[6] *Hoffmann, R.:* Erfahrungen mit der Sprache CASSANDRE. Informatik-Forschungsgruppe Rechnerorganisation und Schaltwerke, Technische Universität Berlin, 1974.

[7] *Jost, W.:* Stand der Arbeiten zur formalen Beschreibung von Hardwarekomplexen mit PL/1. SIEMENS AG, München 1973 (firmeninterner Bericht).

[8] *Jost, W.; Zoschke, M.:* Beschreibungsebenen vermittlungstechnischer Entwicklungsprobleme. SIEMENS AG, München 1972 (firmeninterner Bericht).

[9] *Knobloch, H.-J.:* A RTL-Simulator with a flexible control language and its application to test and verification experiments. Workshop on Computer Hardware Description Languages, Darmstadt 1974.

[10] *Lux, A.:* Die Sprache CASSANDRE. Programmieranleitung. IMAG, Grenoble 1971.

[11] *Piloty, R.:* RTS I (3. Auflage). Institut für Nachrichtenverarbeitung, Technische Hochschule Darmstadt, 1969.

[12] *Zoschke, M.:* Anleitung zur Beschreibung von Funktionseinheiten mit Hilfe von LDS. SIEMENS AG, München 1973 (firmeninterner Bericht).

[13] *Schmitt, T.:* Beschreibung eines Mikroprogrammsteuerwerks durch verschiedene Entwurfssprachen. Diplomarbeit an der Universität Karlsruhe, Fakultät für Informatik, Institut für Informatik IV, 1975.

[14] *Piloty, R.:* Guidelines for a computer hardware description consensus language CONLAN. Institut für Datentechnik, TH Darmstadt, Bericht Nr. RO 76/4, 1976.

[15] *Schmid, D.; Schmitt, T.:* Vergleich und Bewertung einiger Registertransfersprachen. Elektron. Rechenanl. 21 (1979), H. 1.

Die Verfasser danken *W. Jost*, SIEMENS AG, München, für seine große Unterstützung und Hilfe bei der Ausführung dieser Arbeit.

Zum Begriff des Prozesses: am Beispiel von PEARL*

The concept of process: the real-time language PEARL

Elektron. Rechenanl. 20 (1978), H. 6, S. 277 – 282
Manuskripteingang: 27. Februar 1978

von J. F. H. WINKLER
Institut für Informatik III
der Universität Karlsruhe

Der Begriff des (Rechen-)Prozesses entstand innerhalb des Gebietes der Betriebssysteme. Er führte dort zu einer übersichtlichen Struktur und zur Bildung einfacher Schnittstellen zwischen Betriebssystem und Anwenderprogramm und Teilen des Betriebssystems selbst. Das Konzept des Prozesses wurde auch in Programmiersprachen aufgenommen. Insbesondere für die in den letzten Jahren entstehenden Echtzeitsprachen ist es von essentieller Bedeutung. Im folgenden Beitrag wird der Prozeßbegriff der Echtzeitsprache PEARL dargestellt und durch Zustandsübergangdiagramme veranschaulicht, die eine Erweiterung der bisher verwendeten Diagramme darstellen.

The term "process" emerged in the field of operating systems. There it led to a clear structure, and to simple interfaces between the operating system and the user program and between different parts of the operating system. This concept of process (task) was also included in programming languages. It is an essential concept for the real-time languages, which are emerging in the last few years. The following paper deals with the process concept of the real-time language PEARL, and characterizes this concept by means of state-transition diagrams, which are extensions of the diagrams as they are used up to now.

1. Einleitung

Vorbemerkung: Unter einem Prozeß wird im folgenden stets ein Rechenprozeß im Sinne von VDI/VDE 3554 verstanden. Es handelt sich dabei um eine Spezialform des allgemeinen, in DIN 66201 definierten, Prozeßbegriffes. Insbesondere ist nicht der technische Prozeß gemeint, der durch ein PEARL-Programm überwacht, gesteuert und/oder geregelt wird.

Der Begriff des (Rechen-)Prozesses entstand zuerst innerhalb des Gebietes der Betriebssysteme, und zwar als Verwaltungseinheit im Bereich der Ablauforganisation. Eine Reihe von Definitionen ist in [1: S. 4 bis 6] aufgeführt. Weitere Definitionen des Prozeßbegriffes finden sich in [2: S. 145; 3: S. 46; 4: S. 35; 5: S. 5; 6: S. 29]. Die Verwendung dieses Konzeptes führte bei der Betriebssystemkonstruktion zu einer besseren Gliederung und einer übersichtlicheren Organisation von Betriebssystemen und machte gewisse Vorgänge wie z. B. das Umschalten zwischen verschiedenen Einheiten des Ablaufgeschehens (Programmen) explizit. Solche Vorgänge und auch die Schnittstellen zwischen diesen Einheiten des Ablaufgeschehens waren vorher mehr implizit vorhanden [7: S. 321f.; 3: S. 45, 46].

