Entwurf, Beschreibung und Implementation von Systemen mit Hilfe der nebenläufigen Programmiersprache CAP

Franz J. Rammig
Universität Dortmund
Abt. Informatik I

Kurzfassung:

Es wird eine Sprache vorgestellt, die aufbauend auf einem einheitlichen Konzept (zeitbewertete interpretierte Petri Netze) Beschreibungen auf sehr unterschiedlichen Abstraktionsebenen erlaubt. Diese Abstraktionsebenen reichen von der Ebene der Verschaltung von integrierten Bausteinen hin bis zu Warteschlangenmodellen. Auf die Möglichkeit, nebenläufige Prozesse beschreiben zu können, wurde besonders Wert gelegt (CAP steht für Concurrent Algorithmic Programming Language) /RA1/.

In diesem Beitrag wird das Grundkonzept der Sprache, insbesondere die CAP-Netze, erläutert und spezielle Spracheigenschaften von CAP anhand von fünf kleinen Beispielprogrammen vorgestellt. Diese Beispielprogramme sind:
- *auf der IC-Ebene: Beschreibung eines "8-bit-ripple-counter"*
- *auf der RT-Ebene: Beschreibung eines synchronen sequentiellen Komplementierers*
- *auf der RT-Ebene: Beschreibung eines asynchronen sequentiellen Komplementierers*
- *auf der BS-Ebene: Beschreibung eines Resourcenverwalters*
- *auf der Systeminteraktionsebene: Beschreibung einer Tankstelle.*

Abschließend wird die bisher implementierte CAP-Software (Compiler, Interpreter/Simulator, Debugger, Dokumentationsgenerator, optimierender Cross Code Generator, Maschinenmodellgenerator) kurz vorgestellt.

1. Grundkonzept

Das CAP-Grundkonzept läßt sich als alphanumerisches Äquivalent zu zeitbewerteten interpretierten Petri Netzen charakterisieren, wobei die Interpretation (= Datenmanipulation) in PL/1-artigen Sprachkonstrukten ausgedrückt wird.

Für die Beschreibung des Petri Netzes (Kontrollstruktur der Sprache) werden Label zur Identifikation von Stellen benutzt, während "ON CONDITIONS" auf Labeln zur Beschreibung von Transitionen dienen. Formal erhält man dadurch Sprachkonstrukte, die nicht unähnlich zu Dijkstras "Guarded Commands" /DI1/ sind. Z.B.:

ON(A, B) : C : D : F := G+H; entspricht einer Transition mit Eingangsstellen A und B, Ausgangsstellen C und D sowie der Interpretation F := G+H. In diesem Beispiel ist die Transition nicht zeitbewertet.

Dies könnte beispielsweise dadurch geschehen, daß das Terminatorsymbol ";" ersetzt wird durch "DELAY(20).

In Anlehnung an LOGOS /RO1/ gibt es in CAP-Netzen /RA2/ neben der üblichen Petri Netz Transition auch Oder-Transitionen, Entscheidungs-Transitionen und Transitionen zur Blockschachtelung. Weiterhin können Stellen mit einer individuellen Kapazität versehen werden, wobei es Bestandteil der Schaltregel jeder Transition ist, daß nur dann geschaltet werden kann, wenn alle Ausgangsstellen der Transition freie Kapazität haben. (Vergleiche die Einteilung in "Integer Places" und "Boolean Places" in /YB1/.)

