Informatik-Spektrum 2, 219–229 (1979)

Spektrum

Das Prozeßkonzept in Betriebssystemen und Programmiersprachen I*

J. F. H. Winkler

Institut für Informatik 3 der Universität Karlsruhe

Zusammenfassung. Das Konzept des (Rechen-) Prozesses wird in Betriebssystemen, Kommando- und Programmiersprachen verwendet. Die Verwendung dieses Konzeptes brachte eine Vereinfachung der Konstruktion von Betriebssystemen mit sich. Für Echtzeitsprachen ist es ein essentielles Sprachelement. In der vorliegenden Arbeit werden eine Reihe verschiedener Prozeßkonzepte mit einem einheitlichen Begriffsschema beschrieben. In einer Übersicht werden die Prozeßkonzepte von sieben algorithmischen und sechs Echtzeitsprachen anhand von 25 Merkmalen charakterisiert und einander gegenübergestellt.

Summary. The concept of process is used in operating systems, command and programming languages. Using this concept the construction of operating systems becomes much more easier. The concept of process is an essential feature of real-time languages. This paper describes several different realizations of this concept by means of a uniform framework. The process concepts of seven algorithmic and six real-time languages are characterized by 25 features. A survey of this characterization is given in a tabular summary.

Vorbemerkung

Der vorliegende Aufsatz erscheint wegen seines Umfanges in zwei Teilen. Der erste Teil in diesem Heft enthält neben der Einführung die Abschnitte über Betriebssysteme und allgemeine höhere Programmiersprachen. Der zweite Teil, der im ersten Heft 1980 erscheinen wird, beinhaltet die Abschnitte über Echtzeitprogrammiersprachen, eine zusammenfassende Übersicht und das Literaturverzeichnis.

1 Zum Begriff des Prozesses

Nach DIN 66201 ist der Terminus „Prozeß" definiert als „Umformung und/oder Transport von Materie, Energie und/oder Information". Diese Definition ist sehr allgemein gefaßt und ist auf viele Vorgänge in Naturwissenschaft und Technik anwendbar. Es handelt sich dabei um einen inhärent dynamischen Begriff, der stets mit dem Ablauf einer Folge von Einzelereignissen verknüpft ist. Eine Teilklasse der Prozesse sind die Prozesse, die innerhalb von Rechensystemen ablaufen. Je nach Abstraktionsebene kann man dort verschiedene Prozesse unterscheiden. Auf der Abstraktionsebene, auf welcher der Programmierer tätig ist, kann man allgemein von Datenverarbeitungs- oder Datentransformationsprozessen sprechen.

Innerhalb des Rechensystems werden die Datenverarbeitungsprozesse als gewisse Verwaltungseinheiten organisiert. Diese Organisation erfolgt in der Regel durch das Betriebssystem des Rechensystems. Innerhalb des Gebietes der Betriebssysteme hat sich daher ein auf diese Aufgabe zugeschnittener Prozeßbegriff entwickelt [Arb 73 : 2; BG 71 b : 91; CCJ 74 : 340; Dij 68 a : 51; DV 66 : 145; GMD 71 : 5; Hab 76 : 46; HH 73 : 20; Kat 73 : 125; Sal 66 : 22; Say 71 : 315].

Diese Begriffsbildung führte innerhalb der Betriebssysteme zu einer besseren Strukturierung und dient der Bildung klarer und einfacher Schnittstellen zwischen

* Diese Arbeit enthält Ergebnisse aus einem mit Mitteln des Bundesministers für Forschung und Technologie (Kennzeichen DV 5.505) geförderten Forschungsvorhabens des Projektes „Prozeßlenkung mit DV-Anlagen (PDV)" im Rahmen des 3. DV-Programmes der Bundesregierung. Die Verantwortung für den Inhalt liegt jedoch allein beim Autor.

Betriebssystemen und Anwenderprogramm und zwischen Teilen des Betriebssystems selbst. Eine darauf fußende (informale) Definition des Prozeßbegriffes ist dann (vergleiche auch [Wet 78 : 21]):

„Gliederungseinheit des Ablaufgeschehens, welches durch das Betriebssystem organisiert und überwacht wird." (1)

Habermann gibt eine mehr syntaktische Definition: "to consider a process as the execution of a closed procedure operating in a well-defined execution environment" [Hab 76 : 46]. Hierbei kommt jedoch der Organisationsaspekt von (1) nicht zum Ausdruck, denn nicht jede Prozedurausführung wird in einem Rechensystem als Prozeß organisiert. Kombiniert man die Definition (1) mit der von Habermann, dann erhält man die folgende informale Definition des Prozesses (2) (vergleiche auch [See 74 : 360; Win 79 : 17]).

Ein *Prozeß* ist die einmalige Ausführung einer ausgezeichneten Prozedur, wobei diese Prozedurausführung als autonome Einheit im Sinne des Ablaufgeschehens organisiert ist, und die prozedurlokalen Objekte nur zu dieser Prozedurausführung gehören. (2)

Zu einer besonderen einheitlichen Betrachtung kommt man, wenn man alle autonomen Einheiten des Ablaufgeschehens (z. B.: Prozessor, E/A-Gerät, Operateur) als Prozesse auffaßt [SW 78 : 90 f.].

Im folgenden wird von der in Abbildung 1 dargestellten Situation ausgegangen.

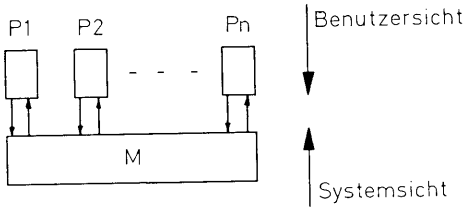

Abb. 1 Prozesse und unterstützende Maschine M

Eine Maschine M (bei *PEARL*-Implementierungen z. B. Rechner + Betriebssystem, Monitor in [Ker 78]) unterstützt den nebenläufigen Ablauf einer Menge von Prozessen Pl, ..., Pn, d. h. die Maschine M hat die Fähigkeit zur Organisation und Koordination des Ablaufs eines Systems nebenläufiger Prozesse. Diese Situation kann unter zwei Aspekten betrachtet werden. Von „oben" aus der Sicht des Benutzers (z. B. *PEARL*-Programmierers) und von „unten" aus der Sicht der Maschine bzw. dessen, der die Maschine realisiert und implementiert. Aus Benutzersicht erscheinen die Prozesse so, wie es die Definitionen (1) und (2) aussagen.

Im folgenden wird das Phänomen „Prozeß" stets aus Benutzersicht betrachtet, d. h. es werden die Eigenschaften und Operationen betrachtet, welche der Benutzer zur Verfügung hat und verwendet. Eigenschaften und Operationen, die nur innerhalb von M eine Rolle spielen, werden im folgenden nicht betrachtet, denn die aus Benutzersicht zu sehenden Eigenschaften und Operationen sind das Primäre; die Maschine M muß dann so konstruiert werden, daß sie die betreffenden Eigenschaften und Operationen realisiert.

