somit in den Zweig  $\alpha$ . Die zweite Alternative wird nie durchlaufen.

Um auf Spezifikationsebene das Verhalten des implementierten Prozeßsystems vorhersagen zu können, müssen folgende Aussagen gelten:

**Definition 2-2 : Prozeßtreu /nach Hofm 84/**

Eine Implementation heißt prozeßtreu, wenn sie die Prozeßstruktur der Spezifikation nicht verändert. Aktionen, die in der Spezifikation im gleichen Prozeß enthalten sind, müssen auch in der Implementierung gleichen Prozeß enthalten sein.

**Definition 2-3 : Behinderungsfrei /nach Hofm 84/**

Eine Implementierung heißt behinderungsfrei, wenn für alle Aktionsfolgen einer Implementierung gilt, daß sie durch eine Aktion fortsetzbar sind, wenn die Spezifikation dies für die entsprechende Aktionsfolge fordert.

Für prozeßtreue und behinderungsfreie Implementationen können wir aus der Spezifikation entscheiden, ob ein Prozeß der Implementation lauffähig ist.

In /Krag 86/ wird ein Transformationswerkzeug beschrieben, das bei der Umsetzung einer PASS-Spezifikation /Flei 83/ in ablauffähigen Code, genau diese Anforderungen der Prozeßtreue und Behinderungsfreiheit erfüllt.

Dort wird die spezifizierte Kommunikation in einem System paralleler Prozesse automatisch in Kommunikationsanweisungen der Programmiersprache "verteiltes PEARL" /FIHo 83/ umgesetzt. Dabei werden auch komplexe Konstrukte, wie die nichtdeterministischen Kontrollanweisungen behinderungsfrei transformiert. Die Prozeßstruktur der Spezifikation wird auf PEARL-Tasks abgebildet.

Unter solchen Voraussetzungen ist es also sinnvoll, das Verhalten einer Implementation auf Spezifikationsebene zu untersuchen. Durch diese Untersuchung kann man Aussagen über das spätere Verhalten der spezifizierten Implementation auch unter Betriebsaspekten machen, wie z. B. Aufteilung eines Prozeßsystems in parallele und quasiparallele Prozesse, die auf einem Monoprozessor bearbeitet werden.

## 2.5 Zusammenfassung

Zur Beschreibung von Programmen zur Prozeßautomatisierung ist der Begriff des Prozesses als Strukturierungsmittel von überragender Bedeutung. Prozesse können zustands- oder ereignisorientiert beschrieben werden. Um Zeitbedingungen, wie sie für Realzeitsysteme typisch sind, ausdrücken zu können, benötigen wir eine zustandsorientierte Sichtweise von Prozessen.

Zur Kommunikation und Synchronisation verwenden Prozesse Botschaften. Botschaften erlauben die einheitliche Modellierung der Kommunikationsbeziehungen zwischen allen Komponenten eines Realzeitsystems. Die Prozeßautomatisierung erfordert Beschreibungsmöglichkeiten zur Darstellung von konjunktiv und disjunktiv verknüpftem Botschaftsaustausch. Außerdem muß eine prioritätengesteuerte Auswahl aus mehreren alternativ angebotenen Botschaften möglich sein. Gepufferter Botschaftsaustausch ist notwendig, um Wartezeiten von zeitkritischen Prozessen bei der Übergabe von Meldungen an "langsame" Prozesse zu vermeiden.

Die Blockierung an Botschaftsoperationen soll zeitüberwacht zu erfolgen, um Ausnahmesituationen, wie z.B. Ausfälle von Teilsystemen, erkennen zu können.

Diese Eigenschaften eines Echtzeitsystems müssen mit Sprachmitteln einer Spezifikationssprache beschreibbar sein. Um zeitliches und logisches Verhalten eines implementierten Prozeßsystems auf Spezifikationsebene bestimmen zu können, muß eine prozeßtreue und behinderungsfreie Implementierung gegeben sein.

### 3. Beschreibung verteilter Systeme

Zur Beschreibung von Systemen paralleler Prozesse existieren eine Reihe von Methoden, die sich in ihrer Ausdrucksfähigkeit und Analysefähigkeit stark unterscheiden.

Die Ausdrucksfähigkeit bezieht sich auf die Möglichkeit, ein System ausreichend detailliert und einfach beschreiben zu können. Unter Analysefähigkeit ist die Eigenschaft zu verstehen, ein System auf gewünschtes Verhalten hin zu untersuchen und Aussagen über Synchronisations- und Kommunikationsverhalten der Prozesse machen zu können. In der Analyse muß die interne und externe Korrektheit der Spezifikation gezeigt werden können.

In der Praxis werden sowohl informelle als auch formale Beschreibungstechniken verwendet. Informelle Techniken, die meist eine Gliederungsstruktur vorgeben und die Beschreibung in freiem Text zulassen, scheiden aus unserer Betrachtung aus, da sie kaum auf gewünschte Eigenschaften hin zu untersuchen sind.

Formale Techniken beruhen meist auf einer der folgenden Beschreibungsmethoden:

- Beschreibung durch (erweiterte) Petri-Netze;
- Beschreibung durch algebraische Kalküle (z. B. CCS, CSP);
- Beschreibung durch erweiterte endliche Zustandsautomaten (z. B. PASS).

In diesem Kapitel werden Petrinetze, CCS und PASS auf ihre Anwendungsfähigkeit zur Beschreibung von Problemen der Prozeßautomatisierung untersucht.

#### 3.1 Petri-Netze

Grundlage der Netz-Theorie bildet die Arbeit von C.A. Petri [Petri 62]. In ihr wurde erstmalig die Idee formuliert, ein System zu untersuchen, indem man temporale und kausale Beziehungen zwischen Ereignissen mit Hilfe graphentheoretischer und algebraischer Methoden analysiert, um Eigenschaften des modellierten Systems abzuleiten.

Charakteristisch für Petri-Netze sind folgende Punkte /Reis 82/:

- Kausale Abhängigkeiten innerhalb einer Menge von Ereignissen können explizit dargestellt werden. Voneinander unabhängige Ereignisse werden nicht auf eine Zeitachse projiziert. Für sie wird die Relation der Nebenläufigkeit oder "Concurrency" eingeführt.
- Systeme können auf verschiedenen Abstraktionsebenen dargestellt werden, ohne die Beschreibungssprache wechseln zu müssen.
- Netzdarstellungen ermöglichen es, Systemeigenschaften nachzuweisen und Korrektheitsbeweise zu führen.

### 3.1.1 Grundbegriffe der Netztheorie

Ein Petri-Netz ist ein endlicher, gerichteter Graph mit zwei verschiedenen Arten von Knoten, Stellen (S) und Transitionen (T). Stellen dienen der Darstellung von Elementarzuständen, mit Transitionen werden Elementarereignisse beschrieben.

**Definition 3-1 : Netz /Reis82/**

*Ein Tripel  $N ::= (S, T, F)$  heißt Netz, falls gilt*

- *S und T sind disjunkte Mengen von Knoten.*
- *$F \subseteq (S \times T) \cup (T \times S)$  ist eine zweistellige Relation, die Flußrelation von N*
- *für  $x \in S \cup T$  heißt  $x^\bullet = \{y \mid y F x\}$  der Vorbereich und  ${}^\bullet x = \{y \mid y F x\}$  der Nachbereich von x*

Graphisch werden Stellen durch Kreise und Transitionen als Kästchen oder Balken dargestellt. Pfeile führen von Stellen zu Transitionen oder von Transitionen zu Stellen.

Mit Hilfe von Stellen, Transitionen und den Beziehungen zwischen beiden Knotenarten, gegeben durch die Flußrelation, kann der Aufbau von Systemen beschrieben werden.

Im einfachsten Fall wird ein System charakterisiert durch eine Menge von Ereignissen und eine Menge von Bedingungen, deren Gültigkeit Voraussetzung für das Eintreten der Ereignisse ist. Dies wird mit Bedingungs-/Ereignisnetzen (B/E-Netze) modelliert. Bedingungen werden durch Stellen, Ereignisse durch Transitionen repräsentiert. Die Tatsache, daß eine Bedingung erfüllt ist, wird durch eine Marke in der entsprechenden Stelle ausgedrückt.

### Definition 3-2 : Markierung

Eine Markierung  $m$  ist eine Abbildung  $m : S \rightarrow \{0,1\}$

Bedingungs-/Ereignis-Netze sind die detailliertesten Darstellungsebenen markierter Netze. Sie sind dadurch gekennzeichnet, daß jede Stelle höchstens eine Marke trägt.

Die Stellen-/Transitionen-Netze (S/T-Netze) sind eine Verallgemeinerung. Hier dürfen Stellen auch mehr als eine Marke tragen. Diese Netze sind insbesondere zur Formulierung von Blockierungsproblemen geeignet. Ein Anwendungsbeispiel ist die Modellierung eines Erzeuger/Verbrauchersystems mit Pufferkapazität. Dabei kann eine Stelle, die den Puffer darstellt, entsprechend der Pufferkapazität eine endliche Anzahl von Marken aufnehmen. Eine Markierung ist in diesem Fall eine Abbildung  $m : S \rightarrow \mathbb{N}_0$ .

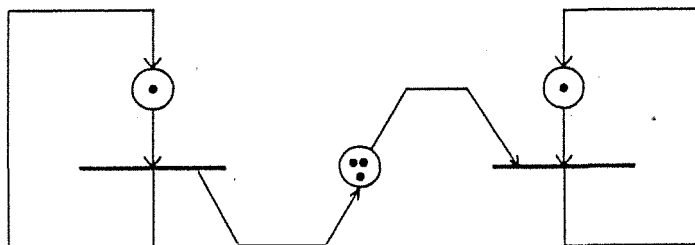


Bild. 3-1 : Erzeuger/Verbraucher System mit Puffer

Solche höheren Netze können durch B/E-Netze implementiert werden. Dazu wird jede Stelle mit Kapazität  $k = n$  durch eine Sequenz von  $n$  Stellen mit Kapazität  $k = 1$  dargestellt (Star 80).

Eine Markierung zeigt für eine bestimmte Situation im System, welche Bedingungen gültig sind und welche nicht. Die Menge der in einer Situation gültigen Bedingungen wird Fall genannt.



**Definition 3-3 : Fall**

Sei  $N : \{B, E; F\}$  ein B/E-Netz.

- Ein Fall  $c$  ist eine Teilmenge der Bedingungsmenge  $B$   
 $c \subseteq B$

Sei  $e \in E$  und  $c \subseteq B$

-  $e$  heißt aktiviert unter  $c$ , falls gilt:  $c \subseteq c_e$ ,  $c_e \subseteq B$ ,  $c$

Sei  $e \in E$ ,  $c \subseteq B$  und  $e$   $c$ -aktiviert.

- Der Folgefall  $c'$  von  $c$  unter  $e$  ist definiert durch

$$c' = (c \setminus c_e) \cup e_o$$

Ein Ereignis  $e$  kann in einem Fall eintreten, wenn die Vorbedingungen von  $e$  erfüllt sind, d.h. die Stellen aus dem Vorbereich der Transition  $e$  eine Marke tragen. Die Nachbedingungen, Stellen des Nachbereichs von  $e$ , dürfen nicht erfüllt sein.

Das "Arbeiten" eines Petri-Netzes wird entsprechend einer Schaltregel definiert. Durch das Schalten einer  $c$ -aktivierten Transition wird eine Markierung  $m$  nach  $m'$  überführt:  $m \rightarrow m'$ . Von jeder Eingangsstelle  $c \in e_o$  wird eine Marke weggenommen und jeder Ausgangsstelle  $c' \in e_o$  wird eine Marke hinzugefügt.

**Definition 3-4 : Schaltregel**

Es gilt  $m \rightarrow m'$  wenn

$$m'(c) = \begin{cases} m(c) - 1 & \text{falls } c \in e_o \\ m(c) + 1 & \text{falls } c \in e_i \\ m(c) & \text{sonst.} \end{cases}$$

Die Elemente der Menge  $\{m \mid m \rightarrow m'\}$  nennt man die Nachfolgemarkierungen von  $m$ .  $\rightarrow^*$  bezeichnet den reflexiven und transitiven Abschluß von  $\rightarrow$ .

In einem Fall  $c$  können mehrere Transitionen aktiviert sein. Das Schalten aller in  $c$  aktivierten Transitionen entsprechend der oben definierten Schaltregel wird zu einem Schritt zusammengefaßt.

### Definition 3-5 : Schritt

Seien  $c$  und  $c'$  Fälle eines B/E-Netzes  $N = (B, E; F)$  und sei  $G \subseteq E$  eine Menge von Ereignissen.

$G$  heißt Schritt von  $c$  nach  $c'$  falls gilt:

- $\forall e \in G : e$  ist  $c$ -aktiviert.
- $e_1 \cdot e_2 \in G \Rightarrow e_1 \parallel e_2 = \emptyset = e_1 \cdot e_2$ .  
(Ereignisse aus  $G$  haben paarweise disjunkte Vor- und Nachbereiche).
- $c' = (c \parallel G) \vee G$ .

Nach dem Schritt  $G$  sind alle Vorbedingungen der Ereignisse in  $G$  nicht mehr erfüllt und alle Nachbedingungen erfüllt.

Das dynamische Verhalten eines Systems wird beschrieben durch die Schritte, die in einem Netz ausführbar sind. Ein Schritt enthält Ereignisse, die unabhängig voneinander eintreten können. Die Ereignisse unterliegen keiner zeitlichen Ordnung; sie treten in beliebiger Reihenfolge oder gleichzeitig ein.

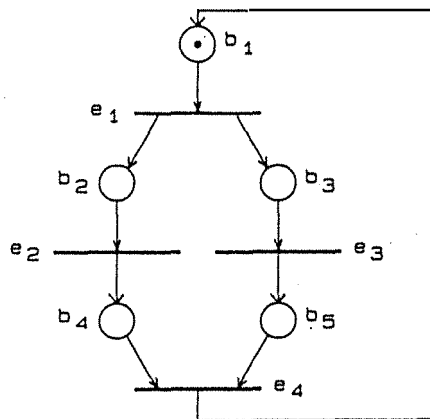


Bild 3-2 : Petri-Netz mit dem Schritt  $\{e_2, e_3\}$

In dem in Bild 3-2 gezeigten Petri-Netz muß Ereignis  $e_1$  vor den Ereignissen  $e_2$  und  $e_3$  eintreten. Die Ereignisse  $e_2$  und  $e_3$  treten entweder gleichzeitig oder in beliebiger Reihenfolge ein. Damit sind sie voneinander unabhängig und können zu einem Schritt zusammengefaßt werden.

In dem Netz von Bild 3-2 besteht zwischen den Ereignissen innerhalb der Mengen  $\{e_1, e_2, e_4\}$  und  $\{e_1, e_3, e_4\}$  eine kausale Beziehung. Die Ereignisse sind jeweils durch eine "Früher-als"-Relation geordnet. Diese Relation, notiert durch " $<$ ",

wird für Knoten  $x, y \in S \cup T$  definiert durch:

$$x < y \text{ gdw. } x F^+ y$$

Die maximalen Mengen der bzgl. " $<$ " geordneten Knoten heißen Linien. Linien sind als sequentielle Teilprozesse des durch ein Netz beschriebenen Gesamtsystems aufzufassen. Linien enthalten sequentiell geordnete Bedingungen und Ereignisse. Analog werden nebenläufige Knoten durch eine Relation definiert:

$$x @ y \text{ gdw. } (\neg x F^+ y \wedge \neg y F^+ x)$$

Dies bedeutet zwei Ereignisse sind nebenläufig und damit unabhängig voneinander, wenn sie nicht durch einen gerichteten nichtleeren Weg verbunden sind.

Die maximalen, bezüglich " $<$ " ungeordneten Mengen von Knoten werden Schnitte genannt. Schnitte, die nur aus Bedingungen bestehen, heißen Scheiben. in Abbildung 3-2 sind z.B.  $\{b_2, b_3\}$  und  $\{b_4, b_5\}$  Scheiben.

Diese Festlegungen spiegeln die Modellvorstellung der Netztheorie wieder. Ereignisse in einem System sind partiell geordnet. Linien ordnen Ereignisse sequentiell. Schnitte beschreiben Situationen im System, in denen Ereignisse parallel unabhängig sind und keine Ordnung angegeben werden kann. Dieses Modell wird als "True Concurrency"-Modell bezeichnet. Es steht im Gegensatz zum "Interleaving"-Modell, das Ereignisse total ordnet (siehe Kap. 3.2).

Scheiben können als Repräsentation eines globalen Systemzustandes aufgefaßt werden. Eine Scheibe charakterisiert den Zustand jedes sequentiellen Teilprozesses, der durch eine Linie gegeben ist. Dazu muß jede Scheibe jede Linie schneiden. Wie /Star 80/ zeigt ist dies bei allen Netzen, die in Prozessbeschreibungen vorkommen, der Fall.

### 3.1.2 Analyse von Petri-Netzen

Das zentrale Problem der Netztheorie ist es, zu zeigen, daß ein Netz bei gegebener Anfangsmarkierung lebendig und sicher ist.

Ein Petri-Netz heißt sicher, wenn keine Marke auf eine schon markierte Stelle gelegt wird. Es existiert ein Algorithmus, der für jedes (endliche) Netz entscheidet, ob das Netz sicher ist.

Die Frage nach der Lebendigkeit ist für das Systemverhalten von Bedeutung.

Wenn ein System durch ein Netz beschrieben wird, so entspricht einem totalen Systemstillstand eine Markierung, bei der keine Transition aktivierbar ist. Ein Fall  $c$  ist also nicht mehr in einen Folgefall  $c'$  überführbar. Solche Netze heißen nicht lebendig. Dabei unterscheidet man verschiedene Lebendigkeitsbegriffe. Diese Konstruktion wird benötigt, um Eigenschaften von Petri-Netzen nachzuweisen.

Definition 3-6 : lebendig /Laut 73/

*Sei  $N$  ein S/T-Netz, sei  $t$  eine Transition und  $m$ , eine Markierung.*

*Dann heißt*

- *$t$  lebendig in  $M$ , wenn es eine Nachfolgemarkierung  $m'$  in  $M$  gibt, in der  $t$  aktiviert ist;*
- *$m$  lebendig, wenn eine Transition in  $m$  schalten kann und*
- *$m$  1-lebendig, wenn alle Folgemarkierungen lebendig sind.*

Kann die Lebendigkeit der Markierungen eines Systems nachgewiesen werden, ist die Verklemmungsfreiheit des Systems bewiesen. Fragen der Lebendigkeit können auf das Erreichbarkeitsproblem zurückgeführt werden. Dabei geht es um die Frage, ob eine gegebene Markierung  $m$  von einer Startmarkierung  $m_0$  aus erreichbar ist, d.h. es soll gezeigt werden, daß  $m_0 \bullet m$ . Bisher ist für allgemeine Netze nicht bekannt, ob dieses Problem entscheidbar ist.

Schränkt man die Klasse der zu untersuchenden Netze jedoch ein auf beschränkte Netze, d.h. Netze deren Stellen eine beschränkte Anzahl von Marken tragen, ist das Problem durch Untersuchung des Fallgraphen lösbar. Der Fallgraph gibt alle Markierungen an, die in einem Netz erreichbar sind.

In Bild 3-3 ist der Fallgraph zum Petri-Netz von Bild 3-2 dargestellt.

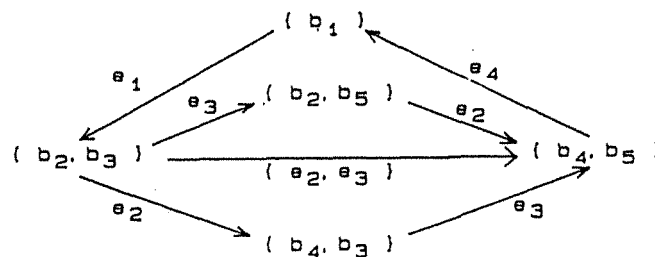


Bild 3-3 : Fallgraph zu Bild 3-2

Der Fallgraph eines allgemeinen Petri-Netzes ist stets endlich, auch wenn die Menge der Markierungen durch Zyklen im Netz unendlich wird und Stellen unbeschränkt Marken erhalten. Die Belegung solcher Stellen mit Marken wird auf ein Spezialsymbol abgebildet. Allerdings ist ein solcher Fallgraph nicht mehr eindeutig. Ein- und derselbe Graph kann zu verschiedenen Netzen gehören. Deshalb ist die Erreichbarkeitsfrage und damit der Nachweis der Lebendigkeit nur für Netze mit beschränkter Markierung entscheidbar.

Um die Lebendigkeit eines Netzes entscheiden zu können, muß außerdem die "Free-Choice"- Bedingung erfüllt sein. Die Bedingung besagt, daß eine Situation, wie sie in Bild 3-4 gezeigt wird, in einem Netz nicht enthalten sein darf. Hier tritt eine Konfusion auf.

Die Stelle  $b_2$  ist Vorbedingung für die Transitionen  $e_1$  und  $e_2$ . Damit befinden sich die beiden Transitionen in einem Konflikt; nur eine von beiden darf schalten. Allerdings ist  $e_1$  zusätzlich von  $b_1$  abhängig. Eine solche Situation heißt "konfus".

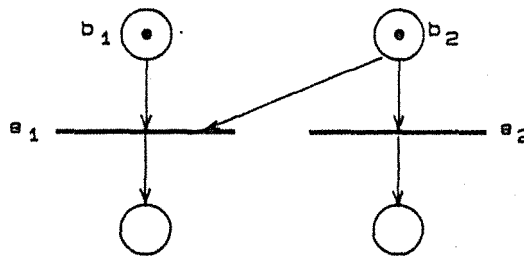


Bild 3-4 : Petri-Netz mit Konfusion

Da die Auswahl, welche Transition schalten darf, von zwei Stellen  $b_1$  und  $b_2$  abhängig ist, ist der Konflikt nicht durch freie Auswahl "systemintern" auflösbar. Voraussetzung für die Analyse der Lebendigkeit ist jedoch, daß eine solche Auswahl durchführbar ist. Free-Choice-Netze sind charakterisiert durch die Bedingung:

$$s, S \text{ " } |s \cdot t| > 1 \Rightarrow (\forall t, S) \text{ gilt } (t = \{s\})$$

In Bild 3-5 ist eine Situation gezeigt, in der der Konflikt zwischen den Transitionen "systemintern" auflösbar ist.

Diese beiden Situationen treten bei der Modellierung von Prozeßsystemen häufig auf. Bild 3-5 entspricht der internen nichtdeterministischen Auswahl von Folgeereignissen in einem Prozeß.

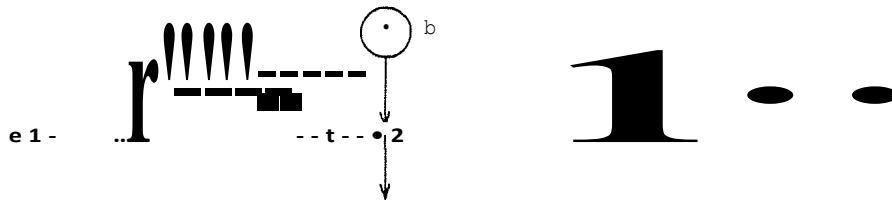


Bild 3-5 : Petri-Netz mit auflösbaren Konflikt

Bild 3-4 beschreibt externen Nichtdeterminismus. Unter der Annahme, daß  $c_1$  und  $b_1$  in einem Prozeß  $P_1$  und  $c_2$ ,  $b_2$  im Prozeß  $P_2$  enthalten sind, hat  $P_2$  in diesem Fall die Auswahl zwischen einem gemeinsamen Ereignis mit  $P_1$  (Synchronisation) oder der Fortsetzung mit dem internen Ereignis  $b_2$ .

### 3.1.3 Ausdrucksfähigkeit

Die Netztheorie erlaubt eine direkte Beschreibung der Parallelität von Ereignissen. Parallelität wird explizit als kausale Unabhängigkeit von Ereignissen in einem System dargestellt.

Es fehlt jedoch eine geeignete Modularisierung eines Gesamtsystems in parallele Prozesse, die unabhängig voneinander spezifiziert werden können. Damit ist die Strukturierung eines Systems nur schwer möglich. Dies beeinträchtigt die Anwendbarkeit der Netztheorie beim Entwurf größerer Systeme erheblich /GMD 88/, da ein Netz alle Ereignisse, die im Gesamtsystem möglich sind, und ihre gegenseitigen Abhängigkeiten enthalten muß.

Eine Zerlegung in Teilnetze, entsprechend dem üblichen Prozeßbegriff (Kap.2\_4), ist zwar möglich, doch ist nur die sequentielle Komposition von Teilnetzen in der Netztheorie enthalten. Zur parallelen Komposition muß die Synchronisation zwischen Ereignissen in verschiedenen Teilnetzen explizit angegeben werden, so daß ein Gesamtnetz entsteht. Daraus resultieren extrem unübersichtliche Netzstrukturen ("Spaghetti-Netz").

Prinzipiell lassen sich jedoch synchrone und asynchrone Kommunikation zwischen Prozessen modellieren.

Zur Beschreibung synchroner und konjunktiv verknüpfter Kommunikation (UND-verknüpftes Senden/Empfangen von Botschaften) müssen mehrere Transitionen

gleichzeitig schalten. Dies kann man in einem Petri-Netz nur durch "Händeschütteln" simulieren.

Bild 3-6 beschreibt die Simulation von gleichzeitigem Schalten durch Händeschütteln.

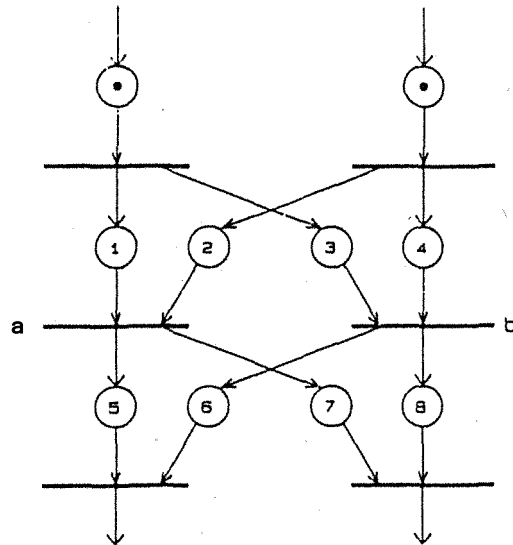


Bild 3-6 : Simulation von gleichzeitigem Schalten /nach Röhr 85/

Marken, auf den Stellen 1 und 3 bzw. 2 und 4 zeigen Bereitschaft zur synchronen Ausführung von a und b an. Nach dem "quasi-gleichzeitigen" Schalten von a und b können die Ausgangstransitionen schalten. Quasi-gleichzeitig bedeutet, daß von außen, vom Rest des Netzes aus, nicht beobachtbar ist, ob eine der Transitionen a, b vor der anderen feuert.

Die Beschreibung der alternativen Fortsetzung durch verschiedene Ereignisse ist auf einfache Weise möglich (s. Bild 3-4 und Bild 3-5). Dabei tritt jedoch eventuell ein Konflikt zwischen Transitionen auf, der nicht systemintern gelöst werden kann. Die Darstellung führt aus der Klasse der analysierbaren Free-Choice-Netze heraus.

Ein ähnliches Problem entsteht, wenn man Prioritätsregeln zur Auswahl aus mehreren aktivierten Transitionen formulieren will. Zur Regelung der Schalterlaubnis werden "Inhibitorkanten" eingeführt /Pete 81/. Grundidee der Inhibitorkanten ist, Transitionen für aktiviert zu erklären, wenn die durch die Inhibitorkante definierte Stelle im Vorbereich der Transition leer ist.

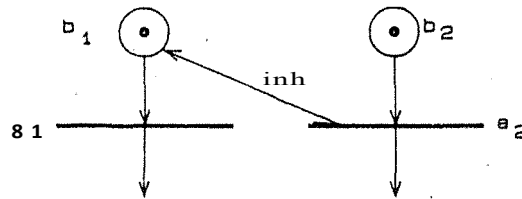


Bild 3-7 : Inhibitorkante zur Prioritätendarstellung

Sind  $b_1$  und  $b_2$  gleichzeitig markiert, hat Transition  $e_1$  Priorität;  $e_2$  wird durch die Inhibitorkante am Schalten gehindert. Allerdings sind Inhibitor-Netze nicht mehr analysierbar; sie besitzen Turing-Mächtigkeit.

Mit erweiterten Petri-Netzen (Timed Petrinetze) können Zeitabläufe in einem System beschrieben werden. Dazu existieren verschiedene Ansätze. In [Stot 86] wird jeder Stelle eine feste Zeit zugeordnet. Eine Marke, die auf eine Stelle gelangt, muß für diese Zeit auf der Stelle "altern", bevor die zugehörige Transition schalten kann. Treffen während der Alterungszeit neue Marken ein, unterliegen diese erst dann dem Alterungsprozeß, wenn die Vorgänger-marke die Stelle verlassen hat. Dieser Modellierungsart liegt die Vorstellung zugrunde, daß jede Stelle über einen eigenen Prozessor verfügt. Marken repräsentieren einen Auftrag, der vom jeweiligen Prozessor bearbeitet wird. Durch eine globale Uhr wird gleichzeitiges Schalten von Transitionen erzwungen. Dazu wird die Schaltregel und die Konstruktion des Fallgraphen neu definiert.

### 3.1.4 Analysefähigkeit

Wesentlich für die Untersuchung von Petri-Netzen ist die Konstruktion des Fallgraphen. Eine Analyse auf Erreichbarkeit ist nur möglich, wenn das zu untersuchende Netz stark einschränkenden Bedingungen genügt. Dies bedeutet, daß ein vorliegendes Netz zunächst auf Einhaltung der Einschränkungen geprüft werden muß.

Der Algorithmus zur Konstruktion des Fallgraphen besitzt exponentielle Laufzeit. Für je  $n$  parallele, unabhängige Ereignisse müssen  $2^n$  Knoten generiert werden. Damit ist die Analyse nur für "kleine" Systeme möglich. Die praktische Anwendbarkeit wird durch die Größe des zu untersuchenden Netzes bestimmt. Deshalb werden zur Untersuchung von Petri-Netzen in der Regel sogenannte



"Token-player" /PROT88/ eingesetzt, die interaktiv durch Benutzer gesteuert das Fortschreiten der Marken im Netz simulieren.