Von zentraler Bedeutung ist nun die Möglichkeit, in CAP das steuernde Petri Netz nicht explizit aufzuschreiben, sondern durch Angabe von "Makrokonstrukten" implizit. Man erhält dadurch strukturierte nebenläufige Programme wobei auch die Vorgehensweise der schrittweisen Verfeinerung unterstützt wird. Diese Makrokonstrukte werden in Anlehnung an PL/1 als DO-Gruppen geschrieben. So bedeutet ein Konstrukt:
DO CONCURRENT I := 1 TO 5 BY 1 SEQUENTIAL;.....END;
daß die Anweisungen in der Gruppe pro Indexwert nebenläufig zueinander ausgeführt werden sollen, die verschiedenen Indizes dagegen sequentiell angewendet werden sollen. Wichtig in diesem Zusammenhang ist die Unterscheidung zwischen DO CONCURRENT und DO PARALLEL. Ersteres bedeutet Nebenläufgkeit (nicht synchronisierte Parallelität), letzteres synchronisierte Parallelität (Gleichzeitigkeit).

2. Eine kurze Einführung in CAP-Netze

2.1 Def.:

APN := (S,T,F) heißt Petri Netz Graph :<=>
S endliche Menge von Stellen
T endliche Menge von Transitionen
$F \subset S \times T \cup T \times S$, $S \cap T = \emptyset$
$\forall x \in S \cup T : \exists y \in S \cup T : ((x,y) \in F \vee (y,x) \in F)$

Wie üblich bezeichnen wir mit $^\bullet t$ die Menge der Eingangsstellen der Transition t, mit $t^\bullet$ die Menge ihrer Ausgangsstellen. Eingangs- und Ausgangsstellen sind geordnet, die Position einer Stelle s bzgl. einer Transition t wird mit $id._t(s)$ bzw. $id_t.(s)$ angegeben. Stellen können Marken enthalten, wobei die Kapazität von Stellen beschränkt sein kann:

cap : S -> $\mathbb{N} \cup \{\infty\}$ (Kapazitätsvereilung)
m : S -> $\mathbb{N} \cup \{o\}$ (totale Markierung)

Mit M wird die Menge aller Markierungen eines gegebenen Petri Netzes bezeichnet.

2.2 Def.: (AND-Transition)

A⊂T. Für alle a ∈ A gilt:
a aktiviert unter einer Markierung m :<=>
$(\forall s \in {}^\bullet a : m(s) > o) \wedge (\forall s \in a^\bullet : m(s) < cap(s))$.
f_a: M -> M, $f_a(m) = m'$ heißt Schalten von a :<=>
a ist aktiviert und $\forall s \in {}^\bullet a : m'(s) = m(s)-1$
$\forall s \in a^\bullet : m'(s) = m(s)+1$
$m'(s) = m(s)$ sonst
Schreibweise im CAP: on(&(e_1,..., e_n)) : a_1:...: a_m
oder einfach: on (e_1,..., e_n)) : a_1;...: a_m

2.3 Def.: (OR-Transition)

O⊂T. Für alle o∈O gilt:
o ist aktiviert unter einer Markierung m :<=>
$(\exists s \in {}^\bullet o : m(s) > o) \wedge (\forall s \in o^\bullet : m(s) < cap(s))$.
f_o: M -> M, $f_o(m) = m'$ heißt Schalten von o :<=>
o ist aktiviert und
$m'(s) = m(s)-1 :<=> s \in {}^\bullet o \wedge id._o(s) = \min\{id._o(s) \mid m(s) > o\}$
$m'(s) = m(s)+1 :<=> s \in o^\bullet$.
Schreibweise in CAP: on(|(e_1,..., e_n)) : a_1:...: a_m

2.4 Def.: (IF-Transition)

D⊂T. Für alle d∈ D gilt:
$|{}^\bullet d| = 1$, $|d^\bullet| = 2$, es gibt ein Datenprädikat P_d, d ist aktiviert unter einer Markierung m :<=>
$(\forall s \in {}^\bullet d : m(s) > o) \wedge (\forall s \in d^\bullet : m(s) < cap(s))$.
f_d: M -> M, $f_d(m) = m'$ heißt Schalten von d :<=>
d ist aktiviert und
$m'(s) = m(s)-1 :<=> s \in {}^\bullet d$
$m'(s) = m(s)+1 :<=> \begin{cases} id_{d^\bullet}(s) = o \wedge P_d = \text{"true"} \\ id_{d^\bullet}(s) = 1 \wedge P_d = \text{"false"} \end{cases}$