Das Konzept des Prozesses wurde auch in höhere Programmiersprachen als Sprachelement aufgenommen (PL/1 [8], Burroughs Extended Algol [9], Concurrent Pascal [10]). Die in den letzten Jahren entstehenden Echtzeitsprachen machen ebenfalls intensiven Gebrauch vom Konzept des (Rechen-) Prozesses, da für die Anwendungsgebiete dieser Programmiersprachen Nebenläufigkeiten charakteristisch sind (Prozeß-Fortran [11], RTL [12 bis 14], PROCOL [13, 14], PEARL [15 bis 20], HAL/S [13], Modula [21], Real-time PL/1 [22]). Im Gegensatz zu den Betriebssystemen, wo das Prozeßkonzept mehr informell verwendet wurde und sich

* Dieser Aufsatz veröffentlicht 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.

keine einheitliche (formale) Definition ergab [23: S. 8; 3: S. 46], ist es bei der Aufnahme dieses Konzeptes in Programmiersprachen notwendig, eine eindeutige und präzise Definition vorzunehmen. Am vollständigsten wurde dies in der Echtzeitsprache PEARL durchgeführt. Die Hauptprobleme, die dabei auftraten, bestanden in der formalen Definition dessen, was ein Prozeß (im Unterschied zu dem in [15] als „Task" bezeichneten Objekt) ist und in der Festlegung der Semantik der Operationen an Prozessen, da in einem Benutzerprogramm jede beliebige Reihenfolge von Operationen an einem Prozeß vorkommen kann. Im Gegensatz dazu definieren Zustandsübergangsdiagramme aus dem Bereich der Betriebssysteme in der Regel nur die Semantik bestimmter Operationsfolgen [24: S.7, 11; 25: S. 203; 26. S. 389, 391; 5: S. 210; 27: S. 102; 28: S. 63].

Im folgenden soll dieses neue Prozeßkonzept erläutert und der Versuch unternommen werden, es durch Zustandsdiagramme zu veranschaulichen.

2. Prozeß und Task

Aus den zahlreichen im Bereich der Betriebssysteme vorliegenden, oft informalen Prozeßdefinitionen [1: S. 4 bis 6; 2: S. 145; 3: S. 46; 5: S. 5; 6: S. 29] kristallisiert sich folgende als allgemein zutreffendste heraus.

> „Ein Prozeß ist die einmalige Ausführung einer (geschlossenen) Prozedur in einer wohldefinierten Umgebung."

Dieser Begriff des Prozesses unterscheidet sich nun von dem in [15] definierten Begriff „Task" durch die Einmaligkeit der Ausführung des zu dem Prozeß gehörenden Programmes, denn der Rumpf einer Task nach [15] kann beliebig oft ausgeführt werden. Dieser Unterschied wird auch schon von Rieder [29: S. 421, 422] betont.

In PEARL-77 [17; 20] unterscheidet man daher die folgenden beiden Begriffe:

- Activity: die einmalige Ausführung des Rumpfes einer Taskdeklaration. Dies wird im folgenden auch mit TA (Task-Activity) abgekürzt.
- Task: ein Objekt zur Verwaltung einer Menge von Activities.

Die zu verschiedenen Tasks gehörenden Mengen von Activities sind stets disjunkt. Unter einer Activity wird genau das verstanden, was in Abschnitt 1 als Prozeß bezeichnet wird. Um jedoch den Zusammenhang mit der Sprachdefinition [17] zu wahren, werden im folgenden weiterhin die Bezeichnungen „Task" und „Activity" verwendet. In [14] und [13] wird statt „Activity" „Parallel Activity (PA)" verwendet.

So wie jedes Programm einen statischen und einen dynamischen Aspekt hat, ist die Task ein Element der statischen Programmstruktur und die Activity die Gliederungseinheit des Ablaufgeschehens, welches durch die Abarbeitung eines PEARL-Programmes definiert wird.

Genaugenommen gilt das für die Task Gesagte nicht für das, was in der Programmniederschrift als Taskdeklaration (TD) auftritt, sondern für jede Inkarnation dieses Objektes während der Abarbeitung des Programmes. Dabei kann es durchaus vorkommen, daß zu einem Zeitpunkt während der Abarbeitung des Programmes mehrere Inkarnationen ein und derselben Taskdeklaration existieren, so wie die lokalen Variablen einer rekursiven Algol 60-Prozedur zu einem Zeitpunkt mehrfach vorhanden sein können.

In der Ausdrucksweise der Betriebssysteme kann man diese Situation folgendermaßen beschreiben: jede Inkarnation einer Taskdeklaration stellt einen Prozeßleitblock [30: S. 8; 5: S. 236; 31: S. 517; 32: S. 24] dar. Zu all diesen Prozeßleitblöcken gehört als Programm dasselbe Programmstück nämlich der Rumpf der betreffenden Taskdeklaration. Zu jedem Prozeßleitblock gehört eine Menge von Prozessen, die streng sequentiell nacheinander ablaufen (abgewickelt werden), d. h. die zu einer Taskinkarnation (TI) gehörende Activitymenge ist totalgeordnet, wobei das Ordnungskriterium der Zeitpunkt der Erzeugung der Activities ist. Die erste TA ist dabei die mit dem frühesten Erschaffenszeitpunkt.

Nachdem bisher mehr die statischen Aspekte der Begriffe „Task", „Taskinkarnation" und „Activity" behandelt wurden, soll in den folgenden Abschnitten auf die dynamischen Aspekte eingegangen werden. Dabei handelt es sich um die Operationen, die auf Taskinkarnationen und/oder Activities ausgeübt werden können und die Wirkungen dieser Operationen.