### 3.2 CCS - ein Kalkül zur Beschreibung paralleler Prozesse

In /Miln 80/ wird CCS (calculus of communicating systems) als Mittel zur Beschreibung und Untersuchung von Systemen paralleler, kommunizierender Prozesse eingeführt. CCS besteht aus einer Sprache zur Beschreibung des Verhaltens kommunizierender Prozesse, Axiomen zur Darstellung der Semantik von Prozeßsystemen und einem Kalkül zur Herleitung der semantischen Gleichheit von CCS-Termen aus ihrem syntaktischen Aufbau. CCS bietet damit gleichzeitig eine formale Sprache zur Spezifikation von Prozeßsystemen und formale Untersuchungsmethoden, um das Verhalten des Systems auf gewünschte Eigenschaften hin zu analysieren.

CCS stellt als Strukturierungseinheit den Prozeß (computing agent) und Operatoren zum Bilden und Strukturieren von Prozessen zur Verfügung. Als Prozeß wird eine Folge von Aktionen betrachtet, die einen Namen tragen. Ein CCS-Ausdruck oder Term repräsentiert diesen Prozeß in Form von beobachtbaren Aktionen.

Prozesse führen unabhängig voneinander Aktionen aus und kommunizieren miteinander über Ports. Die Kommunikation erfolgt synchron und kann nur über komplementäre Ports stattfinden. Dies sind Ports mit dem gleichen Namen, aber unterschiedlicher Richtung (Sende- und Empfangsports). Üblicherweise werden die Namen von Sendeports durch einen Querstrich über dem Namen gekennzeichnet, z. B. Port  $\bar{a}$ . Empfangsports tragen einfache Namen, z. B.  $a$ .

Eine Kommunikationsaktion wird identifiziert durch den Namen des Ports, über den die Nachricht gesendet oder empfangen wird.

Aktionen besitzen keine zeitliche Ausdehnung; sie werden deshalb auch als Ereignisse bezeichnet.

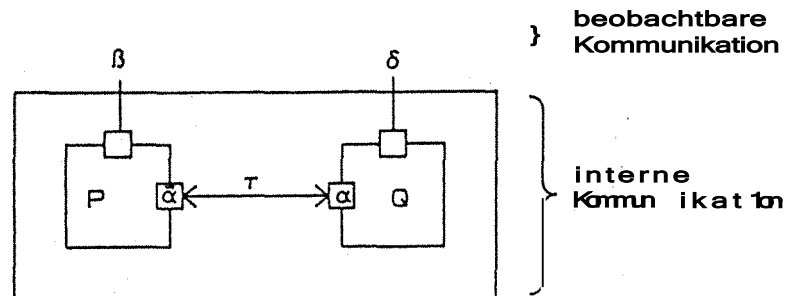


Bild 3-8 : Kommunikationsbeziehungen in CCS

CCS liegt das Konstruktionsprinzip der parallelen Komposition zugrunde. Ein Prozeßsystem wird aufgebaut durch wiederholte Anwendung der parallelen Komposition je zweier Prozesse. Dadurch entsteht ein sequentieller Prozeß, dessen Verhalten nach außen hin nichtdeterministisch erscheint. Beobachtbar von außen ist die externe Kommunikation, das heißt die Kommunikation mit der Umwelt. Die Kommunikation zwischen zwei komponierten Prozessen wird als internes Verhalten bezeichnet, das nach außen hin nicht sichtbar ist. Dieses teilweise Verbergen von Kommunikation wird als Abstraktionsmechanismus gesehen, der eine Beschränkung des Beobachtungsaufwands auf das externe Verhalten ermöglicht.

### 3.2.1 Syntax und Semantik von CCS-Operatoren

Ein Prozeß  $P$  besteht aus einer Folge von Aktionen, die einen Namen tragen. Ein CCS-Ausdruck ist die Repräsentation eines Prozesses. Dazu wird ein Operator benötigt, der eine Folge von Aktionen darstellen kann:

$a..b.nil$  .

Dies bezeichnet einen Prozeß, der zuerst die Aktion  $a$ , dann die Aktion  $b$  ausführt. Weiter kann er keine Aktionen mehr durchführen.  $Nil$  bezeichnet einen Prozeß, der aus der leeren Aktion besteht.

Nichtdeterministische Auswahl wird mithilfe des Summenzeichens "+" dargestellt:

$a.nil + b.nil$  .

Der Prozeß kann entweder die Aktion  $a$  oder die Aktion  $b$  durchführen.

Mit dem "fix"- Operator werden Prozesse rekursiv definiert:

$fix\langle x \rangle . \langle a..b.x \rangle$  .

Dies beschreibt einen Prozeß der unendlich oft das Verhalten  $a..b$  zeigen kann.

Parallele Komposition von Prozessen mit Hilfe des "Parallel"-Operators "/" und der Strukturierung der Ereignisnamen in zwei disjunkte Mengen komplementärer Namen dargestellt.

Die Aktionen  $a$  und  $\bar{a}$  führen zur synchronen Kommunikation zweier Prozesse  $P_1 / P_2$ . Diese synchrone Kommunikation wird im entstehenden sequentiellen Prozeß als Aktion  $T$  bezeichnet. Das Zeichen  $T$  steht für eine von außen nicht beobachtbare Kommunikation.

Umbenennung von Aktionen ist möglich.  $P[\bar{o}/a]$  bedeutet, im Prozeß  $P$  werden alle  $\langle X$ -Aktionen in  $\bar{o}$  umbenannt.

Der "Restriktions"-Operator dient der Einführung von Abstraktion durch Ausblenden von unerwünschten Ereignissen.  $P\bar{o}$  bedeutet, daß die Ereignisse  $\bar{o}$  im Prozeß  $P$  nicht betrachtet werden.

Die CCS-Terme, die das Verhalten von Prozessen beschreiben, können operationell als ein System paralleler Prozesse oder algebraisch als Terme einer Algebra interpretiert werden.

In [Miln 80] wird die Semantik von CCS-Termen operationell definiert. Der operationelle Ansatz wird vielfach als zu wenig abstrakt betrachtet, da die Abbildung von Sprachelementen auf eine zugrunde gelegte abstrakte Maschine als unnatürlich empfunden wird und Beschränkungen bei der Definition der Semantik auferlegt.

Der operationelle Ansatz wurde jedoch gewählt, da dadurch das zustandsabhängige Verhalten paralleler Systeme besser dargestellt werden kann. Das Eintreten von Ereignissen ist immer mit einem Zustandsübergang des betroffenen Prozesses verbunden. Mit einer operationellen Beschreibungsmethode kann ein solcher Vorgang einfach modelliert werden.

Allerdings wurde hier keine feste, abstrakte Maschine zugrunde gelegt, sondern auf einem abstrakten Transitionssystem aufgebaut. CCS-Terme sind selbst Zustände des Transitionssystems. Zustandsübergänge werden als Symbolersetzung, d.h. als syntaktische Transformation, aufgefaßt.

Definition 3-7 : Prozeß (nach [DeNi86])

*Ein Prozeßsystem wird beschrieben durch das Transitionssystem*

$T = (P, A \cup J \cup \{r\}, p)$  wobei

- $P$  Terme von CCS repräsentiert,
- $A$  und  $J$  Mengen von Aktionsnamen sind (Ports und komplementäre Ports),
- $r$  die interne Kommunikation bezeichnet, die durch synchrone Aktionen zustande kommt,
- $E \subseteq P \times (A \cup J \cup \{r\}) \times P$  eine Übergangsrelation ist,
- $p$  einen Prozeß, beschrieben durch einen CCS-Term, darstellt..

Die operationelle Semantik wird induktiv definiert durch eine Menge von Axiomen und Ableitungsregeln über die Struktur von CCS-Ausdrücken. Grundlegend ist das Axiom:

$$c.a.P \quad p$$

Es definiert die Semantik eines sequentiellen Prozeßschrittes. Es wird festgelegt, daß ein Prozeß, der durch den Ausdruck  $a.P$  beschrieben wird, eine Kommunikationsoperation am Port  $a$  ausführen kann und sich dann so verhält, wie durch den Ausdruck  $P$  beschrieben. Seien  $P$  und  $Q$  Prozesse, dann wird  $P \mid Q$  als Schreibweise benutzt für den Ausdruck  $(P,a,Q) E$ ,

Das Verhalten parallel komponierter Prozesse wird durch drei Ableitungsregeln definiert:

$$1) \quad \frac{P \xrightarrow{a} P'}{P/Q \quad P'/Q}$$

$$2) \quad \frac{Q \xrightarrow{a} Q'}{P/Q \quad P/Q'}$$

Der Prozess  $P$  kann die Kommunikationsaktion  $a$  ausführen, anschließend verhält er sich wie durch  $P'$  beschrieben. Komponiert man  $P$  parallel mit dem Prozeß  $Q$ , so kann der entstehende sequentielle Prozeß ebenso  $a$  ausführen. Nach der Ausführung kann sein Verhalten durch die parallele Komposition von  $P'$  und  $Q$  beschrieben werden. In Regel (2) wird dasselbe für den Prozeß  $Q$  festgelegt.

In der folgenden Regel wird die Kommunikation über komplementäre Aktionen festgelegt

$$3) \quad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P/Q \quad 4 \quad P'/Q'}$$

In diesem Fall kommunizieren  $P$  und  $Q$  über die Aktion  $a$  bzw.  $\bar{a}$ . Im komponierten Prozeß wird die Synchronisation durch das  $\tau$ -Symbol als interne Kommunikation dargestellt.

Diese drei Regeln legen fest, wie sich ein System aus zwei komponierten Prozessen verhalten darf. Die enthaltenen Teilprozesse können entweder unabhängig voneinander Aktionen ausführen oder miteinander kommunizieren, so daß eine

T-Aktion im Kompositum entsteht.

Dieser Sachverhalt führt zum Expansionstheorem, das das Verhalten eines Systems paralleler Prozesse durch einen sequentiellen, nichtdeterministischen Prozeß beschreibt. In [Boud87] wird das Expansionstheorem kurz so charakterisiert

Parallelität • Sequentialität + Nichtdeterminismus

Theorem 3-1 : Expansionstheorem

$$P_1 \parallel \dots \parallel P_n = \sum_{i=1}^n \{ a_i (P_1 \parallel \dots \parallel P_i \cdot t \parallel \dots \parallel P_n) \parallel \sum_{j=1}^n P_j \parallel P_i \} \\ + \sum_{i=1}^n \{ r_i (P_1 \parallel \dots \parallel P_i \parallel \dots \parallel P_i \parallel \dots \parallel P_n) \parallel \sum_{j=1}^n P_j \parallel P_i \} \\ \text{wobei } i \neq j < n \text{ und} \\ P_i \beta P_j, \forall P_i \neq P_j \}$$

In einer allgemeineren Form des Expansionstheorems wird die Restriktion von Prozessen auf bestimmte Aktionen berücksichtigt.

Der erste Teilterm der rechten Seite enthält alle Ausführungsmöglichkeiten von unabhängigen Aktionen der Prozesse im System, der zweite die interne Kommunikation zwischen Paaren von Prozessen. Das Expansionstheorem beschreibt damit, wie der gesamte Zustandsraum eines Systems paralleler Prozesse konstruiert wird.

### 3.2.2 Äquivalenz von Prozessen

Gleichungsregeln geben an, wie CCS-Ausdrücke unter Beibehaltung des beobachtbaren Verhaltens umgeformt werden können. Die Gleichungen definieren den Begriff der "beobachtbaren Äquivalenz". Der Äquivalenzbegriff wird benötigt um Eigenschaften von Prozessen überprüfen zu können.

Es wird angenommen, daß ein Beobachter das Verhalten eines Prozeßsystems von außen notiert. Die in einem System auftretenden Ereignisse werden in linear geordneter Form beobachtet. Sind zwei Ereignisse in einem System voneinander unabhängig, d.h. echt parallel möglich, wird dies in Form mehrerer möglicher Beobachtungen beschrieben. Man geht davon aus, daß zwei Ereignisse nicht gleichzeitig auftreten können. Deshalb werden zwei parallele Ereignisse a und b so beschrieben, daß sie in zwei verschiedenen Reihenfolgen, zuerst a dann b

oder zuerst b dann a, auftreten. Diese Modellvorstellung wird als "Interleaving-Modell" bezeichnet.

Interleaving-Modelle beschreiben das Verhalten von Systemen als sequentielle Ereignisfolgen, die auch als Trace oder Bahn bezeichnet werden.

Zwei Prozesse heißen dann beobachtbar äquivalent, wenn sie in jedem Zustand die gleichen beobachtbaren Aktionsfolgen ausführen können.

Obwohl CCS eine aktionsorientierte Beschreibung von Prozessen gibt, spielt also der interne Zustand eines Prozesses eine Rolle. Dies soll an folgendem Beispiel erläutert werden.

Beispiel:

Gegeben seien die beiden Prozesse

$P1 \bullet a.(\beta.nil + \ddot{o}.nil)$  und  $P2 \bullet a.\beta.nil + a.\ddot{o}.nil$

Sie sind nicht äquivalent, obwohl beide genau die Aktionsfolgen  $\{a\}$ ,  $\{a,\beta\}$ ,  $\{a,\ddot{o}\}$  ausführen können.  $P1$  ist jedoch nach der Aktion  $a$  in einem Zustand, in dem er die Wahl zwischen  $\beta$  und  $\ddot{o}$  hat.  $P2$  besitzt nach der Aktion  $a$  diese Wahlmöglichkeit nicht. Wenn ein externer Prozeß  $P3$  die Aktion  $o$  anbietet, kann  $P1$  nach  $a$  eine Kommunikation mit  $P3$  durchführen.  $P2$  kann nur dann kommunizieren, wenn die zweite Alternative ausgewählt wurde.

Gleichungsregeln definieren eine Kongruenzrelation  $\equiv$  auf Prozessen. Für CCS-Ausdröcke, die unter Verwendung des Rekursionsoperators und Operators nicht-deterministische Auswahl (" $+$ ") konstruiert wurden, gelten sie nur unter bestimmten Einschränkungen. Bei der nichtdeterministischen Auswahl muß z. B. die interne Kommunikation berücksichtigt werden. Dies ist in nachfolgendem Beispiel dargestellt.

Beispiel :

Seien  $P1 \bullet a.nil$  und  $P2 \bullet T.a.nil$ , so gilt  $P1::: P2$ .

Die Prozesse  $P1' \bullet a.nil + \beta.nil$  und  $P2' \bullet a.nil + T.\beta.nil$  verhalten sich jedoch nicht gleich, da  $P2$  nach Ausführung der internen Kommunikation nur noch die Aktion  $\beta$  ausführen kann.

Die Forderung nach beobachtbarer Kongruenz unterstützt den modularen Aufbau eines Systems paralleler Prozesse. Es ist möglich ein Prozeßsystem durch parallele

Komposition schrittweise zu konstruieren. Die Reihenfolge, in der man vorgeht ist unwichtig, da die Komposition assoziativ und kommutativ ist

$$P|Q \approx Q|P, \quad (P|Q)|R \approx P|(Q|R)$$

Ersetzt man in einem komplexen System einen Prozeß durch einen anderen, so muß das Verhalten der unveränderten Prozesse nicht neu untersucht werden. Man kann den veränderten Prozeß mit den übrigen Prozessen komponieren.

### 3.2.3 AusdrucksCihlakelt

Der Prozeßbegriff ist grundlegender Bestandteil von CCS. Damit ist die Strukturierung von Prozeßautomatisierungssystemen möglich. Ein Prozeß wird jedoch nur aktionsorientiert beschrieben, d.h. als Folge möglicher Ereignisse. Damit ist die Darstelluns von zeitOberwachten Zuständen nur mit Hilfe zusätzlicher Ereignisse möglich, die Beginn und Ende eines Zeitintervalls repräs ntieren. Der Ablauf der Zeittlberwachung muß durch nichtdeterministische Auswahl zwischen dem geplanten Ablauf und einem Ausnahmeweig dargestellt werden. Der Grund fOr die Ausnahmebehandlung ist nicht modellierbar.

Durch das Verbergen der internen Kommunikation bei paralleler Komposition in dem Symbol T eignet sich CCS auch nicht als Grundlage fOr ein erweitertes Modell, das absolute Zeitbedingungen beschreiben kann (wie z. B. Petri-Netze). Jede interne Kommunikation kann unterschiedliches zeitliches Verhalten besitzen; damit ist sie nicht durch ein Symbol beschreibbar. Ebenso ist bei parallel komponierten Prozessen das Verbergen der internen Kommunikation nicht sinnvoll, da das Zeitverhalten der komponierten Prozesse nach außen hin durchaus von der internen Kommunikation der beiden Prozesse untereinander abhängig ist.

Die Kommunikation entsprechend dem Port-Modell von CCS erfolgt nur synchron. Um asynchronen Nachrichtenaustausch nachbilden zu können, muß fnr die Kommunikation zwischen je zwei Prozessen ein eigener Prozeß definiert werden, der die Speicherfunktion' eines asynchronen Nachrichtenkanals modelliert.

Bei begrenzter Speicherkapazität des Kanals; kann dann aber das Prinzip der schrittweisen Komposition paralleler Prozesse nicht mehr angewendet werden, da Blockierungen am Kanal von der Reihenfolge der Ablage von Nachrichten abhängen können.

Alternativen zwischen Kommunikationsereignissen können ausgedrückt werden. Allerdings ist Kommunikation eine binäre Operation, so daß UND-verknüpfter Nachrichtenaustausch nicht möglich ist.

Auch die Vergabe von Prioritäten bei nichtdeterministischer Auswahl ist nicht möglich,-:

### 3.2.4 Analysefähigkeit

In CCS die Analyse eines Prozeßsystems schrittweise vorgenommen werden. Durch sukzessives Anwenden des Parallel-Operators muß nicht der ganze Zustandsraum eines Prozeßsystems in einem Analysevorgang betrachtet werden ( unter der Voraussetzung der beobachtbaren Kongruenz).

Die parallele Kompositionstechnik erlaubt, daß bei Korrekturen an einzelnen Prozessen, die sich nur in verändertem internen Verhalten eines Teilsystems auswirken, nicht das ganze Prozeßsystem neu untersucht werden muß. Es genügt die Äquivalenz des äußeren Verhaltens mit dem ursprünglichen Systemteil zu zeigen.

Die Äquivalenz wird mit Hilfe der Bisimulationstechnik bewiesen. Damit ist es möglich, das gewünschte Verhalten eines Prozeßsystems durch einen sequentiellen Prozeß zu beschreiben und die Äquivalenz mit dem spezifizierten System paralleler Prozesse nachzuweisen. Eingeführt wird dadurch aber eine formale Vorstufe der Spezifikation, die "irgendwie" aus der Anforderungsdefinition abgeleitet werden muß. Praxisnäher dürfte es daher sein, Teilaspekte eines gewünschten Verhaltens in Form von CCS-Termen zu formulieren. Unter Verwendung der Gleichungsregeln in CCS kann dann versucht werden nachzuweisen, daß der vorgegebene Ausdruck in der Spezifikation des Systems paralleler Prozesse enthalten ist.

Die beschränkte Ausdruckskraft von CCS ermöglicht die Anwendung unkomplizierter Analysemethoden. Die beobachtbare Äquivalenz von Prozessen kann in polynomialer Zeit entschieden werden /KaSm 83/.



### 3.3 PASS - Parallel Activities Specification Scheme

Zur Spezifikation von Prozeßautomatisierungssystemen wird in /Flei 84/ die Beschreibungsmethode PASS vorgestellt. Die Methode wird in /AFHK 85/ auch zur Beschreibung von Kommunikationsprotokollen verwendet. Kommunikationsprotokolle werden als ein Spezialfall von **verteilten** Systemen betrachtet, die aus Paaren von kommunizierenden Prozessen **bestehen**.

PASS benutzt als Strukturierungsmittel den Begriff des Prozesses. Diese Prozesse können sowohl über gemeinsame Objekte als auch mit Hilfe von Botschaften kommunizieren.

Unter Verwendung des Botschaftenmechanismus ist mit PASS eine einheitliche Beschreibung von Rechenprozessen und technischen Prozessen möglich. Wie /Flei 84/ zeigt, sind die Sprachkonstrukte von PASS mächtig genug, um auch das Verhalten technischer Systeme gegenüber der Umwelt mit Hilfe von Botschaftsoperationen zu modellieren;

Eine PASS-Spezifikation besteht aus vier Teilen:

- Die Kommunikationsstruktur beschreibt die Netzwerktopologie graphisch. Es werden alle Prozesse mit ihren Kommunikationsbeziehungen dargestellt.
- Die Ablaufsteuerung definiert das Verhalten eines sequentiellen Prozesses durch graphische Darstellung der möglichen Aktionsfolgen. Dabei wird zwischen Kommunikationsaktionen und internen Berechnungen unterschieden.
- Die Kommunikationsmaschine beschreibt die Art des Botschaftsaustausches. Es wird zwischen synchroner und asynchroner Kommunikation unterschieden.
- Die Benutzermaschine beschreibt die Wirkung interner Berechnungen und die Daten eines Prozesses.

PASS modelliert einen Prozeß mit Hilfe eines erweiterten endlichen Zustandsautomaten. Nach /Boch 83/ werden Zustandsübergänge in einem erweiterten Zustandsautomaten folgendermaßen beschrieben:

FROM       (aktueller Zustand)  
 WHEN       (Kommunikationsanweisung)  
 PROVIDED {Zustandsprädikat}  
 TO           (Folgezustand)  
 BEGIN       {Anweisungsfolge zur Veränderung der lokalen Variablen}  
 END.

Diese allgemeine Form der Übergangsbeschreibung wird in PASS nur eingeschränkt benutzt /Flei 87/. Die Art der Einschränkung ist bei der Beschreibung der einzelnen Sprachelemente erläutert.

### 3.3.1 Strukturierungseinheiten in PASS

PASS verwendet als Basiseinheit zur Strukturierung von Prozeßsystemen den Begriff des Prozesses. Prozesse können zu den Elementen Prozeßgruppe und Prozeßbündel zusammengefaßt werden.

Die verschiedenen Strukturierungseinheiten unterscheiden sich durch die Art der Kommunikation. Innerhalb von Prozeßgruppen und -bündeln können Prozesse über gemeinsame Objekte kommunizieren. Diese Objekte werden in der gemeinsamen Benutzermaschine zusammengefaßt. Prozesse von Gruppen bzw. Bündeln müssen also über einen gemeinsamen Speicher verfügen.

Prozesse von Gruppen oder Bündeln kommunizieren mit Prozessen, die nicht in der eigenen Strukturierungseinheit enthalten sind, über Botschaften. Der Unterschied zwischen einem Bündel und einer Gruppe besteht darin, daß Prozesse einer Gruppe nur anonym über ihren Gruppennamen, nicht aber über ihren Prozeßnamen, angesprochen werden. Für einen Sender oder Empfänger ist nicht unterscheidbar, welcher Prozeß einer Gruppe der tatsächliche Kommunikationspartner ist. Als Adressat oder Absender tritt nur die Gruppe in Erscheinung.

Einzelprozesse werden direkt adressiert und können nur über Botschaften mit anderen Strukturierungseinheiten kommunizieren.

### 3.3.2 Kommunikationsstruktur

Die Kommunikationsstruktur beschreibt, welche Hardware- und Software-Prozesse an einem Prozeßsystem beteiligt sind und welche Kommunikationsbeziehungen

bestehen. In Form eines Kommunikationsdiagramms wird das gesamte Prozeßsystem zusammen mit den auszutauschenden Nachrichten dargestellt. In dieser Darstellung taucht auch der menschliche Benutzer in Form eines Benutzerprozesses auf.

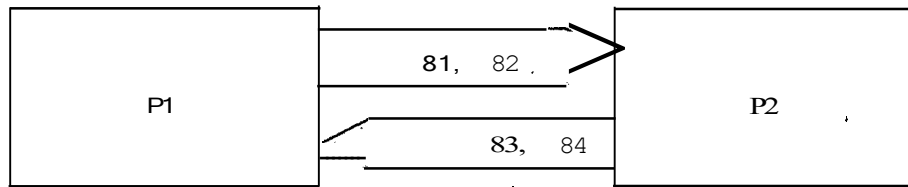


Bild 3-9 : Kommunikationsdiagramm

Strukturierungseinheiten, also Einzelprozesse, Prozeßgruppen und -bindel, werden als Rechtecke gezeichnet. Pfeile, beschriftet mit den Namen der Nachrichten, zeigen vom sendenden zum empfangenden Prozeß. Damit ist die Topologie des Prozeßnetzwerks vollständig festgelegt.

### 3.3.3 Prozeßbeschreibung

Eine Prozeßbeschreibung besteht aus den Teilen Kommunikationsmaschine, Ablaufsteuerung und Benutzermaschine. Die Benutzermaschine ist als private Benutzermaschine jeder Prozeßbeschreibung zugeordnet, als gemeinsame Benutzermaschine wird sie bei Prozeßgruppen und -bindeln von mehreren Prozessen geteilt.

#### I. Die Kommunikationsmaschine

Die Kommunikationsmaschine definiert ein abstraktes Modell des zugrundeliegenden Botschaftsmechanismus. Sie regelt das Senden und Empfangen von Nachrichten und veranlaßt die Ausführung der entsprechenden Operationen.

Entsprechend des in Kapitel 2.3 diskutierten Modells für Botschaftsmechanismen stellt die Kommunikationsmaschine die direkte Adressierung von Prozessen sicher. Lediglich Prozeßgruppen werden eigene Sende- und Empfangsoperationen zugeteilt, die die anonyme Adressierung von Prozessen der Gruppe realisieren.

Die Eigenschaften des Transportmediums sind parametrisierbar. Beim Entwurf des Prozeßsystems kann angegeben werden, ob der Kanal speichernde oder nicht-speichernde Wirkung hat. Entsprechend erfolgt der Nachrichtenaustausch synchron

oder asynchron über einen Wartebereich. Die Größe des Wartebereichs kann spezifiziert werden. Angegeben wird, wie viele Nachrichten gespeichert werden sollen, unabhängig von der Art und Anzahl ihrer Parameter.

Einzelprozesse und Prozesse von Prozeßbtindeln verfügen über eigene Wartebereiche. Den Prozessen einer Gruppe ist stattdessen ein gemeinsamer Wartebereich zugeordnet. Hier ist der Puffer eine Eigenschaft der Gruppe.

Ist einem Prozeß ein Wartebereich zugeordnet, müssen Nachrichten an diesen Prozeß im Wartebereich abgelegt werden. Der sendende Prozeß wird so lange nicht blockiert, wie im Wartebereich noch Platz für eine Nachricht ist. Bei synchroner Kommunikation, bei der kein Wartebereich definiert ist, wird der Sender so lange blockiert, bis der Empfänger die Nachricht entgegennimmt (Rendezvous-Konzept).

Empfangende Prozesse werden immer dann blockiert, wenn eine Nachricht entweder nicht direkt angeboten wird (bei synchroner Kommunikation) oder nicht im Wartebereich vorhanden ist (bei asynchroner Kommunikation).

Nachrichten werden im Wartebereich in der Reihenfolge, in der sie gesendet wurden, abgelegt. Allerdings ist der Empfänger nicht an eine Abarbeitung in dieser Reihenfolge gebunden. Der Empfang erfolgt auftragsgesteuert. Der Empfänger gibt die gewünschte Nachricht und den zugehörigen Absender an. Ist die Nachricht im Wartebereich vorhanden, wird der Empfangswunsch erfüllt. Nur wenn mehrere gleiche Nachrichten, d. h. Nachrichten mit gleichem Namen und vom gleichen Absender, vorhanden sind, bekommt der Empfänger die älteste Nachricht (Fifo-Strategie).

Im Gegensatz zu reinen FIFO-Strategien bei der asynchronen Kommunikation wirkt die Kopplung kommunizierender Prozesse durch die auftragsgesteuerte Entnahme von Nachrichten aus dem Wartebereich weniger stark synchronisierend. Natürlich bleibt die Kausalitätsbedingung, daß eine Nachricht von einem Prozeß erst dann empfangen werden kann, nachdem sie gesendet wurde, erhalten.

## 2 Die Ablaufsteuerung

In der Ablaufsteuerung wird festgelegt, in welcher Reihenfolge ein Prozeß Nachrichten austauscht bzw. welche internen Berechnungen durchgeführt werden. Auf Grund der Ergebnisse solcher Berechnungen und auf Grund der zustandegewonnenen Kommunikation kann in verschiedene Teile der Ablaufsteuerung verzweigt werden.

Die Ablaufsteuerung wird unter Verwendung der Elemente Knoten und Kante

aufgebaut. Sie stellt eine graphische Notation der Zustandsüberföhrungsfunktion des zugrundeliegenden Zustandsautomaten dar.

In der Ablaufsteuerung unterscheidet man Kommunikations- und Internzustände. Ein Kommunikationszustand wird durch ein Rechtecksymbol, ein Internzustand durch einen ovalen Knoten markiert.