Schreibweise in CAP: on(e_1) : if(P_d) then a_0 :;
else a_1 :;

Zwei spezielle AND-Transitionen dienen zum Aufruf von Unternetzen bzw. zur Synchronisation mit deren Fertigsignalen:

C-Transition, in CAP: on(e_1) : a_1 : call a_2;
Man beachte, daß die rufende Instanz über a_1 aktiv bleibt.

R-Transition, in CAP: on(← e_2, e_1) : a_1 :; .

2.5 Def.: (BLKHEAD-Transition)

$H \subset T$. Für alle $h \in H$ gilt:
$|h^\bullet| > 1$, $|{}^\bullet h| = |h'|$, h ist aktiviert unter m :<=>
$\forall s \in {}^\bullet h$, $id_{\cdot h}(s) = o$: $m(s) > o$;
$\exists s \in {}^\bullet h$, $id_{\cdot h}(s) \neq o$: $m(s) > o$;
$\forall s \in h^\bullet$: $m(s) < cap(s)$.
$f_h : M \to M$, $f_h(m) = m'$ heißt Schalten von h :<=>
h ist aktiviert und

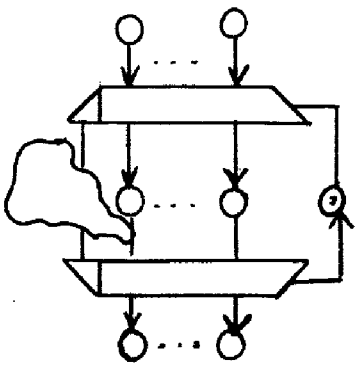
$$m'(s) = m(s)-1 :\Leftrightarrow \quad s \in {}^\bullet h \wedge id_{\cdot h}(s) = o \;\vee\; s \in {}^\bullet h \wedge id_{\cdot h}(s) = \min\{id_{\cdot h}(s) \neq o \mid m(s) > o\}$$

$$m'(s) = m(s)+1 :\Leftrightarrow \quad s \in h^\bullet \;\; id_{h^\bullet}(s) = o \;\vee\; s \in h^\bullet \;\; id_{h^\bullet}(s) = \min\{id_{\cdot h}(s) \neq o \mid m(s) > o\}$$

Schreibweise in CAP: on call(e_1,..., e_n) : a_1: procedure

Als Blockend-Transition wird eine symmetrische Transition mit gleichen Schaltverhalten benutzt.
Schreibweise in CAP: on(e_1) : end

Blockhead- und Blockend-Transitionen kommen nur paarweise als Gebilde folgender Form vor:



Oben sieht man die "Aufrufstellen" e_1 bis e_n, die nach Prioritätenregeln bedient werden. Der bediente Aufruf wird durch Markieren einer korespondierenden Stelle festgehalten, sodaß eine Rückmeldung an die richtige Instanz möglich ist. Die "Rückmeldungs"-Stelle (nach rechts versetzt) steuert die Aktivierbarkeit des Unternetzes (Block).

CAP-Netze sind nun Petri-Netze bestehend aus Transitionen der eben eingeführten Typen. Hinzu kommt eine Vielzahl globaler Beschränkungen, insbesondere zur korrekten Blockschachtelung, auf die hier nicht eingegangen werden soll. Die Transitionstypen wurden aus denen der LOGOS-Netze /RO1/ entwickelt. Ähnliche Transitionen findet man in den Makro-E-Netzen /NO1/, der Hauptunterschied besteht darin, daß dort mit attributierten Marken, in CAP-Netzen mit attributierten Transitionen gearbeitet wird. Schränkt man sich auf AND-, OR-, IF-Transitionen ein (OR ohne Prioritätenregelung), betrachtet nur Transitionen mit Grad (2,1) bzw. (1,2) und Stellen mit unendlicher Kapazität, so erhält man die Netze von Herzog und Yoeli /HY1/. Netze, in denen Stellen mit Kapazität 1 gemischt mit solchen unendlicher Kapazität vorkommen, werden auch in /BY1/ beschrieben. Für jedes CAP-Netz, in dem nur Stellen beschränkter Kapazität vorkommen, kann man ein semantisch äquivalentes Petri-Netz konstruieren /RA2/.