3. Operationen an Taskinkarnationen und Activities

3.1 Unmittelbare (Direkte) Operationen

3.1.1 Vereinbarung (= Schaffen einer TI)

Durch die Abarbeitung einer Taskdeklaration wird eine neue Inkarnation der betreffenden Taskdeklaration geschaffen. Die Menge der Activities dieser Taskinkarnation ist leer. Wenn in der Deklaration eine Priorität angegeben ist, dann gilt diese Priorität für alle TA, die jemals zu dieser Taskinkarnation gehören.

3.1.2 Blockverlassen (= Vernichten einer TI)

Beim Verlassen eines Blockes, bei dessen Betreten eine Taskdeklaration abgearbeitet wurde und somit (nach 3.1.1) eine neue Taskinkarnation geschaffen wurde, wird diese Taskinkarnation wieder vernichtet. Daß dieses Vernichten nur in einem besonderen Zustand der TI möglich ist, darauf wird im Abschnitt 3.3.2 eingegangen werden.

3.1.3 ACTIVATE (=Schaffen einer TA)

Durch die Anweisung „ACTIVATE", die sich stets auf eine bestimmte TI bezieht, wird eine neue TA geschaffen und zur Menge der Activities dieser TI hinzugefügt. War diese Menge vorher leer, so wird sofort mit der Ausführung dieser TA begonnen. Wenn die TA-Menge der TI nicht leer war, dann wird die TA in einen Wartezustand versetzt, den man mit „Warten auf das Verfügbarwerden des (virtuellen) Prozessors der zugehörigen TI" bezeichnen kann. Der Begriff des virtuellen Prozessors wird hier im Sinne von Saltzer [28] verwendet.

Die Priorität dieser neuen TA kann auf zweierlei Art und Weise festgelegt werden. Wenn in der zugehörigen Taskdekla-

ration eine Prioritätsangabe enthalten ist, dann darf die ACTIVATE-Anweisung keine Prioritätsangabe enthalten. Enthält die zugehörige TD keine Prioritätsangabe, dann kann die ACTIVATE-Anweisung eine solche enthalten. Diese Prioritätsangabe wird dann als Teil der Abarbeitung der ACTIVATE-Anweisung ausgewertet und ergibt die Priorität der neugeschaffenen TA. Enthält auch die ACTIVATE-Anweisung keine Prioritätsangabe, dann wird seitens des Systems ein Standardwert als Priorität der neugeschaffenen TA angenommen.

3.1.4 TERMINATE (= Vernichten einer TA)

Durch die Anweisung TERMINATE, die sich stets auf eine bestimmte TI bezieht, wird die erste TA dieser TI (sofern die TA-Menge nicht leer ist) abgebrochen und vernichtet. Dabei spielt der aktuelle Zustand dieser TA keine Rolle. Da ein solch durchgreifendes Abbrechen bei der Betriebsmittelverwaltung zu Problemen führen kann, werden exklusiv belegte Betriebsmittel automatisch freigegeben. Wenn nach dem Entfernen des ersten Elementes die TA-Menge nicht leer ist, dann wird sofort mit der Ausführung des nun ersten Elementes der TA-Menge begonnen.

3.1.5 SUSPEND (= Anhalten einer TA)

Sofern die TA-Menge der betroffenen TI nicht leer ist, wird die erste TA in einen Wartezustand versetzt. Befand sie sich schon in einem durch SUSPEND bewirkten Wartezustand, so wird die erneute Ausführung als Programmlaufbesonderheit (exceptional condition, Irregularität) aufgefaßt und eine entsprechende Reaktion des Systems veranlaßt.

3.1.6 CONTINUE (= Fortsetzen einer TA)

Wenn die TA-Menge der betroffenen TI nicht leer ist, und sich die erste TA in einem durch eine SUSPEND-Anweisung herbeigeführten Wartezustand befindet, dann wird diese TA wieder in den Zustand versetzt, den sie vor der Abarbeitung der betreffenden SUSPEND-Anweisung innehatte. Enthält die zugehörige TD keine Prioritätsangabe, dann darf die CONTINUE-Anweisung eine Prioritätsangabe enthalten, die als Teil der Abarbeitung dieser CONTINUE-Anweisung ausgewertet wird und die anschließend gültige Priorität der betreffenden TA ergibt.

3.2 Einplanungen von Operationen

3.2.1 Zeitpläne

Für die vier in 3.1.3 bis 3.1.6 beschriebenen Operationen an TI und/oder TA kann mittels Zeitplänen (in [17] „schedule-element") die ein- oder mehrmalige Ausführung durch das System vorgemerkt werden. Ein Zeitplan beschreibt eine Menge von Zeitpunkten der Struktur

$ta, ta+dt, ta+2 \cdot dt, \ldots, ta+n \cdot dt, \ldots$

Der Zeitpunkt ta kann festgelegt werden durch:

a) Die Angabe einer Uhrzeit t (AT-Klausel).
   Das bedeutet dann, daß $ta = t$ gilt.

b) Die Angabe einer Verzögerung dta (AFTER-Klausel).
   Das bedeutet dann, daß ta = aktuelle Zeit + dta gilt.

c) Die Angabe eines Interrupts (WHEN-Klausel).
   Der Zeitpunkt eines nach der Eintragung des betreffenden Zeitplanes erfolgenden Auftretens des angegebenen Interrupts ist dann gleich dem Anfangszeitpunkt ta.

d) Kombination aus b) und c) (WHEN ... AFTER-Klausel).

