FORMALE DEFINITION DER SEMANTIK VON PROZESSOPERATIONEN*

J.F.H. Winkler, München

Zusammenfassung

Das Konzept des Rechenprozesses wurde im Bereich der Betriebssysteme entwickelt und dient dort als wesentliche Gliederungseinheit des Ablaufgeschehens. Dieses Konzept wurde auch in Programmiersprachen übernommen, insbesondere in die Echtzeitsprachen, für welche es ein essentielles Sprachelement ist. Die Aufnahme des Prozeßkonzeptes in Programmiersprachen erfordert eine präzise Definition der Attribute von Prozessen und der Semantik von Operationen an Prozessen. In der vorliegenden Arbeit wird der Begriff der Prozeßoperation formal definiert. Zur formalen Beschreibung der Semantik solcher Operationen werden allgemeine Prozeßzustände als Operationsfolgen eingeführt. Mit diesem Ansatz lassen sich Phänomene wie Rückwirkung auf den initiierenden Prozeß und Stauung nicht unmittelbar durchführbarer Operationen beschreiben, was mit den bisher verwendeten Zustandsübergangsdiagrammen nicht möglich ist.

Summary

The process concept evolved in the area of operating systems and is there used as the primary structural unit of the dynamic behaviour. This concept was also incorporated in programming languages, especially the real-time languages for which it is an essential element. The incorporation of the process concept in programming languages requires a precise definition of the process attributes and of the semantics of operations on processes. In this paper the concept of process operation is formally defined. To describe the semantics of such operations formally general process states are defined as sequences of operations. Using this approach some phenomena not describable by the usually used state transition diagrams (reaction on the initiating process, queuing of operations not immediately executable) can be described.

* 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.

1. EINLEITUNG

Das Konzept des Rechenprozesses (im folgenden einfach "Prozeß" genannt) wurde im Bereich der Betriebssysteme entwickelt [Win 79: 11] und führte dort zu einer besseren Strukturierung und klareren Schnittstellen zwischen Betriebssystem und Anwenderprogramm sowie Teilen des Betriebssystems selbst.

Um dem Benutzer (Programmierer) die Möglichkeit zu geben, Nebenläufigkeiten, die in dem seinem Programm zugrunde liegenden Algorithmus vorhanden sind, explizit zu spezifizieren, damit sie von der Maschine (z.B.: Apparatur + Betriebssystem + Compiler) eventuell ausgenutzt werden können, wurde das Prozeßkonzept auch in Programmiersprachen aufgenommen : z.B. PL/1 [IBM 72], Burroughs Extended Algol [Bur 74] oder Ada [Ada 79]. Für die in den letzten Jahren entwickelten Echtzeitsprachen ist das Prozeßkonzept ein essentieller Bestandteil. Beispiele für solche Sprachen sind PEARL [PDV 77a; DIN 66253], Prozeß-Fortran [ABC 79; VDI 78] und Real-Time Basic [Ind 79].

Stellt man dem Benutzer Objekte der Art "Prozeß" zur Verfügung, dann müssen diese in ihren statischen und dynamischen Eigenschaften vollständiger definiert werden als bei der Verwendung innerhalb des Betriebssystems [Win 79: 6,7; Win 79a], da aus einem Benutzerprogramm heraus eine beliebige Folge von Operationen zur Manipulation von Prozessen verlangt werden kann.

Im Bereich der Betriebssysteme wird die Semantik von Operationen an Prozessen meist mit Hilfe von Zustandsübergangsdiagrammen definiert [Han 72: 102; Sal 66: 63; Wet 78: 25]. Diese Diagramme sind in der Regel unvollständig in dem Sinne, als nicht für alle Paare (Zustand, Operation) deren Semantik definiert ist, sondern nur für bestimmte Paare, die innerhalb des Betriebssystems auftreten. Verglichen mit der freien Verfügbarkeit, welche beim Vorhandensein in einer Programmiersprache vorliegt, wird das Prozeßkonzept innerhalb der Betriebssysteme in einer disziplinierten Art und Weise verwendet. Dies hat seinen Grund darin, daß eine Programmiersprache innerhalb ihres Anwendungsgebietes die Formulierung von Programmen für unterschiedliche Probleme erlauben soll; ein Betriebssystem dagegen stellt ein Programm für ein spezielles Problem dar.