2 Attribute von und Operationen an Prozessen

2.1 Allgemeines

Die in Abschnitt 1 herausgearbeitete Prozeßdefinition ist eine Beschreibung des Phänomens „Prozeß" (im Bereich der Betriebssysteme) und keine formale Definition in dem Sinne, daß die Struktur und die Komponenten der Obejekte der Art „Prozeß" präzise festgelegt sind. Eine formale Definition in diesem Sinne ist aber notwendig, wenn man Operationen an Objekten der Art „Prozeß" beschreiben will. Im folgenden wird davon ausgegangen, daß ein Prozeß zum Zwecke der Organisation und Verwaltung des Ablaufgeschehens durch ein strukturiertes Objekt repräsentiert wird. Die Komponenten eines solchen Objektes werden oft als die Attribute des (repräsentierten) Prozesses bezeichnet. Damit erhält man etwa

```
TYPE PROZESS STRUCT [ATTR1 ATYP1,
                     .
                     .                    (3)
                     .
                     ATTRn ATYPn].
```

Sofern keine Mißverständnisse zu erwarten sind, wird im folgenden der Begriff „Prozeß" sowohl für das Objekt nach Definition (2) als auch für das strukturierte Objekt, welches der Repräsentation dient, verwendet.

2.2 Attribute von Prozessen

Objekte der in (3) definierten Art *PROZESS* werden anderswo z. B. als „Prozeßbeschreibung" [SW 78 : 144 f.], „Prozeßleitblock (PLB)" [Wet 78 : 23], "Process Control Block (PCB)" [DM 74 : 236]. "task variable" [Cow 75 : 155] oder „Prozeßbeschreibungsblock" [Huf 77 : 17] bezeichnet. Diese Objekte der Art *PROZESS* können im praktischen Fall eine ganze Anzahl verschiedener Komponenten enthalten; in [SW 78] z. B. bis zu 25. Viele dieser Komponenten sind aus Benutzersicht nicht bedeutsam und dienen nur der internen Organisation der Maschine M.

Aus Benutzersicht sind insbesondere folgende Attribute von Bedeutung:

Identifikation
Zustand
Priorität
Programm

Ein Objekt der Art *PROZESS* kann im allgemeinen Falle nicht mit einem Prozeß gemäß der Definition (2) identifiziert werden, denn eine Prozeßbeschreibung kann nacheinander zur Repräsentation verschiedener Prozesse verwendet werden. Dies ist dann möglich, wenn ein Prozeß nur während eines endlichen Zeitintervalles existent ist (endliche Lebensdauer). In der Regel ist dies für Benutzerprogramme, die im Rechenzentrumsbetrieb als Prozesse abgewickelt werden, der Fall.

Während der Lebensdauer eines Prozesses bleiben Identifikation und Programm konstant. Operationen beziehen sich im folgenden daher nur auf die beiden Attribute „Priorität" und „Zustand".

2.3 Operationen an Prozessen

Geht man von der Definition (2) aus, so folgt, daß Operationen an Prozessen solche Aktionen sind, die der Organisation und Überwachung des Ablaufgeschehens dienen. Dazu gehören:

a) Erzeugen eines Prozesses : ERZEUGEN
b) Starten eines Prozesses : STARTEN
c) Anhalten eines Prozesses : ANHALTEN
d) Fortsetzen eines Prozesses : FORTSETZEN
e) Beenden eines Prozesses : BEENDEN
f) Vernichten eines Prozesses : VERNICHTEN
g) Ändern der Priorität eines Prozesses.

Betrachtet man das Ablaufgeschehen weiter im Detail, dann sind noch folgende Operationen an Prozessen sinnvoll:

h) Auslagern eines Prozesses in den Sekundärspeicher
i) Einlagern eines Prozesses in den Primärspeicher
j) Zuordnen eines Prozesses zu einem Prozessor
k) Auflösung einer Zuordnung (Prozeß, Prozessor).

Die Operationen h) bis k) stellen dabei interne Aktionen der Maschine M dar. Die Operation g) beeinflußt die Ausführung der Operationen h) bis k) indirekt. Die Operationen a) bis g) werden von den Prozessen, d. h. in den den Prozessen zugeordneten Programmen benutzt, um ein System nebenläufiger Prozesse zu schaffen und seinen Ablauf zu koordinieren. Aus Benutzersicht sind daher im folgenden die Operationen a) bis g) wichtig.

In den folgenden Abschnitten wird untersucht, welche Attribute und Operationen für Prozesse in den betrachteten Betriebssystemen und Programmiersprachen jeweils existieren, wobei die Betrachtung stets aus Benutzersicht erfolgt.

3 Prozesse in Betriebssystemen und Kommandosprachen

3.1 Prozesse in Betriebssystemen

Prozesse wurden zuerst intern innerhalb von Betriebssystemen verwendet und dienten dort der besseren Gliederung und Strukturierung des Komplexes Betriebssystem/Anwenderprogramm. Dadurch wurden Vorgänge wie z. B. das Umschalten vom Betriebssystem zum Anwenderprogramm klarer und deutlicher abgrenzbar. Solche Vorgänge waren vorher mehr implizit vorhanden [Gue 63 : 321 f.; Hab 76 : 45, 46]. Auch Teile von Betriebssystemen wurden in Form von Prozessen organisiert.

Wenn die Prozesse nur intern im Betriebssystem vorhanden und nach außen nicht sichtbar sind, dann ergibt eine Betrachtung aus Benutzersicht nichts. Die Prozesse sind in diesem Falle nur aus Systemsicht sichtbar.

Wie in Abschnitt 2 erwähnt, werden die Prozesse in der Maschine M, die neben der Apparatur zumindest den Betriebssystemkern enthält, in der Regel durch Verbunde repräsentiert. Die Zahl der Prozeßattribute schwankt von Betriebssystem zu Betriebssystem. Beispiele sind:

a) 55: MCP der Burroughs 6700/7700 [Bur 73 : 1–10 f.];
b) 17: Kern eines Concurrent Pascal Systems für die PDP 11/45 [Huf 77 : 17 f.]
c) 23: Betriebssystem für PEARL und für die Siemens 310 [SW 78 : 145]
d) 4: Kern einer Modula-Implementierung auf der PDP 11/45 [Wir 77 c : 72].

Operationen, die an den Prozessen vorgenommen werden können, sind meist durch sogenannte Zustandsdiagramme definiert. Als Beispiel dafür sei das von Brinch-Hansen veröffentlichte Diagramm [Han 72 : 102] angegeben (Abbildung 2).

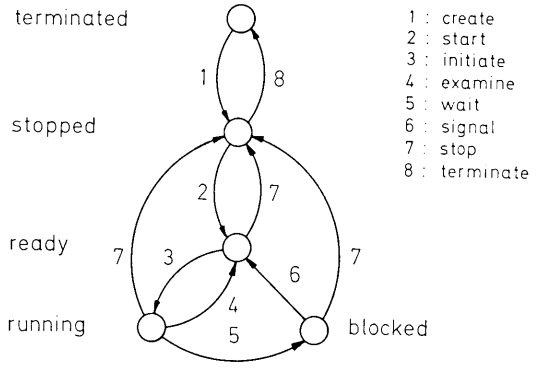

Abb. 2 Zustandsübergangsdiagramm nach Brinch-Hansen

In dem Diagramm in Abbildung 2 ist dargestellt, welche Zustände ein Prozeß während seiner Abwicklung im Rechensystem annehmen kann, und welche Operationen Zustandsänderungen bewirken. Die Operationen „initiate" und „examine" werden nur innerhalb des Betriebssystemkernes angewendet, während die anderen sechs Operationen auch von Prozessen auf andere Prozesse ausgeübt werden können.