Die Zykluszeit dt wird durch die Angabe eines Zeitintervalles (dur-exp7) festgelegt. Ist keine Zykluszeit angegeben, dann enthält der Zeitplan nur ein Element.

Die Anzahl der Elemente eines Zeitplanes kann beschränkt werden durch

a) die Angabe eines Endezeitpunktes te (UNTIL-Klausel). Der Zeitplan besteht dann aus den Zeitpunkten ti, für die gilt:

   $ta \leq ti = ta + i \cdot dt \leq te$.

b) die Angabe eines Zeitintervalles dtp (DURING-Klausel). Der Zeitplan besteht dann aus den Zeitpunkten ti, für die gilt:

   $ta \leq ti = ta + i \cdot dt \leq ta + dtp$.

Dabei ist i eine natürliche Zahl (Null eingeschlossen).

Ist weder die DURING-Klausel noch die UNTIL-Klausel aber eine Zykluszeit angegeben, dann enthält der Zeitplan unendlich viele Zeitpunkte.

3.2.2 Einplanen

In [17] werden die Anweisungen zum Einplanen von Operationen an TI und/oder TA etwas irreführend als bedingte Operationen bezeichnet, obwohl die unmittelbare Wirkung der Abarbeitung einer Anweisung der Form

schedule-1, schedule-2, ..., schedule-*n* task-operation

darin besteht, daß die betreffende „task-operation" für die durch die Zeitpläne „schedule-1" bis „schedule-*n*" festgelegten Zeitpunkte vorgemerkt wird. Die „task-operation" selbst wird dann zu den betreffenden Zeitpunkten vom System durchgeführt.

Einplanungen betreffen immer Taskinkarnationen und nicht individuelle TA, denn zum Zeitpunkt des Einplanens kann nicht vorausgesagt werden, welche Activities von der gerade eingeplanten „task-operation" einmal betroffen sein werden.

Der Vorgang des Vormerkens, d. h. das Einplanen geschieht für jede „task-Operation" separat, also getrennt für ACTIVATE, TERMINATE, SUSPEND und CONTINUE.

3.2.3 Ausplanen

Das Streichen von durch Zeitpläne festgelegten Vormerkungen erfolgt pauschal für alle auf eine TI bezogenen Einplanungen durch die Anweisung PREVENT. Außerdem werden durch diese Anweisung alle zu dieser TI gehörenden TA, ausgenommen die erste TA, vernichtet. Ob eine größere Differenzierung beim Ausplanen notwendig ist, wird erst die praktische Erfahrung bei Verwendung der Sprache zeigen können.

Die Anweisung PREVENT kann selbst auch eingeplant werden. Die Angabe eines Zeitplanes mit mehr als einem Element ist dabei zwar möglich aber sinnlos, da die erste Durchführung zur Beseitigung auch dieses Zeitplanes führt.

3.3 Synchronisation

3.3.1 Explizite Synchronisation

Zur Realisierung von koordinierten Zugriffen auf gemeinsame Daten sind in PEARL zwei Arten von Synchronisationsgrößen vorgesehen, die in PEARL als SEMA(phor) und BOLT bezeichnet werden. Ein SEMA-Objekt ist ein allgemeiner Semaphor im Sinne von Dijkstra [33: S. 124], wobei das Fortsetzen blockierter Activities in der Reihenfolge der Priorität erfolgt [34: S. 146]. Die Objekte vom Typ BOLT sind speziell auf Probleme von der Art des Leser-Schreiber-Problems [35 bis 37] zugeschnitten; man kann sie als Kombination eines binären und eines allgemeinen Semaphors auffassen, wobei der binäre Semaphor zur Synchronisation der Schreiber und der allgemeine zur Synchronisation der Leser dient. Das Verhältnis beider Benutzerklassen ist so definiert, daß der Zugriffswunsch eines Schreibers Leser mit niedrigerer Priorität daran hindert einen Zugriff durchzuführen.

Die Auswirkungen von Operationen an Synchronisationsgrößen betreffen immer einzelne Activities.

3.3.2 Implizite Synchronisation

PEARL ist eine Sprache mit Blockstruktur und im Vereinbarungsteil eines jeden Blockes dürfen Taskdeklarationen enthalten sein. Dies kann zu einer Schachtelung von Taskinkarnationen derart führen, daß im Zuge der Ausführung einer TA eine Taskdeklaration abgearbeitet wird und Activities zu dieser neuen TI geschaffen werden. Da nun die ursprüngliche TA und neue TA parallel mit beliebigen Geschwindigkeiten abgearbeitet werden, kann der Fall eintreten, daß eine TA einen Block verlassen will (womit auch die darin deklarierten Objekte zu vernichten sind), obwohl noch TA zu einer in diesem Block enthaltenen TI existieren. In PEARL ist dieser Fall nun so entschieden, daß eine TA einen Block erst dann verlassen kann, wenn für alle in dem betreffenden Block enthaltenen Taskinkarnationen die Menge der TA leer ist und keine Einplanungen für die Operation ACTIVATE mehr vorhanden sind. Gegebenenfalls wird die TA, die einen Block verlassen will, solange angehalten, bis die eben erwähnte Bedingung erfüllt ist. Man spricht in diesem Zusammenhang von einer impliziten Synchronisation am Blockende. Diese Synchronisation betrifft hauptsächlich die Taskinkarnationen, die in dem zu verlassenden Block enthalten sind, und nicht nur deren individuelle Activities.