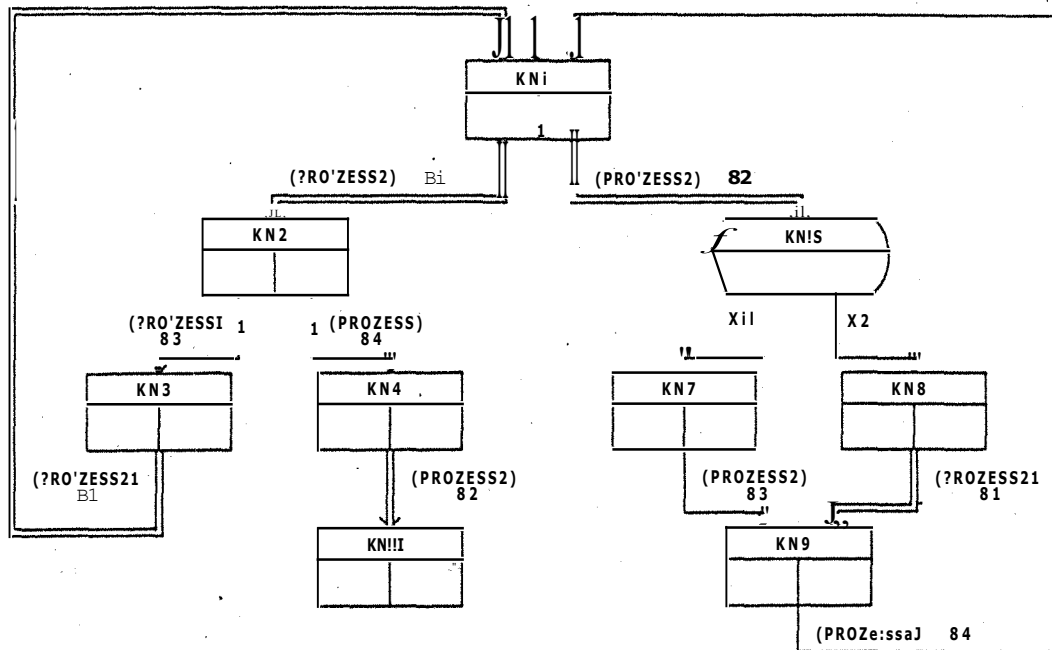


Bild 3-10: Ablaufsteuerung

Kommunikationsknoten symbolisieren Sende- oder Empfangszustände, je nach Art der wegführenden Kanten: einfache Pfeile entsprechen Empfangskanten, Doppelpfeile entsprechen Sendekanten. Kanten sind mit den Namen der Kommunikationspartner und den Botschaftsnamen beschriftet.

In PA ist die Darstellung eines konjunktiv verknüpften und alternativen Nachrichtenaustauschs möglich. Föhren von einem Knoten mehrere Sende- oder Empfangskanten weg, so kann in dem Zustand alternativ eine der Kommunikationsanweisungen ausgeföhrt werden. Die Auswahl ist davon abhängig, ob eine der Vorbedingungen zu den verschiedenen Alternativen gültig ist. Sind mehrere Vorbedingungen gleichzeitig erfüllt, kann eine Auswahl zufällig oder nach Prioritäten erfolgen. Dazu kann in jedem Kommunikationsknoten eine entsprechende Prioritätenregelung angegeben werden.

Sind an einer Kommunikationskante mehrere Prozeßpartner und Nachrichten

spezifiziert, erfolgt die Kommunikation "UND-verknüpft". Ein Zustandsübergang kann erst dann durchgeführt werden, wenn sichergestellt ist, daß alle spezifizierten Nachrichten ausgetauscht werden können.

Das Senden und Empfangen von Nachrichten kann zeitüberwacht erfolgen. Dazu wird im Kommunikationsknoten die Überwachungszeit eingetragen. Eine Kante, die mit "Zeit" beschriftet ist, führt aus dem Knoten zum Nachfolgezustand, in dem nach Ablauf der Zustandsüberwachung eine Ausnahmebehandlung durchgeführt wird.

Bei internen Berechnungen wird zwischen internen Funktionen und internen Operationen unterschieden. Führt man eine interne Operation aus, so werden die lokalen Daten eines Prozesses verändert. Der Zustand der lokalen Daten wird über interne Funktionen abgefragt. Die Unterscheidung erfolgt wieder durch die Art der Kanten, die vom Knoten wegführen. Interne Funktionen sind durch einfache wegführende Kanten, interne Operationen durch wegführende Doppelkanten gekennzeichnet.

Durch die Zweiteilung in Kommunikationsknoten und Internknoten wird die oben beschriebene allgemeine Darstellung der Zustandsüberführung eingeschränkt

- bei Kommunikationsknoten

```
FROM x      {Zustand}
WHEN b1 and b2 {"UND"-verknüpfte Liste von Botschaften}
TO y1      {Folgezustand}
WHEN b3      {alternative Botschaft}
TO y2.
```

Die Kommunikation selbst ist also nicht datenabhängig. Der "provided"-Teil, der den datenabhängigen Kontext beschreibt, fällt hier weg.

- bei Internknoten

```
FROM x      (Zustand)
PROVIDED p   (Prädikat über den Kontext)
TO y        {Folgezustand}
BEGIN ...    {Änderung des Kontexts bei interner Operation}
END.
```

### 3 Die Benutzermaschine

Mit der Ablaufsteuerung wird festgelegt, wann welche Aktionen ausgeführt werden, aber nicht, wie die einzelnen Aktionen definiert sind. Die Beschreibung der Nachrichtenparameter und der Wirkung der internen Funktionen und Operationen erfolgt in der privaten bzw. gemeinsamen Benutzermaschine.

Die Benutzermaschine enthält die lokalen Daten eines Prozesses, die Definition der internen Berechnungen und die Beschreibung der Ausgabefunktionen und Eingabeoperationen. Jeder Nachricht, die gesendet wird, ist eine Ausgabefunktion in der Benutzermaschine des Senders zugeordnet. Diese Funktion ermittelt aus dem internen Zustand eines Prozesses die Werte der Nachrichtenparameter. Ebenso wird für jede Empfangsnachricht eine Eingabeoperation angegeben, die die Veränderung der lokalen Daten durch die Nachrichtenparameter beschreibt.

In der gemeinsamen Benutzermaschine von Prozeßbündeln werden gemeinsame Datenobjekte und die notwendigen Zugriffsfunktionen definiert.

Die Methode, in der die Benutzermaschine spezifiziert wird, bleibt dem Anwender überlassen. Hier sind alle Techniken, wie sie in der sequentiellen Programmierung üblich sind, einsetzbar [Fle1 84].

#### 3.3.4 Ausdrucksfähigkeit

PASS verwendet als Basiselement zur Bearbeitung von Automatisierungssystemen den Begriff des Prozesses. Prozesse können zu Gruppen oder Bündeln zusammengefaßt werden. Welche Art der Zusammenfassung gewählt wird, ist abhängig von dem verwendeten Kommunikationskonzept. Damit ist eine anforderungsgerechte Strukturierung von Systemen zur Prozeßautomatisierung möglich.

Das Kommunikationskonzept in PASS ist sehr variabel. Verwirklicht ist sowohl die Kommunikation über gemeinsame Objekte als auch über Botschaftsmechanismen. Allerdings ist die Darstellung der Synchronisation und Kommunikation über gemeinsame Objekte, die in der gemeinsamen Benutzermaschine auftauchen, nicht festgelegt. Die Auswahl einer geeigneten Beschreibungstechnik bleibt dem Anwender überlassen. PASS bietet nur den Rahmen, in den die Beschreibung geeignet eingeordnet werden muß.

Mit den Botschaftsmechanismen von PASS ist eine synchrone und asynchrone Kommunikation zwischen Prozessen beschreibbar. Die auftragsgesteuerte Entnahme von Botschaften in Verbindung mit alternativen Empfangsanweisungen ermöglicht ein prioritätengesteuertes Reagieren auf Botschaften. Dies ist bei der Modellierung von Prozeßautomatisierungssystemen von wesentlicher Bedeutung. PASS-Kommunikationsknoten entsprechen einer graphischen Darstellung von nichtdeterministischen Kontrollanweisungen, wie sie in Kap. 23 diskutiert wurden. Damit ist eine übersichtliche Formulierung komplexer Kommunikationsbedingungen möglich, die auch Zeitüberwachung von Kommunikationsanweisungen umfaßt.

Die Beschreibung von Zeitabläufen durch Angabe von Ausführungszeiten ist in PASS nicht vorgesehen. Da PASS zustandsorientiert ist, ist eine entsprechende Erweiterung auf einfache Weise möglich (ähnlich Timed-Petrinetzen).

### 3.3.5 Analysefähigkeit

Die Semantik der Sprachelemente von PASS wurde in [Fle84] nur informell ("in Prosa") festgelegt. Deswegen ist eine Analyse von PASS-Spezifikationen in der Entwurfsphase nur von Hand durchführbar. In [Krag 86] wird durch Transformationsregeln, die PASS-Spezifikationen in die Programmiersprache PEARL abbilden, eine formale Beschreibung für einen Teil der Sprachelemente gegeben. Die Transformation beschränkt sich auf die Teile Kommunikationsstruktur und Ablaufsteuerung. Damit wird die Prozeßstruktur und die Kommunikation über Botschaften vollständig in PEARL-Code umgesetzt.

## 3.4 Zusammenfassung

Vorgelegt wurden mit der Petrinetztheorie und CCS zwei Vertreter unterschiedlicher Beschreibungsarten von parallelem Verhalten. Petrinetze beruhen auf einem Modell, das echte Nebenläufigkeit darstellen kann, währenddessen CCS mit dem Interleaving-Modell nur total geordnete Ereignisse kennt.

Der Unterschied zwischen beiden Betrachtungsweisen wirkt sich technisch kaum aus, da auch in der Netztheorie bei der Analyse von Netzen durch die Konstruktion des Fallgraphen total geordnete Ereignisfolgen eingeführt werden.

Durch den Verzicht auf ausdrucksstarke Sprachkonstrukte gewinnt CCS erheblich an Analysefähigkeit. Von besonderer Bedeutung für einen praktischen Einsatz



der Spezifikationsmethode, erscheint die Eigenschaft, Prozeßsysteme strukturiert beschreiben zu können. Die kompositionelle Technik von CCS erleichtert damit den Umgang mit komplexen Systemen ganz wesentlich. Aber selbst unter Einschränkung der Ausdrucksfähigkeit von CCS ist der Nachweis gewünschter Eigenschaften oft nur durch Untersuchung des gesamten Zustandsraums eines parallelen Systems möglich (Expansionstheorem).

Petrinetze hingegen erlauben eine umfangreichere Modellierung von Problemen in der Prozeßautomatisierung. Mit Hilfe erweiterter Petrinetze (Inhibitor-Netze, Timed-Netze) sind die wesentlichen Eigenschaften von Echtzeitsystemen beschreibbar.

Da in der Netztheorie der modulare Aufbau durch Strukturierung eines Systems in Prozesse und Komposition von Prozessen vernachlässigt wird, ist die Verwendungsfähigkeit für komplexe Systeme beeinträchtigt.

Auch die Analysemethoden für beschränkte Netze (Erreichbarkeit) beruhen auf Algorithmen mit exponentieller Laufzeit.

PASS ermöglicht, ähnlich wie CCS, die Strukturierung eines Systems in Prozesse. Besonderer Vorteil von PASS ist die Mächtigkeit des Kommunikationskonzepts, mit dem die bei der Prozeßautomatisierung auftretenden Bedingungen auf einfache und übersichtliche Weise beschrieben werden können. PASS fehlt jedoch eine formale Grundlage, um Analysen durchführen zu können.

Wegen des Analyseaufwands für vollständiges Beweisen von Eigenschaften, werden in der Literatur für CCS und Petrinetze interpretative Untersuchungswerkzeuge vorgeschlagen, die ein dialogorientiertes Testen von Spezifikationen ermöglichen. Diese Werkzeuge basieren auf den operationellen Semantikmodellen der jeweiligen Methoden. Eigenschaften einer Spezifikation können benutzergesteuert durch Simulation des Ablaufs eines gegebenen Spezifikationsmodells abgeleitet werden. Für Teilausschnitte oder kleinere Systeme ist auch eine vollständige Analyse möglich. Im wesentlichen werden dabei die Erreichbarkeit von Zuständen bzw. bestimmter Ereignisfolgen untersucht.

Aus diesen Überlegungen wird deutlich, daß die Erweiterung der Mächtigkeit einer Beschreibungssprache stets mit einer Einschränkung der Analysemöglichkeiten einhergeht. Beim Entwurf komplexer Systeme ist aber eine Unterstützung sowohl durch geeignete Sprachkonstrukte zur Darstellung notwendiger Eigenschaften.

des Systems als auch durch hinreichend mächtige Untersuchungsmethoden zur Überprüfung einer Spezifikation auf gewünschtes Verhalten, notwendig.

Interpretative Ausführung von Spezifikationen erscheint als geeigneter Mittelweg zwischen beiden den beiden Ansprüchen nach Ausdruckskraft und Analysefähigkeit. Allerdings sind dann Werkzeuge, die auf Methoden mit beschränkter Ausdruckskraft beruhen, nicht mächtig .senug.

#### 4. Die Semantik von PASS

Um PASS-Spezifikationen rechnergestützt analysieren zu können, soll in diesem Kapitel ein formales Semantik-Modell entwickelt werden.

Entsprechend den Anforderungen aus Kapitel 2 muß eine Spezifikation zeitliche und implementierungsnahe Abfolge in einem System paralleler Prozesse beschreiben können.

Deswegen wird zunächst in Kap. 4.1 ein Zustandsmodell für Prozesse in PASS entwickelt, das Synchronisation und Betriebssystemeinflüsse modellierbar macht.

In Kap. 4.2 wird ein Zeitmodell beschrieben, das durch ein System lokaler Uhren die zeitliche Ordnung von Aktivitäten in einem verteilten System gestattet. Wichtigster Vorteil des Modells ist, daß auf eine globale Uhr verzichtet werden kann und das "Ticken" einer Uhr nicht als Zustand des Prozeßsystems modelliert werden muß.

Kap. 4.3 stellt später benötigte Grundbegriffe der Theorie der Graphersetzungssysteme zur Verfügung.

In Kap. 4.4 definieren wir die Semantik von PASS-Sprachkonstrukten mit Hilfe von Graphproduktionen eines Graphersetzungssystems. Die Graphproduktionen bauen auf dem Zustands- und Zeitmodell auf.

Kap 4.5 beschreibt die Semantik eines Prozeßsystems als Menge aller gemischten Ableitungsfolgen auf einem Programmgraphen.

##### 4.1 Ein Zustandsmodell für PASS-Prozesse

Die in der Literatur bekannten Methoden zur Beschreibung und Analyse von Systemen paralleler Prozesse legen der Darstellung von Synchronisationsbeziehungen zwischen Prozessen in der Regel ein vereinfachendes Zustandsmodell zugrunde, so daß Prozesse folgende Zustände annehmen können:

- aktiv, d.h. der Prozeß führt spezifizierte Aktionen aus,
- blockiert, d.h. der Prozeß wartet darauf, daß ein kooperierender Prozeß die Ausführung einer anstehenden Kommunikationsanweisung ermöglicht und so eine Synchronisation stattfinden kann,
- ruhend, d.h. der Prozeß hat alle spezifizierten Aktionen ausgeführt und einen Endzustand erreicht.

Dieses Zustandsmodell abstrahiert von den Gegebenheiten in einer Laufzeitumgebung. Aktionen paralleler Prozesse sind in diesem Modell echt unabhängig voneinander. In einer Implementierung müßte jeder Prozeß über einen eigenen Prozessor verfügen, um dieses Verhalten realisieren zu können.

Da in einem verteilten System die Anzahl der Prozessoren jedoch in der Regel wesentlich kleiner ist als die Anzahl der Prozesse, gibt es ein Nebeneinander von parallelen und quasi-parallelen Prozessen.

Auf einem Prozessor laufen Prozesse quasi-parallel ab. Dadurch sind die Aktionen quasi-paralleler Prozesse der Verwaltung eines Betriebssystems unterworfen. Um die durch den Prozeßumschalter des Betriebssystems bewirkte lineare Ordnung von Aktionen quasi-paralleler Prozesse analysieren zu können, benötigen wir ein Zustandsmodell, das es gestattet, implementationsabhängige Eigenschaften zu untersuchen:

- Quasi-parallele Prozesse einer Gruppe konkurrieren um den Prozessor. Die Reihenfolge der Prozessorvergabe wird durch Prozeßprioritäten bestimmt.
- Die erzeugte lineare Ordnung hängt vom Typ des Prozeßumschalters ab. Es gibt unterschiedliche Strategien, wann und wie eine Auswahl unter konkurrierenden Prozessoren getroffen wird.

Zur abstrakten Darstellung eines Betriebssystems erweitern wir das Zustandsmodell. Der Zustand "aktiv" wird unterteilt in die Zustände "aktiv" und "lauffähig":

Prozesse können die in Bild 4-1 gezeigten Zustandsübergänge durchführen:

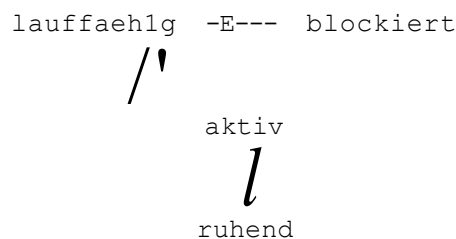


Bild 4-1 : Zustandsmodell

Der zusätzliche Zustand "lauffähig" ist notwendig, um das Konkurrieren quasi-paralleler Prozesse um die Zuteilung des Prozessors modellieren zu können.

Zur Untersuchung des Verhaltens eines Prozeßsystems werden zunächst alle Prozesse, die über einen eigenen Prozessor verfügen, in den Zustand aktiv versetzt. Quasi-parallele Prozesse, die innerhalb einer Gruppe um einen Prozessor konkurrieren, gelten als lauffähig. Die Auswahl aus den lauffähigen Prozessen erfolgt nach einem vorgegebenen Auswahlverfahren. In der Prozeßautomatisierung wird in der Regel die Auswahl nach Prioritäten verwendet; andere Kriterien können Betriebsmittelbedarf (z. B. Haupt-/Externspeicher) oder Laufzeit sein.

Im Startzustand eines Systems sind höchstens so viele Prozesse aktiv, wie Prozessoren vorhanden sind.

Nur ein aktiver Prozeß kann Botschaften senden oder empfangen und interne Berechnungen durchführen. Je nach Prozessorvergabe-strategie bestehen nach einer Kommunikationsanweisung zwei Möglichkeiten:

- a) der Prozeß bleibt weiter aktiv (nichtverdrängende Strategie),
- b) der Prozeß wird in den Zustand lauffähig versetzt und muß wieder auf die Prozessorzuteilung warten (verdrängende Strategie).

Bei Ausführung einer internen Berechnung erfolgt kein Prozeßwechsel; der aktive Prozeß bleibt im aktuellen Zustand.

Kann ein aktiver Prozeß eine Kommunikationsanweisung nicht ausführen, wird er in den Zustand "blockiert" versetzt. Aus diesem Zustand kann er nur durch eine entsprechende Kommunikationsanweisung eines kooperierenden Prozesses oder durch den Ablauf einer Überwachungszeit befreit werden. Dadurch führt der Prozeß einen Übergang nach "lauffähig" aus.

Im Zustand "lauffähig" ist ein Prozeß bereit, eine Aktion durchzuführen. Dazu muß er sich zunächst erfolgreich um die Zuteilung des Prozessors bewerben. Die Auswahl soll nach Prioritäten erfolgen. Sie wird vom Prozeßumschalter durchgeführt. Er ordnet dem ausgewählten Prozeß den Zustand "aktiv" zu.

Die Zahl der Mitbewerber kann unterschiedlich sein. Bei Prozessen, die allein über einen Prozessor verfügen, ist keine Auswahl notwendig; sie werden sofort in den Zustand "aktiv" versetzt.

Ein Prozeß, der nach Ausführung einer Aktion terminiert, geht in den Zustand "ruhend" über. Aus diesem Zustand führt keine Aktivierung wieder heraus.

## 4.2 Zeit in Realzeitsystemen

Will man mehrere Prozesse bezüglich ihres Verhaltens in Relation setzen, so benötigt man in verteilten Systemen einen geeigneten Zeitbegriff.

Bei Programmen zur Automatisierung technischer Prozesse ist es in der Regel notwendig, Zeitbeziehungen zwischen Ereignissen, hervorgerufen durch den technischen Prozeß und Reaktionen des kontrollierenden Programms auszudrücken. Echtzeitsysteme müssen in der Lage sein auf Ereignisse innerhalb fester Zeitschranken zu reagieren.

Nimmt man etwa Informationssysteme, z. B. die Flugreservierung in einem Reisebüro, so werden hier typischerweise Leistungsanforderungen gestellt, wie: " 98% aller Anfragen müssen in weniger als 5 Sekunden bearbeitet werden ",

Für Systeme zur Prozeßautomatisierung sind solche Aussagen noch nicht präzise genug. Hier lautet eine typische Anforderung : " Spätestens 0,5 Sekunden nach Eintreffen einer Alarmmeldung muß unter allen Betriebsbedingungen eine Reaktion in Form eines Steuerbefehls erfolgen ". Es wird also ein Grenzwert vorgegeben, der unter keinen Umständen überschritten werden darf.

Für den Entwerfer eines Echtzeitsystems bedeutet dies, daß nicht nur der logische Ablauf, sondern auch die zeitliche Abfolge von Aktionen im Prozeßsystem Bestandteil der Spezifikation sein muß.

In der Anforderungsdefinition für Prozeßautomatisierungssysteme werden Bedingungen der folgenden Art festgelegt

- 1.) Es liegen nicht mehr als  $x$  Zeiteinheiten zwischen zwei Ereignissen, die von außen, d.h. vom technischen System, kommen. Damit wird z. B. das zyklische Auftreten von Statusmeldungen eines technischen Prozesses beschrieben, die als Unterbrechungsbotschaften vom Rechenprozeß rechtzeitig entgegengenommen werden müssen.
- 2.) Es liegen nicht mehr als  $x$  Zeiteinheiten zwischen einem äußeren Ereignis und einer Reaktion darauf. Damit wird das Antwortverhalten eines Rechenprozesses auf eine Botschaft, die vom technischen Prozeß gesendet wird, zeitlich festgelegt.
- 3.) Es liegen nicht mehr als  $x$  Zeiteinheiten zwischen einer Reaktion des Rechenprozesses und dem nächsten äußeren Ereignis. Damit wird eine Zeitgrenze

festgelegt, die zur Überwachung des technischen Systems auf Störfälle hin, benutzt wird. Meldet sich der Prozeß erst nach Ablauf der Überwachungszeit, wird das als Fehler erkannt und eine Fehlerreaktion eingeleitet.

- 4.) Es liegen nicht mehr als  $x$  Zeiteinheiten zwischen zwei Reaktionen des Prozeßsystems. Damit wird die Bearbeitungsdauer begrenzt, die bestimmt wird durch interne Berechnungen der Prozesse und/oder durch Verwaltungszeiten, die z. B. das Betriebssystem für die Prozeßumschaltung bei quasi-parallelen Prozessen benötigt.

Analog können diese Anforderungen als Minimum-Bedingungen formuliert werden, so daß mindestens  $x$  Zeiteinheiten zwischen Ereignissen liegen müssen.

Aus der aufgeführten Liste von Zeitbeziehungen zwischen den verschiedenen Ereignissen werden zwei unterschiedliche Aspekte der Formulierung von Leistungsanforderungen deutlich:

- Formulierung von Restriktionen, die durch ein implementiertes Prozeßsystem - und damit auch von der Spezifikation als abstrakte Beschreibung des Prozeßsystems - erfüllt werden müssen.

Die Erfüllung solcher Bedingungen, wie sie z.B. in Punkt 4) auftreten, kann durch Analyse der Spezifikation überprüft werden. Dies erfolgt durch Zuordnung von Ausführungszeiten zu einzelnen Aktionen und Untersuchung der möglichen Aktionsfolgen zwischen zwei Ereignissen, für die ein maximales/minimales Zeitintervall vorgegeben ist.

Die Analyse hat das Ziel, eine in der Spezifikation gegebene Problemlösung auf Übereinstimmung mit der Anforderungsdefinition zu überprüfen.

- Formulierung von Zeitbedingungen, die sich direkt auf Ablauffolgen im Spezifikationsmodell auswirken.

Der Ablauf wird abhängig davon, wann Ereignisse eintreten. Dies ist z. B. oben in der Forderung 3) der Fall. Der Rechenprozeß wartet auf das Eintreffen einer Meldung. Erhält er sie nicht "rechtzeitig", verzweigt er zu einer Fehlerreaktion, die im Normalfall nicht durchgeführt wird. In der Spezifikationssprache müssen Konstrukte vorhanden sein, um solche Bedingungen formulieren zu können.

Für beide Zwecke benötigen wir ein Modell, mit dessen Hilfe es möglich ist,

den Ablauf von Zeit zu beschreiben. In verteilten Systemen ist es schwierig, zu einem universellen Zeit- und Geschwindigkeitsbegriff zu gelangen, da das unabhängige Verstreichen von Zeit an verschiedenen Orten modelliert werden muß.

#### 4.2.1 Ein Modell zur Beschreibung von Zeit

Wenn wir den zeitlichen Ablauf eines Systems kommunizierender Prozesse beobachten wollen, benötigen wir eine Uhr, die es erlaubt, dem Auftreten von Ereignissen einen Zeitpunkt zuzuordnen.

In der Realität existieren keine globalen Uhren, die für zwei Orte die gleiche Zeit anzeigen (Heisenberg'sche Unschärferelation). Verteilte Systeme können immer nur über lokale Uhren verfügen, die für genau einen Prozessor gültig sind. Damit entstehen Probleme, Ereignisse an verschiedenen Orten in eine lineare zeitliche Reihenfolge zu ordnen, da für die Ordnung die Ganggenauigkeit der Uhren von Bedeutung ist. Gleichzeitigkeit von Ereignissen ist von der Übereinstimmung der einzelnen Uhren abhängig.

Prinzipiell kann man bei der Untersuchung von Spezifikationen von dieser Zeiteigenschaft abstrahieren und eine globale Uhr fordern. Bei der Analyse eines solchen Prozeßsystems wird eine globale Uhr als zusätzlicher Prozeß eingeführt. Dieser Prozeß synchronisiert das Zeitverhalten aller parallelen Prozesse. Ein solches Verfahren wird z.B. in [Stot 86] zur Analyse des Zeitverhaltens in 'Timed'-Petri-Netzen beschrieben. Das Fortschreiten der Zeit wird durch das "Ticken" des Uhrprozesses modelliert. Jeder Tick bewirkt einen Zustandsübergang im verteilten System und wird in der Markierung des Netzes festgehalten. Feuern können Transitionen jedoch erst, nachdem die Marken ein definiertes Zeitintervall auf einer Stelle verbracht haben. So kann ein Zustandswechsel allein durch das Ticken auftreten oder zusammen mit Feuern einer oder mehrerer unabhängiger Transitionen verbunden sein. Der zeitliche Abstand zwischen zwei Ereignissen ergibt sich dann durch die Anzahl der vollzogenen Zustandswechsel im System.

Diese Vorgehensweise hat den Nachteil, daß der Zustandsraum des betrachteten Systems durch die Einführung der globalen Uhr stark ausgeweitet wird. Dadurch tritt eine "Zustandsexplosion" auch schon bei kleineren Systemen auf. Außerdem ist die Einbettung der Uhr in botschaftsorientierte Spezifikationssprachen nur auf unnatürliche Weise möglich. Der Uhrprozeß muß entweder



als gemeinsames Objekt modelliert werden, was der Botschaftsorientierung widerspricht, oder als Prozeß, der an alle anderen Prozesse des Systems eine Zeitbotschaft versendet. Der Empfang solcher Botschaften ist in der Spezifikation von Benutzern aber nicht vorgesehen.

#### 4.2.2 Lokale und globale Uhren

Im folgenden soll nun ein Zeitmodell so konstruiert werden, daß es in die Spezifikationsmethode PASS auf einfache Weise integriert werden kann. Dazu wird ein System aus lokalen Uhren, die jeweils einen Prozessor zugeordnet sind, verwendet. Durch Vergabe von Zeitstempeln beim Botschaftsaustausch der Prozesse eines Prozeßsystems kann man eine globale Uhr konstruieren, so daß jeder Prozeß den globalen Zeitpunkt feststellen kann, zu dem ein Kommunikationsereignis im System stattfindet. Verwendet wird dazu ein Verfahren nach /Larnp 78/, das dahingehend erweitert wird, daß auch Gleichzeitigkeit durch synchronen Nachrichtenaustausch erfaßt wird.

Die Prozesse, die wir untersuchen, sind diskrete Systeme. Zustandsänderungen treten plötzlich und in zeitlich getrennten Schritten auf. Es genügt daher, das Modell einer virtuellen, diskreten Zeit zugrunde zu legen.

##### Definition 4-1: virtuelle Zeit

Eine virtuelle Zeit  $t \in \{T, \prec\}$  ist eine abzählbare Menge von Zeitpunkten, die durch die Relation " $\prec$ " wohlgeordnet ist. Durch " $\prec$ " ist eine irreflexive, transitive Ordnung gegeben.

Mit dieser Definition wird keine feste Metrik für Intervalle zwischen zwei aufeinanderfolgende Zeitpunkte angegeben. Der Ablauf von Zeit wird als nicht kontinuierlich, also nicht durch eine stetige Funktion beschrieben, angesehen. Die Zeit macht "Sprünge". Durch diese Definition der Zeit begrenzen wir den Zustandsraum in unserem Modell. Zur Darstellung von Zeitintervallen verwenden wir keine "Tick"-Ereignisse als Basis, die feste Abstände voneinander haben und zu Zustandsänderungen im System führen. Wir abstrahieren von diesen Basisereignissen und verwenden den Begriff der Zeitdauer.

#### Definition 4-2 : Zeitdauer

Sei  $T$  eine Menge von Zeitpunkten, mit beliebigen  $x, y, z \in T$  und einer "Früher-als" Relation .

Eine Zeitdauer ist eine Funktion  $d : T \times T \rightarrow \mathbb{N}_0$  mit den Eigenschaften

- 1)  $d(x,y) = 0 \Leftrightarrow x = y$
- 2)  $d(x,y) > 0 \Leftrightarrow x < y$
- 3)  $d(x,y) + d(y,z) = d(x,z)$ , mit  $x < y, y < z$

Mit  $d$  wird also ein zeitlicher Abstand zwischen zwei Zeitpunkten definiert. Damit können wir nun den Begriff der Aktivität einführen.

#### Definition 4-3 : Aktivität

Eine Aktivität ist ein Paar  $A = (e, d)$ , wobei  $e$  ein Element der Ereignismenge eines Prozesses ist und  $d$  den Abstand zum nächsten Zeitpunkt angibt, zu dem wieder ein Ereignis eintreten kann.

Diese Definition der Aktivität wird üblicherweise bei der Beschreibung von Prozessen vermieden; eine Aktivität wird in ein Paar von Ereignissen, die Anfang und Ende einer Aktivität markieren, zerlegt /Hoar 85/. Dadurch wird der Zustandsraum eines zu untersuchenden Systems allerdings unnötig vergrößert.