Das Diagramm ist unvollständig in dem Sinne, als nicht angegeben ist, wie die sechs frei verfügbaren Operationen in jedem der fünf Zustände wirken; z. B. ist nicht ersichtlich, wie die Operation „terminate" im Zustand "blocked" wirkt. Die Operation "wait" führt nur in dem Falle in den Zustand "blocked", wenn ein in der Operation angesprochener Semaphor bereits auf 0 heruntergezählt worden ist. Im anderen Falle ändert sich der Zustand des Prozesses nicht, wohl aber der des Semaphors.

Da ein Prozeß mehrmals von "terminated" nach "terminated" gebracht werden kann, unterscheidet sich das dem Diagramm in Abbildung 2 zugrunde liegende Prozeßkonzept von dem in Definition (2), da dort die einmalige Ausführung einer Prozedur als separater Prozeß aufgefaßt wird.

3.2 Prozesse in Kommandosprachen

In Kommandosprachen wurden erste Ansätze gemacht, Prozesse als für den Benutzer (Programmierer) sichtbare Objekte zu formulieren. Die rein sequentielle Abfolge von einzelnen (sequentiellen) Prozessen, wie die Folge der Job-steps in JCL des OS 360/370 [Dav 77 : 358 f.], benutzt das Konzept der nebenläufigen Prozesse noch nicht in seiner ganzen Breite. Dies ist jedoch der Fall, wenn aus einem Prozeß heraus nebenläufige Prozesse gestartet werden können und der startende Prozeß ebenfalls weiterläuft.

Ein erster Ansatz dazu ist in der Kommandosprache der Univac Serie 1100 [Spe 71] in Form des „START"-Kommandos gemacht. Durch dieses Kommando wird ein unabhängiger Run gestartet. Ein Run entspricht dabei dem, was bei IBM 360/370 oder Burroughs 6700/7700 als Job bezeichnet wird. Kommunikation zwischen solch nebenläufigen Runs ist über Dateien möglich, die zum Zwecke der Synchronisation exklusiv eröffnet werden können. Dieses Prozeßkonzept läßt sich daher durch folgende Merkmale charakterisieren:

a) keine Hierarchie innerhalb der Menge der Prozesse.
b) keine direkte Steuerung von Prozessen.
c) echte Nebenläufigkeit.
d) Synchronisation mittels Dateien.

Ähnliche Möglichkeiten gibt es auch in der Work-Flow-Language (WFL) der Burroughs 6700/7700 [Bur 73 : 5–1. . 5–30]. Innerhalb eines Jobs, der zugleich auch einen Prozeß darstellt, können weitere Prozesse gestartet werden. Nebenläufige Prozesse werden durch die PROCESS-Anweisung erzeugt und zugleich gestartet. Ein solcher Prozeß ist die einmalige Ausführung eines Programmes, welches in einer anzugebenden Datei oder einer innerhalb des Jobs vereinbarten Subroutine (Prozedur) enthalten ist. Zur Steuerung des Prozeßablaufes kann dem Prozeß ein benanntes Ablaufkontrollobjekt (Prozeßbeschreibung) in der PROCESS-Anweisung zugeordnet werden. Mittels dieses Ablaufkontrollobjektes, welches in WFL als "task" bezeichnet wird, ist eine Koordinierung der nebenläufigen Prozesse möglich durch:

a) Warten auf verschiedene Bedingungen wie Zeitablauf, Beendigung eines Prozesses, bestimmten Zustand eines Prozesses.

b) Abfragen und Setzen von Taskattributen (das sind die Komponenten des Ablaufkontrollobjektes).

Weitere Hilfsmittel zur Koordination der nebenläufigen Prozesse sind nicht vorhanden.

Zusammenfassend läßt sich dieses Prozeßkonzept wie folgt charakterisieren:

a) zweistufige Hierarchie (im statischen Sinne): Job auf Stufe 1, Unterprozesse eines Jobs auf Stufe 2.

b) Prozesse werden durch task-identifier benannt und können direkt gesteuert werden.

c) echte Nebenläufigkeit

d) Synchronisation mittels des Dekker'schen Algorithmus.

4 Prozesse in Programmiersprachen

4.1 Allgemeines

Der Tendenz folgend, dem Benutzer (Programmierer) die Möglichkeit zu geben, wachsenden Einfluß auf das Ablaufgeschehen zu nehmen und durch das Rechensystem gegebene Möglichkeiten von Nebenläufigkeiten auszunutzen, fand das Konzept des Prozesses auch Eingang in die Programmiersprachen (PL/1 [IBM 72]; Burroughs Extendend Algol [Bur 74]; Concurrent Pascal [Bri 75 c]; Ada [Ada 79]). Als für das Gebiet der Echtzeitprogrammierung ebenfalls höhere Programmiersprachen entwickelt wurden, wurde auch in diese Sprachen das Prozeßkonzept aufgenommen ([BS 77 a]; Prozeß-Fortran [CCH 78; VDI 78]; PEARL [PDV 77, 77 a]; Modula [Wir 77]; Real-Time Basic [Ind 79]). Für das Anwendungsgebiet dieser Sprachen sind Nebenläu-

figkeiten geradezu charakteristisch, und daher war hier die Aufnahme des Prozeßkonzeptes notwendig.

Stellt man dem Benutzer Objekte der Art „Prozeß" zur Verfügung, dann müssen diese Objekte in ihren statischen und dynamischen Eigenschaften vollständig und eindeutig definiert werden. Wendet man diesen Grundsatz auf die Operationen an, welche auf Prozesse ausgeübt werden sollen, dann bedeutet dies, daß ein Zustandsdiagramm, wie es in Abschnitt 3.1 dargestellt wurde, vollständig in dem Sinne sein muß, daß für jede Kombination (Prozeßzustand, Prozeßoperation) ein Folgezustand des betroffenen Prozesses definiert sein muß. Dafür sind mehrere Gründe maßgebend:

a) der Benutzer kann Programme erstellen, welche beliebige Reihenfolgen von Operationen emittieren.

b) in einem System relativ schwach gekoppelter Prozesse, deren Ablaufgeschwindigkeiten nicht präzise vorausgesagt werden können, muß damit gerechnet werden, daß eine Prozeßoperation auf einen beliebigen Zustand des angesprochenen Prozesses trifft. Wegen der schwachen Kopplung kann dies auch nicht in jedem Falle als Fehler betrachtet werden, auch wenn der aktuelle Zustand des betroffenen Prozesses nicht „geeignet" ist zur Durchführung der gewünschten Operation. (Z. B. Anhalten eines noch nicht gestarteten Prozesses oder vernichten eines noch nicht beendeten Prozesses).

c) es muß mit fehlerhaften Prozessen gerechnet werden, deren Verhalten nicht vorhersehbar ist.

Die im Zusammenhang mit Betriebssystemen veröffentlichten Zustandsdiagramme sind im obigen Sinne meist unvollständig [Arb 73 : 7, 11; De 71 : 203; Do 72 : 389, 391; DM 74 : 210; Han 72 : 102; Sal 66 : 63]. Dies hat seinen Grund darin, daß in Betriebssystemen diese Operationen nur im Bereich des Kernes (Monitors) verwendet werden, und daß bei der Durchführung der Operationen eine starke Kopplung zwischen Kern und Prozeß derart besteht, daß dem Kern der Zustand des betroffenen Prozesses genau bekannt ist und sich durch Einwirkung von dritter Seite während der Aktivität des Kernes nicht ändert [Neh 75 : 248]. Die Initiierung einer Prozeßoperation in einem nicht geeigneten Zustand des betroffenen Prozesses ist dann als Fehlverhalten des betreffenden Betriebssystemes zu betrachten.