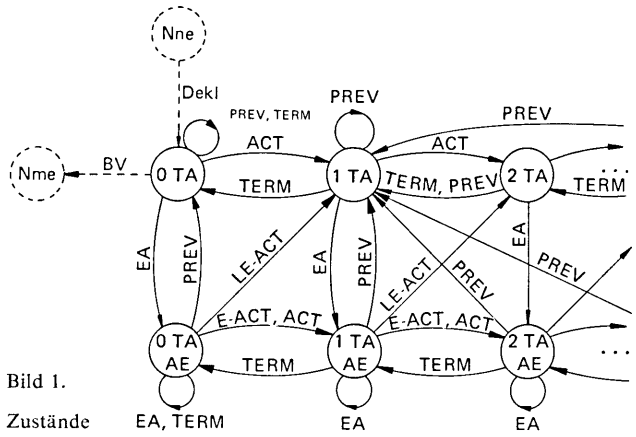

Bild 1.

Zustände

Nne: noch nicht existent.

Nme: nicht mehr existent.

n TA: Die Menge der zu der betreffenden TI gehörenden TA hat *n* Elemente.

AE: Einplanungen für ACTIVATE vorhanden.

*n*TA/AE: Konjunktion von *n* TA und AE.

Zustandsübergänge

Dekl: Abarbeitung der betreffenden Taskdeklaration.

BV: Verlassen des Blockes, in dessen Vereinbarungsteil die betreffende Taskdeklaration unmittelbar enthalten ist.

ACT: Abarbeitung einer ACTIVATE-Anweisung.

TERM: Durchführung der Operation TERMINATE.

PREV: Durchführung der Operation PREVENT.

EA: Einplanung von ACTIVATE, d. h. Abarbeitung einer Einplanungsanweisung für ACTIVATE.

LE-ACT: Der Zeitpunkt, an dem das letzte Mal eine eingeplante ACTIVATE-Operation durchgeführt werden soll, ist eingetreten. Die Operation wird durchgeführt und die Einplanung gestrichen.

E-ACT: Ein Zeitpunkt, an dem die eingeplante ACTIVATE-Operation durchgeführt werden soll, ist eingetreten, und dies ist nicht der letzte dieser Zeitpunkte. Die Operation wird durchgeführt.

4. Zustandsdiagramme

Was in den Abschnitten 2 und 3 über Taskinkarnation und Activity gesagt wurde, soll nun anhand von zwei Zustandsdiagrammen zusammengefaßt und veranschaulicht werden.

Dabei wird ein Diagramm für Taskinkarnationen (Bild 1) und eines für Activities (Bild 2) aufgestellt. Diese Trennung ergibt eine übersichtlichere Darstellung als die Zusammenfassung in einem Diagramm, die auch verschiedentlich vorgenommen wurde [38: S. 100, 167; 39: S. 213, 29; 40: S. 25; 16: S. 137]. Die Abgrenzung der beiden Diagramme ergibt sich daraus, ob ein Vorgang nur eine Activity oder mehr Komponenten einer Taskinkarnation betrifft, wie z. B. Zeitpläne.

4.1 Zustandsdiagramm für Taskinkarnationen

Eine Taskinkarnation wird geschaffen bei der Abarbeitung einer Taskdeklaration im Zuge des Betretens eines Blockes. Dadurch geht die TI vom Zustand „Noch-nicht-existent" in den Zustand „0 TA" über. Nur wenn sich die TI in diesem Zustand befindet, kann die betreffende Blockinkarnation wieder verlassen werden. In dieser Beschränkung kommt die in Abschnitt 3.3.2 erwähnte implizite Synchronisation zum Ausdruck.

Der Zustand einer TI wird charakterisiert durch die Anzahl der für diese TI noch abzuwickelnden Activities und die Tatsache, ob Einplanungen der Operation ACTIVATE für diese TI vorhanden sind. Diese beiden Variationsmöglichkeiten sind in Bild 1 in den zwei Dimensionen der Ebene dargestellt. Von links nach rechts steigt die Zahl der noch abzuwickelnden TA; in den Zuständen der oberen Zeile des Diagramms sind keine Einplanungen für ACTIVATE vorhanden und in den unteren Zuständen ist dies der Fall.

Das Diagramm erstreckt sich im Prinzip beliebig weit nach rechts. In einer konkreten Implementierung kann die Anzahl der für eine TI noch abzuwickelnden TA durch eine Größe „nbuff" beschränkt sein [17: S. 1.4/17], so daß sich dann ein endliches TI-Zustandsdiagramm ergibt.

Wegen der Bedeutung der ACTIVATE-Einplanungen für den TI-Zustand spielt es bei der ACTIVATE-Operation eine Rolle, ob sie durch eine ACTIVATE-Anweisung oder durch eine Einplanung initiiert wird. Dies ist in Bild 1 durch die drei Zustandsübergänge „ACT", „E-ACT" und „LE-ACT" berücksichtigt. Bei den Operationen PREVENT und TERMINATE ist es ohne Bedeutung, ob die betreffende Operation durch eine entsprechende Anweisung oder durch eine Einplanung initiiert wird.