Mit dem Begriff der Aktivität werden zwei Ziele erreicht:

- In einem PASS-Spezifikationsmodell haben wir Kommunikationsanweisungen und interne Berechnungen als Aktionen. Diese Aktionen sind atomar, d.h. während ihrer Ausführung können in einem Prozeß keine weiteren Ereignisse auftreten. Mit dem Begriff der Aktivität können wir diesen atomaren Charakter bewahren. Eine Aktivität wird nur durch ihr Startereignis und eine Zeitdauer beschrieben.
- Mit dem Begriff der Aktivität können sowohl zeitorientierte als auch "zeitlose" Untersuchungen modelliert werden. Setzt man den Abstand  $d = 0$ , wird die Aktivität zur Aktion ohne Zeitdauer. Damit ist sie nur durch ein Ereignis beschrieben. Zur zeitorientierten Analyse von Spezifikationen kann der Abstand  $d$  vom Entwerfer vorgegeben werden. Damit sind Ausführungszeiten von Aktionen des Spezifikationsmodells beliebig dimensionierbar,

so daß der Einfluß unterschiedlicher Bearbeitungszeiten auf das Modellverhalten untersucht werden kann.

Um den zeitlichen Ablauf von Prozessen eines verteilten Systems zu modellieren, muß nun jeder Aktivität, d.h. jedem Startereignis einer Aktivität, ein Zeitpunkt  $C(x)$  zugeordnet werden.

**Definition 4-4 : Uhr**

Sei  $T$  eine Menge von Zeitpunkten.

Die Abbildung  $C: T \rightarrow \mathbb{N}_0$ , die jedem Zeitpunkt eine natürliche Zahl zuordnet, bezeichnen wir als Uhr.

Mit Hilfe der Uhr müssen drei Beziehungen zwischen Aktivitäten beschrieben werden;

- a) Senden und Empfangen bei synchronem Nachrichtenaustausch erfolgen gleichzeitig.
- b) Das Empfangen einer Nachricht, die asynchron gesendet wird, folgt dem Senden der Nachricht.
- c) Zwei aufeinander folgende Aktivitäten eines Prozesses folgen auch zeitlich aufeinander.

Im Fall a) existiert natürlich streng genommen keine echte Gleichzeitigkeit zwischen Sende- und Empfangsereignis. In einer Implementation sorgt ein Protokoll dafür, daß zwei Prozesse synchron Nachrichten austauschen können. Gemäß diesem Protokoll werden Nachrichten auf dem Transportmedium ausgetauscht, die die Synchronisation sicher stellen. Der zeitliche Aufwand dafür wird in der Aktivität "synchroner Nachrichtenaustausch" modelliert. Hier wird vereinfachend angenommen, daß das Protokoll beiden beteiligten Kommunikationspartnern gleichzeitig die Mitteilung macht, daß der Nachrichtenaustausch erfolgreich durchgeführt wurde und beide Prozesse damit ihre Aktivität "Senden" bzw. "Empfangen" beenden.

Im Fall b) wird dagegen die Tatsache formuliert, daß ein Prozeß eine gesendete Nachricht zu einem beliebigen Zeitpunkt erhalten kann. Für diesen Zeitpunkt wird nur gefordert, daß er nach dem Sendezeitpunkt liegt.

Mit den folgenden Regeln können wir die zeitliche Konsistenz der verteilten lokalen Uhren eines Systems sicherstellen /nach Lamp 78/:

Regel 1:

Bei synchronem Nachrichtenaustausch zwischen den Prozessen P und Q werden die Uhren  $C_p$  und  $C_q$  so gestellt, daß gilt:

$$C_p(x) + d(x) = C_q(y) + d(y)$$

wobei x das Sendeereignis und y das Empfangsereignis darstellen und  $d(x) = d(y)$  der zeitliche Aufwand zur Übertragung einer Nachricht ist.

Regel 2:

Sendet P eine Nachricht an Q, so wird  $C_p(x)$  als Zeitstempel übertragen: beim Empfangen wird sichergestellt, daß

$$C_p(x) + d(x) < C_q(y)$$

Regel 3:

Zwischen zwei Aktivitäten x und y des Prozesses P, die aufeinander folgen, stellt P die Uhr  $C_p$  weiter, so daß

$$C_p(x) + d(x) < C_p(y)$$

Bei allen Regeln bezeichnet  $d(x)$  die in der Spezifikation angegebene Zeitdauer, die zur Ausführung der Aktivität benötigt wird.

Durch Anwendung dieser Regeln bei der Bearbeitung von Aktivitäten erhalten wir eine logische globale Uhr, die sich als kartesisches Produkt der lokalen Uhren ergibt:

$$C = p \cdot c_p$$

Damit weiß jeder Prozeß, zu welchen globalen Zeitpunkt  $C(x)$  eines seiner Ereignisse stattfindet, und er kann erfahren, wann ein bestimmtes Ereignis in einem anderen Prozeß stattgefunden hat.

Für alle Aktivitäten x, y im System ist damit die Bedingung sichergestellt, daß gilt:

$$\text{wenn } x \rightarrow y, \text{ d.h. } y \text{ ist kausal abhängig von } x, \text{ dann } C(x) < C(y)$$

Mit dieser Bedingung ist es möglich, die Aktivitäten in einem verteilten System zeitlich zu ordnen und Kommunikation unter Zeitaspekten zu betrachten.

Durch die Konstruktion der verteilten Uhr ist auch ausgeschlossen, daß sich Nachrichten von einem Prozeß an einen anderen zeitlich überholen. Diese Eigenschaft wird auch von fast allen Protokollen zur Kommunikation in verteilten Systemen sichergestellt (z.B. durch Sequenznummern für Datenpakete).

### 4.3 Graphersetzunessysteme

In diesem Abschnitt werden die wesentlichen Begriffe aus der Theorie der Graphgrammatiken zitiert, die zur operationellen Definition der Semantik der Sprachelemente von PASS benötigt werden. Verwendet wird dabei der algorithmische Ansatz zur Definition von Graphersetzungs-systemen, wie er in /Nagl79/ zu finden ist. Er wird um einen geeigneten Attributierungsbegriff erweitert, um Zeitabläufe und Prioritätensteuerung darstellen zu können.

Benötigt wird der Begriff des attributierten, gerichteten, knoten- und kantenmarkierten Graphen zur Repräsentation von PASS-Ablaufsteuerungen in Form eines Programmgraphen. Mit Hilfe von Graphproduktionen wird die Semantik von PASS-Sprachelementen beschrieben. Die Menge aller Produktionen bildet einen abstrakten Interpreter, der die Ableitung aller zulässigen Zustandsübergänge in einem mit PASS spezifizierten Prozeßsystem erlaubt.

#### 4.3.1 Graphen und Einbettung von Uotereraphea

Im folgenden bezeichnet  $I_v$  eine endliche Menge von Knotenmarkierungen und  $I_e$  eine endliche Menge von Kantenmarkierungen.  $K$  sei eine endliche Menge von Knoten. Jeder Knoten kann über Attribute verfügen. Mit  $AB$  bezeichnen wir die Attributzuordnung  $AB: I_v \rightarrow P(ATT)$ .

$ATT$  ist die Menge aller zulässigen Attribute. Die Attributzuordnung  $AB$  legt für alle Knoten einer bestimmten Markierung fest, welche Attribute zugeordnet sind.

#### Definition 4-5 : Attributierter, knoten- und kantenmarkierter Graph

Ein attributierter, gerichteter, knoten- und kantenmarkierter Graph über  $E_v$  und  $E_p$  ist ein Quintupel  $d = (K, p, \beta, AB, val)$  mit

- $K$  ist endliche Knotenmenge.
- $p = \{ p_a \mid p_a \in E \times K \times K \}$  ist eine Menge von Relationen für beliebige  $a \in I_p$ .
- $\beta : K \rightarrow E_y$  ist die Knotenmarkierungsfunktion, die jedem Knoten ein Symbol aus  $E_y$  zuordnet.
- $AB : E_y \rightarrow \text{PUTT}$  ist Attributzuordnung, die angibt, über welche Attribute ein Knoten verfügt.
- $val : K \times A \rightarrow I_B$  ist eine Funktion, die jedem Knoten einen Attributwert zuordnet.  $WB$  gibt den zulässigen Wertebereich eines Attributs an.

Mit  $d(2_v, r, E, AB)$  bezeichnen wir die Menge aller Graphen über  $2_v$  und  $I_E$  unter der Attributzuordnung  $AB$ .

Ein Element  $e = (k_1, k_2) \in p_a$  wird als gerichtete Kante, die vom Knoten  $k_1$  zum Knoten  $k_2$  führt, aufgefaßt. Die Kante ist mit  $a \in L_E$  markiert. Eine mit  $a$  markierte Kante bzw. ein mit  $v$  markierter Knoten heißt  $a$ -Kante bzw.  $v$ -Knoten.

Zwischen zwei Knoten darf es mehrere Kanten geben, die sich jedoch in ihrer Markierung und/oder Richtung unterscheiden müssen. Durch Angabe der Quell- und Zielknoten, sowie der Markierung, werden Kanten eindeutig bezeichnet.

Knoten werden durch die Knotenbezeichnung identifiziert. Graphisch stellen wir Knoten durch Rechtecke dar. Die Knotenmarkierung steht innerhalb des Symbols, die Knotenbezeichnung außerhalb.

Eine  $a$ -Kante vom Knoten  $k_1$  zum Knoten  $k_2$  wird als Pfeil von  $k_1$  nach  $k_2$  gezeichnet, der die Markierung  $a$  trägt. Der Übersichtlichkeit wegen werden wir Kantenmarkierungen bei der Darstellung von PASS-Programmgraphen durch unterschiedliche Pfeilkanten ausdrücken (z.B. Doppelpfeil  $\Leftrightarrow$  oder einfachen Pfeil  $\rightarrow$ ).

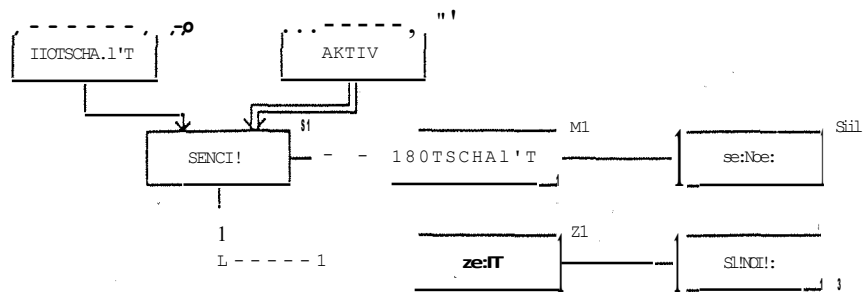


Bild 4-2 : Ein knoten- und kantenmarkierter Graph

Dieser Graph enthält die Knoten  $\{A1, M0, M1, S1, S2, S3, Z1\}$ . Das Knotenmarkierungsalphabet  $!_y$  ist gegeben durch  $!_y = \{\text{Aktiv, Sende, Zeit, Botschaft}\}$ , das Kantenmarkierungsalphabet  $!_E$  durch  $!_E = \{\Rightarrow, -->, - ->\}$ .

Dieser Graph repräsentiert folgenden Ausschnitt aus einer PASS-Ablaufsteuerung.

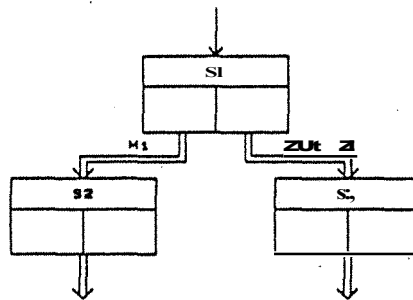


Bild 4-3: PASS-Ablaufsteuerung

Der gezeigte PASS-Prozeß enthält die Kommunikationsanweisungen S1, S2 und S3. Mit der Anweisung S1 wird die Botschaft M1 gesendet. Der Einfachheit halber ist in diesem Beispiel der Empfänger weggelassen worden. Der Nachfolgezustand von S1 ist S2. Der Botschaftsaustausch ist zeitüberwacht, mit Nachfolgezustand S3.

Zur Darstellung der Zeitüberwachung benötigen wir noch die Attributierung der Knoten. Dazu sei die Menge der Attribute gegeben durch

$AIT = \{\text{Timeout, Uhr, Überwachung}\}$ .

Wir definieren eine Attributzuordnung AB:

Aktiv {Uhr, Überwachung}

Sende {Timeout}

Der Knoten mit der Markierung *Aktiv* erhält also die Attribute *Uhr* und *Überwachung*, die Knoten mit der Markierung *Sende* das Attribut *Timeout*.

Mit Hilfe der Funktion *val* ordnen wir dem Knoten SI die Überwachungsdauer 5 Zeiteinheiten zu :  $val(SI, \text{Überwachung}) := 5$

Der Wertebereich für die Attribute *Überwachung* und *Ausführung* sei vom Typ INTEGER . Für die übrigen Knoten gelten die folgenden Initialisierungen:

$val(A1, \text{Uhr}) := 0;$

$val\{A1, \text{Überwachung}\} = 0;$

$\forall k_i \in K \text{ mit } \beta(k_i) = \{\text{Botschaft}\} \text{ gilt: } val(k_i, \text{Ausführung}) := 2;$

Nach diesem Prinzip können wir PASS-Ablaufsteuerungen durch attributierte, gerichtete, knoten- und kantenmarkierte Graphen darstellen.

Definition 4-6 : Untergraph

*Ein attributierter, markierter Graph  $d'$ ,  $d' \in \mathcal{E}_Y, \mathcal{E}_E AB$  heißt Untergraph von  $d$ ,  $d' \in \mathcal{E}_V, \mathcal{LE}, AB$ , geschrieben  $d' \in d$  gdw.*

- $K' \in K$ ,
- $\forall a, \mathcal{LE} : p \wedge s (Pa n (K' \times K'))$ , d.h. Jede Kante, die zwei Knoten aus  $K'$  verbindet, gehört zu  $d'$ .
- $\beta' = \beta|_{K'}$ , d. h. die Knotenmarkierungsfunktion wird eingeschränkt auf  $K'$ .
- $AB' = AB|_{K'}$ , d. h. die Attributzuordnung ist genauso wie für  $K$  eingeschränkt auf den Bereich  $K'$ .

Die Definition besagt, daß bei dem Test, ob  $d'$  Untergraph von  $d$  ist, nur die Struktur des Graphen mit dem Wirtsgraphen verglichen werden muß. Die Attributzuordnung eines Knoten  $k' \in d'$  muß die gleiche wie die des entsprechenden Knotens  $k \in d$  sein muß. Aktuelle Attributwerte müssen nicht übereinstimmen.



**Definition 4-7: Komplementärer Graph**

Sei  $d' \in d$  mit  $d', d, d(E, E) \in ABJ$ .

Der Untergraph  $d(K-K')$  ist der zu  $d'$  komplementäre Untergraph (Kurzschreibweise:  $d - d'$ ).

$K-K'$  bezeichnet die Menge aller Knoten  $k, K$ , die nicht in  $K'$  enthalten sind.

Im folgenden werden einige Definitionen aufgeführt, die wir benötigen, um die Kanten zwischen einem Untergraphen und seinem komplementären Graphen zu beschreiben und die Quell- bzw. Zielknoten von ein- bzw. auslaufenden Kanten des Untergraphen zu bestimmen.

Wir führen folgende Bezeichnungen ein:

$$\text{In}_a(d', d) = \text{Pa}_n((K, K') \times K).$$

$$\text{Out}_a(d, d') = \text{Pa}_n(K' \times (K, K')).$$

Durch  $\text{In}_a(d', d)$  ist die Menge aller a-Kanten, die von  $d-d'$  nach  $d'$  führen, durch  $\text{Out}_a(d, d')$  die Menge aller a-Kanten, die in umgekehrte Richtung führen, festgelegt.

**Definition 4-8 : Einbettung**

Das Tupel  $EM(d, d') = (\text{In}_\delta(d', d), \text{Out}_\delta(d, d'))$  heißt Einbettung von  $d'$  in  $d$ .

Die Einbettung legt alle Kanten zwischen dem Untergraphen  $d'$  und dem komplementären Graphen fest. Um die zugehörigen Quell- und Zielknoten in dem zum betrachteten Untergraphen komplementären Graphen zu beschreiben, werden Ausdrücke verwendet, die durch Kombination von Operatoren entstehen.

#### Definition 4-9 : Operator

Sei  $E ::= EV \vee \{ La / \parallel ' EE \} \vee \{ Ra \text{ falls } a \in \mathbb{R}E \} \vee \{ C, v, \cup \}$ .

Operatoren sind Wörter aus  $E$ , die ausschließlich nach folgenden Regeln gebildet sind:

a)  $La$  und  $Ra$  sind Operatoren für beliebige  $a \in E$ .

b) Falls  $E$  ein Operator ist, so ist

$\{CE\}$  und  $\{FE\}$  ein Operator, mit  $v \in E$ .

c) Falls  $E$  und  $F$  Operatoren sind, so sind

$EF$ ,  $(E \vee F)$  und  $(E \cap F)$  Operatoren.

$op(E)$  sei die Menge aller nach diesem rekursivem Schema gebildeten Wörter aus  $E$ .

Die Operatoren  $L_a$ ,  $R_a$  für  $a \in E$  heißen Elementaroperatoren, alle anderen zusammengesetzte Operatoren.

#### Definition 4-10 : Interpretation

Seien  $d, d', d(EV, EE, AB)$ ,  $d'E, k, K$  und  $E, F, op(E)$ .

Eine Interpretation ordnet für beliebige  $d, d', k$  einem Operator  $E$  eine Teilmenge der Knotenmenge  $K-K'$ , was als  $A^{d',d}(k)$  geschrieben wird, zu:

a)  $L^{d',d}(k) := \{ l \in K-K' \mid (l, k) \in p_g \}$

ist die Menge aller Knoten aus  $K-K'$ , von denen eine 11-Kante zu  $k$  läuft ( $L$  steht als Abkürzung für "links von").

$R^{d',d}(k) := \{ k \in K-K' \mid (k, l) \in p_g \}$

ist die Menge aller Knoten aus  $K-K'$ , zu denen eine 11-Kante von  $k$  aus führt ( $R$  steht als Abkürzung für "rechts von").

b)  $(CAJ^{d',d}(k)) := (k \in K-K' \mid \neg (k \in L^{d',d}(k) \cup R^{d',d}(k)))$

liefert das Komplement von  $A^{d',d}(k)$  in  $K-K'$ .

$$(v_1 \dots v_n A^{d,d}(k)) := (k / k, A^{d,d}(k) \cup B(k), (v_1, \dots, v_{n-1}))$$

bestimmt die Untermenge von  $A^{d,d}(k)$ , deren Knoten mit einem der Zeichen des Wortes  $v_1 v_2 \dots v_n$  markiert sind.

$$cl A \cdot B^{d,d}(kl) := (k / 3 k' : k', acJ^{d,d}(kl) \cup k, A^{d,d}(k)) J$$

ist die Menge aller Knoten, die sich durch die Hintereinanderschaltung von Operatoren ergibt.

$$(A \cup B)^{d,d}(k) := A^{d,d}(k) \cup B^{d,d}(k)$$

$$(A \cap B)^{d,d}(k) := A^{d,d}(k) \cap B^{d,d}(k)$$

sind die Mengen, die sich durch Verzweigung bzw. Parallelschaltung von Operatoren ergeben.

Im folgenden seien einige Beispiele für Operatoren und ihre Interpretationen gegeben. Wir nehmen an, daß der in Bild 4-2 abgebildete Graph als Untergraphen den Graphen  $d'$  mit einem Knoten  $A1$  besitzt.

- $CR_{\rightarrow}(A1)$  ist dann die Menge aller Knoten  $k \in K - K_0$ , die nicht Zielknoten von  $\bullet \rightarrow$ -Kanten sind, die von  $A1$  ausgehen. Es ergibt sich die Menge  $\{M0, M1, Z1, S2, S3\}$ .
- $L R_{\rightarrow}(A1)$  liefert den Knoten  $M0$ .
- $Sende R R_{\rightarrow} / A1$  liefert alle Knoten  $k \in K - K'$  die mit *Sende* markiert sind und über die angegebene Kantenfolge erreichbar sind. Dies sind die Knoten  $S2$  und  $S3$ .

#### 4.3.2 Graphersetzen

##### 4.3.2.1 Sequentielle Graphersetzen

Sequentielle Graphersetzungssysteme sind eine Verallgemeinerung der formalen Chomsky-Grammatiken über Zeichenketten. Ein Ableitungsschritt in einer Grammatik über Zeichenketten besteht darin, ein Teilwort einer gegebenen Zeichenkette, das in der linken Seite einer Produktion enthalten ist, durch das Wort der rechten Seite der Produktion zu ersetzen. Dieser Ersetzungsvor-

gang wird auf Graphgrammatiken übertragen. Allerdings reicht hier die Angabe einer Regel  $(d_l, d_r)$ , die die Ersetzung eines Untergraphen  $d_l$  in einem Wirtsgraphen  $d$  durch den Graphen  $d_r$  fordert, nicht aus.

Zusätzlich muß angegeben werden, wie der Untergraph  $d_r$  in den Wirtsgraphen eingebettet werden soll. Dazu muß festgelegt werden, durch welche Kanten die Knoten von  $d_r$  mit beliebigen Knoten des Wirtsgraphen verbunden sein sollen. Dazu verwendet man die oben eingeführten Operatoren.

Für ein- oder auslaufende Kanten der rechten Seite einer Ersetzungsregel kann explizit angegeben werden, von welchen beliebig "weit entfernten" Knoten des Wirtsgraphen sie wegführen bzw. in welchem Knoten sie enden sollen.

#### Definition 4-11 : Attributierte Graph-Produktion

**Eine attributierte Graphproduktion über  $E \rightarrow E$  ist ein 4-Tupel**

**$p = (d_l, d_r, E, BR)$  mit**

- **$d_l, d_r, d (E \rightarrow E)$ , wobei  $d_l$  die linke Seite und  $d_r$  die rechte Seite der Produktion  $p$  genannt werden.**
- **$E$  die Einbettungsüberführung ist, beschrieben durch**  

$$E = (l_a, r_a, E, E), \text{ mit den Einbettungskomponenten}$$

$$l_a = \bigcup_{\lambda=1}^q A_\lambda(k'_\lambda) \times \{k''_\lambda\} \text{ und}$$

$$r_a = \bigcup_{\lambda=1}^m \{k''_\lambda\} \times A_\lambda(k'_\lambda),$$

wobei  $k'_\lambda \in K_l, k''_\lambda \in K_r, A_\lambda, op(I;J, m \geq 1, q \geq 1)$  und  $a, E$  vorausgesetzt wird.

- **$BR$  ist eine endliche Menge von Attributberechnungsregeln mit**  

$$BR = \{br | i \in WB(att, J) \rightarrow \exists WB(attn) \rightarrow WB(att) \mid att, ATTr, \text{ wobei}$$

die  $att$  mit  $1 \leq i \leq n$  und  $n = n(p, att)$  Attribute von Knoten  $k_j, K_j$  mit  $1 \leq j \leq |K_j|$  sind, und

$att$  ein Attribut eines Knoten  $k, K$ .

$ATTr$  ist die Menge der Attribute von Knoten des Teilgraphen  $d_r$

Eine Attributberechnungsregel ordnet Attributen von Knoten der rechten Sei-

te einer Produktion neue Attributwerte zu. Diese Attributwerte werden berechnet aus den Werten einiger Attribute von Knoten der linken Seite. Welche Attribute benötigt werden, ist abhängig von den aktuellen Produktionen.

Beispiel:

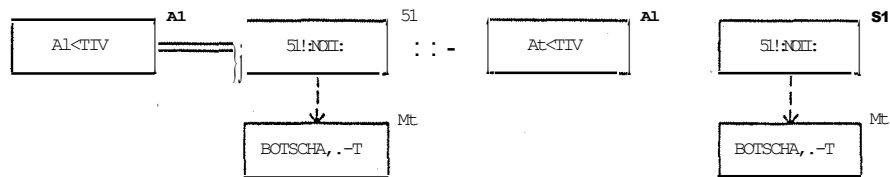


Bild 4-4: Produktion

E  $E_d$  erweitert um  $r_{\rightarrow}(A1) \cdot (A1, R(M1))$

BR:  $brf_d$  erweitert um  $val(A1, Uhr) : \bullet val(A1, Uhr) + val(M1, Ausführung)$

Durch die Kurzschreibweise Eid wird ausgedrückt, daß alle Einbettungskanten unverändert übernommen werden. Eid heißt daher identische Einbettung. Zusätzlich wird eine neue Einbettungskante durch die Erweiterung eingeführt. Es soll eine Doppelpfeilkante vom Knoten A1 zum Pfeilnachfolger von M1 führen.

Analog wird die Attributierung identisch übernommen. Im Beispiel setzen wir folgende Attributierung voraus:

AB: Aktiv {Uhr, Überwachung}  
Botschaft {Ausführung}

Die Berechnungsregel gibt an, daß die Attribute für alle Knoten, außer dem Knoten A1, unverändert bleiben. Im Knoten A1 wird das Attribut *Uhr* verändert.

Mit Hilfe derartiger Produktionen können wir festlegen, wie Graphen sequentiell ersetzt werden.

**Definition 4-12 : Direkt sequentiell ableitbar**

Ein markierter, attributierter Graph  $d' \in d\{Ev, E_F, AB\}$  heißt **aus**  
 $d, d(Ev, :[E, AB])$  **direkt sequentiell ableitbar mittels der Produktion**  
 $p = (d_1, dr E, BRJ, gdw.$

a)  $d_1 \neq d, dr \subseteq d', d - d_1 = d - dr$ .

b)  $In_a(d_r d') = I_a^d 1^d$  und

$Out; /dr d' = r_a^d 1^d$  für beliebige  $a, EE$ .

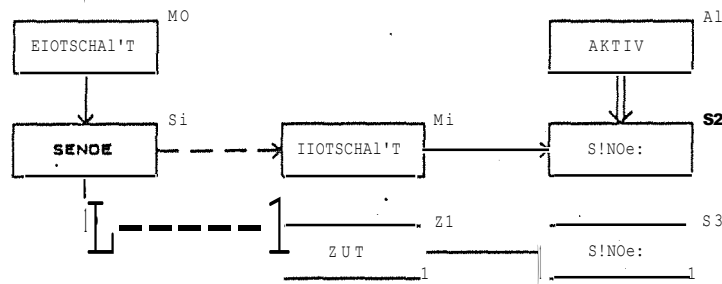
c)  $val(k'.att) = val\{k, att\} \forall att, AB(k')$  und

$k, kfd-d_1, k', k(d-d_1)$ .

Die Definition a) drückt die Ersetzung des Untergraphen  $d_1$  durch  $d$ , aus; b) legt fest, wie die ein- und auslaufenden Kanten von  $dr$  zum unveränderten Teil des Wirtsgraphen festgelegt werden; c) fordert, daß Attribute vom Knoten des Wirtsgraphen, die nicht in  $d$ , enthalten sind, unverändert bleiben. Für die Knoten von  $d$ , legt die Produktion durch die Berechnungsregeln fest, welche Attribute unverändert bleiben und welche neu berechnet werden.

**Beispiel:**

Durch Anwendung der Produktion  $p_1$  aus obigem Beispiel auf den Graphen in Bild 4-2 wird folgender Graph abgeleitet



mit dem Attributwert  $val(A1, Uhr) \cdot 2$

**Bild 4-5: Abgeleiteter Graph**

Die Produktion  $p_1$  löscht im Graphen die Doppelpfeilkante, die vom Knoten A1 zum Knoten S1 führt. Die Einbettungsüberführung gibt an, daß eine solche Kante aus dem eingesetzten Untergraphen  $d$ , zu dem Knoten gezogen werden

soll, der Ober eine Pfeilkante mit S2 verbunden ist. Die Attributierung wird nur im Knoten A1 verändert. Dort wird die Uhr um den Inhalt von Ausführung erhöht.

Nach diesen Definitionen können wir den Begriff des sequentiellen Graphersetzungssystems festlegen.

**Definition 4-13 : Sequentielles Graphersetzungssystem**

*Ein sequentielles Graphersetzungssystem ist ein Tupel*

$G = \langle E_v, E_p, A_v, A_p, d, P, AB, -, \text{val} \rangle$ , mit

- $E_v, E_p$  endliche, nichtleere Mengen, das Knoten- und das Kantenmarkierungsalphabet.
- $A_v \subseteq E_v, A_p \subseteq E_p$  heißen terminales Knoten- und Kantenmarkierungsalphabet,
- $E_v - A_v$  und  $E_p - A_p$  heißen nichtterminal markierte Knoten- und Kantenmarkierungsalphabet.
- $d(E_v, E_p, AB)$  heißt Startgraph.
- $P$  sei eine endliche Menge von Produktionen  $p = (d, dr, E, BR)$  mit  $d \in E_v, dr \in E_p, BR \subseteq E_v \times E_p$  mit  $d \in E_v, dr \in E_p$  u  $(d;)$ .
- $-;$  sei die direkte sequentielle Ableitung gemäß Definition 4.3-8.

Leitet man aus einem gegebenen Startgraphen mit Hilfe einer Grammatik beliebige Graphen ab, so enthalten diese im allgemeinen sowohl terminal als auch nichtterminal markierte Knoten und Kanten. Solche abgeleiteten Graphen heißen Graphsatzformen. Graphsatzformen können weiter abgeleitet werden, bis nur noch terminal markierte Knoten und Kanten im Graphen enthalten sind.

#### 4.3.2.2 Programmierte Ersetzungen

Die Theorie der Graphersetzungssysteme zeigt, daß die bis hierher eingeführten Graphgrammatiken universell sind, d.h., daß damit alle aufzählbaren Graphsprachen erzeugt werden können. Um jedoch mit Graphgrammatiken Sachverhalte einfach darstellen und Ableitungsschritte kontrolliert durchführen zu können, werden programmierte Graphersetzungssysteme eingeführt. Solche Ersetzungs-

systeme enthalten Kontrolldiagramme, die die Anwendung von Produktionen steuern. Mit sequentiell programmierten Ersetzungsschritten können komplexe Graphmanipulationen überschaubar formuliert werden. Durch Hinzunahme von steuernden Kontrolldiagrammen werden umfangreiche Einbettungsüberführungen in einfachere überführt, was sich positiv auf die Übersichtlichkeit, das Maß an Zuverlässigkeit und die Wartbarkeit der Graphalgorithmen auswirkt /Nagl79/;

#### **Definition 4-14 : Kontrolldiagramm**

Ein Kontrolldiagramm ist ein zusammenhängender, markierter Graph. Das Knotenmarkierungsalphabet umfaßt die Produktionenmenge  $P$ , das Kantenmarkierungsalphabet ist die Menge  $\{Ja, Nein\}$ . Der ausgezeichnete Startknoten ist minimal und es gibt immer einen Haltknoten (maximaler Knoten). Bei der Knotenmarkierung handelt es sich entweder um die Produktionsnamen oder um die Namen weiterer Kontrolldiagramme (Unterkontrolldiagramm). Je nach Art der Knotenmarkierung spricht man von Produktionsknoten oder Aufrufknoten. Die Kanten sind entweder mit "Ja" oder mit "Nein" markiert.