Im Zusammenhang mit der Initiierung einer Prozeßoperation in einem nicht „geeigneten" Zustand tritt auch das Problem der Rückmeldung an den auftraggebenden (initiierenden) Prozeß auf, denn eine Vervollständigung des Übergangsdiagramms in der Art, daß für fehlende Kombinationen (Zustand, Operation) eine Schleife eingefügt wird, kann nicht in jedem Fall als zufriedenstellend angesehen werden. Dadurch würde eine solche Operation als Leeroperation behandelt. Es sind jedoch auch andere Formen der Reaktion auf eine nicht unmittelbar durchführbare Operation denkbar [Win 79 : 128, 129]. So kann der Auftraggeber einer solchen Operation auch beendet oder beendet und vernichtet werden. Falls durch Operationen anderer Prozesse in der Zukunft noch ein Systemzustand eintreten kann, in welchem die gewünschte Operation durchführbar wird, kann der betreffende Auftraggeber bis zum Eintreten eines solchen Systemzustandes angehalten werden. Zur Beschreibung solcher Wirkungen von Prozeßoperationen genügen die bisher verwendeten Zustandsübergangsdiagramme, in welchen nur der Zustand eines Prozesses dargestellt wird, nicht mehr; es müssen Methoden verwendet werden, mit denen die simultane Zustandsänderung mehrerer Prozesse beschrieben werden kann [Win 79].

Im folgenden werden die Prozeßkonzepte einer Reihe von Programmiersprachen unter fünf Gesichtspunkten untersucht:

1. Prozeßkonzept: dabei wird auf den in Abschnitt 1 entwickelten Prozeßbegriff Bezug genommen.

2. Prozeßhierarchie: gibt es eine Hierarchie der Prozesse, und wenn ja, wie wird sie gebildet?

3. Direkte Prozeßsteuerung: dies bezieht sich auf die in Abschnitt 2.3 unter a) bis g) angegebenen Prozeßoperationen.

4. Indirekte Prozeßsteuerung: dies bezieht sich auf Synchronisation und Kommunikation; dabei werden Prozesse indirekt beeinflußt (z. B. Fortsetzen beim Hochzählen eines Semaphors).

5. Priorität und Parameter.

Die betrachteten Sprachen werden in zwei Gruppen eingeteilt: algorithmische- und Echtzeitsprachen. Während die Zahl der algorithmischen Sprachen, in welchen das Prozeßkonzept enthalten ist, noch klein ist, sind bereits wesentlich mehr Echtzeitsprachen mit Prozeßkonzept entwickelt worden, als in einer solchen Arbeit behandelt werden können. Außerdem ist die Entwicklung auf dem Gebiet der Echtzeitprogrammierung, wo der Übergang zur höheren Programmiersprache noch im Gange ist, sehr stürmisch, so daß damit zu rechnen ist, daß einige der behandelten Sprachen, die teils als Normentwürfe vorliegen, noch Änderungen erfahren und neue Sprachen auf der Szene erscheinen werden.

Wenn im folgenden im Zusammenhang mit Sprachen mit Blockstruktur von vereinbarten Objekten die Rede ist, dann sind damit stets einzelne Inkarnationen der betreffenden Vereinbarung gemeint, denn im allgemeinen kann es zu einem Zeitpunkt mehrere Inkarnationen einer Vereinbarung geben [Win 78 : 278].

4.2 Prozesse in algorithmischen Sprachen

4.2.1 PL/1 [IBM 72]

4.2.1.1 Prozeßkonzept. In PL/1 wird ein Prozeß erzeugt und gestartet durch die Abarbeitung einer Anweisung der Form (4.2-1).

```
CALL proz-bez akt-par TASK (task-bez) EVENT (ev-bez)
                PRIORITY (ausdruck);                    (4.2-1)
```

Dabei sind nicht alle Teile dieser Anweisung obligatorisch. Wenn "TASK (task-bez)" nicht angegeben ist, dann wird eine anonyme Taskvariable benutzt. Der so geschaffene Prozeß kann nicht referiert werden. Ein Prozeß im Sinne der Definition in Abschnitt 1 wird gebildet durch die Kombination einer Prozedur und einer Taskvariablen. Die Taskvariable kann als Abstraktion der Prozeßbeschreibung (s. Abschnitt 2.2) aufgefaßt werden. Eine Taskvariable enthält nur ein Attribut, nämlich die Priorität des Prozesses. Eine Prozedur kann mit beliebigen Taskvariablen und eine Taskvariable mit beliebigen Prozeduren kombiniert werden. Zu einem Zeitpunkt kann eine Taskvariable höchstens für einen Prozeß benutzt werden. Nacheinander kann eine Taskvariable für mehrere Prozesse benutzt werden, die sich zeitlich nicht überlappen. Eine individuelle Bezeichnung der einzelnen Prozesse erfolgt nicht, vielmehr wird durch eine Taskvariable eine Klasse von Prozessen bezeichnet, nämlich die, für welche die betreffende Taskvariable als Ablaufkontrollobjekt verwendet wird.

4.2.1.2 Prozeßhierarchie. Prozeduren und Taskvariable können als lokale Objekte in einem Block oder einer Prozedur vereinbart werden. Da PL/1 eine Sprache mit Blockstruktur ist, können infolgedessen auch Prozesse geschachtelt sein, nämlich dann, wenn Prozedur und Taskvariable eines Prozesses B in der Prozedur eines Prozesses A lokal vereinbart sind. Dadurch ergäbe sich die Möglichkeit eine Unterprozeßrelation zu definieren. In PL/1 ist die Unterprozeßrelation jedoch nicht aufgrund einer solchen Schachtelung sondern über den Vorgang des Erzeugens und Startens definiert:

> P2 ist Unterprozeß der Blockinkarnation B von P1 genau dann, wenn die CALL-Anweisung, durch welche P2 erzeugt wurde, im Zuge der Abarbeitung von B (als Teil von P1) ausgeführt wurde. (4.2-2)

P2 ist an B gebunden und wird beendet, wenn P1 B verläßt. Durch diese Relation ist die Menge der Prozesse eines PL/1-Programmes als Baum strukturiert mit dem Hauptprogramm als Wurzel.

Die Definition (4.2-2) ist allerdings zu restriktiv, was das folgende Beispiel zeigt:

```
1. B1 : BEGIN;
2       P1 : PROC; . . . END P1;
3            DCL T1 TASK;
4            B2 : BEGIN;
                    .                                   (4.2-3)
                    .
5                   CALL P1 TASK (T1);
                    .
                    .
6            END B2;
7   END B1;
```

Der durch die Anweisung in Zeile 5 erzeugte und gestartete Prozeß wird bei Verlassen des Blockes B2 abgebrochen, obwohl dies aus Gültigkeitsbereichsgründen erst bei Verlassen von B1 notwendig wäre.

Eine implizite Synchronisation am Blockende findet nicht statt; wenn eine Blockinkarnation verlassen wird, dann werden alle daran gebundenen Prozesse abgebrochen.