Das Diagramm in Bild 1 ist vollständig in dem Sinne, als es alle zur Laufzeit möglichen Zustandsübergänge enthält. Im Zustand „Noch-nicht-existent" kann z. B. nicht die Operation ACTIVATE auf diese TI ausgeübt werden, da dies durch die Regeln über den Gültigkeitsbereich von Bezeichnern verhindert wird, die bereits bei der Übersetzung des Programmes überprüft werden.

Ebenso kann es nicht vorkommen, daß vom Zustand „1 TA" ein Zustandsübergang „E-ACT" erfolgt, denn wenn keine Einplanung vorhanden ist, kann auch keine ACTIVATE-Operation aus einer Einplanung heraus initiiert werden.

In Basis-PEARL [16] sind Taskdeklarationen nur auf Modulebene zulässig. Das hat zur Folge, daß es zu diesen TD jeweils genau eine TI gibt, die bei Programmstart alle existent sind. Daher erhält man das TI-Diagramm für Basis-PEARL, indem man die in Bild 1 gestrichelten Teile wegläßt.

4.2 Zustandsdiagramm für Taskactivities

Da die TA dem im Bereich der Betriebssysteme entwickelten Konzept des Prozesses entspricht, ähnelt das in Bild 2 dargestellte Zustandsdiagramm für TA den bekannten Prozeßzustandsdiagrammen [27: S. 102; 26: S. 391; 1: S. 3; 30: S. 12, 14].

Durch die Operation ACTIVATE wird eine TA erzeugt. Diese TA kann erst dann abgearbeitet werden, wenn alle vor ihr für dieselbe TI erzeugten TA beendet worden sind und sich im Zustand „Nme" befinden. Durch die Operation ACTIVATE findet ein Übergang von „Noch-nicht-existent" in „Warten-auf-virtuellen-Prozessor" statt. Auch die zugehörige TI erfährt dadurch einen Zustandsübergang (s. Bild 1).

Im Zustand „WvP" ist die TA nur beeinflußbar durch die Operation PREVENT, und zwar beseitigt diese Operation alle die TA der betroffenen TI, die sich in diesem Zustand befinden. Wenn sich die zugehörige TI in einem Zustand „n TA" oder „n TA/AE" befindet mit $n > 0$, dann befindet sich genau eine TA in einem der vier Zustände „WC", „A", „WS" und „WCS", und n-1 TA befinden sich im Zustand „WvP". Durch die Operation PREVENT werden diese n-1 TA beseitigt. Dies ist auch in Bild 1 deutlich erkennbar, in welchem fast alle mit „PREV" markierten Kanten im Zustand „1 TA" enden.

Während des eigentlichen Ablaufs befindet sich eine TA in einem der vier Zustände „A", „WC", „WS" und „WCS", wovon die letzten drei Wartezustände sind. In allen diesen vier Zuständen kann eine der Operationen CONTINUE, SUSPEND und TERMINATE auf die TA ausgeübt werden. Der Zustandsübergang „Bel-S-bel" kann nur im Zustand „A" durch die TA selbst bewirkt werden, denn er beruht auf der Ausführung einer belegenden Synchronisationsoperation (REQUEST, ENTER, RESERVE) durch die TA. Daher gehen von „WC", „WS" und „WCS" keine mit „Bel-S-bel" markierten Kanten aus. Das Freigeben einer Synchronisiergröße ist für eine TA nur dann von Bedeutung, wenn sie sich im Zustand „WS" befindet und auf das Freigeben dieser Synchronisiergröße wartet. In diesem Falle wird die TA in den Zustand „A" versetzt und wiederholt die belegende Synchronisationsoperation.

Wird die TA im Zustand „WS" von der Operation SUSPEND betroffen, dann wird sie in den Zustand „WCS" überführt. In diesem Zustand nimmt sie nicht mehr am Wettbewerb um die zu belegende Synchronisiergröße teil, d. h. während sich eine TA im Zustand „WCS" befindet, ist es für sie nicht von Bedeutung, welche Synchronisationsoperationen auf die betreffende Synchronisiergröße ausgeübt werden. Wird die TA durch eine CONTINUE-Operation festgesetzt, so wird sie in den Zustand „A" versetzt und wiederholt die belegende Synchronisationsoperation. Je nach dem aktuellen Zustand der Synchronisiergröße ist diese Operation dann erfolgreich oder nicht.

Für alle hier vorkommenden Operationen ist es unerheblich, ob sie unmittelbar durch eine entsprechende Anweisung oder durch eine Einplanung initiiert werden.

Da in Basis-PEARL [16] eine TA die SUSPEND-Operation nur auf sich selbst ausüben kann, erhält man das TA-Zustandsdiagramm für Basis-PEARL, indem man in Bild 2 die gestrichelten Teile wegläßt.