Zwischen zwei Knoten gibt es höchstens eine Kante. Führt aus einem Knoten eine mit Nein markierte Kante heraus, so besitzt der Knoten genau eine weitere, Ja-markierte Kante. Besitzt ein Knoten nur Ja-markierte, herauslaufende Kanten, so können dies beliebig viele sein. Ein Aufrufknoten besitzt nur Ja-markierte auslaufende Kanten.

#### **Definition 4-15 : sequentiell, programmiert ableitbar**

*Sei  $kd$  ein Kontrolldiagramm und seien  $kd_1, \dots, kd_n$  die zugehörigen Unterkontrolldiagramme.*

*Seien  $D, D', D(I \rightarrow v, EE, AB)$ .*

*$D'$  ist aus  $D$  mittels des Kontrolldiagramms  $kd$  und seiner Unterkontrolldiagramme sequentiell ableitbar, gdw. es einen Berechnungsdurchlauf durch  $kd$  gibt, der mit dem Startknoten von  $kd$  beginnt, mit einem Haltepunkt von  $kd$  endet und bei dem es eine Folge  $D = D_1 \dots D_n = D'$  gibt, so daß*



- entweder  $D_{\mu} \rightarrow D_{\mu-1}$  für jede Knotenmarkierung  $(k)$  und Kantenmarkierung  $= T$
- oder  $D_{\mu} = D_{\mu+t}$  und Kantenmarkierung  $= F$ ist, für  $t \leq \mu \leq m-1$

Bei dem Berechnungsdurchlauf werden der durchlaufene Knoten, die Knotenmarkierung und die Kantenmarkierung in der Reihenfolge des Durchlaufs notiert. Auf diese Weise erhält man die Abwicklung des Kontrolldiagramms.

**Definition 4-17 : Programmierter Ersetzungsschritt**

Ein programmierter Ersetzungsschritt ist ein Durchlauf durch ein Kontrolldiagramm, wobei als primitive Aktionen nur sequentielle Graphersetzungen ausgeführt werden.

In Kapitel 4.4 werden programmierte Ersetzungen zur Beschreibung einer Sendoperation in PASS gezeigt.

#### 4.4 Eine Grapharammatik für PASS

In diesem Abschnitt soll mit Hilfe der definierten Begriffe die Struktur einer PASS-Spezifikation durch einen Programmgraphen repräsentiert werden. Graphproduktionen beschreiben die Semantik von PASS-Sprachelementen operationell. Durch Anwendung der Produktion können alle erlaubten Zustände eines verteilten Systems abgeleitet werden.

In [Fede 86] wurden der Programmgraph und die Produktionen bereits beschrieben, deswegen soll hier nur eine kurze Übersicht gegeben und die Erweiterungen durch die Attributierung erläutert werden. In Kapitel 5 wird dann ein Mechanismus zur Steuerung der Ableitungsschritte vorgestellt.

##### 4.4.1 Der Programmgraph

Die Grundlage für die Erstellung eines Programmgraphen ist die PASS-Ablaufsteuerung eines Prozesses. Im Programmgraphen wird die Information notiert, in welcher Reihenfolge Kommunikationsanweisungen und interne Operationen/-

Funktionen ausgeführt werden bzw. welche Alternativen zur Auswahl von Ausführungsfolgen bestehen. Dadurch ist die statische Semantik eines Prozesses gegeben.

Der Programmgraph wird ergänzt um Verwaltungsknoten, die z.B. den Prozesszustand (blockiert, aktiv usw.) und die Zuordnung von Prozessen zu Prozessoren repräsentieren. Die Produktionen des Graphersetzungssystems wirken in erster Linie verändernd auf diese Verwaltungsknoten. Der durch die PASS-Ablaufsteuerung gegebene Anteil des Programmgraphen bleibt konstant.

Im Programmgraphen werden die Knoten und Kanten der PASS-Ablaufsteuerungen stets als Knoten dargestellt. Knoten aus PASS bezeichnen wir im folgenden als Aktionsknoten; Kanten in PASS als Namensknoten. Kanten des Programmgraphen stellen Reihenfolge- oder Zuordnungsbeziehungen zwischen verschiedenen Knoten her.

In Bild 4-6 ist das verwendete Knoten- und Kantenmarkierungsalphabet beschrieben. Die Markierung der Kanten des Programmgraphen wird aus Gründen der Übersichtlichkeit durch unterschiedliche graphische Symbole notiert.

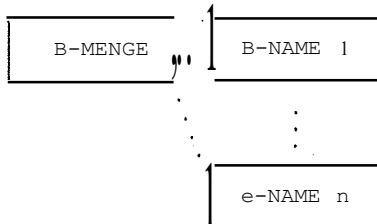
Aktionsknoten mit den Markierungen *Sende* bzw. *Empfange* repräsentieren Kommunikationsknoten von PASS. Jedem solchen Knoten ist durch einen Einfachpfeil, der zu einem Sammelnamensknoten führt, eine Menge von Botschaften zugeordnet. Führen mehrere Kanten zu verschiedenen Sammelnamensknoten, können die mit dem Namensknoten verbundenen Botschaften alternativ gesendet oder empfangen werden. Botschaften, die durch eine "UND"-Verknüpfung zusammengehören, werden über genau einen Sammelnamensknoten erreicht. Vom Sammelnamensknoten führen mehrere gepunktete Kanten zu den konjunktiv verknüpften Einzelnamensknoten. Bei Bearbeitung der zugehörigen Aktionsknoten müssen alle so verbundenen Botschaften gleichzeitig ausgetauscht werden können.

Durch einen einfachen Pfeil als Kante wird zwischen zwei Aktionsknoten über den Sammelnamensknoten die Reihenfolgebezeichnung hergestellt. Führt aus einem Knoten der PASS-Ablaufsteuerung keine Kante heraus, wird der Knoten im Programmgraphen mit der terminalen Markierung *Stop* bezeichnet.

Die Art der Kommunikation zwischen zwei Prozessen wird in Synchron- bzw. Wartebereichsknoten festgehalten.



**Aktionsknoten mit Markierung:**  
**Stop, Sende, Empfang oder Funktion**



**Namensknoten, wobei B-Name i durch die i-te Botschaft im Programmgraphen an dieser Stelle ersetzt werden muß.**



**Prozeßzustandsknoten mit der Markierung:**  
**Aktiv (A), Lauffähig (L), Wartend (W),  
 Ruhend (R)**



**Verwaltungsknoten für einen Prozessor**



**globaler Verwaltungsknoten für alle  
 Prozessoren**



**Befehlszählerkante**



**Eintragskante in Wartebereich**



**Nachfolgeknoten-Kante**



**Kante vom Sammelnamensknoten zu  
 den Einzelnamensknoten**



**·Knoten für synchronen Nachrichtenaustausch**



**Wartebereichsknoten für asynchronen  
 Nachrichtenaustausch**

**Bild 4-6 : Knoten- und Kantenmarkierungsalphabet /nach Fede 86/**

Zu jedem Prozeß existiert ein Prozeßknoten, dessen Markierung den aktuellen Zustand des Prozesses, entsprechend dem in Kap. 4.1 diskutierten Zustandsmodell wiedergibt. Über einen Doppelpfeil ist der aktuell betrachtete Knoten des zugehörigen Prozesses erreichbar. Diese Kante hat die Funktion eines Befehlszählers. Über die Prozessorknoten werden Prozesse genau einem Prozessor zugeordnet. Über Nachfolgekanten (einfacher Pfeil) sind die Prozeßknoten aller Prozesse, die auf einem Prozessor quasi-parallel bearbeitet werden sollen, erreichbar. Die Doppelpfeilkante führt zum Prozeßknoten des Prozesses, der gerade den Prozessor zugeteilt bekommen hat, also aktiv ist.

Über einen globalen Verwaltungsknoten sind alle Prozessoren des verteilten Systems durch Nachfolgekanten verbunden.

#### 4.4.2 Attributierung

Den verschiedenen Knoten sind Attribute zur Darstellung von Zeit, Priorität und Wartebereichsbelegung zugeordnet. Knoten, die mit *Sende* bzw. *Empfange* markiert sind, erhalten das Attribut *Timeout*. Es wird aus den entsprechenden Kommunikationsknoten der PASS-Ablaufsteuerung übernommen.

Zur Modellierung des Zeitbedarfs für die Durchführung einer Kommunikation wird sowohl den Kommunikationsknoten als auch den mit *Botschaft* markierten Knoten das Attribut *Ausführungszeit* zugeordnet. Damit kann der Kommunikationsaufwand in zwei Anteile zerlegt werden:

- Zeitverbrauch durch Aufruf des Kommunikationsdienstes eines Betriebssystems durch das Attribut im Kommunikationsknoten.
- Zeitaufwand für die Übertragung der entsprechenden Botschaft an den entfernten Partner durch das Attribut im *Botschafts-Knoten*.

Der Botschaftsaustausch kann also entsprechend dem in Bild 4-7 dargestellten Zeitdiagramm nachgebildet werden.

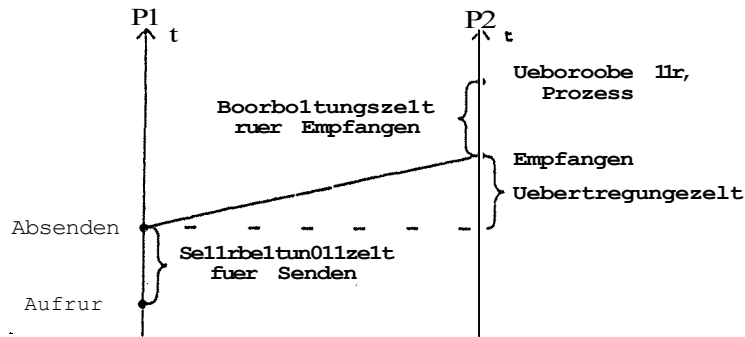


Bild 4-7 : Zeitablauf bei der Kommunikation

Damit können sowohl die Eigenschaften des Transportmediums zum Nachrichtenaustausch als auch das Zeitverhalten des Betriebssystems, das in einer Implementierung verwendet wird, im Spezifikationsmodell untersucht werden. Die Initialisierung der Attribute mit konkreten Zeitwerten erfolgt über eine spezielle Dialog-Schnittstelle, da die Werte nicht in der PASS-Ablaufsteuerung angegeben werden sollen. Dies hat den Vorteil, daß eine Spezifikation zunächst zeitunabhängig vorgenommen werden kann. Sind keine Attributwerte definiert, werden sie bei der Anwendung von Graphproduktionen und Ausführung der zugehörigen Berechnungsregeln mit dem Wert 0 vorbesetzt. Damit ist eine Untersuchung des Spezifikationsmodells ohne Berücksichtigung der Zeit möglich.

Bei der Definition von Ausführungszeiten sind mehrere Vorgehensweisen möglich:

- Alle Ausführungszeiten  $t_s$ ,  $t_e$  und  $t_u$  erhalten jeweils für alle Knoten einen bestimmten Wert. Damit kann ein homogenes, verteiltes System charakterisiert werden, in dem auf allen Prozessoren dasselbe Betriebssystem arbeitet und das Übertragungsnetzwerk feste Übertragungszeiten, unabhängig von Entfernung und Botschaftstyp, garantiert. Für Realzeitsysteme wird in der Regel ein solches Übertragungsmedium gefordert. Man verwendet deswegen das Token-Ring-Verfahren für Nahverkehrsnetze. Es garantiert maximale Übertragungszeiten, die nicht überschritten werden.
- Für Teilmengen von Knoten werden gleiche Ausführungszeiten definiert. Angewandt auf Zeiten  $t_u$  für die Übertragung, kann damit ein inhomogener Netzaufbau nachgebildet werden. Liegt zum Beispiel eine Netzwerkstruktur zugrunde, die aus Prozessen besteht, die in einem Nahverkehrsnetz kommu-

nizieren, aber auch über ein Weitverkehrsnetz mit einer bestimmten Teilmenge anderer Prozesse, kann das entsprechende Zeitverhalten angegeben werden. Alle *Botschafts-Knoten* mit Botschaften an Prozesse im Nahverkehrsnetz bzw. Weitverkehrsnetz erhalten jeweils einheitliche Werte  $t_u$ . Analoges Vorgehen ist auch für inhomogene Betriebssysteme vorgesehen. In diesem Fall sind die Kommunikationsknoten betroffen.

- Jedem Kommunikationsknoten bzw. *Botschafts-Knoten* wird explizit eine der Zeiten  $t_u$  bzw.  $t_s$  zugeordnet.

*Botschafts-Knoten* erhalten zusätzlich zur Ausführungszeit die Attribute *Zeitstempel* und *Priorität*. Bei der Anwendung von Sende- und Empfangsproduktionen muß, wie in Kapitel 4.2, diskutiert wurde, das Kausalitätsprinzip gewahrt bleiben. Eine Botschaft kann erst dann empfangen werden, wenn sie vorher gesendet wurde. Entsprechend den Regeln zur Konstruktion einer globalen Uhr, kann die Empfangsproduktion nur zu einem Zeitpunkt angewendet werden, der zeitlich nach dem Sendezeitpunkt liegt. Im Attribut *Zeitstempel* wird daher der Absendezeitpunkt  $t_{fu}$  jeder Botschaft festgehalten. Dieses Attribut wird dynamisch bei der Ableitung von Programmgraphen berechnet.

Mit Hilfe des Attributs *Priorität*, das in der PASS-Ablaufsteuerung angegeben wird und stets konstant bleibt, wird beim alternativen Senden bzw. Empfangen von Botschaften eine Auswahl getroffen. Ist es möglich, mehrere alternative Kommunikationsanweisungen auszuführen, wird die mit der höchsten Priorität bevorzugt. Zulässig sind ganzzahlige Prioritätswerte  $P_p$ , wobei entsprechend üblicher Praxis niedrigere Werte höherer Dringlichkeit entsprechen.

Dem Prozeßknoten ist ebenfalls das Attribut *Priorität* zugeordnet. Hier wird es als Auswahlkriterium bei der Prozessorvergabe an einen Prozeß aus einer Menge lauffähiger Prozesse verwendet.

Prozessorknoten tragen die Attribute *Uhr* und *Überwachung*. Mit Hilfe von *Uhr* wird die lokale Uhr eines jeden Prozessors gemäß dem Zeitmodell modelliert. Bei der Anwendung von Produktionen, die die Bearbeitung von Kommunikationsanweisungen bzw. internen Funktionen/Operationen beschreiben, wird über Berechnungsregeln das Attribut *Uhr* entsprechend der *Ausführungszeit* modifiziert. Damit kann das diskrete Fortschreiten der Zeit modelliert werden. Bei der Kommunikation wird zusätzlich die lokale Uhr des Absende-Prozesses übertragen

(Zeitstempel), um die globale Uhr nach den Regeln in Kapitel 4.2 zu realisieren. Beim Empfänger muß dann evtl. die lokale Uhr weitergestellt werden.

Das Attribut *Überwachungsende* wird benutzt, um zeitü.berwachte Kommunika-tion zu modellieren. Bei Ableitung des Programmgraphen unter Berü.cksichti-gung von Ausfnhrungszeiten wird das Überwachungsende durch Addition der Werte von *Uhr* und *Timeout* ermittelt. Die der Zeitkante in der PASS-Ablaufsteue-rung entsprechende Produktion wird erst dann anwendbar, wenn die Werte von *Uhr* und *Überwachungsende* ü.bereinstimmen. Ohne Berü.cksichtigung von Ausffh-rungszeiten wird die Zeitkante so behandelt, wie eine alternative Kommunikations-anweisung, die keiner Priorität unterworfen ist und ohne Einwirkung von außen zu einem spontanen Zustandsü.bergang fü.hrt.

Zur Definition des Wartebereichs bei asynchroner Kommunikation dienen die Attribute *WB-Größe* und *WB-Belegung*. Das erste Attribut gibt die maximale Anzahl von Plätzen im Wartebereich an. Dieses Attribut ist statisch in der PASS-Kommunikationsmaschine vorgegeben. *WB-Belegung* zeigt an, wie viele Plätze davon bei der Ableitung von Graphsatzformen aktuell belegt sind. Beim Senden wird ein Platz belegt, beim Empfangen freigegeben.

Damit können wir nun die Bestandteile des Programmgraphen folgendermaßen zusammenfassen:

- Knotenmarkierungsalphabet
  - !<sub>v</sub> • {Sende, Empfange, Funktion, F-Ergebnis, Botschaft, Sammelbotschaft, Zeit, S, WB, aktiv, blockiert, lauffähig, ruhend, V, Z }
- Kantenmarkierungsalphabet
  - !<sub>E</sub> { •>' ' --->, "">, !! }
- Menge der Attribute
  - ATT • {Ausführung, Timeout, Uhr, Überwachung, Zeitstempel, Priorität, WB-Größe, WB-Belegung}
- Attributzuordnung
 

AB: {Sende, Empfange}	→	{Ausführung, Timeout}
{Botschaft}	→	{Ausführung, Zeitstempel, Priorität}
{F-Ergebnis}	→	{Ausführung}

{aktiv, blockiert,  
lauffähig, ruhend}

{Uhr, Überwachung, Priorität}

- Wertzuordnung

val : K x ATT    INTEGER

mit Initialisierung

$(\forall k \in K \text{ mit } \beta(k) = \{ \text{aktiv, blockiert, lauffähig} \})$   
 $\text{Cval}(k, \text{Uhr}) := 0$   
 $\text{val}(k, \text{Überwachung}) := 0$ ;  
 $(\forall k_i \in K \text{ mit } \beta(k_i) = \{ \text{aktiv, lauffähig} \})$   
 $\text{Cval}(k_i, \text{Priorität}) := PP_i$ ;  
 $(\forall k_1 \in K \text{ mit } \beta(k_1) = \{ \text{Sende, Empfange} \})$   
 $(\text{val}(k_1, \text{Ausführung}) := A_1$   
 $\text{val}(k_1, \text{Timeout}) := T_1$ );  
 $(\forall k_1 \in K \text{ mit } \beta(k_1) = \{ \text{F-Ergebnis} \})$   
 $(\text{val}(k_1, \text{Ausführung}) := AF_1$ ;  
 $(\forall k_1 \in K \text{ mit } \beta(k_1) = \{ \text{Botschaft I} \})$   
 $(\text{val}(k_1, \text{Zeitstempel}) := 0$   
 $\text{val}(k_1, \text{Ausführung}) := AB_1$   
 $\text{val}(k_1, \text{Priorität}) := PB_1$ );

#### 4.4.3 Ein Beispiel: Das synchrone Senden einer Botschaft.

Am Beispiel einer Produktion, die das Senden von Botschaften beschreibt, soll dargestellt werden, wie programmierte, attributierte Ersetzungen zur Definition der Semantik von Sprachelementen von PASS verwendet werden. Die programmierte Ersetzung besteht aus einer Reihe von Einzelersetzungen, die über ein Kontrolldiagramm gesteuert werden. Das Kontrolldiagramm legt fest, welche Einzelproduktionen in welcher Reihenfolge anzuwenden sind. Programmgraph und Einzelproduktionen sind so konstruiert, daß bei der Abwicklung eines Kontrolldiagramms nur der mit der ersten Einzelproduktion ausgewählte Untergraph bearbeitet wird. Dies wird über den Prozeßzustandsknoten und die Befehlszählerkante erreicht.

Diese Befehlszählerkante hat gleichzeitig den Zweck, den Ort des gesamten

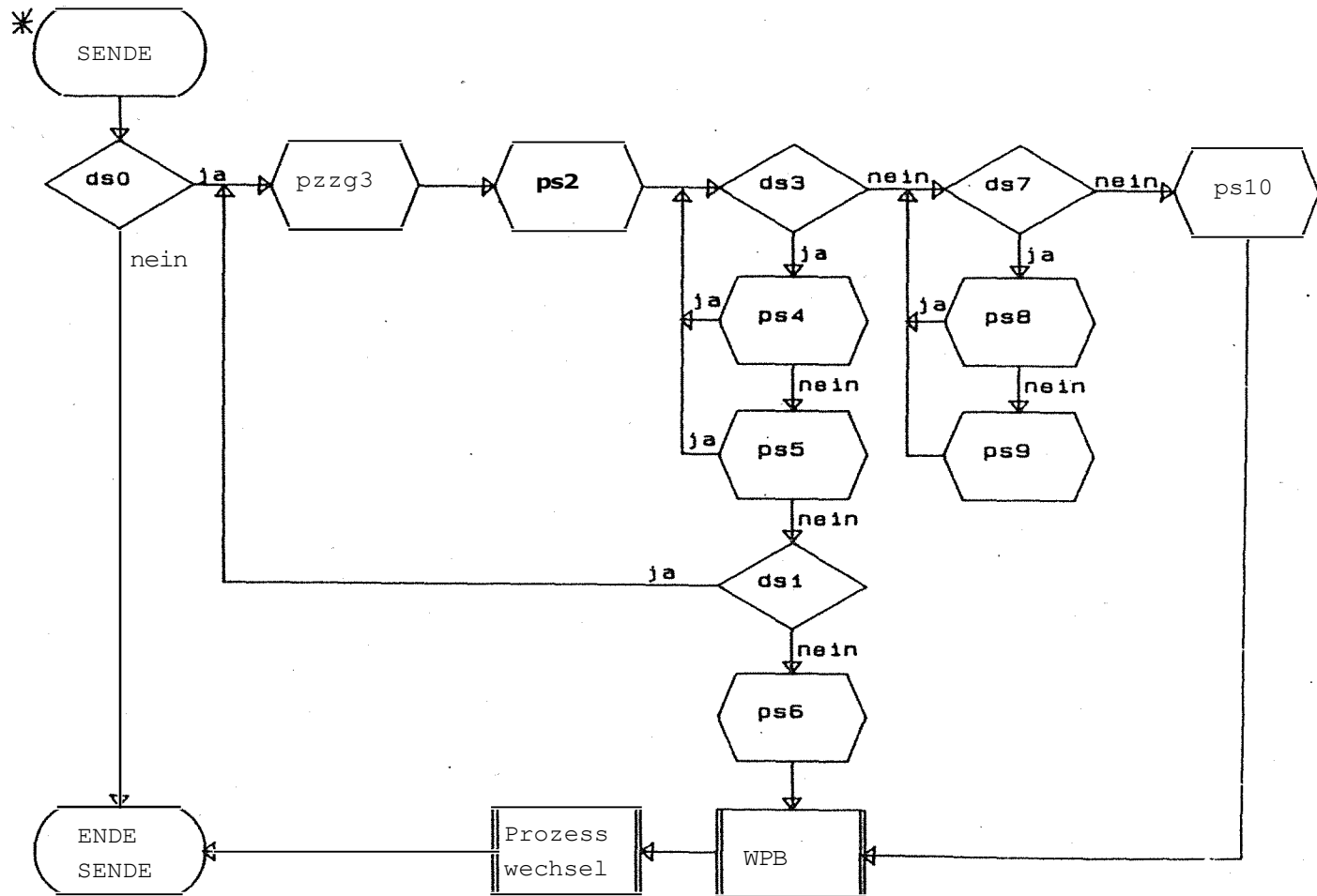


programmierten Ersetzungsschrittes festzulegen. Der Untergraphentest auf Vorhandensein der linken Seite im Wirtsgraphen arbeitet nicht auf konkreten Graphen, sondern auf abstrakten Graphen. Es wird also ein zur linken Seite der Produktion bis auf Knotenbezeichnungen äquivalenter Untergraph gesucht und ersetzt. Knotenbezeichnungen entsprechen den Namen der Knoten und Kanten in einer PASS-Ablaufsteuerung. Davon soll bei Anwendung der Produktionen abstrahiert werden. Durch die Abstraktion werden Produktionen prinzipiell an beliebigen Stellen im Programmgraphen anwendbar. Dies wird durch die Befehlszählerkante jedoch verhindert. Dies erlaubt eine effiziente Implementierung des Untergraphentests, der zur Auswahl einer geeigneten Produktion notwendig ist.

In den Produktionen verwenden wir als Hilfsbezeichnung für Knoten natürliche Zahlen, um die Einbettungsüberführung und Attributierung von Knoten eindeutig beschreiben zu können.

Ein Kontrolldiagramm zusammen mit seinen Unterkontrolldiagrammen ist ein hierarchischer Graph, der als Programmknoten die sequentielle Anwendung einer Produktion bzw. einen Unterkontrolldiagrammaufruf enthält. In der graphischen Notation besitzen Kontrolldiagramme einen Start- und Endeknoten. Durch Rauten werden Abfragen gekennzeichnet, in denen Untergraphentests formuliert werden. Sechseckige Knoten enthalten als Operationen die Anwendung von Produktionen.

Bild 4-8 zeigt das Sendekontrolldiagramm. Anschließend soll eine typische Einzelproduktion erläutert werden. Eine ausführliche Darstellung findet sich in /Fede86/. Im folgenden wird eine überarbeitete und um Attributierung erweiterte Fassung vorgestellt.



Das Sendekontrolldiagramm enthält die Knoten PS2 bis PS10. Dies sind sequentielle Ersetzungsschritte. DSO, DSI, DS3 und DS7 enthalten Untergraphentests.

Die Produktion PZZG3 wird bei Bearbeitung alternativer Kommunikationsanweisungen von Bedeutung. Über diese Produktion wird der durch alternative Anweisungen entstehende Nichtdeterminismus der Ableitungsschritte aufgelöst. Hier wird eine Auswahl aus alternativen Kommunikationsanweisungen gemäß einer bestimmten Strategie getroffen. Ist eine Prioritätenregelung gegeben, ist die Auswahl deterministisch unter den möglichen, durch eine erfüllte Vorbedingung gekennzeichneten Alternativen.

Das Unterkontrolldiagramm WPB kontrolliert die Bearbeitung der blockierten Prozesse. Nach Abschluß der Sendebearbeitung werden diese Prozesse daraufhin geprüft, ob durch das Senden ihre jeweilige Blockierungsbedingung ungültig geworden ist und sie lauffähig gesetzt werden können.

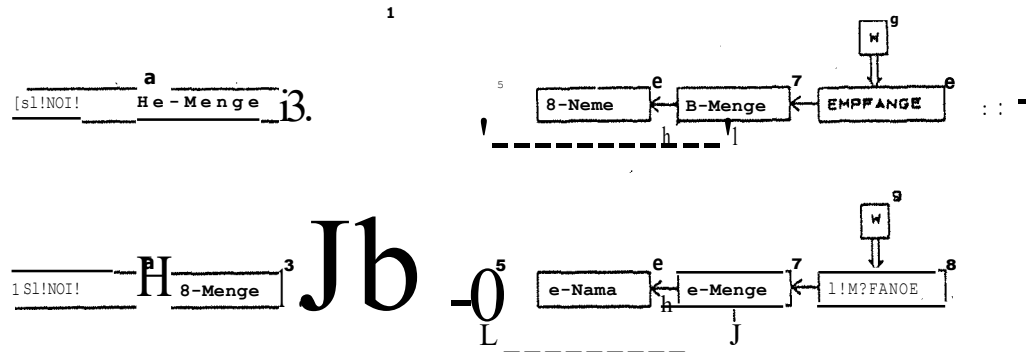
Mit dem Unterkontrolldiagramm PW wird für quasi-parallele Prozesse die Prozessorzuordnung gesteuert. Nach der vorgewählten Prozessorvergabe-strategie wird aus der Menge der lauffähigen Prozesse des betrachteten Prozessors ein Prozeß ausgewählt und aktiv gesetzt. Am Beispiel einiger Einzelproduktionen sei hier die Abwicklung des Kontrolldiagramms erläutert.

Mit DSO wird zunächst geprüft, ob die Sendeproduktion als Ganzes anwendbar ist.

Damit ist für eine Ablaufsteuerung genau ein Ort definiert, an dem die Produktion angewendet werden kann. Bedingung ist, daß eine Befehlszählerkante vom Prozeßzustandsknoten, der mit *aktiv* markiert ist, zu einem Sende-Knoten führt.

Die Produktion PS4 beschreibt das Vorgehen bei synchronem Nachrichtenaustausch.

PS4:



BR : BRid erweitert um

$val(L(1) Uhr) : val(L(1) Uhr) + val(2, Ausführung)$

Bild 4-9 : Produktion PS4

Die Produktion stellt die Ausführbarkeit einer Sendeanweisung fest. Wartet der Empfänger bereits auf die angegebene Botschaft, wird eine Hilfskante zur entsprechenden Botschaft kreiert. Damit ist die Botschaft noch nicht übertragen. Es muß erst noch überprüft werden, ob weitere mit "UND"-verknüpfte Botschaften vorliegen (DS3). Deshalb wird in der Attributberechnungsregel nur die lokale Uhr um die Bearbeitungszeit für den Betriebssystemaufruf weitergestellt. Die Uhr ist Attribut des zugehörigen Prozessorknotens, der durch eine Nachfolgekante mit dem Prozeßzustandsknoten *aktiv* verbunden ist.

Können alle Nachrichten übertragen werden, setzt PS8 den Empfänger *lauf-*fähig und eine Befehlszählerkante auf die nächste Aktion.