4.2.1.3 Direkte Prozeßsteuerung. Die in (4.2-1) angegebene Anweisung ist eine Kombination von Erzeugen und Starten. Eine separate Ausführung dieser Prozeßoperationen ist nicht möglich.

Außer der in (4.2-1) angegebenen Initiierung sind folgende direkte Prozeßbeeinflussungen möglich:

a) ein Prozeß kann sich selbst beenden (END, EXIT, RETURN).

b) ein Prozeß kann alle zu dem betreffenden Programm gehörenden Prozesse beenden (STOP).

c) ein Prozeß kann sich selbst für eine vorgegebene Zeitspanne anhalten (DELAY).

d) die Priorität eines Prozesses kann abgefragt und gesetzt werden.

Weitere direkte Beeinflussungsmöglichkeiten gibt es nicht.

4.2.1.4 Indirekte Prozeßsteuerung. Indirekte Beeinflussungen von Prozessen sind zum Zwecke der Synchronisation möglich. Dazu dienen sogenannte EVENT-Variable und die WAIT-Anweisung. Eine EVENT-Variable kann zwei verschiedene Werte (0,1) annehmen. Durch die WAIT-Anweisung wartet der diese Anweisung ausführende Prozeß so lange, bis die darin angegebenen EVENT-Variablen den Wert 1 haben. Das Setzen auf den Wert 1 kann durch verschiedene Ereignisse wie Ende eines Prozesses oder Ende einer E/A-Anweisung oder auch explizit programmiert erfolgen.

Diese Hilfsmittel stellen allerdings kein vollständiges Synchronisationsmittel dar, da die WAIT-Anweisung den Wert der angesprochenen EVENT-Variablen unverändert läßt.

4.2.1.5 Priorität und Parameter. Die Priorität der Prozesse eines PL/1-Programmes kann Werte von 0 bis MAX annehmen, wobei MAX in der Kommandosprache angegeben wird. Im Quellprogramm werden stets relative Prioritäten (relativ zum ausführenden Prozeß) angegeben. Diese werden jedoch stets intern auf absolute Werte aus dem oben angegebenen Intervall umgerechnet. Die Priorität eines benannten Prozesses kann jederzeit abgefragt und gesetzt werden und ist für interne Verwaltungszwecke des Betriebssystems von Bedeutung.

Wie aus (4.2-1) hervorgeht, kann ein Prozeß Parameter haben. Neben Datenobjekten können auch Prozeduren und Taskvariable als Prozedur – und damit auch als Prozeßparameter auftreten.

4.2.2 Burroughs Extended Algol [Bur 74]

4.2.2.1 Prozeßkonzept. In Burroughs Extended Algol wird in einer PROCESS-Anweisung (4.2-4) eine Prozedur mit einer Task kombiniert und dadurch ein Prozeß erzeugt und gestartet.

PROCESS proz-bez akt-par [task-bez]; (4.2-4)

Eine Task kann als Abstraktion der Prozeßbeschreibung (s. Abschnitt 2.2) aufgefaßt werden. Eine Task ist ein Verbund mit 30 Komponenten, die dem Programmierer zugänglich sind. Im Betriebssystem (MCP) sind 42 Taskattribute zugänglich. Für die Kombinationsmöglichkeiten von Prozedur und Task gilt dasselbe wie in PL/1 (s. Abschnitt 4.2.1.1).

4.2.2.2 Prozeßhierarchie. Prozeduren und Tasks können in Blöcken und Prozeduren vereinbart werden. Prozesse sind auch hier an Blockinkarnationen gebunden, und zwar ist ein Prozeß P an die Blockinkarnation gebunden, die von allen Blockinkarnationen, von welchen er abhängig ist, als letzte geschaffen wurde („kritischer Block"). Da diese Blockinkarnation von genau einem anderen Prozeß Q geschaffen wurde, läßt sich auch hier eine Unterprozeßrelation definieren, welche zu einer Baumstruktur führt mit dem Programm als Wurzel. Diese Unterprozeßrelation ist hier konsequent entsprechend den Gültigkeitsbereichsanforderungen definiert. Eine implizite Synchronisation am Blockende findet nicht statt. Will ein Prozeß P den kritischen Block eines seiner Unterprozesse verlassen, dann werden sowohl P als auch alle Prozesse, für die dieser Block kritischer Block ist, abgebrochen.

4.2.2.3 Direkte Prozeßsteuerung. Die in (4.2-4) angegebene Anweisung ist eine Kombination von Erzeugen und Starten. Eine separate Ausführung dieser Prozeßoperationen ist nicht möglich.

Weitere direkte Prozeßbeeinflussungen sind möglich durch Manipulation der zugehörigen Task-Komponente STATUS. Dadurch sind folgende Operationen realisierbar: Anhalten, Fortsetzen und Beenden. Ein Prozeß kann diese Operationen stets auf sich selbst ausüben (MYSELF.STATUS), auch wenn er nicht im Zugänglichkeitsbereich des ihm zugeordneten task-bez arbeitet. Eine direkte Fremdbeeinflussung ist nur im Zugänglichkeitsbereich des betreffenden task-bez möglich (task-bez. STATUS).

Mittels der WAIT-Anweisung kann sich ein Prozeß selbst für eine bestimmte Zeitspanne oder bis zu einem Zeitpunkt, zu dem eine sogenannte EVENT-Variable gesetzt ist, anhalten.

4.2.2.4 Indirekte Prozeßsteuerung. Indirekte Beeinflussung zum Zwecke der Koordination bei der Benutzung gemeinsamer Datenobjekte ist möglich mittels Objekten der Art EVENT, die im wesentlichen eine Kombination der EVENT-Variablen von PL/1 und des binären Semaphors darstellen. Weiterhin ist eine indirekte Beeinflussung mittels Objekten der Art INTERRUPT möglich, die an Objekte der Art EVENT angeschlossen werden können. Einem INTERRUPT wird eine unmarkierte Anweisung zugeordnet. Wenn nun die Anweisung „CAUSE (event-bez)" durchgeführt wird, dann werden die unmarkierten Anweisungen all der INTERRUPT-Objekte durchgeführt (als parameterlose Prozedur), welche an das mit event-bez bezeichnete EVENT-Objekt angeschlossen (ATTACHED) und ermöglicht (ENABLED) sind. Dieser Mechanismus arbeitet ähnlich, wie das Ansprechen von sogenannten ON-Anweisungen beim Auftreten von Unterbrechungen in PL/1.

4.2.2.5 Priorität und Parameter. Objekte der Art TASK und PROCEDURE können als Prozedur- und damit auch als Prozeßparameter auftreten. Eine Prozedur und damit ein Prozeß kann Wert- und Namensparameter haben. Die Priorität eines Prozesses kann einen Wert von 0 bis 99 annehmen. Maßgebend ist der Wert zum Zeitpunkt des Startens.

4.2.3 Algol 68 [FKL 76]

4.2.3.1 Prozeßkonzept. In Algol 68 ist das Konzept des Prozesses nur implizit vorhanden, und zwar innerhalb der kollateralen und der parallelen Klausel. Die kollaterale Klausel ergibt völlig unabhängige Prozesse, die zwar auf gemeinsame Daten zugreifen können, diese Zugriffe jedoch nicht koordinieren können. In der parallelen Klausel stehen die allgemeinen Semaphore von Dijkstra [Dij 68 a] als Koordinierungsmittel zur Verfügung. Die beiden Klauseln haben die in (4.2-5) und (4.2-6) angegebene Form wobei $n \geq 0$ gilt.