3. Spezielle Spracheigenschaften

Nachfolgend werden anhand von Beispielprogrammen spezielle Spracheigenschaften vorgestellt. Es ist jedoch keinesfalls so, daß die aufgeführten Eigenschaften nur für die jeweilige Abstraktionsebene relevant seien. Sie sollen nur jeweils besonders deutlich gemacht werden.

3.1 Strukturbeschreibung auf IC-Ebene (Anhang 1)

Das Beispiel zeigt eine hierarchische Strukturbeschreibung. Dies wird dadurch möglich, daß das "LIKE"-Attribut auch auf Prozeduren und Funktionen angewandt werden darf (statisches Analogon zu SIMULA-Klassen). Im Beispiel werden dadurch zwei Inkarnationen der Beschreibung eines 4-Bit-Zählers und pro Inkarnation vier Inkarnationen der Beschreibung eines D-Flipflops erzeugt. Als weitere Besonderheiten fallen auf:

- Strukturen werden in PASCAL-Notation geschrieben
- Funktionen können auch vom Typ Struktur sein
- Es gibt eine Vielzahl von logischen Operatoren (angegeben ist die ASCII-Notation: & für und, ! für oder, / für nicht)
- Neben Variablen mit Speicherwirkung gibt es auch solche, die immer dann einen neuen Wert erhalten, wenn eine Variable, von der sie abhängen einen neuen Wert bekommt (IMPLICIT-Variable, vergleichbar mit TERMINAL in CDL /CH1/).

In der vorliegenden Beschreibung wurde von der Fiktion absoluter

Gleichzeitigkeit ausgegangen.

In den Zeilen 6 bis 13 wird das einzelne Flipflop beschrieben. Das Flipflop wird gesteuert von der negativen Flanke des Signals C. Diese wird durch den Vergleich von altem und neuem Wert identifiziert. Die Zeilen 10 und 11 enthalten die eigentlichen Flipflop-Gleichungen. Hierzu ist zu beachten, daß in einer Funktion auf sich selbst referiert werden kann, dann allerdings nicht im Sinn eines Funktionsaufrufs sondern nur im Sinn einer Referenz auf den aktuellen Wert einer gleichnamigen Variable. In der Zeile 16 werden drei weitere Kopien des Flipflops angelegt und in den Zeilen 17 bis 22 zum einem "ripple-counter" verschaltet. Damit ist ein 4-bit-Zähler definiert. In Zeile 26 wird eine Kopie dieses Zählers angelegt. Diese beiden 4-bit-Zähler werden in den Zeilen 29 und 30 zu dem gewünschten 8-bit-Zähler verschaltet.

3.2 Verhaltensbeschreibung auf RT-Ebene (Anhang 2)

Hier ist in CAP ein CDL-Beispiel von Chu /CH1/ aufgeschrieben. Daher ist auch hier keine Nebenläufigkeit vorhanden. Man sieht, daß Register-Transfer-Beschreibungen auf natürliche Art und Weise auf heute übliche Programmierkonventionen übertragen werden. Als weitere Besonderheiten fallen auf:

- FIXED-Zahlen lassen sich bzgl. Darstellungsart und Länge genau spezifizieren. So bedeutet (3,UN) bespielsweise 3-bit-Wortlänge und "unsigned integer".
- Konkatenation wird wie in PL/1 geschrieben (!! in ASCII)

Das Programm wird gesteuert von einem endlichen Automaten mit drei Zuständen, codiert in der Variable S in 1-aus-3-code. Im Zustand "001" (Zeilen 8 bis 12) wird der Folgezustand "010" bestimmt, A unter Komplementierung des rechtesten Bit rundgeshiftet und die Einzelkomplementiervorgänge gezählt. Im Zustand "010" wird entschieden, ob man fertig ist oder nicht. Der Automat wird synchron betrieben, die Implusform wird in Zeile 6 angegeben.