PS8:

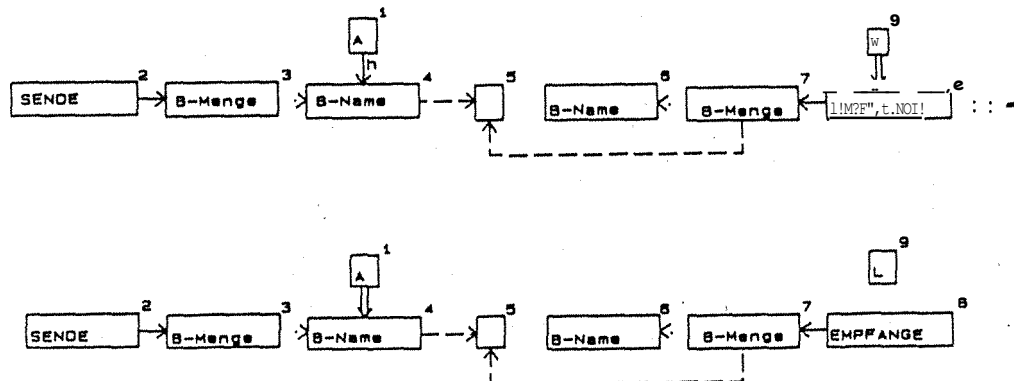


Bild 4-10 : Produktion PS8

BR(PS8) : BRid erweitert um

$$\begin{aligned} (1) \text{ val } (L \text{ (I), Uhr)} &= \max ((\text{val } (L \text{ (I), Uhr)} + \text{val } (4, \text{Ausführung})), \\ &\quad (\text{val } (L \text{ (9), Uhr)} + \text{val } (4, \text{Ausführung}))) \\ \text{val } (L \text{ (9), Uhr)} &= \max ((\text{val } (L \text{ (I), Uhr)} + \text{val } (4, \text{Ausführung}) + \\ &\quad + \text{val } (8, \text{Ausführung})), \\ &\quad (\text{val } (L \text{ (9), Uhr)} + \text{val } (8, \text{Ausführung}))) \end{aligned}$$

$$(2) \text{ val } (L \text{ (9), Überwachung)} = 0, \text{ val } (L \text{ (I), Überwachung)} := 0$$

Die Berechnungsregel (1) stellt die Konsistenz der Uhren beim Sender und Empfänger her. (2) setzt das Attribut *Überwachung* wieder zurück. Die Produktion PS10 löscht beim Sender die Hilfskanten und setzt den Befehlszähler weiter.

Die Produktionen PS5 und PS9 wirken in analoger Weise beim asynchronen Senden von Botschaften.

#### 4.5 Gemischte Ersetzungen

Bisher wurden nur sequentielle Ersetzungen auf markierten Graphen betrachtet, bei denen in jedem Ableitungsschritt genau ein Untergraph durch einen anderen ersetzt wird und der Restgraph unverändert bleibt.

Da wir Graphersetzen jedoch zur operationellen Untersuchung von Systemen paralleler Prozesse einsetzen wollen, benötigen wir einen Ersetzungsbegriff, der auf solche parallelen Systeme anwendbar ist.

Die Theorie der parallelen Ersetzungssysteme legt fest, daß in einem parallelen Ersetzungsschritt der gesamte Graph überschrieben wird, und es keinen unveränderten Teil gibt. Die Menge der Untergraphen  $d/$  bildet eine Zerlegung des Wirtsgraphen. Solche parallelen Ersetzungssysteme werden z.B. in der Biologie zur Beschreibung des Zellwachstums eingesetzt. In einem Wachstumsschritt verändern sich alle Zellen, repräsentiert durch Knoten eines Graphen, gleichzeitig.

Für unsere Anwendung ist eine solche Definition nicht brauchbar, da sich Prozesse zwar an verschiedenen Orten gleichzeitig ändern, aber dabei nur genau eine Zustandsänderung durchführen, so daß der größte Teil der Prozesse unverändert bleibt.

Gemischte Ersetzungen sind eine Mischform von sequentiellen und parallelen Ersetzungen. Hier werden parallele Ableitungen nur auf einen Teil des Wirtsgraphen durchgeführt, der Rest bleibt unverändert. Eine gemischte Ersetzung besteht aus der parallelen Anwendung einer endlichen Menge von sequentiellen Produktionen.

In [Nagl 79] zur Beschreibung der Ersetzung sowohl eine Einbettungs- als auch Verbindungsüberführung definiert.

Die Einbettungsüberführung beschreibt, wie gewohnt, die Kanten zwischen dem eingesetzten Graphen  $d$ , und dem unveränderten Teil des Wirtsgraphen. Die Verbindungsüberführung gibt an, welche Kanten zwischen parallel eingesetzten Graphen  $d_i$  der gleichzeitig angewandten Produktionen existieren sollen. Hier wird eine vereinfachte Definition verwendet, da zwischen parallel eingesetzten Graphen  $d$ , keine Verbindung hergestellt werden muß.

**Definition 4-17 : gemischt ableitbar**

*Sei  $d, d(Ev, EE, AB)$ ,  $d(E)$  und  $d', d(Ev, E2, ABI)$*

*Dann heie  $d'$  direkt aus  $d$  gemischt ableitbar (abgekrzt  $d \xrightarrow{g} d'$*

*gdw. es eine Menge knotendisjunkter Produktionen  $\{p_1, \dots, p_n\}$  gibt,*

*so da gilt:*

*$d \xrightarrow{d_i} d' \wedge d \xrightarrow{d_j} d' \wedge \dots \wedge d \xrightarrow{d_n} d'$  fr  $1 \leq n$*

**b) Die Kanten zwischen  $d' - \bigcup d_i$  und den  $d_i$  sind entsprechend Definition 4-12 festgelegt.**

**c) Berechnungsregeln drfen nicht verndernd auf die gleichen Attribute gemeinsamer Knoten wirken.**

Die Definition schliet als Spezialfall die sequentielle Ersetzung ein; das ist dann der Fall, wenn genau eine Produktion zur Anwendung kommt. Die Anzahl der parallel anzuwendenden Produktionen wird nicht vorgeschrieben.

Beschreibt man jede Anweisung, die in einer PASS-Ablaufsteuerung enthalten ist, durch eine Graphproduktion, die den Zustand des PASS-Programmgraphen entsprechend der Semantik der Anweisung transformiert, ist die Semantik eines Prozesystems mit Hilfe gemischter Ableitungen darstellbar.

In gemischten Ableitungen sind folgende beiden Extremfälle enthalten:

- Jeder Ableitungsschritt bewirkt maximale Parallelität. Das bedeutet, jeder Prozeß des Prozeßsystems kann in einem Ableitungsschritt eine Zustands-transformation durchführen. Dies ist bei einer Implementation dann gegeben, wenn jeder Prozeß über einen eigenen Prozessor verfügt. Voraussetzung ist, daß parallele Ableitungsschritte nicht auf demselben Untergraphen Änderungen durchführen wollen, d.h. daß die linken und rechten Seiten der parallelen Produktionen disjunkt sind.
- In jedem Ableitungsschritt wird nur eine sequentielle Ersetzung durchgeführt. Das ist z.B. dann der Fall, wenn Prozesse quasi-parallel auf einem Prozessor laufen. Dann kann jeweils nur genau ein aktiver Prozeß Änderungen am Programmgraphen verursachen. Die dazu quasi-parallelen Prozesse werden in der durch den Ablaufplan vorgegebenen Reihenfolge bedient.

#### **4.6 Das Semantikmodell**

Mit dem oben beschriebenen Graphersetzungssystem kann man alle Transformationen am Programmgraphen beschreiben, die erlaubten Zustandsübergängen in einem PASS-Spezifikationsmodell entsprechen. Die Semantik eines Systems paralleler, kommunizierender Prozesse kann so als die Menge aller zulässigen Ableitungsfolgen, die mit gegebenen Programmgraphen beginnen, definiert werden. Der Programmgraph ergibt sich aus der Menge der PASS-Ablaufsteuerungen des Prozeßsystems und einer initialen Attributierung.

Der Programmgraph stellt die Struktur eines Prozeßsystems dar. Die definierte Produktionenmenge legt die Bedeutung der Sprachelemente der zugrundeliegenden Spezifikationssprache PASS fest. Die Produktionen sind attribuiert. Über die Attributberechnung kann das Verhalten eines virtuellen Betriebssystems und der Ablauf von Zeit modelliert werden. Die Anwendung der Produktionen auf den Programmgraphen legt eine Funktion fest, die den Programmgraphen in die Menge aller gemischten Ableitungsfolgen abbildet.

Damit ist ein vollständiges Semantikmodell definiert. Es besteht aus den drei Teilen:

- syntaktischer Bereich : Menge der Programmgraphen,
- semantischer Bereich : Menge aller Ableitungsfolgen,
- Semantik der PASS-Sprachelemente, die den syntaktischen Bereich in den semantischen abbildet.

Um das Verhalten von Systemen in allen Prozessen untersuchen zu können, muß die Menge aller zulässigen Ableitungsfolgen eines konkreten Programmgraphen überprüft werden. Diese Menge können wir operationell durch einen abstrakten Interpretierer erzeugen.

Die Menge der Graphproduktionen bildet einen Interpretierer, der das Ergebnis jedes Ableitungsschrittes durch die Modifikation des Programmgraphen sichtbar macht. Die Semantikdefinition durch einen Interpretierer ist notwendig, weil es auf die zeitliche Koordination von Prozessen zur Laufzeit ankommt. Jeder Ableitungsschritt entspricht einer Zustandsänderung im verteilten System.

Die gemischten Ableitungsfolgen beschreiben also das Verhalten eines Systems als Folge von Zustandsänderungen, die gleichzeitig (bei paralleler Anwendung von sequentiellen Ableitungsschritten) oder sequentiell auftreten können.

Die Angabe aller möglichen Ableitungsfolgen für einen gegebenen Programmgraphen stellt eine trace-orientierte Beschreibung des Verhaltens paralleler Prozesse dar. Eine Ableitungsfolge repräsentiert die parallele und sequentielle Ausführung von Aktionen. Im folgenden interessieren wir uns dafür, wie man solche Ableitungsfolgen gewinnen kann.



## 5. Simulation von Systemen paralleler Prozesse

In Kapitel 4 wurden die formalen Grundlagen gelegt, um das Verhalten von parallelen Prozessen, die mit Hilfe von PASS spezifiziert wurden, ableiten zu können. Die Semantik von PASS wurde mit Hilfe von Graphproduktionen definiert. Auf diesen Produktionen baut ein Interpreter auf, der Spezifikationen ausführbar macht. Das Verhalten eines Prozeßsystems wird durch Transformationen auf dem Programmgraphen sichtbar gemacht.

Dazu werden die einzelnen Prozesse durch den Interpreter entsprechend den Synchronisationsanweisungen, den vorgegebenen Betriebssystemeigenschaften und dem spezifizierten Zeitverhalten parallel komponiert. In Kap. 5.1 soll die Realisierung der parallelen Komposition durch den Interpreter und ihre Darstellung an der Benutzerschnittstelle erläutert werden.

Durch die "Animation" eines Spezifikationsmodells erhält man direkt aus der Spezifikation einen Prototypen des entworfenen Prozeßsystems.

Die Kernfrage bei der Arbeit mit einem Prototypen ist die Frage nach der Ausführbarkeit von bestimmten Aktionsfolgen oder der Erreichbarkeit von Zuständen. Daraus läßt sich ableiten, ob eine vorgegebene Anforderungsdefinition erfüllt werden kann (externe Korrektheit).

Eine automatische Analyse des Modellverhaltens auf interne Eigenschaften, wie z.B. Verklemmungsfreiheit, durch Aufzählung des Zustandsraums ist für größere Systeme wegen des exponentiellen Wachstums des Zustandsraums mit der Anzahl von Zuständen praktisch nicht durchführbar.

In Kap. 5.2 werden deshalb Verfahren vorgestellt, die eine Beschränkung des zu untersuchenden Zustandsraumes ermöglichen. Der Benutzer des Interpreters kann interessierende Teilbereiche eines Spezifikationsmodells zur Untersuchung auswählen. Dabei ist auch die Simulation unvollständiger Prozeßsysteme möglich.

Der Ablauf der Simulation wird entweder interaktiv durch den Benutzer des Interpreters oder teilweise automatisch durch Verwendung geeigneter Steuerungsmechanismen bestimmt. Die Einbettung der Steuerung in die Graphproduktionen wird in Kap. 5.3 dargestellt.

Kap. 5.4 schließt mit der Beschreibung einiger Untersuchungen, die mit Hilfe des Interpreters am Prototypen möglich sind.

## 5.1 Darstellung des Modellverhaltens

### 5.1.1 Benutzersicht

Die komplexe Struktur des Ablaufs in einem System paralleler Prozesse erfordert eine visuelle Darstellung der Vorgänge im System. Textuelle Beschreibung des Ablaufs ist in diesem Fall extrem unübersichtlich.

Gerade beim Bearbeiten und Verbessern eines Entwurfs ist es notwendig, einen Überblick über die Systemstruktur zu bekommen. Da die Spezifikationsmethode PASS ohnehin graphisch orientiert ist, bietet es sich an, den Modellablauf in Symbolen von PASS zu beschreiben.

Der Interpreter macht daher Zustände und mögliche Zustandsübergänge mit Hilfe von PASS-Knoten und -Kanten am Bildschirm sichtbar. Dazu wird der Bildschirm in verschiedene Fenster aufgeteilt, die den unterschiedlichen Prozessen des Systems zugeordnet sind. Ein Fenster kann als ausgezeichnetes Fenster ausgewählt werden, das einen speziell zu beobachtenden Prozeß zeigt. Übersteigt die Anzahl der Prozesse im System die verfügbare Fensterzahl, werden aktive Prozesse auf die zur Verfügung stehenden Fenster verteilt, sodaß ein Fenster mehreren Prozessen zugeordnet ist.

Zu jedem Fenster wird die aktuelle Prozessoruhr eines Prozesses eingeblendet. Der Benutzer erhält damit Informationen über den zeitlichen Ablauf der Simulation.

Die visuelle Darstellung des parallelen Ablaufs von Prozessen hat als formale Grundlage die Beschreibung des Modellablaufs durch gemischte Ableitungen. Die Zustandsänderung eines Prozeßsystems durch einen gemischten Ableitungsschritt wird in den verschiedenen Fenstern direkt sichtbar.

### 5.1.2 Realisierung paralleler Abläufe durch gemischte Ableitungen

Durch das in Kapitel 4.2 definierte Zeitmodell werden die Aktionen paralleler Prozesse entlang einer Zeitachse linear geordnet. Es wurde ein System lokaler Zeitachsen (Uhren) definiert, die beim Nachrichtenaustausch ständig synchronisiert werden, so daß eine globale Zeitachse entsteht. Damit sind auch alle Aktionen

global geordnet. Diese Vorgehensweise ist zur Untersuchung von Prozessen geeignet, über deren zeitliche Abläufe Aussagen gemacht werden, d.h. deren Aktivitäten und Ausführungszeiten spezifiziert werden.

Läßt man die lineare Ordnung der zeitlichen Abfolge von Aktionen unberücksichtigt, treten noch folgende Ordnungskriterien auf:

- sequentielle Reihenfolgen in einem Prozeß;
- Sequentialisierung quasi-paralleler Prozesse durch den Prozeßumschalter;
- Synchronisation durch Kommunikationsanweisungen.

Flir unabhängige Aktionen paralleler Prozesse kann keine Ordnung angegeben werden. Da man die relativen Bearbeitungsgeschwindigkeiten der Prozesse nicht kennt, können sie in jeder möglichen Reihenfolge oder gleichzeitig auftreten.

Betrachten wir die Aktionen, die bei der Ausführung eines PASS-Prozeßsystems auftreten können, und deren Beschreibung durch Graphproduktionen:

- Interne Funktionen und Operationen

Diese PASS-Elemente haben nur lokale Auswirkungen auf den Prozeß, zu dem sie gehören. Sie können parallel zu jeder Aktion eines anderen Prozesses ausgeführt werden, der nicht auf dem gleichen Prozessor liegt.

- Synchrone Sende- und Empfangsanweisungen

An synchroner Kommunikation sind im einfachsten Fall genau zwei Prozesse beteiligt. Die Sende- und Empfangsanweisung muß gleichzeitig mit der Empfangsanweisung des Kommunikationspartners bearbeitet werden. Zu anderen, nicht beteiligten Prozessen besteht keine Beziehung. Deren Aktionen können unabhängig davon ausgeführt werden.

- Asynchrone Sende- und Empfangsanweisungen

Asynchrone Sende- und Empfangsanweisungen stellen zunächst keine direkte Beziehung zwischen dem Sende- und Empfangsprozess her. Einzige Voraussetzung für das Senden ist, daß der Empfänger noch Platz in seinem Wartebereich hat. Allerdings werden Sende- und Empfangsanweisungen verschiedener Absender an den gleichen Empfänger durch den Wartebereich des Empfängers sequenzialisiert. Da bei PASS das Empfangen auftragsgesteuert ist, d.h. für die Verwaltung des Wartebereichs keine feste Strategie (z.B. FIFO) vorliegt, treten Reihenfolgeprobleme nur

dann auf, wenn die Wartebereichsgrenzen erreicht werden. Werden mehr Sendeanforderungen gestellt, als Platz im Wartebereich vorhanden ist, muß ein Teil der Sender blockiert werden.

Diese Eigenschaften parallel ausführbarer Aktionen lassen sich durch gemischte Ableitungsschritte formal beschreiben. Parallel anwendbar sind nur solche Produktionen, die entweder knotendisjunkt oder äquivalent bis auf Knotenbezeichnungen sind. Dies bedeutet, daß parallel entweder voneinander verschiedene Aktionen oder gleichartige Aktionen, die aber unterschiedliche Prozesse betreffen, ausführbar sind.

Eine parallele Ersetzung drückt folgenden Sachverhalt aus:

$$\begin{array}{c} 82 \\ \bullet \\ A, \%, C \\ \hline /4 \\ \bullet \\ 81 \end{array}$$

Bild 5-1: Parallele Ersetzung

Vom Systemzustand A führt die Anwendung der Produktionen P1 und P2 zum Zustand C. Dabei ist die Reihenfolge der Anwendung beliebig. Die parallele Anwendung (P1 + P2) der Produktionen P1 und P2 führt zum gleichen Ergebnis (Church-Rosser-Eigenschaft).

Während die sequentielle Anwendung einzelner Produktionen das Vorwärtsschreiten eines Prozesses darstellt, wird durch einen parallelen Ersetzungsschritt das nebenläufige Verhalten eines Prozeßsystems charakterisiert. Die Menge der möglichen Sequentialisierungen eines parallelen Schrittes bildet eine Äquivalenzklasse, deren Elemente durch die Relation "Sequenz" in Bezug stehen. Parallele Ersetzungsschritte repräsentieren eine ganze Klasse möglicher Einzelschritte von Prozessen.

Für eine Implementierung gemischter Ableitungsschritte im Interpreter muß durch die Konstruktion der Produktionen sichergestellt sein, daß die Church-Rosser Eigenschaft in allen Situationen erhalten bleibt. Dann ist eine einfache Realisierung eines gemischten Ableitungsschritts durch sequentielle Anwendung von Produktionen möglich.

Für interne Funktionen und Operationen entstehen keine Probleme, da sie, wie oben beschrieben, nur lokale Wirkung haben.

Synchrone Kommunikation wird durch genau einen programmierten Ersetzungsschritt modelliert, der sowohl Sende- als auch Empfängerprozesse enthält. Hier kann es zu Überschneidungen bei paralleler Anwendung nur dann kommen, wenn folgende Situation gegeben ist

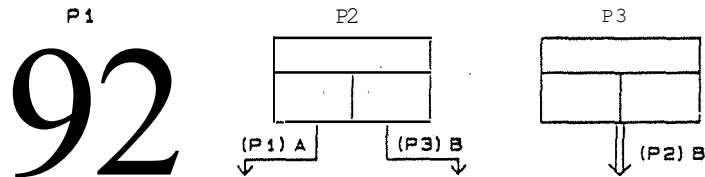


Bild 5-2: Synchrone Kommunikation

Die Prozesse P1 und P3 sind bereit, die Botschaft A bzw. B an P2 zu senden. In diesem Fall sind die Aktionen A und B aber nicht parallel unabhängig. Die Voraussetzung der Knotendisjunktheit der anzuwendenden Produktionen ist nicht gegeben. Durch Anwendung einer Produktion "Synchrones Senden" wird P2 so verändert, daß die zweite Sendeanweisung nicht mehr durchgeführt werden kann. Eine Sequentialisierung erfolgt durch Mechanismen zur Auflösung von alternativer Auswahl, wie sie in Kapitel 5.3 beschrieben werden.

Bei asynchroner Kommunikation muß, wie oben beschrieben, untersucht werden, ob mehrere Kommunikationspartner parallel Nachrichten im Wartebereich eines Prozesses ablegen wollen. In diesem Fall unterliegt die sequentielle Bearbeitung der asynchronen Kommunikation einer Einschränkung.

Zwei Situationen sind zu unterscheiden:

- a) Durch Bearbeitung aller Sendewünsche von parallelen Prozesse wird die Wartebereichskapazität nicht überschritten.

In diesem Fall ist eine beliebige Sequentialisierung der Sendeanweisungen möglich. Da in PASS der Wartebereich nicht nach einer Fifa-Strategie, sondern auftragsgesteuert geleert wird, ist die Reihenfolge, in der Nachrichten abgelegt werden, ohne Bedeutung.

- b) Bei Erfüllung aller Sendewünsche wird die Kapazität des Wartebereichs überschritten.

In diesem Fall müssen wir auswählen, welche Prozesse Nachrichten ablegen sollen und welche blockiert werden. Über einen Hilfskantenmechanismus werden zunächst alle sendebereiten Prozesse ermittelt.

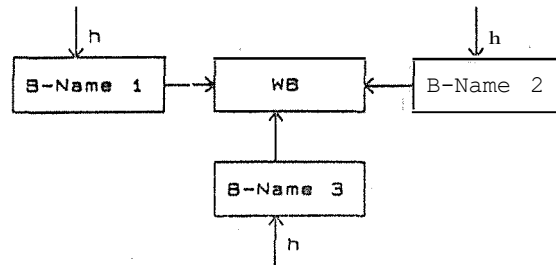


Bild 5-3: Asynchrone Kommunikation

Die Auswahl unter allen sendebereiten Prozessen erfolgt durch die Einbetungsüberführung einer Produktion, die einen Auswahl-Operator zur Verfügung stellt. Dieser Auswahloperator trifft die Entscheidung, welche Hilfskanten wieder zu löschen sind, so daß die zugehörigen Prozesse aktiv werden können. Der Auswahlmechanismus ist in Kapitel 5.3 beschrieben.

## 5.2 Beschränkung des Zustandsraummodells

### 5.2.1 Simulation unvollständiger Spezifikationen

Die Produktionen zur Beschreibung von PASS-Kommunikationsanweisungen verlangen, so wie sie bisher vorgestellt wurden, das Vorhandensein eines Kommunikationspartners, der die entsprechende Botschaft sendet oder entgegennimmt.

Diese Forderung ist für einen praktischen Einsatz des Interpreters jedoch zu streng:

- das Verhalten der Umwelt (Benutzer, technischer Prozeß) muß unter dieser Voraussetzung als eigener PASS-Prozeß vorliegen;
- die Simulation einer Spezifikation kann sich nicht auf Teilsysteme eines Systems paralleler Prozesse beschränken.

Deswegen sind die Produktionen so implementiert, daß Nachrichten an Partner oder von Partnern, die nicht in der Kommunikationsstruktur einer PASS-Spezifikation enthalten sind, stets gesendet oder empfangen werden können. Die entsprechenden Kommunikationsanweisungen werden entweder wie Leeranweisungen behandelt oder, bei interaktivem Betrieb des Simulators, durch eine Meldung an den Benutzer ersetzt. Der Benutzer kann dann entscheiden, ob die entsprechende Botschaft ausgetauscht werden soll.

Durch diese Vorgehensweise ist die Konsistenz einer Spezifikation sichergestellt, da explizit Bezug auf die Kommunikationsstruktur genommen wird. Damit ist es möglich, auch bei unvollständiger Spezifikation Nachrichten zu erkennen, die nicht korrekt spezifiziert wurden, d.h. also z.B. von keinem Prozeß gesendet, aber trotzdem empfangen werden sollen.

Die Beschränkung auf Teilsystem bietet den Vorteil, in einem überschaubaren Bereich das Verhalten von Prozessen untersuchen zu können. Damit können einfache Synchronisationsfehler zwischen den Prozessen des betrachteten Teilsystems gefunden werden.

Allerdings kann sich dieses Teilverhalten ändern, wenn andere Prozesse zum Teilsystem hinzugenommen werden. Eine korrekte Modellierung von Teilsystemen ist wegen der in der Simulation berücksichtigten Beschreibungsmittel dann möglich, wenn

- alle Prozesse, die asynchron Nachrichten im Wartebereich eines zu untersuchenden Prozesses ablegen, berücksichtigt,
- alle quasi-parallel auf einem Prozessor ablaufenden Prozesse simuliert und
- Prioritäten bei der Auswahl von Alternativen vollständig ausgewertet

werden können.

### 5.2.2 Zyklen 111Abbussteuerungem

Durch Zyklen in der Ablaufsteuerung eines Prozesses wird der Zustandsraum eines Prozeßsystems potentiell unendlich. Deshalb fährt der Interpreter eine Prüfung auf Zyklen durch und meldet dem Benutzer, wenn in der Simulation der Startknoten eines Zyklus erreicht wurde.

### Definition 5-1: Pfad, Zyklus

Ein Pfad in einer PASS-Ablaufsteuerung ist gegeben durch eine nichtleere Menge von Knoten  $(k_1, \dots, k_n)$ , wobei je zwei Knoten  $k_i$  und  $k_{i+1}$  mit  $1 \leq i < n$  durch eine Kommunikations- oder Internkante verbunden sind und kein Knoten mehrfach enthalten ist.

Ein Pfad heißt Zyklus, wenn  $k_1$  und  $k_n$  identisch sind. Der Knoten  $k_1$  heißt Startknoten des Zyklus.

Enthalten Zyklen nur interne Operationen und Funktionen, werden sie bei der Simulation genau einmal durchlaufen. Mehrmalige Bearbeitung ändert nur das Zeitverhalten eines Prozesses. Startknoten werden deshalb wie Unterbrechungspunkte behandelt. Hier kann die Zeit für einen Durchlauf des Zyklus berechnet und vom Benutzer abgefragt werden. Der Interpreter markiert den Zyklus als bearbeitet und verzweigt in andere Alternativen der Ablaufsteuerung.

Sind Kommunikationsanweisungen im Zyklus enthalten, wird ähnlich vorgegangen. Am Startknoten des Zyklus wird eine Momentaufnahme über alle Wartebereiche der Prozesse eines Systems gemacht. Vergleicht man den Zustand aller Wartebereich vor Eintritt und nach Verlassen des Zyklus, können Auswirkungen des Durchlaufs als Differenz des Nachrichtenbestands in Wartebereichen ausgedrückt werden. Tritt eine Differenz auf, ist bei wiederholtem Durchlauf eine Blockade an den Wartebereichen möglich. Kapitel 6 zeigt ein Beispiel, in dem unendliches zyklisches Verhalten zweier Prozesse auf diese Weise überprüft werden kann.

## S.3 Steuerung des Modellablaufs

### S.3.1 Auswahl alternativer Zustandsübergänge

In bestimmten Zuständen hat ein Prozeß die Möglichkeit, alternative Zustandsübergänge auszuwählen.

Eine interne Funktion zur Abfrage lokaler Daten hat in der Regel mehrere Ergebniskanten, die zu verschiedenen Folgezuständen führen.

Ein Kommunikationsknoten kann das Warten auf eine von mehreren alternativen Botschaften darstellen, wobei jeweils in einen unterschiedlichen Folgezustand übergegangen wird. Bei der Bearbeitung von Wartebereichen muß die Auswahl unter parallel möglichen Sendewünschen für das ganze Prozeßsystem getroffen werden.



Solche Alternativen auf Spezifikationsebene werden zur Laufzeit der Prozesse entschieden durch:

- (1) Datenabhängigkeiten,
- (2) Verhalten der Umwelt (Benutzer, technischer Prozeß)
- (3) Zeitbedingungen
- (4) Auswahlalgorithmen des Betriebssystems.

Die Bedingungen (3) und (4) können in der Simulation einer Spezifikation modelliert werden.

Die Einflüsse durch die Bedingungen (1) und (2) äußern sich als Auswahlentscheidungen auf Spezifikationsebene. In PASS-Ablaufsteuerungen ist nur der Kontrollfluß innerhalb von Prozessen festgehalten. Über verarbeitete Daten wird nichts ausgesagt. Da der Interpreter nur auf dem Programmgraphen, gegeben durch die PASS-Ablaufsteuerungen, arbeitet, ist der datenabhängige Kontrollfluß nur indirekt durch Auswahl von Ergebniskanten modellierbar.

Eine Begrenzung der Zustandskombinationen kann dadurch erreicht werden, daß der Benutzer in den Modellierungsablauf, d.h. in die Konstruktion von Ableitungsfolgen, eingeschaltet wird. Der Benutzer kann sein Wissen über sinnvolle Auswahl von Alternativen zur parallelen Komposition in den Simulationsablauf einbringen. Damit ist der Test auf Erreichbarkeit von Zuständen begrenzt.

Da der Interpreter als Entwurfshilfsmittel, das die Erstellung einer Spezifikation unterstützen soll, gedacht ist, ist diese Vorgehensweise sinnvoll. Der Benutzer erhält dadurch auch größtmöglichen Einfluß auf den Ablauf des Spezifikationsmodells.

Geht es jedoch darum, unvollständige Abläufe oder Überspezifikationen der Kommunikation zu finden, muß umgekehrt der Benutzer mit seiner vorgefaßten Meinung über zulässige Ablauffolgen ausgeschaltet werden. Dann übernehmen Steuerungsmechanismen die Aufgabe der Auswahl aus Alternativen. Das dann entstehende Problem einer vollständigen Konstruktion aller möglichen Zustandskombinationen ist jedoch NP-vollständig /Tayl 83/.

### 5.3.2 Einbettung von Steuermechanismen in die Graphproduktionen

Auswahlentscheidungen müssen bei der Bearbeitung interner Funktionen und Operationen, sowie bei Kommunikationsknoten mit alternativem Senden oder Empfangen getroffen werden. Die Einbettung des Auswahlmechanismus soll anhand der Produktionen zur Bearbeitung von internen Funktionen und Operationen in PASS gezeigt werden. Hier erlaubt der einfache Kontext eine kurze Darstellung. In gleicher Weise erfolgt die Auswahl bei der Bearbeitung von Kommunikationsknoten. Dabei wird die Auswahl jedoch zusätzlich durch Prioritäten beeinflusst.