BEGIN unit-1, unit-2, ..., unit-n END (4.2-5)
PAR BEGIN unit-1, unit-2, ..., unit-n END (4.2-6)

In [FKL 76] ist die Semantik dieser Klauseln ohne Bezugnahme auf echte Nebenläufigkeiten definiert. Einer nebenläufigen Abarbeitung der verschiedenen „unit's" einer solchen Klausel steht jedoch nichts im Wege, wenn nur die Atomarität der inkompatiblen Aktionen [FKL 76 : 2.1.4.2.e] gewährleistet wird.

4.2.3.2 Prozeßhierarchie. Da eine „unit" selbst wieder eine kollaterale oder parallele Klausel enthalten kann, ergibt sich auch hier ein hierarchischer Aufbau. Dennoch gibt es keine Unterprozeßrelation in dem bisher verwendeten Sinne, da die Semantik der Klauseln (4.2-5) und (4.2-6) besagt, daß es sich um eine Aufspaltung in n Unteraktionen und eine anschließende Zusammenführung handelt. Da diese Aufspaltung über mehrere Stufen erfolgen kann, erhält man im allgemeinen Falle eine Baumstruktur, in welcher die Blätter ausführbare Aktionen darstellen. Eine Aktion A ist niemals kollateral zu einer Aktion B, aus welcher sie durch Aufspaltung hervorging.

Das Problem der Synchronisation am Blockende existiert hier nicht.

4.2.3.3 Direkte Prozeßsteuereung. Eine direkte Prozeßsteuerung ist nicht möglich.

4.2.3.4 Indirekte Prozeßsteuerung. Eine indirekte Beeinflussung der Prozesse erfolgt durch die Synchronisationsoperationen „up" (≙ der V-Operation von Dijkstra) und „down" (≙ der P-Operation) möglich.

4.2.3.5 Priorität und Parameter. Das Konzept der Priorität kommt in Algol 68 nicht vor. Prozesse können nicht als Parameter auftreten, sie können jedoch beliebige Parameter haben, denn eine „unit" in (4.2-5) oder (4.2-6) kann z. B. ein Prozeduraufruf sein.

4.2.4 Concurrent Pascal [Bri 75c]

4.2.4.1 Prozeßkonzept. In Concurrent Pascal kann ein abstrakter Datentyp, der ähnlich wie eine Simula-Klasse [DMN 70 : 15 f.] strukturiert ist, mit dem zusätzlichen Attribut „PROCESS" versehen werden (4.2-7).

```
TYPE myprocesstype =                                    (4.2-7)
PROCESS param; vereinb-folge BEGIN anw-folge END
```

Variable eines solchen Typs sind dann Prozesse im hier verwendeten Sinn. Ein solcher Prozeß wird erzeugt durch die Abarbeitung der betreffenden Variablendeklaration (4.2-8).

```
VAR myprocess : myprocesstype;                          (4.2-8)
```

4.2.4.2 Prozeßhierarchie. Die Definition von Prozeßtypen kann geschachtelt werden. Damit ist auch eine Schachtelung von Prozessen möglich. Alle in Prozeßtypen und solchen Typen, in welchen Prozeßtypen vereinbart werden können, vereinbarten Objekte sind sogenannte permanente Objekte, deren Lebensdauer von ihrer Erschaffung bis zum Ende der Abarbeitung des gesamten Programms dauert, unabhängig davon, ob das Objekt, zu welchem sie lokal gehören, terminiert wird [Bri 75c:26, 37].

Diese Vorschrift hat zur Folge, daß das Problem der impliziten Synchronisation am Gültigkeitsbereichsende nicht existiert.

4.2.4.3 Direkte Prozeßsteuerung. Erzeugt wird ein Prozeß durch die Abarbeitung einer Variablendeklaration. (4.2-8). Gestartet wird ein solcher Prozeß durch eine INIT-Anweisung (4.2-9).

```
INIT myprocess;                                         (4.2-9)
```

Diese Anweisung bewirkt, daß die „anw-folge" des zugehörigen Prozeßtyps als sequentieller Prozeß nebenläufig zu allen existierenden Prozessen ausgeführt wird. Auf eine Prozeßvariable darf höchstens einmal eine INIT-Anweisung ausgeübt werden.

Weitere direkte Beeinflussungsmöglichkeiten existieren nicht.

4.2.4.4 Indirekte Prozeßsteuerung. Eine Synchronisation von Prozessen beim Zugriff zu gemeinsamen Objekten ist mittels Monitoren [Hoa 74; Kam 74b] möglich.

4.2.4.5 Priorität und Parameter. Das Konzept der Priorität von Prozessen ist in Concurrent Pascal nicht enthalten.

Prozesse können Monitore und Konstanten als Parameter haben. Prozesse können nicht als Parameter auftreten.

4.2.5 Platon [SS 75a]

4.2.5.1 Prozeßkonzept. In einer PROCESS-Vereinbarung wird ein Prozeßtyp vereinbart. Die Syntax dieser Vereinbarung stimmt im wesentlichen mit der Syntax der Pascal-Prozedurvereinbarung überein. Eine Inkarnation eines solchen Prozeßtyps wird durch eine CREATE-Anweisung erzeugt (4.2-10).

```
CREATE shadow-var LIKE proz-id akt-par
                WITH store-size;                        (4.2-10)
```

Die shadow-var dient zur Identifizierung der so geschaffenen Prozeßinkarnation. Zu einem Prozeßtyp können mehrere Inkarnationen erzeugt werden, die bei gleichzeitiger Existenz durch verschiedene shadow-var bezeichnet werden müssen. Eine shadow-var kann nacheinander zur Bezeichnung verschiedener Inkarnationen verwendet werden.

Eine mittels (4.2-10) geschaffene Inkarnation repräsentiert eine Menge von Prozessen, die hintereinander ausgeführt werden und alle von dem durch proz-id bezeichneten Prozeßtyp sind. Alle diese Prozesse werden durch die shadow-var bezeichnet. Die Eindeutigkeit dieser Bezeichnung ist dadurch gewährleistet, daß zu einem Zeitpunkt höchstens ein solcher Prozeß existiert.

4.2.5.2 Prozeßhierarchie. Prozeßvereinbarungen können textuell geschachtelt sein. Ein Prozeß kann nur Inkarnationen von solchen Prozeßtypen schaffen, deren Vereinbarung unmittelbar in seinem Prozeßtyp enthalten ist. Diese Bedingung wird durch die Gültigkeitsbereichsregeln gewährleistet. Durch diese Bedingung ist die Relation „A ist unmittelbarer Unterprozeß von B" definiert. Die transitive Hülle davon ist dann die Unterprozeßrelation.

Eine Synchronisation am Blockende wird nicht vorgenommen; wenn ein Prozeß A beendet wird, dann werden alle seine Unterprozesse ebenfalls beendet und vernichtet.

4.2.5.3 Direkte Prozeßsteuerung. Zur direkten Prozeßsteuerung stehen folgende Anweisungen zur Verfügung:

CREATE: dadurch wird eine Inkarnation eines Prozeßtyps erzeugt (s. Abschnitt 4.2.5.1).

