stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IN 26/83

J.A. BERGSTRA & J.J. SMEETS

FORMELE ONTWIKKELING VAN SCHERMVERLOOPSCHEMA'S EN INTERAKTIEVE PROGRAMMA'S

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, Kruislaan 413, Amsterdam, The Netherlands.

The Mathematical Centre, founded 11 February 1946, is a non-profit institution for the promotion of pure and applied mathematics and computer science. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Formele ontwikkeling van schermverloopschema's en interaktieve programma's

Formal development of screen flow diagrams and interactive programs

J.A. Bergstra & J.J. Smeets *)

ABSTRACT

Uitgaande van de bij VOLMAC gehanteerde VSP-methode geven wij een theoretisch kader voor het implementeren van conversationele en pseudo-conversationele programma strukturen.

Een formeel begrip van schermverloopschema's met operaties ⊕ en ⊘ vormt de basis van dit theoretisch kader.

KEY WORDS & PHRASES: *schermverloopschema's, conversationele programmastructuur, pseudo-conversationele programmastructuur, interactief systeem*

---

*) Volmac TT (Toptraining) B.V.

1. INLEIDING

In dit artikel willen wij een bijdrage leveren aan de methodiek van het construeren van interaktieve programma's. Uitgangspunt is de bij VOLMAC gehanteerde VSP methode [1].
De theorie, die wij uiteenzetten levert een symbolische notatie voor schermverloopdiagrammen, waarin funktietoetsen voor de volgende twee doelen gebruikt kunnen worden.

1) Een sprong maken binnen een nivo van het (meestal hiërarchisch gelaagde SV-diagram).
2) Een sprong maken naar een dieper nivo van het SV-diagram.

Er is nog een derde toepassing van funktietoetsen voor ons van belang.

3) Lokaal gebruik van funktietoetsen; in dit geval is het gebruik van een funktietoets hetzelfde als 'enter' en het geven van een standaardvulling van het scherm.

Om de analyse te vereenvoudigen zijn we er hier vanuit gegaan dat lokaal gebruik van funktietoetsen niet optreedt, temeer daar dit aspekt vrij eenvoudig valt te implementeren (al dienen zich onvermijdelijk verschillende varianten aan).

De struktuur van dit stuk is als volgt.
Allereerst formuleren we enkele technische uitgangspunten. Daarna gaan we uitgebreid in op schermverloopdiagrammen als hulpmiddel voor het noteren van interaktieve strukturen.
Vervolgens wordt uitgebreid aandacht besteed aan implementaties van de schermen in termen van een hanteerbare pseudocode.
De hoofdmoeilijkheid die hier op onze weg ligt bestaat uit het in kaart brengen van een groot aantal verschillende varianten. Een van deze varianten: pseudocode voor verwerkingsformat-gestruktureerde pseudo-conversationele programma's ligt bevredigend dicht bij de VSP oplossing van het hier besproken ontwerpprobleem en deze variant beschouwen wij als mogelijke theoretische fundering van de VSP-oplossing.

2. TECHNISCHE UITGANGSPUNTEN

Om de gedachten te bepalen volgt een zestal punten:

a) De programmastruktuur wordt direkt afgeleid uit de dialoogstruktuur. (Soortgelijke ideëen komen voor in JANSEN [2] en JANSSENS [3]).
De dialoogstruktuur wordt weergegeven met behulp van hiërarchische geordende schermverloopdiagrammen.

b) Zoals in VSP beperkingen worden opgelegd aan de voorkomende gegevensstrukturen zo worden nu beperkingen opgelegd aan de dialoogstrukturen (geen recursie!)

c) De eenvoudigste SV-diagrammen zijn flow charts met twee soorten punten: schermen en verwerkingsprocessen.
De belangrijkste extra faciliteit welke hier aan de orde komt, wordt gevormd door de funktietoetsen.

d) Funktietoetsen hebben twee gebruiksvormen (betekenismogelijkheden):
   *) intikken van een funktietoets resulteert in een sprong door een SV-diagram welke niet conform de natuurlijke besturing is.
   **) aanroep van een (interaktieve) procedure waarbij na afloop van de procedure weer hetzelfde scherm wordt getoond als voorafgaande aan de aanroep.

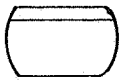
e) Wanneer een funktietoets p in een SV-diagram niet een van de beide betekenissen uit d) heeft dan wordt 'by default' de volgende betekenis gegeven: na intoetsen van p verschijnt op het scherm de mededeling dat p niet van toepassing is en dat men iets anders moet verzinnen.

f) Bij het schrijven van de pseudocode gebruiken we een ALGOL-PASCAL-achtige notatie. Hierbij geven we routines voor het gehele systeem bestaande uit monitorprogramma én gebruiker. Dat wil zeggen er is een procedure user (met een niet-deterministische body) die de gebruiker achter z'n monitor modelleert. User wordt in de monitorprogramma's als subroutine aangroepen.