Die Produktion zur Bearbeitung interner Aktionen muß im wesentlichen das Weiterschalten der Befehlszählerkante zu einem der möglichen Folgezustände beschreiben.

P1:



$E_1$  erweitert durch  $r_1 \Rightarrow (1, ZZG(R(2)))$

Bild 5-4: Produktion P1

P2:



$E_1$  erweitert durch  $r_1 \Rightarrow (1, R(3))$

BRid erweitert durch  $val(L(I), Uhr) := val(L(1), Uhr) + val(3, Ausführung)$

Bild 5-5 : Produktion P2

Über die Einbettungsüberführung der Produktion P1 wird die Auswahl der Ergebniskante gesteuert. Dazu wird ein Operator ZZG definiert, der aus allen Nachfolgeknoten des Funktionsknotens genau einen Knoten auswählt. Die Nachfolge-Knoten werden durch den Operator R+ geliefert ("Rechts-Nachfolger").

Die Produktion P2 hat im wesentlichen die Aufgabe, die lokale Uhr um den Zeitaufwand zur Berechnung des jeweiligen Funktionsergebnisses weiterzustellen und die nachfolgende Aktion zu bestimmen. Genauso wird bei Kommunikationsknoten verfahren.

Bei Aufruf der Operation ZZG können die verschiedenen Auswahlmechanismen wirksam werden:

a) dialoggesteuerter Modellablauf

Dem Benutzer werden alle Rechtsnachfolger des Knotens 2 mitgeteilt. Dies entspricht der Angabe aller zulässigen Funktionsergebnisse. Aus dieser Menge kann ein Element ausgewählt werden. Der gewählte Zustandsübergang wird in der Einbettungsüberführung durchgeführt und die Befehlszählerkante auf das entsprechende Funktionsergebnis gerichtet.

b) Auswahl durch Zufallszahlengenerator

An Stelle des Benutzers kann ein Zufallszahlengenerator eine der Kanten auswählen. Die gewählte Kante wird in einer Sperrliste, die im Funktions-Knoten verankert ist, festgehalten. Dadurch wird die Kante bei wiederholtem Durchlauf solange für eine Auswahl gesperrt, bis auch alle übrigen Kanten einmal ausgewählt wurden. Eine andere Möglichkeit wäre die Auswahl entsprechend einer vorgegebenen Übergangswahrscheinlichkeit. Durch Zuordnung von Wahrscheinlichkeiten zu jedem Funktionsergebnis wird der Zufallszahlengenerator so gesteuert, daß bestimmte Kanten entsprechend oft ausgewählt werden. Allerdings ist dies mit hohem Aufwand bei der Beschreibung des Spezifikationsmodells verbunden, so daß diese Auswahlart bisher noch nicht vorgesehen ist.

#### c) Auswahl nach Zielfunktion

Um die Untersuchung des Modellablaufs nach bestimmten Untersuchungskriterien zu unterstützen, sind Zielfunktionen bei der Auswahl von alternativen Zustandsübergängen vorgesehen. Diese Funktionen erlauben ein schwachstellenorientiertes Untersuchen eines Spezifikationsmodells. Durch "look-ahead"-Techniken können Pfade, ausgehend von einem aktuellen Zustand, bewertet und so die Entscheidung für einen bestimmten Zustandsübergang getroffen werden. Als Kriterien sind wählbar:

- Ausführungszeiten bis zum Erreichen eines Zielzustands,
- Kommunikationsbelastung bis zum Erreichen eines Zielzustands.

Es kann jeweils zwischen einer Minimum- oder Maximumbedingung gewählt werden. Damit ist das Testen zeitkritischer Pfade und der Dimensionierung von Wartebereichen möglich.

#### d) Auswahl nach Prioritäten

Dieses Auswahlverfahren wird bei Kommunikationsknoten angewandt, bei denen Prioritäten für alternative Botschaften definiert sind. Liegen mehrere Botschaften gleichzeitig vor, oder können sie alternativ gesendet werden, wird die Botschaft mit der höchsten Priorität bevorzugt. Ist auf Grund der Priorität eine eindeutige Auswahl möglich, werden beim automatischen Ablauf keine anderen Steuermechanismen wirksam. Im interaktiven Betrieb kann der Benutzer ebenfalls nur die ausgewählte Alternative aktivieren.

### 5.4 Analysefähigkeit des Interpreters

Der hier entwickelte Interpreter ist durch die Verwendung des definierten Semantik-Modells formal begründet. Über die durch PASS-Sprachelemente eingeführte Mächtigkeit der Ausdruckskraft hinaus, erlaubt er die Untersuchung von Zeitabläufen und die Berücksichtigung von Einflüssen, die Betriebssystemeigenschaften auf das spezifizierte Prozeßsystem ausüben.

Diese Fähigkeiten des Interpreters sind von besonderer Bedeutung, wenn eine Spezifikation auf bestimmte Eigenschaften hin untersucht werden soll. Die Frage, ob eine gewünschte Kommunikation zustande kommt oder ob ein angegebener Zustand erreicht wird, ist durchaus unterschiedlich zu beantworten. Die

Antwort ist abhängig vom betrachteten Zustandsraum des verteilten Systems. Bild 5-6 soll diese Tatsache veranschaulichen.

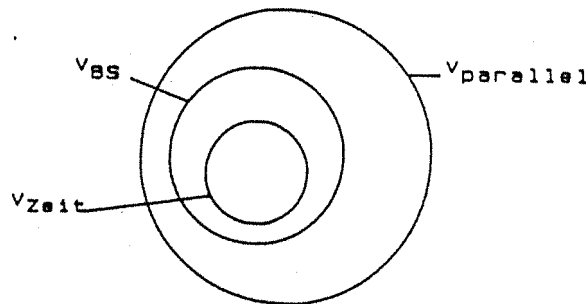


Bild 5.6 : Verhalten im verteilten System

Legt man der Spezifikation ein abstraktes Modell zugrunde, das die Unabhängigkeit aller Prozesse von einer Prozessorverwaltung vorsieht, läßt sich das Verhalten  $V_{parallel}$  realisieren. Betriebssystemstrategien zur Verwaltung quasiparalleler Prozesse führen zu einem Ablaufplan, der nur noch das eingeschränkte Verhalten  $V_{BS}$  zuläßt. Aussagen über Eigenschaften, wie Erreichbarkeit eines bestimmten Zustands, die auf einer Untersuchung des Raumes  $V_{parallel}$  beruhen, sind daher nicht sinnvoll.

Lebendigkeitsaussagen, wie Verklemmungsfreiheit des Prozeßsystems, gelten dagegen natürlich auch im eingeschränkten Bereich. Allerdings wird die Untersuchung durch Einführung von Restriktionen gemäß einem Ablaufplan im vorliegenden Fall sogar einfacher, da der zu konstruierende Zustandsraum beschränkt wird. Das gleiche gilt, wenn man Restriktionen durch Behandlung von Ausführungszeiten für Aktivitäten von Prozessen untersucht. Hier werden ebenso Ereignisse im Prozeßsystem total geordnet, wodurch das realisierbare Verhalten wesentlich eingeengt wird.

Durch die rechnergestützte Ausführung eines Spezifikationsmodells können folgende typische Fragestellungen, die bei der Entwicklung verteilter Systeme auftreten, beantwortet werden:

- Kann mit dem gegebenen Modell ein gewünschtes Verhalten, das für bestimmte Situationen in Form von Zustands- oder Aktionsfolgen beschrieben wird, erreicht werden?

In der Anforderungsdefinition von Prozeßautomatisierungssystemen werden

häufig Szenarien von möglichen Verhaltensweisen des technischen Prozesses vorgegeben. Sie können mit Hilfe des Interpreters ausgeführt werden, um zu zeigen, daß das Verhalten der Spezifikation in gegebenen Teilbereichen mit der Anforderungsdefinition vertraglich ist.

- Ist das Problem tiberspezifiziert?

Umgekehrt können alle möglichen Verhaltensweisen des Modells zwischen zwei vorgegebenen Zuständen des Prozeßsystems durch den Interpreter sichtbar gemacht werden. Damit entdeckt man Überspezifikationen, die sich in unerwünschter oder Überfillssiger Kommunikation zeigen.

- Können die geforderten Antwortzeiten eingehalten werden?

Für zeitkritische Ablaufpfade interessieren Ausführungszeiten, um die Reaktionszeit des Prozeßsystems auf Nachrichten von außen (technischer Prozeß, Benutzer) zu ermitteln.

Außerdem wird dadurch eine geeignete Dimensionierung der Zeitüberwachung (Time-Out) von Kommunikationsanweisungen möglich.

- Welchen Einfluß hat das Betriebssystem?

Verändert man die Betriebssystem-Parameter, wird die Abhängigkeit des Kommunikationsverhaltens von der Betriebssystemumgebung deutlich. Die Einflüsse von Prozessorvergabestrategie und Prioritäten der quasi-parallelen Prozesse können zu unterschiedlichen Verklemmungssituationen führen. Dadurch ist es möglich, das Maß der Portabilität einer gewählten Problemlösung festzustellen.

- Welchen Kommunikationsaufwand erfordert das System?

Durch die Ausführung der Spezifikation können Leistungsmessungen bereits am Modell erfolgen. Gemessen wird, welche und wie viele Nachrichten zwischen den beteiligten Prozessen ausgetauscht werden. Das Wissen über diesen Kommunikationsaufwand ist Voraussetzung zur Bestimmung der Kommunikationskosten einer gewählten Netzwerktopologie. Durch Verlagerung von Prozessen auf andere Prozessoren kann die optimale Verteilung von Prozessen ermittelt werden. Solche Prozeßkonfigurationen müssen dann wieder auf Korrektheit untersucht werden. Der Interpreter erlaubt durch einfaches Parametrisieren der Systembeschreibung solche Untersuchungen.

## 6. Untersuchungen mit dem Interpreter - ein Beispiel

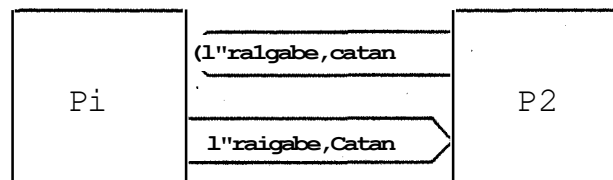
Als einfaches Beispiel zur Darstellung einiger Einsatzmöglichkeiten des Interpreters soll ein Prozeßsystem von zwei Prozessen auf gewünschte Eigenschaften untersucht werden.

Die beiden Prozesse tauschen miteinander Datenpakete aus. Damit ein Prozeß Daten senden kann, muß er von seinem Kommunikationspartner die Erlaubnis zum Senden erhalten ( "Freigabe empfangen"). Sobald ein Prozeß im Besitz der Freigabe ist, kann er ein Datenpaket senden. Darauf geht er wieder in den Anfangszustand über, wartet auf die Freigabe oder erteilt selbst die Freigabe zum Senden von Datenpaketen.

Ein Prozeß hat also die Möglichkeiten, auf die Sendeberechtigung zu warten und dann Daten zu senden oder die Sendeberechtigung abzugeben und dann Daten zu empfangen. Es soll ausgeschlossen sein, daß beide Prozesse gleichzeitig Datenpakete senden oder empfangen wollen.

Die Spezifikation in PASS besteht aus Kommunikationsstruktur und Ablaufsteuerung.

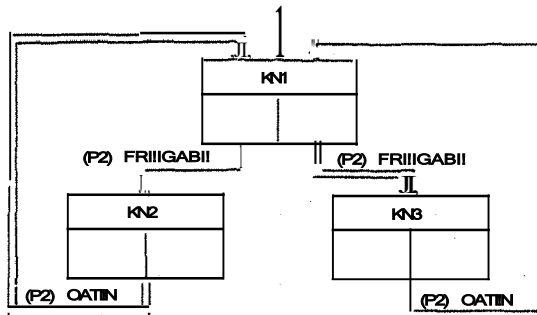
### 1. Kommunikationsstruktur



## II. Elemente des Prozeßsystems

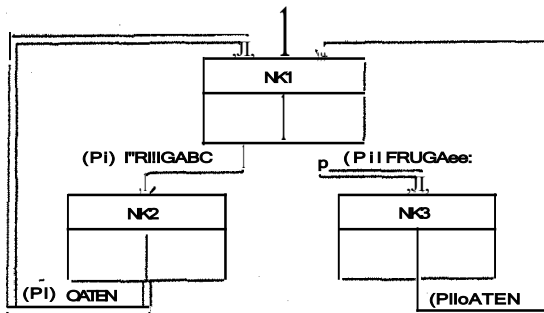
### EINZELPROZESS: P1

1. Kommunikationsmaschine: Wartebereich O
2. Ablaufsteuerung



### EINZELPROZESS: P2

1. Kommunikationsmaschine: Wartebereich 0
2. Ablaufsteuerung



Die Spezifikation ist verkürzt dargestellt, da die Beschreibung der Benutzermaschinen für jeden Prozeß fehlt. Da diese Information für den Interpreter jedoch nicht von Bedeutung ist, wurde sie weggelassen.

Kommunikationsstruktur und Ablaufsteuerungen werden mit Hilfe eines graphischen Editors erstellt. Der Editor ist in /KRAG86/ beschrieben. Über eine Dateischnittstelle erhält der Interpreter die notwendigen Informationen zum Aufbau des Programmgraphen für das Prozeßsystem.



## Schritt 1 : Definition des Prozeßsystems

Der Interpreterlauf beginnt mit der Aufforderung zur Definition des zu simulierenden Prozeßsystems. Der Benutzer gibt an, in welcher Datei die Kommunikationsstruktur des Prozeßsystems zu finden ist.

Nach Auswertung dieser Datei werden zusätzliche Informationen zur Simulation unter Betriebsaspekten erfragt (z. B. Verteilung der Prozesse auf Prozessoren, Betriebssystemstrategien usw.).

Mit vordefinierten Masken wird der Benutzer geführt, bis die Definition des Prozeßsystems vollständig ist.

Für das Beispiel-Prozeßsystem ist die Definition der Prozessorzuordnung in Bild 6-1 dargestellt.

Prozessorname : FIROZE:SSOR 1	
Strategie : "n 11 "	
Prozesse : P1 P2	X

Bild 6-1: Definition der Prozessorzuordnung

Der Benutzer trägt in die Maske die Prozessorbezeichnung, unter der der Prozessor bei der Simulation geführt wird, und die Prozessorvergabestrategie ein. Zulässige Strategien sind "verdrängend" und "nicht verdrängend"; wird keine Strategie ("nil") angegeben, darf dem Prozessor höchstens ein Prozeß zugeteilt werden.

Im unteren Bereich der Maske erscheinen alle Prozesse, die noch keinem Prozessor zugeordnet sind. Durch Markierung werden diejenigen Prozesse ausgewählt, die von dem aktuellen Prozessor quasiparallel bearbeitet werden.

In einer weiteren Maske werden jedem Prozeß die Attribute Wartebereichsgröße, Priorität und Dateiname der Ablaufsteuerung zugeordnet.

In diesem einfachen Beispiel benötigen wir keine Zeitsimulation, deshalb werden in der Maske "Ausführungszeiten" keine Zeiteinträge notwendig.



befindlichen Textfenster werden die Prozessorzuordnung und die Uhren der Prozessoren angezeigt. Da in unserem Fall keine Ablaufzeiten angegeben sind, bleiben die Uhren auf Null stehen.

Ein weiteres Fenster gibt die auszuführenden Simulationsschritte an. Der Benutzer kann sich in dieser Situation, durch Auswahl einer Ausgangskante der Zustandsknoten, entscheiden, welche Kommunikationsanweisung ausgeführt werden soll. Kanten sind in jedem Prozeß entsprechend der graphischen Darstellung von links nach rechts durchnummeriert.

Wird in Prozeß P2 die Kante 2 ausgewählt, entspricht dies der Durchführung der Sendeanweisung "(P1)Freigabe". Diese Anweisung wird ausgeführt, da P1 in einer Alternative des Kommunikationsknotens KNI auf die Botschaft "Freigabe" von P2 wartet. Die Anwendung der Graph-Produktion "Synchrones Senden" führt beide Prozesse in den jeweiligen Folgezustand über.

Wird bei einem Prozeß eine Kante ausgewählt, hat dies bei synchroner Kommunikation auf beide beteiligten Prozesse eine Auswirkung, falls die Anweisung ausführbar ist. Die Auswahl der Kante 1 in Prozeß P1 hätte den gleichen Zustandsübergang zur Folge gehabt. Die Graph-Produktion "Synchrones Empfangen" prüft, ob Prozeß P2 sendebereit ist. Da P2 in einer Alternative die gewünschte Botschaft anbietet, kann die Produktion angewendet werden.

Im Anfangszustand des Systems sind zwei verschiedene Zustandsübergänge möglich. Entweder P1 oder P2 sendet die Botschaft "Freigeben" und der jeweilige Partnerprozeß empfängt die Botschaft.

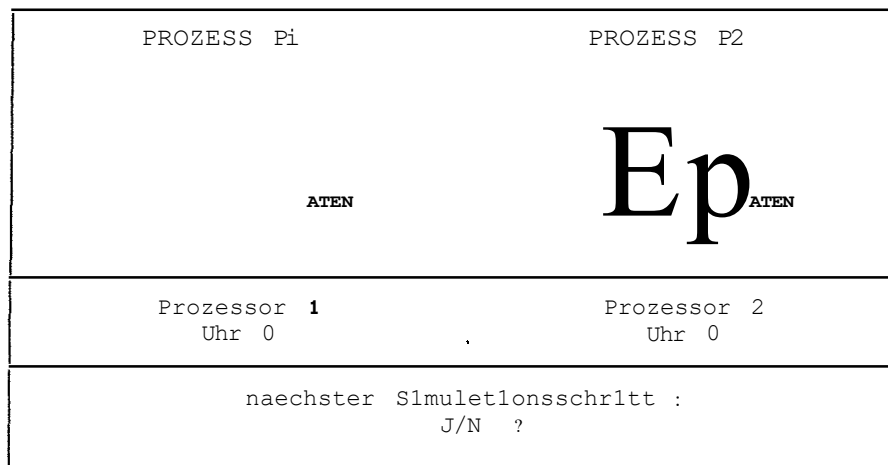
Eine Ereigniskombination bei der beide Prozesse senden oder empfangen kann nicht eintreten. In der ereignisorientierten Sichtweise von CCS wird eine solche Kombination untersucht. Sie führt bei einer parallelen Komposition der beiden Prozesse zu dem Verhalten "nil". Dies wird als unerwünschte Kommunikation oder auch als Verklemmung gesehen. Das Entfernen solcher "nil"-Übergänge wird in /CCS88/ als Absorptionsregel eingeführt zur Vereinfachung von CCS-Ausdrücken, die einen zusammengesetzten Prozeß beschreiben.

In dem hier verwendeten Semantik-Modell für PASS sind in den Zuständen NKI und KNI nur synchrone Kommunikations-Produktionen anwendbar, die zu erwünschter Kommunikation führen.

Die Empfangsproduktion enthält die Kausalitätsbedingung, nach der das Empfangen

nur möglich ist, wenn die Botschaft vom Partnerprozeß angeboten wird. Die Sendeproduktionen sind wegen der Bedingung "synchrone Kommunikation" nur anwendbar, wenn der Partner die Alternative mit Botschaftsempfang auswählt. Beim automatischen Ablauf der Simulation wird diese Tatsache zur Steuerung des Ablaufs ausgenutzt, um überflüssige Tests auf Anwendbarkeit von Produktionen zu eliminieren. Im Anfangszustand des Beispielsystems sind daher auch nur zwei Tests notwendig, um alle möglichen Zustandsübergänge zu finden.

Bild 6-3 zeigt den Systemzustand, der erreicht wird, wenn P2 die Freigabe erteilt. Jetzt kann Prozeß P1 Daten senden. P2 befindet sich im entsprechenden Empfangszustand. Damit ist ein Zustandsübergang verbunden, der wieder in den Ausgangszustand des Prozeßsystems führt. Aus dem Ausgangszustand ist ebenso die umgekehrte Kombination ableitbar: Prozeß P1 erteilt die Freigabe und P2 überträgt die Daten.



**Bild 6-3 : Datenaustausch**

Aus den vorgenommenen Untersuchungen kann man folgende Schlußfolgerungen ziehen:

- Das spezifizierte Protokoll ist in der Lage, Daten zu übertragen. Die beiden Partnerprozesse erreichen die Zustandskonfiguration { (P1)Daten senden, (P2)Daten empfangen } bzw. { (P1)Daten empfangen, (P2)Daten senden }. Dieses Verhalten ist für alle Zyklendurchläufe gesichert, da beide Prozesse den Anfangszustand im jeweiligen Zyklus erreichen.

- Das Prozeßsystem zeigt unendliches Verhalten. Bei einmaligem Durchlauf aller Zyklen treten keine Verklemmungen auf. Es werden auch keine Bedingungen verletzt, die eine Wiederholung der Zyklen undurchführbar machen könnten.

### Schritt 3 : Untersuchuna unter verluderten Betriebsparametern

#### a) asynchrone Kommunikation

Das obige Beispiel gibt das Verhalten eines einfachen Protokolls unter idealisierten Bedingungen wieder. Es wurde angenommen, daß beide Protokollpartner direkt und damit synchron kommunizieren können.

Nun soll ein Übertragungsmedium zwischen beiden Partnern modelliert werden. Jeder Prozeß kann eine Botschaft an die Übertragungsleitung abgeben, die von dieser zwischengespeichert wird. Damit besteht keine direkte Kopplung zwischen den Prozessen. In der Spezifikation wird dies ausgedrückt durch einen Wartebereich der Größe eins, der jedem Prozeß vorgelagert ist.

In der PASS-Beschreibung nur die Kommunikationsmaschine geändert werden. Durch den Wartebereich wird die Kommunikation jetzt asynchron ausgeführt. Die Ablaufsteuerungen bleiben unverändert.

Im Anfangszustand des Prozeßsystems {KN1, NK1} sind wie bisher die Aktionen { (P2)Freigabe senden, (P1)Freigabe empfangen } bzw. umgekehrt { \_(P2)Freigabe empfangen, (P1)Freigabe senden } ausführbar. Durch den Wartebereich werden zusätzliche Zustandsübergänge möglich. Jeder Prozeß kann die Botschaft "Freigabe" im Wartebereich des Empfängers ablegen. Das Prozeßsystem kann den Zustand {KN3, NK3 } annehmen.

In diesem Zustand prOft der Interpreter mit Hilfe der Produktion "Asynchrones Empfangen", ob eine Botschaft "Daten" vom entsprechenden Partner angeboten wird, d.h. im Wartebereich vorliegt. Da dies nicht der Fall ist, werden beide Prozesse blockiert. Im Prozeßsystem ist keine Aktion mehr ausführbar. Ein Verklemmungszustand wurde erreicht (s. Bild 6-4).

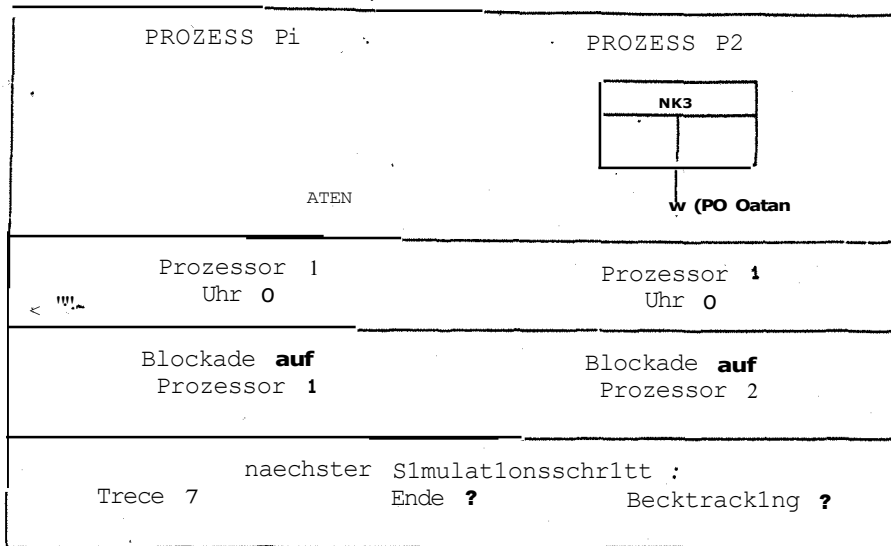


Bild 6-4 : Verklemmungs-zustand

Im Trace ist der Systemablauf festgehalten, der zur Verklemmung fñhrte. Der Benutzer kann Ober das Trace-Menñ den Ablauf untersuchen. Der Simulator bietet die Mñglichkeit, durchgefñhrte Ableitungsschritte wieder rñckgãngig zu machen. Aus dem Trace werden die angewendeten Graph-Produktionen gesucht und durch Ausfñhrung der dazu inversen Produktionen ein froherer Systemzustand hergestellt. Dadurch ist es zum Beispiel mñglich, P1 und P2 wieder in den Anfangszustand zuri.ickzufñhren und dort eine andere Alternative zu wãhlen.

#### b) Implementierung auf einem Monoprozessor

Das Verhalten des Prozeßsystems mit den Prozessen P1 und P2 soll jetzt unter der Annahme, daß beide Prozesse von einem Prozessor bearbeitet werden, untersucht werden. Beide Prozesse sollen wieder Ober einen Wartebereich kommunizieren.

Durch Ånderung der Systembeschreibung wird das Spezifikationsmodell neu definiert. Beide Prozesse werden dem Prozessor 1 zugeordnet. Das Betriebssystem verwendet eine verdrãngende Strategie zur Prozessorvergabe.

Das Verhalten des Prozeßsystems ändert sich durch diese Implementierung gegentñber des zweiten Beispiels nur unwesentlich. Die parallele Aktion { (P2)Freigabe senden, (P1)Freigabe senden } tritt nicht mehr auf. Sie ist durch eine sequentialisierte Folge der Einzelaktionen ersetzt. Die Reihenfolge wird dadurch



wurde. Da P1 noch am Datenempfang blockiert ist, erfolgt ein Prozeßwechsel und P2 wird aktiv. Im nächsten Simulationsschritt hat P2 jetzt nur die Möglichkeit, die Botschaft "Freigabe" aus dem Wartebereich zu entnehmen. Die Prioritätenrelation verhindert das Senden der Botschaft "Freigabe" an P1. Der vorher noch mögliche Verklemmungszustand ist nicht mehr erreichbar.

Untersucht man diese Lösung in der Konfiguration von Beispiel a), asynchrone Kommunikation, stellt man fest, daß sie hier keine Auswirkung hat. Durch die Unabhängigkeit der Sendeoperationen in der Mehrrechner-Umgebung, kann der Fall eintreten, daß beide Prozesse gleichzeitig senden. Dies wird durch die Priorität nicht verhindert. Erst die Sequentialisierung der Aktionen durch den Ablaufplan des Monoprozessors läßt die Prioritätenregelung wirksam werden.

Für den Entwerfer des Prozeßsystems ergeben sich durch die Untersuchung folgende Hinweise:

- In einer Mehrprozessorumgebung muß die Kollision, die durch gleichzeitiges Senden entsteht, geeignet aufgelöst werden, um eine Verklemmung zu vermeiden. Eine Möglichkeit zur Auflösung der Kollision ist die Einführung einer variablen Zeitliberwachung.
- In einer Einrechnerumgebung kann die Kollision durch einfache Prioritätenregelung verhindert werden. Damit ist kein zusätzlicher Synchronisationsaufwand notwendig; das Protokoll wird effizienter.



## 7. Aufbau des Interpreten

Der Interpreter ist Bestandteil einer Entwicklungsumgebung für verteilte Systeme zur Prozeßautomatisierung /Holl 89/. Er ist in Verbindung mit dem Transformationswerkzeug /Krag 86/ zu sehen, das PASS-Spezifikationen erfaßt und durch Programmsynthese automatisch PEARL-Code erzeugt.

Bild 7-1 zeigt den Zusammenhang zwischen beiden Werkzeugen.

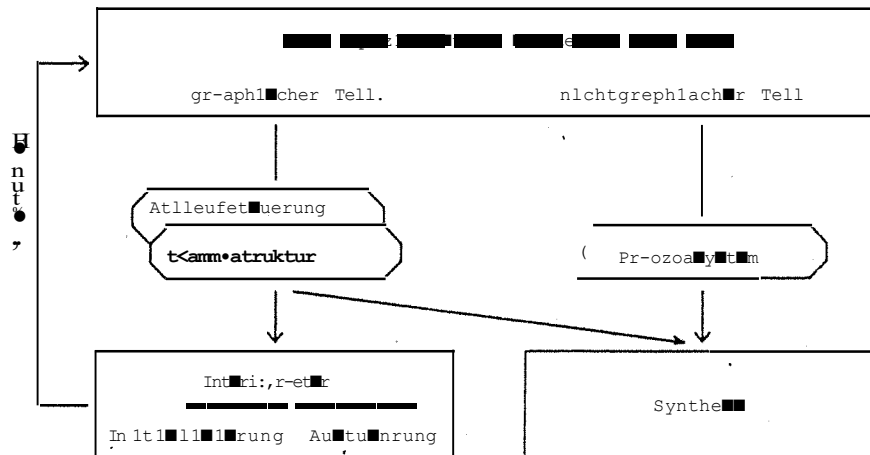


Bild 7-1 : Programmierungsumgebung für verteiltes PEARL /Holl 89/

Dateien, die die Kommunikationsstruktur eines Prozeßsystems und die Ablaufsteuerungen der dazu gehörenden Prozesse enthalten, bilden die Schnittstelle zwischen beiden Werkzeugen. Der Interpreter benötigt nur Informationen, die im graphischen Teil der PASS-Spezifikation enthalten sind.

Dabei werden folgende Listen übernommen:

- aus der Kommunikationsstruktur:
  - Liste aller Prozesse,
  - Liste aller Prozeßblöcke,
  - Liste aller Prozeßgruppen,
- aus den Ablaufsteuerungen:
  - Liste aller Knoten.