START: diese Operation bezieht sich stets auf eine unmittelbare Unterprozeßinkarnation. Existiert zu dieser Inkarnation kein Prozeß, so wird einer erzeugt und gestartet. Existiert zu dieser Inkarnation ein Prozeß, der angehalten ist, so werden er und diejenigen seiner Unterprozesse, die ebenfalls angehalten sind, fortgesetzt.

STOP: anhalten eines Prozesses und seiner Unterprozesse.

HALT: mittels dieser Anweisung beendet sich ein Prozeß selbst; alle seine Unterprozesse werden beendet und vernichtet.

REMOVE: beenden und vernichten einer unmittelbaren Unterprozeßinkarnation und aller ihrer Unterprozesse.

4.2.5.4 Indirekte Prozeßsteuerung. Eine indirekte Steuerung von Prozessen ist zum Zwecke von Synchronisation und Kommunikation möglich. Der allgemeine Fall ist die Kommunikation, die durch die Übergabe von Nachrichten realisiert wird, wobei Semaphore als Übergabestelle verwendet werden. Wird die leere Nachricht übergeben, so handelt es sich um den Fall der reinen Synchronisation.

Nachrichten werden in Nachrichtenbehältern untergebracht, von welchen statisch feste Vorräte definiert werden können.

Bei der Kommunikation werden meistens nur Verweise auf Nachrichtenbehälter manipuliert, lediglich beim Einfüllen und der Entnahme der Nachrichten wird auf die Behälter selbst zugegriffen. Dieser direkte Zugriff ist nur innerhalb der LOCK-Anweisung möglich.

Ein Vorrat von Nachrichtenbehältern ist eine Variable vom Typ:

SHAREDSET (Anzahl, Behältertyp).

Ein Semaphor kann entweder eine Warteschlange von Verweisen auf Nachrichtenbehälter oder eine Warteschlange von Prozessen enthalten, je nachdem ob mehr Verweise auf Nachrichtenbehälter bei ihm abgeliefert als angefordert wurden oder umgekehrt.

Mit der ALLOCATE-Anweisung wird ein Verweis auf einen Nachrichtenbehälter beschafft und einer Variablen vom Typ REFERENCE zugewiesen. Dieser Verweis kann mittels der SIGNAL-Anweisung an einen Semaphor geschickt werden, wodurch die betreffende REFERENCE-Variable wieder undefiniert wird. Mit der WAIT-Anweisung kann ein Verweis auf einen Nachrichtenbehälter von einem Semaphor abgeholt werden und einer REFERENCE-Variablen zugewiesen werden. Durch RETURN wird eine REFERENCE-Variable auf „nil" gesetzt und der Nachrichtenbehälter, auf welchen die Variable verwiesen hatte, an den Vorrat zurückgegeben.

Die Kommunikationsanweisungen sind so definiert, daß zu jedem Zeitpunkt höchstens eine REFERENCE-Variable einen Verweis auf einen bestimmten Nachrichtenbehälter enthalten kann [SS 75:5]. Dadurch ist ein koordinierter Nachrichtenaustausch gewährleistet.

4.2.5.5 Priorität und Parameter. Das Konzept der Priorität kommt in Platon nicht vor. Prozesse können nur Datenobjekte als Wertparameter und Semaphore als Referenzparameter haben. Prozesse können nicht als Parameter auftreten.

4.2.6 OSL/2 [Als 71]

4.2.6.1 Prozeßkonzept. Ein Prozeß ist die einmalige Ausführung einer Prozedur. Die Operation E+S (Erzeugen und Starten) kann auf drei verschiedene Arten durchgeführt werden:

a) SPAWN (procedure-statement).
Dadurch wird ein unbenannter, paralleler Prozeß erzeugt und gestartet. Das Programm dieses Prozesses ist bestimmt durch die in dem „procedure-statement" angegebene Prozedur.

b) APPEND (procedure-statement).

Dadurch wird ein unbenannter, paralleler Prozeß erzeugt und gestartet. Das Programm dieses Prozesses ist gegeben durch die in dem „procedure-statement" angegebene Prozedur. Außerdem ist der erzeugte Prozeß an den Block, in welchem die APPEND-Anweisung unmittelbar enthalten ist, derart gebunden, daß dieser erst verlassen werden kann, wenn der Prozeß beendet ist.

c) INITIATE (procedure-statement, proc-var).
Dadurch wird ein benannter, paralleler Prozeß erzeugt und gestartet. Die Benennung erfolgt mittels „proc-var". Das Programm dieses Prozesses ist bestimmt durch die in dem „procedure-statement" angegebene Prozedur.

In den Fällen son SPAWN und INITIATE ist keine Aussage über eine evt. erforderliche Synchronisation an der Blockstruktur gemacht.

4.2.6.2 Prozeßhierarchie. Prozeduren können textuell geschachtelt sein. Der Unterprozeß ist allerdings nicht durch diese Schachtelung definiert, sondern wie folgt:

a) durch SPAWN wird ein Nachbarprozeß zu dem diese Anweisung ausführenden Prozeß geschaffen.

b) durch APPEND und INITIATE werden unmittelbare Unterprozesse zu den diese Anweisungen ausführenden Prozessen erzeugt.

Dadurch ist eine Unterprozeßrelation definiert, die mehrere maximale Elemente haben kann.

4.2.6.3 Direkte Prozeßsteuerung. Ein Prozeß kann sich selbst beenden durch die Anweisung TERMINATE. Benannte Prozesse (die durch INITIATE geschaffen worden sind) können durch die folgenden drei Anweisungen gesteuert werden:

a) SUSPEND (proc-var) · Anhalten.
b) RESTART (proc-var) · Fortsetzen.
c) TERMINATE (proc-var) · Beenden.

4.2.6.4 Indirekte Prozeßsteuerung. Zum Zwecke der Synchronisation sind allgemeine Semaphore und die P- und V-Operation vorhanden.

4.2.6.5 Priorität und Parameter. Das Konzept der Priorität ist in OSL/2 nicht enthalten. Prozesse können als Prozedur- und damit auch als Prozeßparameter auftreten. Prozesse können beliebige Wert-, Referenz- und Namensparameter haben, lediglich bei SPAWN müssen alle Parameter durch Wertübergabe übergeben werden.

4.2.7. Ada [Ada 79]

4.2.7.1 Prozeßkonzept
In Ada können Objekte der Art TASK deklariert werden. Ein solches TASK-Objekt entspricht einer Menge von Prozessen im Sinne von Abschnitt 1, wobei sich die Lebensdauer der Prozesse einer solchen Menge nicht überlappen. Eine TASK-Deklaration besteht aus zwei Teilen; der Spezifikation (4.2-11), welche die nach außen sichtbaren Komponenten definiert, und dem Rumpf (4.2-12), welcher das Programm der zugehörigen Prozesse definiert.

```
TASK task-bez
IS vereinb-folge                          (4.2-11)
[PRIVATE vereinb-folge]
END task-bez;

TASK BODY task-bez
IS vereinb-folge
[BEGIN anw-folge                          (4.2-12)
[EXCEPTION ausn-reakt-folge]]
END task-bez;
```

4.2.7.2 Prozeßhierarchie. Vereinbarungen von Objekten der Art TASK können in Blöcken, Prozeduren, Moduln und Tasks enthalten sein, so daß TASK-Deklarationen auch geschachtelt sein können. Die Unterprozeßrelation ist allerdings nicht über diese Schachtelung sondern über die Abarbeitung der TASK-Deklaration definiert: Prozeß B ist unmittelbarer Unterprozeß von Prozeß A genau dann, wenn Prozeß A die zu B gehörige TASK-Deklaration abgearbeitet hat. Die transitive Hülle dieser Relation ist dann die allgemeine Unterprozeßrelation. Diese hat Baumstruktur mit dem Gesamtprogramm als Wurzel.