Bei der Übernahme des Prozeßkonzeptes in Programmiersprachen ging man bei der Definition der Semantik von Operationen an solchen Objekten oft so vor, daß man sich eine Menge von Operationen vorgab (z.B. ACTIVATE, SUSPEND, CONTINUE und TERMINATE in PEARL), und die Semantik dieser Operationen mit Hilfe der in den oben erwähnten Zustandsübergangsdiagrammen enthaltenen Zuständen zu definieren versuchte. Dabei traten Schwierigkeiten auf, die ihren Grund in der oben erwähnten Unvollständigkeit und disziplinierten Verwendung haben.

In der vorliegenden Arbeit wird der Begriff der Prozeßoperation formal definiert. Dabei wird unter einer Prozeßoperation eine Operation verstanden, die auf einen Prozeß als Ganzes ausgeübt wird (z.B. Starten, Anhalten, Beenden). Zur Definition der Semantik solcher Prozeßoperationen werden

systematisch aufgebaute und algorithmisch manipulierbare Zustände definiert, die im wesentlichen aus Operationsfolgen bestehen, und die eine beliebig feine Unterscheidung zweier Zustände bezüglich der zugrundeliegenden Operationenmenge erlauben.

Mit diesem Ansatz lassen sich eine Reihe von Phänomenen formal beschreiben, die bisher bei der Definition von Prozeßoperationen Schwierigkeiten bereitet haben. Ein Beispiel dafür stellen die gestauten ("buffered") Aktivierungen von Tasks in PEARL dar. Außerdem wird gezeigt, daß sich damit Prozeß- und Synchronisationsoperationen sowie Operationen an nichtprivaten Datenobjekten einheitlich darstellen lassen.

2. FORMALE DEFINITION DER PROZESSOPERATION

Im folgenden wird von der in [Win 79: 17] gegebenen, informalen Prozeßdefinition ausgegangen :

> 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. (2-1)

Seien PM eine endliche Menge von Prozessen und ZM eine endliche, nichtleere Menge von Zuständen und PM n ZM = ∅. Ein solches Paar (PM,ZM) wird als Prozeßsystem bezeichnet.

Eine Abbildung :

$$SZ : PM \rightarrow ZM \qquad (2\text{-}2)$$

ordnet jedem Prozeß p ∈ PM einen Zustand z ∈ ZM zu und kann als Systemzustand bezeichnet werden. Die Menge der Systemzustände (SZM) eines Prozeßsystems (PM,ZM) ist dann gegeben durch (2-3)

$$SZM((PM,ZM)) = Abb(PM, ZM) \qquad (2\text{-}3)$$

wobei Abb(A,B) für die Menge aller Abbildungen von A in B steht. Eine Prozeßoperation in einem Prozeßsystem (PM,ZM) ist eine Abbildung der Art (2-4).

```
        PM * Pm(PM) * SZM -> SZM
             (p,Z,x) |-> y
```

mit : Pm(A) ist die Potenzmenge von A minus {Ø} , (2-4)
p ist der Auftraggeber,
Z ist die Menge der Zielprozesse,
x ist der Vorzustand ,
y ist der Nachzustand und
SZM steht für SZM((PM,ZM)).

Die Menge aller Prozeßoperationen in einem Prozeßsystem (PM,ZM) wird mit POM((PM,ZM)) bezeichnet.

Während bei Verwendung von Zustandübergangsdiagrammen stets nur der Zustand eines Prozesses dargestellt wird, und daher auch nur die Änderung des Zustandes eines Prozesses

ausgedrückt werden kann, erlaubt die Definition (2-4) wesentlich allgemeinere Zustandsänderungen; im allgemeinsten Falle kann (2-5) gelten.

$$\forall p \in PM \; [\; x(p) \neq t(pl,Z,x)(p) \;] \; . \qquad (2\text{-}5)$$

Dann haben alle Prozesse des Prozeßsystems durch die Operation t in der betreffenden Situation eine Zustandsänderung erfahren. Je nachdem, welche Prozesse eine Zustandänderung erfahren, kann man folgende Wirkungen unterscheiden [Win 79: 60,61]:

Hauptwirkung : $p \in Z \cup x(p) \neq t(pl,Z,x)(p)$
Rückwirkung : $x(pl) \neq t(pl,Z,x)(pl)$
Nebenwirkung : $p \in PM-(Z \cup \{pl\}) \cup x(p) \neq t(pl,Z,x)(p)$.