3.3 Asynchrone Verhaltensbeschreibung auf RT-Ebene (Anhang 3)

Hier ist im Prinzip der gleiche Komplementierer in einer asynchronen Realisierung aufgeschrieben. Das Netz ist explizit angegeben, an die Stelle des Zustandsregisters S treten die Stellen des CAP Netzes (Label).

3.4 Beschreibung von Resourcenverwaltern (Anhang 4)

Die Blockhead-Transition modelliert zusammen mit der Blockend-Transition einen Monitor /HO1/, d.h. jede CAP-Prozedur, die mehrfach aufgerufen wird, ist als Monitor aufzufassen. Im Gegensatz zum Orginalansatz wird in CAP neben der Resourcenvergabe auch die Resourcenmanipulation vom Monitor übernommen (Resource als abstrakter Datentyp).

Als weitere Besonderheiten fallen auf:
- Bei Zeichenketten ist auch der Code spezifizierbar (hier ASCII)
- E/A geschieht durch simple Verwendung von File-Variablen (wie in XPL /MK1/)

3.5 Warteschlangenmodelle (Anhang 5)

Indem man bei der Oder-Transition und bei der Blockhead-Transition neben der starren Prioritätenvergabe auch andere Vergabealgorithmen zuläßt, und außerdem Entscheidungstransitionen einführt, die analog entscheiden, erhält man die Bausteine für eine Warteschlangensimulation, ohne die für Petri Netze relevanten Analysen zu beeinträchtigen. Warteschlangen endlicher Kapazität lassen sich elegant mittels Stellen endlicher Kapazität modellieren.

CAP erlaubt zudem, der Kontrollstruktur eine nebenläufige Unterbrechungsstruktur zu überlagern (zur Problematik von Unterbrechungsmechanismen in nebenläufigen Systemen siehe /GR1/). In der Simulation werden Unterbrechungssignale von programmierbaren Zufallsgeneratoren dargestellt.

4. Implementierte und in Kürze fertiggestellte CAP-Software

Implementiert sind:
- CAP-Compiler CAPCOM. Zielsprache ist die Zwischensprache CAPID
- CAP-Interpreter CAPSIM. Interpretiert direkt CAPID, zugleich hervorragender ereignisgesteuerter Simulator
- Dokumentationsgenerator CAPDOC. Erzeugt nebenläufige Struktogramme /NS1/ aus strukturierten CAP-Programmen, eine Netzdarstellung sonst
- Testhilfesystem CAPTEST. Erlaubt das interaktive Austesten von CAP-Programmen und von Maschinensprachprogrammen, die auf in CAP beschriebenen Maschinen laufen
- Codegeneratorfamilie CAPCCL. Erzeugt weitgehend Zielprozessorunab-

hängig optimierten Code für verschiedene Zielprozessoren.
- Binder CAPLIN. Erlaubt die externe Übersetzung von CAP-Programmen.
- Generator für Maschinenmodelle CAPVM. Erzeugt aus der Codetabelle eines Prozessors das Gerippe seiner Beschreibung in CAP.

Eine Beschreibung der CAP Software ist in /RA3/ enthalten.