Am Ende eines Gültigkeitsbereiches findet eine implizite Synchronisation derart statt, daß ein Prozeß einen Gültigkeitsbereich erst dann verlassen darf, wenn zu den darin enthaltenen TASK-Objekten keine Prozesse mehr existieren.

4.2.7.3 Direkte Prozeßsteuerung. Durch eine Anweisung der Form (4.2-13) wird zu einer Reihe von TASK-Objekten je ein Prozeß erzeugt und gestartet.

```
INITIATE task-bez-folge;                  (4.2-13)
```

Zu einem Zeitpunkt darf zu einem TASK-Objekt höchstens ein Prozeß existieren.

Ein Prozeß kann sich für eine bestimmte Zeitdauer anhalten durch die Ausführung einer DELAY-Anweisung (4.2-14).

```
DELAY einf-ausdr;                         (4.2-14)
```

Wie üblich wird ein Prozeß beendet, wenn er das Ende des Rumpfes der zugehörigen TASK-Deklaration erreicht (und keine lokalen Prozesse mehr existieren).

Eine zweite Möglichkeit der Beendigung ist durch die Anweisung der Form (4.2-15) gegeben.

```
ABORT task-bez-folge;                     (4.2-15)
```

Dadurch werden die betreffenden Prozesse und alle ihre Unterprozesse bedingungslos abgebrochen und vernichtet.

4.2.7.4 Indirekte Prozeßsteuerung. Eine indirekte Steuerung von Prozessen ist zum Zwecke von Synchronisation und Kommunikation möglich. Für beides wird ein einheitlicher Mechanismus verwendet, der auf der Definition von Nachrichteneingängen (ENTRY) (4.2-16) und Annahmeanweisungen (ACCEPT) (4.2-17) beruht, wobei eine Annahmeanweisung für einen Nachrichteneingang nur in dem Prozeß enthalten und abgearbeitet werden kann, in dem dieser Nachrichteneingang deklariert ist.

```
ENTRY eing-bez [form-param];                    (4.2-16)

ACCEPT eing-bez [form-param]
DO anw-folge                                    (4.2-17)
END eing-bez;
```

Die Nachrichteneingänge entsprechen etwa den „ports" in [Wal 72], den Nachrichteneingänge in [BWW 76:156f.] oder den „common procedures" in [Bri 78]. Nachrichteneingänge sehen von außen wie Prozeduren aus und werden auch ebenso aufgerufen. Zu einem Nachrichteneingang kann es mehrere Annahmeanweisungen geben, d. h. gewissermaßen mehrere Rümpfe. Dies unterscheidet sie von gewöhnlichen Prozeduren.

Hat der Nachrichteneingang Parameter, so kann eine Kommunikation in beliebiger Richtung stattfinden. Diese findet statt, wenn im Prozeß, in welchem der Nachrichteneingang deklariert ist, eine Annahmeanweisung initiiert wurde und der Nachrichteneingang von einem zweiten Prozeß aufgerufen wurde, wobei die Reihenfolge dieser beiden Ereignisse gleichgültig ist; sie müsssen nur beide eingetreten sein. Die sich nun anschließende Abarbeitung der Annahmeanweisung wird als Rendezvous bezeichnet. Der rufende Prozeß wird so lange angehalten, bis das Rendezvous beendet ist. Die Abarbeitung der Annahmeanweisung erfolgt durch den Prozeß, zu welchem der betreffende Nachrichteneingang gehört. Das Rendezvous kann als einseitig anonym bezeichnet werden, da nur der Aufruf eines Nachrichteneinganges prozeßspezifisch (gegebenenfalls in der qualifizierten Form proz-bez . eing-bez) erfolgt; in der ACCEPT-Anweisung wird nur der Bezeichner des Nachrichteneinganges selbst angegeben.

Aufrufe von Nachrichteneingängen werden so lange gestaut, bis sie in einem Rendezvous bedient werden können. In einem Rendezvous wird genau ein Aufruf eines Nachrichteneingangs bedient und eine zugehörige Annahmeanweisung einmal abgearbeitet. Das Entstauen gestauter Aufrufe erfolgt nach FIFO.

Mit der Annahmeanweisung (4.2-17) kann jeweils nur auf den Aufruf genau eines Nachrichteneinganges gewartet werden.

Mit der Auswahlanweisung (4.2-18), die dem „alternative construct" aus [Dij 75] ähnelt, kann auf den Aufruf mehrerer Nachrichteneingänge oder den Ablauf einer Zeitdauer gewartet werden.

```
SELECT [WHEN bedingung ⇒]
          ausw-alternative
       [OR [WHEN bedingung ⇒]
        ausw-alternative]*
       [ELSE anw-folge]
END SELECT;                                     (4.2-18)
ausw-alternative ::=
     annahme-anw [anw-folge]
     verzögerungs-anw [anw-folge]
```

Die Auswahlalternativen können noch von Bedingungen („guard" in [Dij 75]) abhängen. Eine fehlende Bedingung ist stets wahr. Bei der Abarbeitung einer Auswahlanweisung werden nur die Alternativen berücksichtigt, deren Bedingungen wahr sind. Diese Alternativen sind dann „offen". Wenn mehrere offene Alternativen abgearbeitet werden können, dann wird genau eine davon ausgewählt und abgearbeitet, wobei die Auswahl nichtdeterministisch erfolgt. Wenn mindestens eine Alternative offen ist, dann wird gegebenenfalls gewartet (wenn keine Aufrufe für die betreffenden Nachrichteneingänge vorliegen). Wenn alle Alternativen geschlossen sind und eine ELSE-Klausel vorhanden ist, dann wird diese abgearbeitet. Wenn alle Alternativen geschlossen sind und keine ELSE-Klausel vorhanden ist, dann wird die Fehlerbedingung SELECT_ERROR gesetzt.

Wenn man die Auswahlanweisung in eine Schleife einbettet, dann erhält man eine Konstruktion, die dem „repetitive construct" in [Dij 75] entspricht.

4.2.7.5 Priorität und Parameter. Jeder Prozeß hat eine Priorität. Der Wertebereich der Prioritätswerte ist ein implementationsabhängiges Intervall der ganzen Zahlen. Beim Start eines Prozesses erhält der gestartete Prozeß dieselbe Priorität wie derjenige, der die Startanweisung ausführt. Das Gesamtprogramm hat eine implementationsabhängige mittlere Priorität. Ein Prozeß kann seine eigene Priorität jederzeit setzen und die Priorität der ihm bekannten Prozesse abfragen.

Prozesse haben keine Parameter und können nicht als Parameter auftreten.

Eingegangen: 24. 4. 1979; in überarbeiteter Form: 14. 8. 79

J. F. H. Winkler
Institut für Informatik 3
Universität Karlsruhe
Zirkel 2
D-7500 Karlsruhe 1