Operationen nach (2-4) werden in [Win 79a] als Operationen zur direkten Prozeßsteuerung bezeichnet, da die Prozesse als Zielobjekte der Operationen auftreten.

3. DEFINITION KONKRETER PROZESSOPERATIONEN

Im folgenden werden sechs konkrete Operationen zur direkten Steuerung von (Rechen-) Prozessen betrachtet und eine Methode zur Definition der Semantik dieser Operationen entwickelt. Die sechs Operationen sind : Erzeugen, Starten, Anhalten, Fortsetzen, Beenden und Vernichten. Diese Operationen können in verschiedenen Varianten definiert werden. Es stellt sich nun die Frage, welche Zustände Prozesse annehmen können sollen, auf die diese sechs Operationen ausgeübt werden, wobei inbesondere für jedes Paar (Zustand, Operation) der Folgezustand definiert sein soll.

Sei nun OB die in (3-1) definierte Menge.

$$OB = \{E, S, A, F, B, V, A2, F2\} \qquad (3\text{-}1)$$

Als Prozeßzustände werden nun Worte aus der Menge $(a,e)OB^*$ betrachtet, d.h. im wesentlichen Folgen von Operationen. Durch "a" wird der Anfangszustand bezeichnet, in welchem sich alle Prozesse befinden, auf die noch nie eine Prozeßoperation ausgeübt wurde. Der Endzustand "e" ≠ "a" ist notwendig, um der in Definition (2-1) geforderten Einmaligkeit Rechnung zu tragen. Befindet sich ein Prozeß in diesem Zustand, so kann sich sein Zustand niemals mehr ändern. Dies wird als Axiom des Endzustandes bezeichnet (3-2).

Axiom des Endzustandes:

$$\forall t \in POM\ \forall p1,p2 \in PM\ \forall Z \in Pm(PM)\ \forall x \in SZM \qquad (3\text{-}2)$$
$$[\ x(p1) = \underline{e} \Rightarrow t(p2,Z,x)(p1) = \underline{e} = x(p1)\ ].$$

Innerhalb von Formeln werden die Prozeßzustände stets unterstrichen, um sie von den restlichen Zeichen zur Formulierung der Formeln zu unterscheiden. Im laufenden Text werden die Zustände in Doppelapostrophe eingeschlossen.

Da bei einer Realisierung von Prozeßoperationen nur endlich viele verschiedene Prozeßzustände auftreten können, muß in der Menge $(a,e)OB^*$ eine Äquivalenzrelation definiert werden. Durch das Axiom (3-2) werden viele Zustände als äquivalent erklärt, nämlich alle $u\underline{e}$ äq $v\underline{e}$, mit $u,v \in OB^*$, wenn u einen Prozeß von "a" in "e" überführt und v beliebig ist.

Ein Beispiel für die Abwicklung eines Prozesses mittels der oben erwähnten sechs Operationen und die dabei auftretenden Zustände ist in (3-3) gegeben.

$$\underline{a} \xrightarrow{E} \underline{aE} \xrightarrow{S} \underline{aES} \xrightarrow{A} \underline{aESA} \xrightarrow{F} \underline{aES} \xrightarrow{B} \underline{eE} \xrightarrow{V} \underline{e} \qquad (3\text{-}3)$$

Dabei tritt eine weitere Äquivalenz auf: $\underline{aESAF}$ äq $\underline{aES}$.

In (3-3) ist der "normale" Ablauf eines Prozesses dargestellt. Berücksichtigt man die oben aufgestellte Forderung nach Vollständigkeit, dann muß z.B. auch für die in (3-4) dargestellte Situation ein Folgezustand festgelegt werden.

$$\underline{aE} \xrightarrow{A} ? \qquad (3\text{-}4)$$

In (3-4) soll auf einen Prozeß, der erzeugt aber noch nicht gestartet worden ist, die Operation "Anhalten" ausgeübt werden. Dies ist nicht unmittelbar durchführbar, da von Anhalten sinnvollerweise erst nach dem Starten gesprochen werden kann. Es gibt nun mehrere Möglichkeiten, einen Folgezustand für die Situation (3-4) zu definieren (3-5).

$$\underline{aE} \xrightarrow{A} \begin{cases} \underline{aE} & : \text{Ignorieren} \\ \underline{aEa} & : \text{Stauen ohne Auftraggeber} \\ \underline{aEa\{p\}} & : \text{Stauen mit Auftraggeber} \end{cases} \qquad (3\text{-}5)$$

Das Stauen einer nicht unmittelbar durchführbaren Operation ist dann sinnvoll, wenn in der Zukunft noch eine Situation eintreten kann, in welcher die gestaute Operation wirken kann. Dies ist für den Fall (3-4) durch die Operation "Starten" möglich.