Elemente des Typs Knoten enthalten die Komponenten Knotenart, Knotentext und einen Zeiger auf eine verkettete Liste von Kanten. Diese Kantenliste stellt

die logische Verbindung zwischen zwei Knoten der Ablaufsteuerung her. Sie enthält alle von einem Knoten wegführenden Kanten und deren Zielknoten. Dies genügt, um den Programmgraphen, der die Grundlage für die Ausführung eines Spezifikationsmodells bildet, zu erstellen.

## **7.1 Der Interpreter aus Benutzersicht**

Die Benutzung des Interpreters erfolgt in 3 Phasen.

### **1) Initialisierung**

In dieser Phase wird die Simulation des Prozesssystems vorbereitet. Der Programmgraph wird aufgebaut und entsprechend den Benutzerangaben attribuiert.

Dazu werden vom Benutzer folgende Angaben menügesteuert erfragt:

- Namen der Dateien, die die Kommunikationsstruktur und Ablaufsteuerungen enthalten,
- Zuordnung von Prozessen zu Prozessoren,
- Betriebssystemstrategie für Prozessorvergabe,
- Größe des Wartebereichs von Prozessen bzw. Prozeßgruppen,
- Ausführungszeiten für Prozeßwechsel und Betriebssystemaufrufe,
- Bearbeitungszeit für interne Funktionen und Operationen,
- Übertragungszeit für Botschaften.

Diese Angaben werden auf einer Datei gespeichert. Der Inhalt der Datei kann zusammen mit dem Ergebnisprotokoll von Simulationsläufen ausgegeben werden, um eine vollständige Dokumentation über die verwendeten Konfigurationsparameter zu erhalten.

Die Angaben können jederzeit verändert werden. Bis auf die Information über die Prozessorzuordnung wirken sich die Parameter nur auf die Attributierung des Programmgraphen aus. Bei einer Änderung muß nicht der Programmgraph selbst neu aufgebaut werden, sondern nur die Attributwerte entsprechend initialisiert werden.

Die Angabe der Ausführungszeiten für Knoten und Kanten einer Ablaufsteuerung erfolgt mehrstufig:

- keine Zeitsimulation

Alle Ausführungszeiten besitzen den Wert Null. Beim Simulationsablauf werden Time-Out-Kanten wie Botschaftskanten behandelt und in die Steuerungsstrategie mit einbezogen.

- gleiche Zeiten für alle Kommunikationsanweisungen

Für eine grobe Abschätzung des Kommunikationsaufwandes können allen Kommunikationsanweisungen feste Übertragungs- und Betriebssystembearbeitungszeiten zugeordnet werden. Interne Funktionen und Operationen werden als Ereignisse ohne Zeitdauer betrachtet.

- Zuordnung einer individuellen Ausführungszeit zu jeder Aktion

Jede Botschaft und jede interne Funktion oder Operation kann mit einer spezifischen Übertragungs- und Bearbeitungszeit attribuiert werden. Alle Werte sind mit 0 initialisiert. Die Belegung kann also auch gezielt für einzelne Knoten und Kanten erfolgen, wenn die Simulation auf bestimmte interessierende Aktivitäten begrenzt sein soll.

## 2) Simulationsphase

In der zweiten Phase wird der eigentliche Testvorgang abgewickelt. Der Benutzer entscheidet, ob die Untersuchung des Spezifikationsmodells automatisch oder interaktiv erfolgen soll.

Beim automatischen Ablauf kann zwischen den verschiedenen Steuermechanismen ausgewählt werden.

Bei zufallsgesteuertem Ablauf der Simulation wählt ein Zufallszahlengenerator bei alternativen Kommunikationsanweisungen eine Kommunikationskante aus bzw. wählt eine von mehreren Funktions/Operationskanten. Tritt eine Verklemmungssituation auf, wird das in der Trace-Datei vermerkt und der Ablauf mit Hilfe der inversen Produktionen auf die letzte Anweisung, bei der eine Auswahl erfolgte, zurückgesetzt.

Beim zielgesteuerten Ablauf hat der Benutzer die Auswahl unter den Zielkriterien zur Erreichung einer bestimmten Zustandskonfiguration:

- zeitliche kürzeste Pfade finden
- Wartebereiche voll benutzen
- Pfad\_e mit geringsten Kommunikationsaufwand finden

Die Pfade der einzelnen Prozesse zu einem bestimmten Zielzustand werden dann nach diesen Kriterien geordnet und untersucht.

Der Untersuchungsbereich kann durch Setzen von Haltepunkten eingeschränkt werden. Dazu werden Knoten der verschiedenen Prozesse als terminale "Stop"-Knoten definiert. Bei Erreichen solcher Stopknoten werden Pfade nicht mehr weiterverfolgt.

Ist das Simulationsziel erreicht, wird die Kontrolle an den Benutzer zurückgegeben.

Beim interaktiven Ablauf kann der Benutzer jeden Schritt des Simulators einzeln steuern. Dazu steht ihm ein Hauptmenü: zur Verfügung, das sechs verschiedene Untermenüs aufruft.

Im Menü "Trace" kann die Erstellung einer neuen Trace-Liste initialisiert, die letzte Aktion oder der gesamte bisherige Testverlauf abgerufen werden.

Das Menü "Wartebereich" stellt Funktionen zum Abfragen und Manipulieren einzelner Wartebereiche zur Verfügung. So kann z.B. die Belegung eines Wartebereichs ermittelt oder die Größe verändert werden.

Im Menü "Programmgraph" werden Funktionen zur Kontrolle des Programmgraphen aufgerufen. Man erhält Informationen über den aktuell bearbeiteten Knoten, dessen Vorgänger- und Nachfolgerknoten und Kommunikationspartner. Pfade, die von aktuellen Knoten zu einem Zielknoten, können aufgelistet werden.

Im Menü "Einzelprozeß" werden Attribute von Prozessen abgefragt und verändert. Dies bezieht sich auf Priorität und Zustand.

Das Menü "Prozessorverwaltung" enthält Funktionen, die Informationen liefern über:

- Prozesse, die einem Prozessor zugeordnet sind,
- lauffähige Prozesse des Prozessors,
- Betriebssystemstrategie auf dem Prozessor,
- Uhrzeit,
- Haltepunkte.

In diesem Menü können Uhrzeit und Haltepunkte auch gesetzt werden.

Funktionen des Menüs "Systembeschreibung" erlauben die vollständige Attribu-

tierung des Programmgraphen.

Ist der interaktive Ablauf gewählt, wird der aktuelle Simulationsstand graphisch am Bildschirm angezeigt (s. Kap. 6).

### 3) Auswertung

Die durchgeführten Aktionen im Spezifikationsmodell werden auf einer Trace-Datei festgehalten. Hier wird eingetragen, welcher Knoten bei welchem Prozeß bearbeitet wurde. Ist der Test unter Zeitsteuerung abgelaufen, wird auch die Prozessorzeit festgehalten. Die Trace-Datei liefert die Datenbasis zu Leistungsaussagen über ein Prozeßsystem. Im automatischen Ablauf durchgeführte Aktionenfolgen können analysiert werden.

Es kann die Kommunikationsbelastung zwischen verschiedenen Prozessoren oder Prozessen gemessen werden. Diese Information ist wichtig, um eine geeignete Netzwerktopologie definieren zu können. Durch die Zuordnung von Prozessen zu Prozessoren werden die Kommunikationskosten im verteilten System bestimmt. Will man diese minimieren, muß der Kommunikationsaufwand bestimmt werden.

Ebenso können aus dieser Datei statistische Angaben über zeitliche Abläufe gewonnen werden. Dies ist wichtig, um Leistungsengpässe zu ermitteln. Durch Verändern von Parametern, wie Betriebssystemverwaltungszeiten oder Übertragungszeiten, ist die Untersuchung von Umgebungsbedingungen bei einer Realisierung des simulierten Prozeßsystems möglich.

## 7.2 Interner Aufbau des Interpreters

In Bild 7-2 ist die Modul-Hierarchie des Interpreters dargestellt. Zwischen den Modulen besteht eine Benutzbeziehung, dargestellt durch Pfeile. Achtecke beschreiben Datenstrukturen, Doppelpfeile die zugehörigen Zugriffsoperationen.

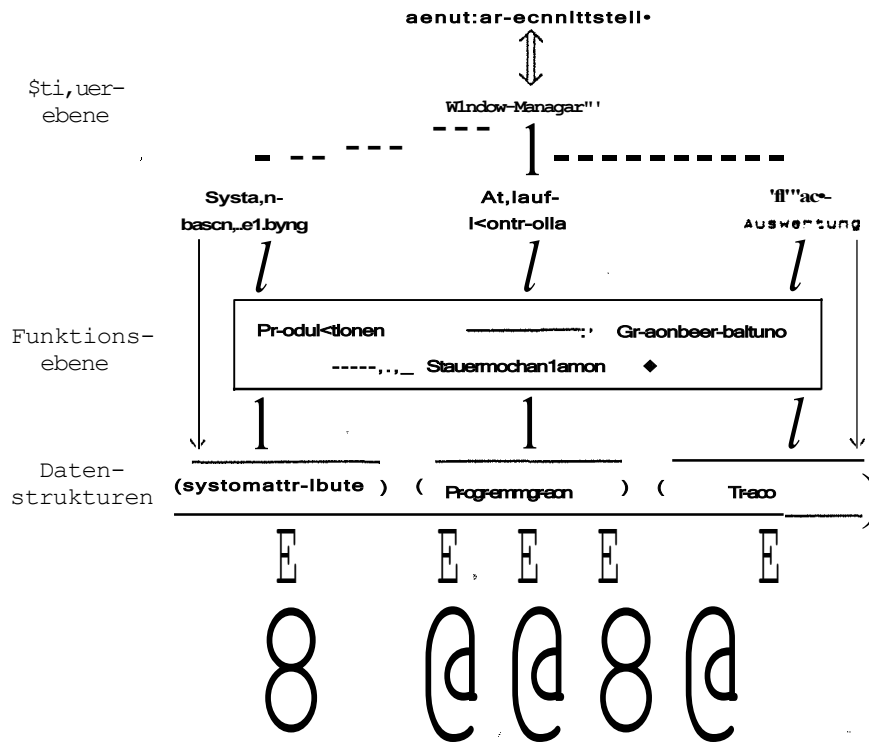


Bild 7-2: Aufbau des Interpreters

D 1 - D 5 stellen folgende Dateien dar:

- D 1 : Systembeschreibung aus der Initialisierungsphase
- D 2 : PASS-Kommunikationsstruktur
- D 3 : PASS-Ablaufsteuerungen (eine Datei für jeden Prozeß)
- D 4 : Textdatei (enthält Texte der Komm.-Struktur und Ablaufsteuerungen)
- D 5 : Trace-Datei

### 7.2.1 Window-Manager

Die Benutzerschnittstelle des Interpreters wird im wesentlichen durch die Verwendung von Fenstertechniken gestaltet. Es wurde ein Fenstersystem entworfen, das vier verschiedene Typen von Fenstern enthält:

- Menue-Fenster zur Systembeschreibung
- Eingabe-Fenster zur Eingabe von Steuerbefehlen an den Interpreter und zur Systemabfrage,
- Graphi-Fenster zur Angabe der erreichten Systemzustände,
- Textfenster zur Ausgabe von Interpretermeldungen, wie z.B. aktuelle Uhrzeit, Hiltepunkte, Systemzuständen.

Kommandos, die im Eingabe-Fenster an den Interpreter gegeben werden, können an die Module Systembeschreibung, Ablaufkontrolle und Trace-Auswertung gerichtet sein. Eingaben werden als Zeichenstring weitergereicht. Die Analyse der Kommandos erfolgt in den jeweiligen Modulen.

### 7.2.2 Steuerebene

Die Steuerebene enthält die drei Module Systembeschreibung, Ablaufkontrolle und Trace-Auswertung.

Das Modul "Systembeschreibung" definiert die Menüs, die vom Benutzer Informationen über implementationsabhängige Details erfragen. Diese Grundmenüs werden mit der in der Datei Kommunikationsstruktur enthaltenen Beschreibung des Spezifikationsmodells gefüllt. Dazu verwendet das Modul Zugriffsfunktionen, die im Modul "Graphbeschreibung" der Funktionsebene enthalten sind. Über diese Funktionen wird auch ein aufgebauter Programmgraph neu attribuiert, wenn die Systembeschreibung geändert wurde.

Wird die Zuordnung von Prozessen zu Prozessoren neu definiert, muß die Struktur des Programmgraphen teilweise geändert werden. Es werden Kanten zwischen dem Prozeßknoten und einem Prozessorknoten gelöscht bzw. neu eingerichtet.

Das Modul "Trace-Auswertung" enthält Funktionen zur Bearbeitung der Trace-Datei. Damit werden gespeicherte Interpreterabläufe gelesen und formatiert an den Benutzer ausgegeben. [n dieses Modul gehören auch die Algorithmen zur statistischen Auswertung (z.B. Kommunikationskosten) der Trace-Information.

Die Ablaufkontrolle enthält die Teilmodule:

- Dialogmodul

Dieses Modul stellt den Window-Manager die notwendige Information über den Interpretationsablauf zur Verfügung. Damit kann die graphische Ausgabe aufgebaut und gesteuert werden.

Meldungen über den Interpretationsablauf werden in Klartext umgewandelt. Information, die im Programmgraphen in codierter Form abgelegt ist (z.B. Name des Zustandsknoten), wird durch Zugriff auf eine Textdatei in Bezeichner des Spezifikationsmodells rückübersetzt.

Eingaben durch den Benutzer werden geprüft und zur Bearbeitung weitergeleitet:

- 1) Abfragen über den aktuellen Zustand des Programmgraphen und den Aufbau (z.B. welche Zyklen oder Pfade enthalten sind) werden durch Aufruf der Funktionen des Moduls Graphbearbeitung erledigt. Ebenso können damit Haltepunkte gesetzt werden.
- 2) Der Anstoß des interaktiven oder automatischen Ablaufs wird an das Modul Ablaufsteuerung weitergeleitet.

- Modul Ablaufsteuerung

Dieses Modul steuert den Aufbau des Programmgraphen, kontrolliert die Produktionen, die zum Ablauf des Spezifikationsmodells angewendet werden und führt das Backtracking durch Aufruf inverser Produktionen beim automatischen Ablauf durch.

Verwendet werden alle Funktionen, die die Funktionsebene anbietet.

### 7.2.3 Funktionsebene

Die Funktionsebene stellt die Produktionen zur Manipulation des Programmgraphen, entsprechend der Semantik der PASS-Sprachelemente zur Verfügung. Zu jeder Produktion existiert auch eine inverse Produktion, die durchgeführte Aktionen rückgängig machen kann. Dazu werden Zugriffsfunktionen des Datenmoduls "Trace" verwendet, um Einträge lesen und evtl. löschen zu können.



Die Produktionen sind in /Fede 86/ beschrieben. Sie wurden erweitert, um die Attributierung und den Einbau von Steuermechanismen zu realisieren. Dazu werden Routinen zur Graphbearbeitung (Traversieren des Graphen, Suchen von Pfaden unter Auswertung von Attributen, Suchen von Zyklen) aufgerufen. Diese Routinen bedienen sich ebenfalls der Steuermechanismen, um Kriterien, wie z.B. Finden von zeitlich kürzesten Pfaden, zu erfüllen.

## 8. Erfahrungen

Ziel dieser Arbeit war die Entwicklung eines Interpreters, der im Rahmen einer Programmierungsumgebung für verteilte Echtzeitanwendungen die Überprüfung von Spezifikationen ermöglicht. Im Rahmen der Arbeit wurden drei Schwerpunkte gesetzt:

- Untersuchung der Anforderungen an die Ausdruckskraft einer Spezifikationssprache zur Beschreibung der besonderen Eigenschaften von Realzeitsystemen und Vergleich mit existierenden Spezifikationstechniken.
- Formale Fundierung der Spezifikationsmethode PASS durch Definition eines Semantikmodells mit Hilfe eines Graphersetzungssystems. Damit ist Kommunikations-, Zeit- und Betriebssystemverhalten beschreibbar.
- Entwicklung eines Interpreters, der PASS-Spezifikationen ausführbar macht und so die Validierung von Spezifikationen unterstützt.

Mit dem hier entwickelten Verfahren und dem zugehörigen Interpreter ist ein Nachbilden der Komposition und des Kommunikationsverhaltens paralleler Prozesse möglich, allerdings ohne Berücksichtigung des datenabhängigen Verhaltens der Prozesse. Das entsprechende Sprachmittel, die Benutzermaschine, besitzt noch keine formale Fundierung. Hier ist ein Schwerpunkt für weitere Untersuchungen zu sehen.

Erste Einsätze des Werkzeugs im praktischen Betrieb scheinen die realisierten Ansätze prinzipiell zu bestätigen. Insbesondere können Fehler schnell in der Spezifikation gefunden und Änderungen der Spezifikation überprüft werden.

Die Implementierung des Interpreters auf einem Personalcomputer erreicht die Grenzen der technischen Gegebenheiten. Rechenleistung und Speicherplatz ermöglichen nur eine Untersuchung kleinerer Probleme. Hier muß ein Übergang zu leistungsfähigeren Systemen erfolgen.

• Danksaauna

An dieser Stelle möchte ich mich bei Herrn Dr. P. Holleczeck und Herrn Prof. Dr. H.J. Schneider für die Betreuung dieser Arbeit herzlich bedanken.

Herrn Prof. Dr. F.Hofmann danke ich für die Übernahme des Zweitgutachtens.

Mein Dank gilt Frau P. Bächle für ihren unermüdlichen Kampf mit unleserlichen Manuskripten, Herrn J. Laskowski für die Erstellung der Zeichnungen und meinen Kollegen für sorgfältiges Korrekturlesen.

Herzlichen Dank möchte ich auch Frau B. Holleczeck für die moralische Unterstützung in kritischen Tagen sagen.

Frau Christiane Feder hat mich ertragen, auch wenn ich in der Hektik der Arbeit besonders unleidlich war, und war stets diskussions- und hilfsbereit zur Stelle. Ihr sei besonders gedankt.

## Literaturverzeichnis

- /AFHK 85/ C.Andres, A.Fleischmann, P.Holleczeck, U.Hittmer, R.Kummer: "Eine Methode zur Beschreibung von Kommunikationsprotokollen",  
in: Kommunikation in Verteilten Systemen I, Proceedings GI/NTG Fachtagung Anwendungen, Betrieb, Grundlagen, Karlsruhe, März 1985
- /Boeh 83/ B.Boehm: "Software Engineering Economics" Prentice Hall International, Englewood Cliffs, 1983
- /Boch 83/ G.v.Bochmann: "Concepts for Distributed Systems Design", Springer Verlag, 1981
- /Boud 87/ G.Boudol, I.Kastellani: "On the Semantics of Concurrency: Partial Orders and Transition Systems" TAPSOFT 87 Vol.1, LNCS 249, Springer Verlag, 1987
- /CCS 88/ A.Giacalone, S.Smolka: "Integrated Environments for Formally Well-Founded Design and Simulation of Concurrent Systems", IEEE Transactions on SE, Vol. 14 No. 6, Juni 1988
- /DeNi 87/ R.DeNicola, M.Hennessy: "CCS without  $\tau$ ", TAPSOFT 87, Vol. 1, LNCS 249, Springer Verlag, 1987
- /Dijk 75/ E.W.Dijkstra: "Guarded Commands, Nondeterminacy and Derivation of Programs", Communications ACM, August 1975
- /Fede 86/ Chr.Feder: "Ein Interpretierer für PASS-Spezifikationen basierend auf einem Graphersetzungssystem", Diplomarbeit am IMMD II, Universität Erlangen, 1986
- /Flei 84/ A.Fleischmann: "Ein Konzept zur Darstellung und Realisierung von verteilten Prozeßautomatisierungssystemen",  
Dissertation Universität Erlangen, 1984
- /Flei 87/ A.Fleischmann: "Description of the Specification Technique PASS", IBM Technical Report, ENC Heidelberg, 1987
- /FIHo 83/ A.Fleischmann, P.Holleczeck, G.Klebes, R.Kummer:  
"Synchronisation und Kommunikation verteilter Automatisierungsprogramme", Angewandte Informatik, Vieweg Verlag 7/1983
- /Färb 84/ G.Färber: "Architektur zukünftiger Prozeßrechnersysteme", Prozeßrechner 84, Informatik Fachberichte, 1984
- /GMD 88/ "Theorie verteilter Systeme"  
Der GMD-Spiegel 2/3 88, GMD
- /Göhn 81/ P.Göhner: "Ingenieurgerechte Spezifikation der Synchronisierung paralleler Rechenprozesse"  
Dissertation Universität Stuttgart, 1981

- /HAnd 84/ W.HAndler, U.Herzog, F.Hofmann, H.J.Schneider: "Multiprozessoren für breite Anwendungsgebiete: Elangen General Purpose Array", GI/NTG-Fachtagung Architektur und Betrieb von Rechnersystemen, Informatik Fachberichte Nr. 78, Springer Verlag 1984
- /Hoar 78/ C.A.R.Hoare: "Communicating Sequential Processes", Communications ACM, August 1978
- /Hoar 85/ C.A.R.Hoare: "Communicating Sequential Processes", Prentice Hall International, Englewood Cliffs, 1985
- /Hofm 84/ F.Hofmann: "Betriebssysteme: Grundkonzepte und Modellvorstellungen", B.G.Teubner Verlag, Stuttgart, 1984
- /Holl 89/ P.Holleczer; "A Programming Environment for distributed Realtime Applications", Hawaiian International Conference on System Sciences, zur Veröffentlichung 1989
- /KaSm 83/ P.Kanellakis, S.Smolka: "CCS expressions, finite state processes, and three problems of equivalence, Proc. of 2nd ACM Symp. on Principles of Distributed Computing, Montreal, August 1983
- /Kera 82/ S.Keramidis: "Eine Methode zur Spezifikation und korrekten Implementierung von asynchronen Systemen" , Arbeitsberichte des IMMD, Erlangen, 1982
- /Kimm 79/ R.Kimm, W.Koch, W.Simmonsmeier, F.Tontsch: "Einführung in Software Engineering", de Gruyter Verlag, 1979
- /Krag 86/ G.Kragl: "Eine Programmierumgebung für verteiltes Pearl", Dissertation Universität Erlangen, 1986
- /Kreo 86/ H.J.Kreowski: "Is Parallelism already Concurrency, Derivations in Graph Grammars", in : Ehrig et al. : Graph Grammars and their Applications to Computer Sciences", LNCS 291, Springer Verlag, 1986
- /Kumm 83/ R.Kummer: "Entwicklung von Betriebssystembausteinen für Guarded Regions und Botschaftsoperationen im Rahmen eines Realzeitprogrammiersystems", Diplomarbeit am IMMD IV der Universität Erlangen, 1983
- /Lamp 78/ L.Lamport "Time, Clocks, and the Ordering of Events in a Distributed System", Communications ACM 21/7, 1978
- /Laut 73/ K.Lautenbach: "Exakte Bedingungen der Lebendigkeit für eine Klasse von Petri-Netzen", Ber.Nr.BMFT-GMD-82, Ges. für Mathematik und Datenverarbeitung, St. Augustin, 1973
- /Levi 81/ P.Levi: "Betriebssysteme für Realzeitanwendungen", Datakontext Verlag, Köln, 1981
- /Miln 80/ R.Milner: "A Calculus of Communicating Systems", LNCS , Springer Verlag, 1980

- /Nagl 79/ M.Nagl: "Graph Grammatiken",  
Vieweg Verlag 1979
- /Nehm 84/ J.Nehmer: "Systemarchitektur von Realzeitsystemen" ,  
Informatik Spektrum Bd 7 Heft 2 April 1984
- /Nehm 85/ J.Nehmer: "Softwaretechnik für verteilte Systeme", Springer Verlag,  
1985
- /PROT 88/ BiUington/Wheeler/Wilbur-Ham: "PROTEAN: A High-Level Petri Net  
Tool for the Specification and Verification of Communication Protocols",  
IEEE Trans. on SE, Vol. 14 No. 3, März 1988
- /Reis 82/ W.Reisig: "Petrinetze", Springer Verlag, 1982
- /Röhr 86/ J.Röhrich: "Parallele Systeme", Informatik-Fachberichte Nr. 117,  
Springer Verlag, 1986
- /Star 80/ P.Starke: "Petri-Netze", VEB Deutscher Verlag der Wissenschaften,  
Berlin 1980
- /Stot 86/ D. Stotts: "Software Modeling with Timed Petri Nets in the PFG  
Environment", Technical Report CS-TR-1712, University of Maryland,  
1986
- /Syrb 78/ M.Syrbe: "Basic Principles of Advanced Process Control System  
Structures", Proc. 7th Triennial World Congress, IFAC, Helsinki,  
Pergamon Press New York 1978
- /Timm 82/ M.Timm: "Grundlagen von Anforderungs- und Entwurfsspezifikationen  
im Prozeß der Software-Entwicklung", GMD-Studien Nr. 66, Gesell-  
schaft für Mathematik und Datenverarbeitung, St. Augustin, 1982
- /Tayl 83/ R.Taylor: "Complexity of Analyzing the Synchronisation Structure  
of Concurrent Programs", Acta Informatica Vol. 19, 1983

## Stichwortverzeichnis

- Ablaufplan (26)
- Ablaufsteuerung (50), (53)
- Ableitbar
  - direkt sequentiell (79)
  - gemischt (95)
  - sequentiell, programmiert (81)
- Ableitung
  - gemischt (99)
- Abstraktion
  - horizontal (24)
- Adressierung
  - direkt (22)
- Aktionsknoten (83)
- aktiv (62).
- Aktivität (67)
- Äquivalenz
  - CCS (46)
- Attribut (70), (78)
- Attributberechnungsregel (77)
- Attributierung (85)
- Attributzuordnung (70)
- Auswahl
  - nichtdeterministisch (23)
- Auswahl-Operator (107)
- Befehlszähler (85)
- Behinderungsfrei (29)
- Benutzermaschine (50), (56)
- bewachte Anweisung (22)
- blockiert (62)
- CCS (42)
  - Analysefähigkeit (49)
  - Ausdrucksfähigkeit (48)
- Church-Rosser Eigenschaft (101)
- Echtzeitsystem (3), (12)
  - Eigenschaften (14)
- Struktur (13)
- Einbettung (74)
  - identische (78)
- Einzelnamensknoten (83)
- Empfangsanweisungen
  - asynchron (100)
  - synchron (100)
- Ereignisse
  - disjunktiv verknüpft (19)
  - konjunktiv verknüpft (19)
  - vorrangige Behandlung (19)
  - zeitlich überwacht (20)
- Erreichbarkeit (37), (98)
- Ersetzung
  - gemischt (95)
- Ersetzungsschritt
  - programmiert (82)
- Expansionstheorem (46)
- Fall (34)
- Fallgraph (37)
- Fenster
  - Interpreter (99)
- Free-Choice-Netz (38)
- Funktion
  - intern (100)
- Graph
  - attribuiert (71)
  - komplementär (74)
- Graph-Produktion
  - attribuiert (77)
- Graphersetzungssystem
  - programmiert (80)
  - sequentiell (76)
- Guard (22)
- Guarded region (22)

Inhibitor-kante (40)	zielgerichtet (109)
Interleaving-Modell (47)	zufallsgesteuert (108)
Interne Funktion (55)	Modellprozeß (17)
Interne Operation (55)	Netz (32)
Interpretation (75)	Operation
Interpreter	intern (100)
Analysefähigkeit (109)	Operator (75), (76)
Aufbau (127)	Ordnung
Interpretierer	linear (100)
abstrakt (97)	PASS (50)
Kanal {22}	Analysefähigkeit (57)
<b>Kantenmarkierung</b> (70)	Ausdrucksfähigkeit (56)
<b>Knotenbezeichnung</b> (71)	Petri-Netz (31)
<b>Knotenmarkierung</b> (70), (71)	Analysefähigkeit (41)
<b>Kommunikation</b>	Ausdrucksfähigkeit (39)
asynchron (22), (53)	erweitert (41)
intern (47)	sicher (36)
synchron (22), (53)	Pfad (105)
<b>Kommunikationsmaschine</b> (50), (52)	Port (21)
<b>Kommunikationsmechanismen</b>	Programmgraph (82)
botschaftsorientiert (21)	Bestandteile (88)
<b>Kommunikationsprobleme</b> (17)	Prototyping
<b>Kommunikationsstruktur</b> (50), (51)	evolutionäres (7)
<b>Komposition</b>	experimentelles {7}
parallel (48)	exploratives (7)
Konfusion (38)	Prozeß {24}
Kontrolldiagramm (81 ), (90)	quasi-parallel (62)
Kontrollebene (13)	technisch (12)
Korrektheit	Prozeßautomatisierung (2)
extern (5), (98)	Prozeßknoten (85)
intern (5)	Prozesse
lauffähig (62)	quasi-parallel (3)
Lebendigkeit (37)	Prozessorknoten (85)
Linie (36)	Prozessorvergabe-strategie (28}
Markierung (33)	nichtverdrängend (27)
Meßbarkeit (6)	verdrängend (27)
Modellablauf	Prozeßtreu (29)
dialoggesteuert (108)	Prozeßumschalter (27)
prioritätsgesteuert (109)	Realzeitsystem (3)



Sammelnamensknoten (83)	Zustandsautomat
Schaltregel (34)	erweitert endlich (50)
Schnitt (36)	Zustandsmodell (60)
Schritt (35)	Zustandsübergang
Semantikmodell	alternativ (105)
PA (97)	Zyklus (105)
Sendeanweisungen	
asynchron (100)	
synchron (100)	
Simulation	
Auswertung (126)	
automatisch (124)	
Initialisierung (123)	
interaktiv (125)	
Software-Entwicklungszyklus (4)	
Spezifikation	
unvollständig (104)	
Timed-Petri-Netz (65)	
True Concurrency-Modell	
(36).	
Uhr (68)	
global (66),(69)	
lokal (66)	
Uhren	
verteilte (68)	
Untergraph (73)	
Untergraphentest (90)	
Unterkontrolldiagrammen (90)	
Validierung (5)	
Verteiltes PEARL (23)	
Wächter (22)	
Wartebereich (22), (53), (100)	
Wartebereichsknoten (83)	
Zeit	
virtuelle (66)	
Zeitbegriff (63)	
Zeitdauer (67)	
Zeitüberwachungs-Bedingung (23)	