Anhang 1: Beschreibung eines 16 bit Aufwärtszählers (IC-Ebene)

```
 1 function CNT8(CNT)returns struct(LOW struct((AO,A1,A2,A3) bit(1)),
 2                                  HIGH struct((A4,A5,A6,A7) bit(1));
 3
 4   function CNT4L(CNT) returns struct((BO,B1,B2,B3) bit(1));
 5
 6     function FF1(D,C) returns struct((NQ,Q) bit(1));
 7       dcl(C,D,AC,D) bit(1);
 8       do parallel;
 9          AC := C;
10          FF1.Q  := FF1.Q&(C>=AC) ! D&(C<AC);
11          FF1.NQ := \(FF1.Q&(C>=AC) ! D&(C<AC));
12       end;
13     end; /*FF1*/
14
15     dcl CNT bit(1),
16         (FF2,FF3,FF4) like FF1;
17     do  parallel delay(20);
18         CNT4L.BO := FF1.Q(\BO,CNT);
19         CNT4L.B1 := FF2.Q(\B1,BO);
20         CNT4L.B2 := FF3.Q(\B2,B2);
21         CNT4L.B3 := FF4.Q(\B3,B2);
22     end;
23   end; /*CNT4L*/
24
25   dcl CNT bit(1),
26       CNT4H like CNT4L,
27       CARRY bit(1) implicit(CNT4L.B3);
28   do  parallel;
29       CNT8.LOW  := CNT4L(CNT);
30       CNT8.HIGH := CNT4H(CARRY);
31   end;
32 end; /*CNT8*/
```

Anhang 2: Beschreibung eines synchronen seriellen Komplementierers (RT-Ebene)

```
 1 on call(COMPLEMENT): procedure;
 2    dcl A bit(5), C fixed(C,un) init(0),
 3        FINI bit(1) init("0"), S bit(-3) init("100"),
 4        P bit(1) implicit;
 5    do parallel while \FINI;
 6       P := \P delay(up 10, down 1);
 7       if P then do parallel delay(5);
 8          if substr(S,0) then do parallel;
 9             S := "010";
10             A := \substr(A,0) !! substr(A,1,4);
11             C := C+1;
12          end; else;
13          if substr(S,1) then if C=5 then S := "001";
14                                     else S := "100";
15                          else;
16          if substr(S,2) then do parallel;
17             S := "000";
18             FINI := "1";
19          end;
20       end;
21    end;
22 end;
```

Anhang 3: Beschreibung eines asynchronen seriellen Komplementrierers (RT-Ebene)

```
 1 on call(COMPLEMENT_ASYNCH): START : procedure;
 2    dcl A bit(5), C fixed(3,un) init(0),
 3       (GO_ON,START,OP1,OP2,OP1_DONE,OP2_DONE,FINI) label;
 4    on(!(GO_ON,START)): OP1 : OP2 :;
 5    on(OP1): OP1_DONE : A := \substr(A,0)!! substr(A,1,4);
 6    on(OP2): OP2_DONE : C := C+1;
 7    on(OP1_DONE,OP2_DONE): if C=5 then FINI :;
 8                                  else GO_ON :;
 9 on(FINI): end;
```

Anhang 4: Beschreibung eines Resourcenverwalters (BS-Ebene)

```
1  on call(CONSOLE_MONITOR): export(WRITE,WRITE_READ) procedure;
2     dcl file LINE char(ascii,-82);
3     dcl (CR,LF) char(ascii,1) init("(3)55","(3)52");
4
5     on call (WRITE): procedure(TEXT);
6        dcl TEXT char(ascii,-80);
7        LINE := TEXT !! CR !! LF;
8     end;
9
10    on call(WRITE_READ): procedure(OUTTEXT,INTEXT);
11       dcl (OUTTEXT,INTEXT) char (ascii,-80);
12       do sequential;
13          LINE := OUTTEXT !! CR !! LF;
14          INTEXT := LINE;
15       end;
16    end;
      /* Aufruf z.B. mit : */
      :
      do concurrent
      :
      call  CONSOLE_MONITOR.WRITE(MESSAGE);
      :
      call CONSOLE_MONITOR.WRITE_READ(MESSAGE,ANSWER);
      :
      end
```