Wenn eine Operation nicht unmittelbar durchführbar ist, dann kann auch eine Rückwirkung auf den Auftraggeber erfolgen; er kann z.B. beendet werden oder solange angehalten werden, bis die gestaute Operation durchgeführt werden kann.

Um alle im Zusammenhang mit Stauungen auftretenden Phänomene modellieren zu können, wird im folgenden die in (3-6) angegebene Menge von Zuständen zugrunde gelegt, wobei in konkreten Fällen stets eine endliche Teilmenge davon als Zustandsmenge auftritt.

$$ZM : \underbrace{(a,e)OB^*}_{\text{Hauptzustand}} \underbrace{(ob(eps,\{PM\}))^*}_{\text{Stauzustand}} \qquad (3\text{-}6)$$

ob = {e, s, a, f, b, v},
PM = {p1, p2,...,pn},
eps steht für das leere Wort.

Insgesamt gibt es die im folgenden aufgeführten Möglichkeiten der Wirkung auf Auftraggeber bzw. Zielprozeß.

Wirkung auf den Auftraggeber :

a) eps : keine Zustandsänderung.

b) Am : anhalten; dies kann verwendet werden, wenn eine Operation nicht unmittelbar durchführbar ist und daher beim Zielprozeß gestaut werden soll.

c) Bm : beenden.

d) BmVm : beenden und vernichten.

Wirkung auf den Zielprozeß :

a) eps : keine Zustandsänderung.

b) W : wirken; der Hauptzustand wird verändert, der Stauzustand bleibt gleich.

c) We : wirken und entstauen; der Haupzustand wird verändert, der Stauzustand wird reduziert.

d) So : stauen ohne Auftraggeber; der Hauptzustand bleibt gleich, der Stauzustand wird verlängert.

e) Sm : stauen mit Auftraggeber; der Hauptzustand bleibt gleich, der Stauzustand wird verlängert.

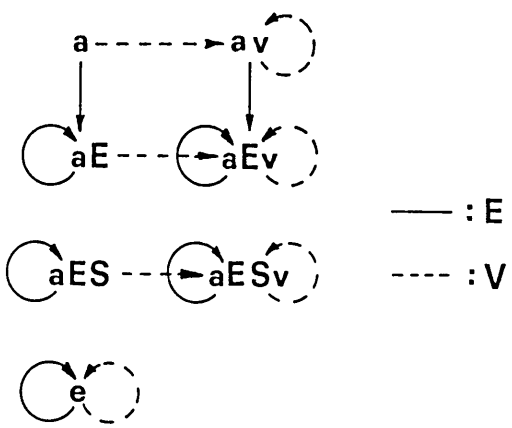
f) Va : virtuell abarbeiten: der Haupzustand bleibt gleich, der Stauzustand wird reduziert.

Von den 24 rein theoretisch möglichen Kombinationen treten bei der Definition von Prozeßoperationen zehn auf [Win 79 : 129].

Die Anwendung der oben definierten Zustände und Wirkungsarten soll am Beispiel der beiden Operationen E und V dargestellt werden. Dabei wird die Zustandsmenge (3-7) zugrunde gelegt.

$$ZM = \{\underline{a}, \underline{av}, \underline{aE}, \underline{aEv}, \underline{aES}, \underline{aESv}, \underline{e}\} \qquad (3\text{-}7)$$

Die Wirkung auf die Zielprozesse wird in dem Diagramm (3-8) angegeben.



(3-8)

Diagramme der Art (3-8) sind sehr anschaulich, erlauben aber nicht die Darstellung aller Zustandsänderungen. Beim Übergang $\underline{a}$ -{V}->$\underline{av}$ kann z.B. nicht zum Ausdruck gebracht werden, daß der betreffende Auftraggeber angehalten werden soll.