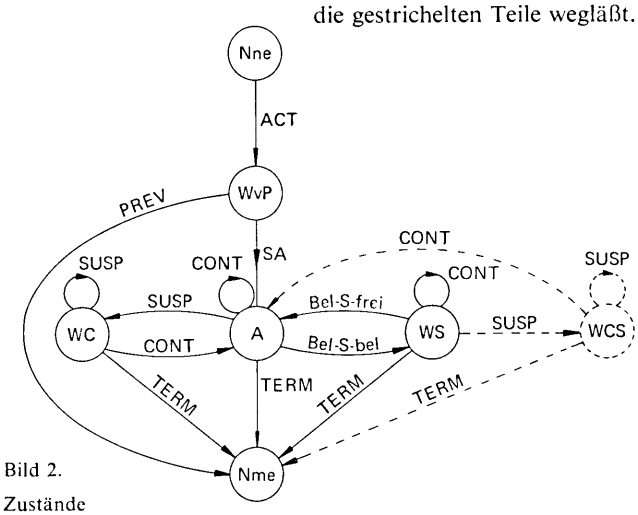

Bild 2.

Zustände

Nne:	noch nicht existent.
WvP:	Warten auf virtuellen Prozessor.
A:	Aktiv.
WC:	Warten auf CONTINUE.
WS:	Warten auf Freiwerden einer Synchronisiergröße.
WCS:	Warten auf CONTINUE und das Freiwerden einer Synchronisiergröße.
Nme:	nicht mehr existent.

Zustandsübergänge

ACT:	Durchführung der ACTIVATE-Operation.
SA:	Systemaktion: zuordnen zum freien virtuellen Prozessor.
SUSP:	Durchführung der SUSPEND-Operation.
CONT:	Durchführung der CONTINUE-Operation.
Bel-S-bel:	Versuch, eine belegte Synchronisiergröße zu belegen.
Bel-S-frei:	Freigabe einer belegten Synchronisiergröße, auf deren Freigabe die betreffende TA wartet.
TERM:	Durchführung der TERMINATE-Operation.

5. Schlußbemerkung

Der vorstehende Abriß des Prozeßkonzeptes der Echtzeitsprache PEARL gibt nicht alle Einzelheiten wieder, da dies den Rahmen einer solchen Arbeit überschreiten würde. Für weitere Details wie

- die Taskoperation RESUME
- die Prioritätsattribute SYS und REL
- die Unterschiede in der Angabe der Zykluszeit von Zeitplänen mittels EVERY und ALL
- die USING-Klausel bei der ACTIVATE-Operation
- die Operationen auf Listen von Synchronisiergrößen

sei der interessierte Leser daher auf die Sprachberichte [16] und [17] verwiesen.

Hier konnten nur die Grundzüge des PEARL-Prozeßkonzeptes herausgearbeitet werden, wobei einerseits darauf hingewiesen werden sollte, daß die Aufnahme eines Konzeptes in eine Programmiersprache eine klärende und normierende Wirkung hat, und andererseits die drei Begriffe, die dieses Prozeßkonzept charakterisieren, eingeführt und erläutert wurden.

Zusammenfassend kann man diese drei Begriffe folgendermaßen charakterisieren:

- Taskdeklaration: ist die Schablone für alle zugehörigen Taskinkarnationen und enthält den Programmcode, der allen zu diesen Taskinkarnationen gehörenden Taskausführungen gemeinsam ist.
- Taskinkarnation: ist die einmalige Realisierung einer Taskdeklaration. Sie ist ein Datenobjekt, welches zur Verwaltung einer Klasse von Taskausführungen dient.
- Taskausführung: (in PEARL als „activity" bezeichnet) ist die einmalige Ausführung der in der zugehörigen Taskdeklaration enthaltenen Anweisungsfolge, in einer Umgebung (Adreßraum), deren lokale Komponenten nur zu dieser individuellen Taskausführung gehören.

Für eine kritische Durchsicht des Manuskriptes und wertvolle Hinweise bin ich *H. Wettstein*, *C. Stoffel* und *H. Vogt* zu Dank verpflichtet.

6. Literatur

[1] *Winkler, J. F. H.:* Zur Terminologie der Ablaufsteuerung. Universität Karlsruhe, Institut für Informatik III, 1973.

[2] *Dennis, Jack B.; van Horn, Earl C.:* Programming Semantics for Multiprogrammed Computations. CACM 9 (1966), 3, 143–155.

[3] *Habermann, A. N.:* Introduction to Operating System Design. Science Research Associates Inc., Chicago etc. 1976.

[4] *Krönig, Dirk* (Hrsg.): Praxis von Sprachen, Programmiersystemen und Programmgeneratoren. Carl Hanser, München usw. 1976.

[5] *Madnick, Stuart E.; Donovan, John J.:* Operating Systems. Mc Graw-Hill, New York etc. 1974.

[6] *Brinch Hansen, Per:* Operating System Principles. Prentice Hall Inc., Englewood Cliffs N. J. 1973.

[7] *Güntsch, Fritz Rudolf:* Einführung in die Programmierung digitaler Rechenautomaten. Walter de Gruyter, Berlin 1963.

[8] International Business Machines Corp.: IBM System/360 Operating System. PL/1(F) Language Reference Manual. Order No. GC 28-8201-4, 5. ed. Dec. 1972.

[9] Burroughs Corporation: B 6700/7700 ALGOL. Language Reference Manual. No. 5000649, 20 May 1974.

[10] *Brinch Hansen, Per:* Concurrent Pascal Report. Information Science, California Institute of Technology, June 1975.