Anhang 5: Simulationsgerippe einer Tankstelle mit 4 Zapfsäulen und zwei Tankwarts. Die Tankstelle hat eine 10 PKW fassende Einfahrt und 2 Ausfahrten, die jeweils 2 PKW fassen und auf eine gemeinsame Ausfahrt vereinigt werden. Für den Tankvorgang werden 7 Minuten, für das Ausfahren auf die Straße 5 Minuten angesetzt.

```
 1 on call (GAS_STATION): BUSY  :  procedure;
 2    dcl (Z1,Z2,Z3,Z4,W1,W2,EXIT) label(1), /*cap. = 1*/
 3        (E1,E2)                  label(2), /*cap. = 2*/
 4        WAIT_QUEUE               label(10); /*cap. = 10*/
 5    dcl (ENTER,FIVE_O_CLOCK) interrupt label;
 6
 7    on interrupt(ENTER): WAIT_QUEUE :
 8      do;.../*description of station entering*/...end
 9
10    on interrupt(FIVE_O_CLOCK): do; disable(ENTER); end;
11
12    on(WAIT_QUEUE): rdm-server(Z1,Z2,Z3,Z4):;
13    on(fifo(Z1,Z2,Z3,Z4)): rdm-server(W1,W2):;
14    on(W1): E1 : do sequential delay (7);
15                 : /* description of filling procedure */
16                 end;
17    on(W2): E2 : do sequential delay(7);
18                 : /* description of filling procedure */
19                 end;
20    on(rdm(E1,E2)): EXIT : delay(5);
21 end;
```

Literatur:

/CH1/: Y.Chu:
Introducing CDL
IEEE Computer, December 1974

/DI1/: E. W. Dijkstra:
Guarded Commands, Nondeterminacy and Formal Derivation of Programs
CACM Vol. 18, No. 8 (1975)

/GR1/: K. Gröning:
Zur Darstellung und Simulation von Interrupts in parallelen Systemen durch Petri-Netze

Diplomarbeit, Abt. Informatik, Univ. Dortmund (1979)

/HO1/: C. A. R. Hoare:
Monitors: An Operating Systems Structuring Concept
CACM Vol. 17, No. 10 (1974)

/HY1/: O. Herzog & M. Yoeli:
Control Nets for Asynchronous Systems, Part 1
Technion Haifa, Dept. of Computer Science, Techn. Report 74 (1976)

/MK1/: W. M. McKeeman et al.:
A Compiler Generator
Prentice Hall (1970)

/NO1/: J. D. Noe, G. J. Nutt:
Macro E-Nets for Representation of Parallel Systems
IEEE ToC Vol. C-22, No. 8 (1973

/NS1/: I. Nassi, B. Shneiderman:
Flowchart Techniques for Structured Programming
SIGPLAN Notices, Nor. 1976

/RA1/: R. J. Rammig:
An Introduction to the Concurrent Algorithmic Programming Language CAP, or Looking at CAP with the revised Ironman's Eyes
Forschungsbericht Nr. 80 d. Abt. Informatik d. Univ. Dortmund (1979)

/RA2/: F. J. Rammig:
Überlegungen zur Kontrollstruktur einer Computer Hardware Beschreibungssprache
Forschungsbericht Nr. 62 d. Abt. Informatik d. Univ. Dortmund, (1978)

/RA3/: F. J. Rammig:
The Concurrent Programming Language CAP and the CAP CAD-System
Ersch. in: Proceedings EUROMICRO '79 Göteborg August 1979

/RO1/: C. W. Rose:
LOGOS and the Software Engineer
FJCC (1972)

/YB1/: M. Yoeli, Z. Barzilai:
On Behavioral Description of Communication Switching Systems
Technion - Israel Institute of Technology, CS Dept.
Technical Report No. 99 (1977)