In (3-9) und (3-1Ø) sind die beiden Operationen als mathematische Funktionen vom Typ PM*PM*SZM -> SZM definiert. Dabei bedeuten Hz(x,p) den Hauptzustand von x(p), Sz(x,p) den Stauzustand von x(p) und Subs(x,p,z) = (x - {(p,x(p))}) u {(p,z)}. Die Fallunterscheidung ist in Anlehnung an die von McCarthy eingeführte Notation [Mc 6Ø: 184] formuliert.

```
E(p1, p2, x) =
  [Hz(x,p1) = aES  =>
     [Hz(x,p2) = a => Subs(x,p2,aESz(x,p2));                    (3-9)
      Hz(x,p2) ≠ a => x ];
   Hz(x,p1) ≠ aES => x]      ∎
```

```
V(p1, p2, x) =
  [Hz(x,p1) = aES  =>
     [x(p2) ∈ {a, aE, aES} =>                                  (3-10)
          Subs(x,p2,x(p2)v);
      x(p2) ∉ {a, aE, aES} => x ];
   Hz(x,p1) ≠ aES => x]      ∎
```

In (3-9) und (3-10) findet noch keine Rückwirkung auf den Auftraggeber statt, es wird aber zum Ausdruck gebracht, daß sich ein Auftraggeber in einem aus einer Menge von bestimmten Zuständen befinden muß, wenn er eine Operation initiieren will (Axiom des laufenden Auftraggebers [Win 79: 155]). In (3-11) ist eine Variante von V angegeben, bei welcher ein Auftraggeber beim Stauen angehalten wird.

```
V(p1, p2, x) =
  [Hz(x,p1) = aES  =>
     [x(p2) ∈ {a, aE, aES} =>                                  (3-11)
          Subs(A2(p1,x), p2, A2(p1,x)(p2)v{p1});
      x(p2) ∉ {a, aE, aES} => x ];
   Hz(x,p1) ≠ aES => x ]     ∎
```

```
A2(p, x) =
  [Hz(x,p) = aES => Subs(x,p,aESA2Sz(x,p));                    (3-12)
   Hz(x,p) ≠ aES => x ]      ∎
```

In (3-14) ist eine Variante der Operation "Erzeugen" angegeben, bei welcher auch entstaut wird. Dabei wird die in (3-13) gegebene Zustandsmenge zugrunde gelegt, die sich von (3-7) dadurch unterscheidet, daß im Hauptzustand "a" auch die Operation "Starten" gestaut sein kann. Diese wird dann bei Ausübung von "Erzeugen" entstaut, wodurch ein direkter Übergang vom Hauptzustand "a" in "aES" erfolgt.

```
    {a, as, av, asv, aE, aEv, aES, aESv, e}                    (3-13)
```

```
E(p1, p2, x) =
  [Hz(x,p1) = aES  =>
     [Hz(x,p2) = a  =>                                         (3-14)
        [s in Sz(x,p2) => Subs(x,p2,aESSz(x,p2)-s);
         ¬(s in Sz(x,p2)) => Subs(x,p2,aESz(x,p2)) ];
      Hz(x,p2) ≠ a  =>  x ];
   Hz(x,p1) ≠ aES  =>  x ]      ∎
```

4. EINORDNUNG DER SYNCHRONISATIONSOPERATIONEN

Sei DM eine endliche Menge von nichtprivaten Datenobjekten, d.h. solchen Datenobjekten, die jeweils von mindestens zwei Prozessen aus referierbar sind. Im folgenden sollen alle p $\in$ PM alle d $\in$ DM referieren können. Sei DWM eine endliche, nichtleere Menge von Werten, die ein Datenobjekt annehmen kann.

Die Menge SZM der Systemzustände in einem System von Prozessen und nichtprivaten Datenobjekten ist nach (4-1) definiert.

$$SZM = \{a \cup b \mid a \in Abb(PM,ZM),\ b \in Abb(DM,DWM)\} \qquad (4\text{-}1)$$

Je nachdem welche Objekte als Zielobjekte auftreten, lassen sich die zwei in (4-2) und (4-3) angegebenen Typen von Operationen definieren.

$$PM * Pm(PM) * SZM \rightarrow SZM \qquad (4\text{-}2)$$
$$PM * Pm(DM) * SZM \rightarrow SZM \qquad (4\text{-}3)$$

Inhaltlich lassen sich die folgenden drei Arten von Operationen unterscheiden :

a) eine Prozeßoperation tp ist eine Operation vom Typ (4-2), durch welche höchstens die Zustände von Prozessen verändert werden :

$$\forall d,p,Z,x\ [\ tp(p,Z,x)(d) = x(d)\ ]. \qquad (4\text{-}4)$$

b) eine reine Datenoperation td ist eine Operation vom Typ (4-3), durch welche höchstens die Zustände von Datenobjekten verändert werden :

$$\forall p1,p2,Z,x\ [\ td(p1,Z,x)(p2) = x(p2)\ ]. \qquad (4\text{-}5)$$

c) eine Synchronisationsoperation ts ist eine Operation vom Typ (4-3), durch welche sowohl die Zustände von Prozessen als auch die von Datenobjekten verändert werden.