[11] VDI/VDE 3556: Prozeß-Fortran 75. VDI/VDE Richtlinie, März 1978.

[12] *Barnes, J. G. P.:* Real time languages for process control. Comp. Journal 15 (1972), 1, 15–17.

[13] *Schwald, A.; Baumann, R.:* PEARL im Vergleich mit anderen Echtzeitsprachen. Regelungstechnik 25 (1977), 11, 337–344.

[14] *Rössler, R.; Schenk, K.:* LTPL-European Group. Language-Comparison. Universität Erlangen, Physikalisches Institut, 1975.

[15] *Timmesfeld, K. H.; Schürlein, B.; Rieder, P.* u. a.: PEARL – A proposal for a process- and experiment automation realtime language. Ges. für Kernforschung Karlsruhe, KFK-PDV 1, 1973.

[16] PDV: Projekt Prozeßlenkung mit DV-Anlagen. Basic PEARL-Language description. Ges. für Kernforschung Karlsruhe, KFK-PDV 120, 1977.

[17] PDV: Projekt Prozeßlenkung mit DV-Anlagen. Full PEARL-Language description. Ges. für Kernforschung Karlsruhe, KFK-PDV 130, 1977.

[18] *Brandes, J.; Eichentopf, S.; Elzer, P.* u. a.: PEARL – The Concept of a Process- and Experiment-oriented Programming Language. elektr. datenverarbeitung 12 (1970), 429–442.

[19] *Eichenauer, B.; Haase, V.; Holleczek, P.* u. a.: PEARL – eine prozeß- und experimentorientierte Programmiersprache. Angew. Informatik (1973), 363–372.

[20] *Kappatsch, A.:* Was ist PEARL? Elektron. Rechenanl. 19 (1977), 6, 284–290.

[21] *Wirth, N.:* Modula: a Language for Modular-Multiprogramming. Software – Pract. + Exp. 7 (1977), 1, 3–35.

[22] *Freiburghouse, R. A.:* Proposed Extensions to PL/I for Real-Time Applications. SIGPLAN Notices 12 (1977), 7, 26–42.

[23] *Coffman, Edward G. Jr.; Denning, Peter J.:* Operating Systems Theory. Prentice Hall Inc., Englewood Cliffs N.J. 1973.

[24] Arbeitsgruppe E2 „Programmierung von Prozeßrechnern". Ein einfaches Zustandsmodell für strikt eingriffsgesteuerten Echtzeitbetrieb. Math. Institut der TU München, Bericht 7306, Mai 1973.

[25] *Denning, Peter J.:* Third Generation Computer Systems. Computing Surveys 3 (1971), 4, 175–216.

[26] *Donovan, John J.:* Systems Programming. Mc Graw Hill, New York etc., 1972.

[27] *Hansen, Per Brinch:* Short-Term Scheduling in Multiprogramming Systems. Operating Systems Review 6 (1972), 1–2, 101–105.

[28] *Saltzer, J. H.:* Traffic Control in a Multiplexed Computer System. MAC-TR-30 (Ph. D. Thesis), MIT Cambridge Mass. July 1966.

[29] *Rieder, P.:* Prozeßzustände bei Echtzeitprogrammiersprachen = [41: S. 413–424].

[30] *Wettstein, H.:* Betriebssysteme. Skriptum. Universität Karlsruhe, Institut für Informatik III, 1977.

[31] *Jordain, Philip B.; Breslau, Michael:* condensed computer encyclopedia. Mc Graw Hill, New York etc. 1969.

[32] *Tsichritzis, Dionysios C.; Bernstein, Philip A.:* Operating Systems. Academic Press, New York etc. 1974.

[33] *Dijkstra, E. W.:* Hierarchical Ordering of Sequential Processes. Acta Informatica 1 (1971), 115–138.

[34] *Liskov, Barbara H.:* The Design of the Venus Operating System. CACM 15 (1972), 3, 144–149.

[35] *Courtois, P. J.; Heymans, F.; Parnas, D. L.:* Concurrent Control with "Readers" and "Writers". CACM 14 (1971), 10, 667–668.

[36] *Courtois, P. J.; Heymans, F.; Parnas, D. L.:* Comments on "A Comparison of Two Synchronizing Concepts by P. B. Hansen". Acta Informatica 1 (1972), 375–376.

[37] *Hansen, Per Brinch:* A Comparison of Two Synchronizing Concepts. Acta Informatica 1 (1972), 190–199.

[38] BBC-PEARL-Subset. Prozeßautomatisierungssprache. BBC-Prozeß-Datenverarbeitung DP 100. ZEK-ED, Dez. 1973, D ZEK 40053D.

[39] *Koch, G.; Kussl, V.:* Prozeßorientierte Programmiersprachen. Formen und Funktionen. In Lecture Notes in Computer Science 7, S. 119–218. Springer, Berlin usw. 1974.

[40] *Gottwald, H.-J.; Schoknecht, H.:* PEARL – Eine leistungsfähige Echtzeit-Programmiersprache. Regelungstechnik 26 (1978), 1, 23–27.

[41] *Krüger, Gerhard; Friehmelt, Rüdiger* (Hrsg.): Fachtagung Prozeßrechner 1974. Springer, Berlin usw. 1974.