5. ZUSAMMENFASSUNG

Der Begriff der Prozeßoperation wird eingeführt und formal definiert in Form einer Abbildung. Um die Semantik solcher Prozeßoperationen präzise definieren zu können werden systematisch aufgebaute und algorithmisch manipulierbare Zustandsbezeichnungen eingeführt. Damit gelingt es, bei der Beschreibung der Semantik von Prozeßoperationen auch solche Phänomene zu erfassen, deren präzise Beschreibung bisher Schwierigkeiten bereitet hat. Als Beispiel dafür wird das Stauen einer nicht unmittelbar durchführbaren Operation behandelt. Abschließend wird eine einheitliche Darstellung für Prozeß-, reine Daten- und Synchronisationsoperationen gegeben.

6. LITERATUR

ABC 79 Arthur,A.; Bearden,F.E.; Caro,R.M. et al.
Industrial Real-Time FORTRAN.
Proposal of the Purdue Europe Technical
Committee 1 on Industrial Real-Time FORTRAN,
PE TC1, 2/79, March 1979.

Ada 79 Preliminary Ada Reference Manual.
SIGPLAN Notices 14,6(1979) Part A.

Ada 80 Reference Manual for the Ada Programming
Language. US DoD, July 1980.

Bur 74 Burroughs Corporation
B6700/B7700 ALGOL Language Reference Manual.
No. 5000649, 20 May 1974.

Han 72 Hansen, Per Brinch
Short-Term Scheduling in Multiprogramming Systems.
Operating Systems Review 6,1-2(1972) 101..105.

IBM 72 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.

Ind 79 Industrial Real-Time Basic.
European Workshop on Industrial Computer
Systems, TC2 on Industrial Real-Time Basic,
Report IRTB-E 79/1, 9-Apr-79.

Mc 60 McCarthy,John
Recursive Functions of Symbolic Expressions and
their Computation by Machine, Part I.
CACM 3,4 (1960) 184..195.

PDV 77a PDV: Projekt Prozeßlenkung mit DV-Anlagen
Full PEARL Language Description.
Ges. f. Kernforschung Karlsruhe, KFK-PDV 130, 1977.

Sal 66 Saltzer,J.H.
Traffic Control in a Multiplexed Computer System.
MAC-TR-30 (PhD-Thesis), MIT, Cambridge Mass., 1966.

VDI 78 VDI/VDE 3556
Prozeß-Fortran 75.
VDI/VDE Richtlinie, Entwurf März 1978.

Wet 78 Wettstein, Horst
Aufbau und Struktur von Betriebssystemen.
Carl Hanser, München usw. 1978.

Win 79 Winkler, J.F.H.
Eine Theorie der Prozeßoperationen.
Dissertation, Universität Karlsruhe, Juli 1979.

Win 79a Winkler, J.F.H.
Das Prozeßkonzept in Betriebssystemen und Programmiersprachen.
Informatik Spektrum 2,4(1979) 219..229,
3,1(1980) 31..40.

Dr.J.F.H. Winkler
Siemens AG D AP GE 4
Otto-Hahn-Ring 6
D-8000 München 83

Implementierungssprachen für nichtsequentielle Programmsysteme

Herausgeben von

Dr. rer. nat. Jürgen Nehmer
Professor an der Universität Kaiserslautern

B. G. Teubner Stuttgart 1981

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Implementierungssprachen für nichtsequentielle Programmsysteme / hrsg. von Jürgen Nehmer. - Stuttgart : Teubner, 1981.
(Berichte des German Chapter of the ACM ; Bd. 7)
ISBN 3-519-02426-8

NE: Nehmer, Jürgen (Hrsg.); Association for Computing Machinery / German Chapter: Berichte des German ...

Printed in Germany
Gesamtherstellung: J. Illig, Göppingen
Umschlaggestaltung: W. Koch, Sindelfingen