3. BESCHRIJVING VAN DE SCHERMVERLOOPDIAGRAMMEN

3.1 Zoals in de inleiding al werd opgemerkt is een SV-diagram hiërarchisch geordend. Deze hiërarchie beschrijft de nivo-sprongen die samenhangen met betekenis (**) van de funktietoetsen. We beginnen met het beschrijven van SV-diagrammen met één nivo.

Deze bestaan uit een diagram (graaf) met punten en pijlen. Er zijn twee soorten punten:



Scherm. (Het bovenste deel duidt de bouwroutine voor het scherm aan).

en

Verwerkingsproces.

Er zijn ook twee soorten pijlen:

$\longrightarrow$ Natuurlijke besturing

$\xrightarrow{p}$ besturingssprong gelabeld door een funktietoets p (met p gelabelde pijl).

Zij FT een vaste (eindige) verzameling van funktietoetsen. De elementen ervan duiden we aan met p, q, r en met p_i, q_i, r_i .... Elke $p \in FT$ kan als label van een pijl van de 2e soort optreden.

Er gelden voor SV-diagrammen nu de volgende regels: (elk diagram dat aan de regels voldoet is een 'toegestaan' SV-diagram).

*) Elk scherm heeft ten hoogste één uitgaande pijl voor natuurlijke besturing en per $p \in FT$ ten hoogste één uitgaande met p gelabelde pijl.
Wanneer het scherm geen uitgaande pijlen heeft is het een eindpunt in het diagram. Op zo'n eindpunt eindigt de in het diagram beschreven dialoog.

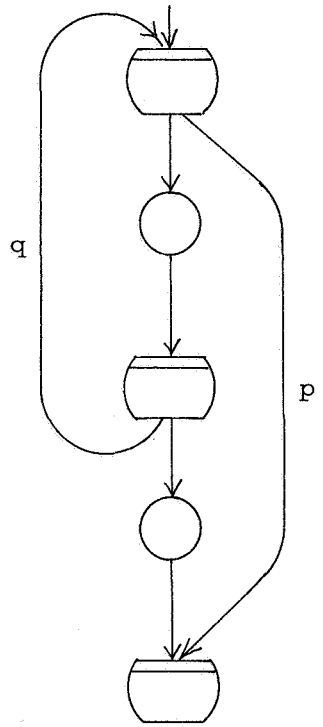
**) Elke verwerkingsproces heeft precies één ingaande pijl voor natuurlijke besturing en een aantal (≥ 1) uitgaande pijlen van deze soort.

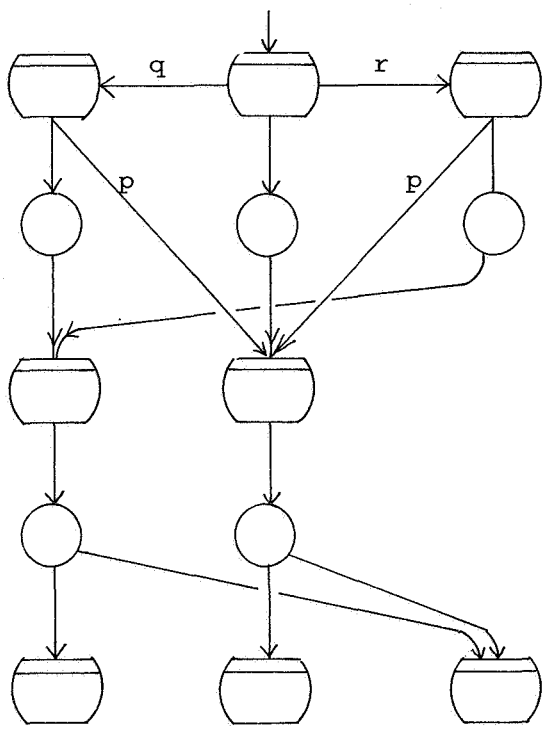
***) De pijlen voor natuurlijke besturing lopen van een scherm naar een verwerkingsproces en omgekeerd.
De gelabelde pijlen lopen altijd van scherm naar scherm.

****)Er is één inkomende pijl voor natuurlijke besturing die naar het beginscherm leidt.

Enige naamgeving is geen overbodige luxe:
We duiden de SV-diagrammen aan met D, D_0, D_1, D_2 .....
Binnen een diagram duiden we de schermen aan met S^0, ... S^k en de verwerkingen met V^0, .. , V^l.
Bij verschillende diagrammen ontstaat hier de dubbele indicering D_i met S_i, ..., $S_i^{k(i)}$ en V_i^0, ..., $V_i^{l(i)}$

3.2 Twee voorbeelden:





Hierbij moet worden opgemerkt, dat bij intikken van een funktietoets en overgang naar het volgende beeld, de door de gebruiker toegevoegde scherminhoud niet wordt verwerkt en verloren gaat.

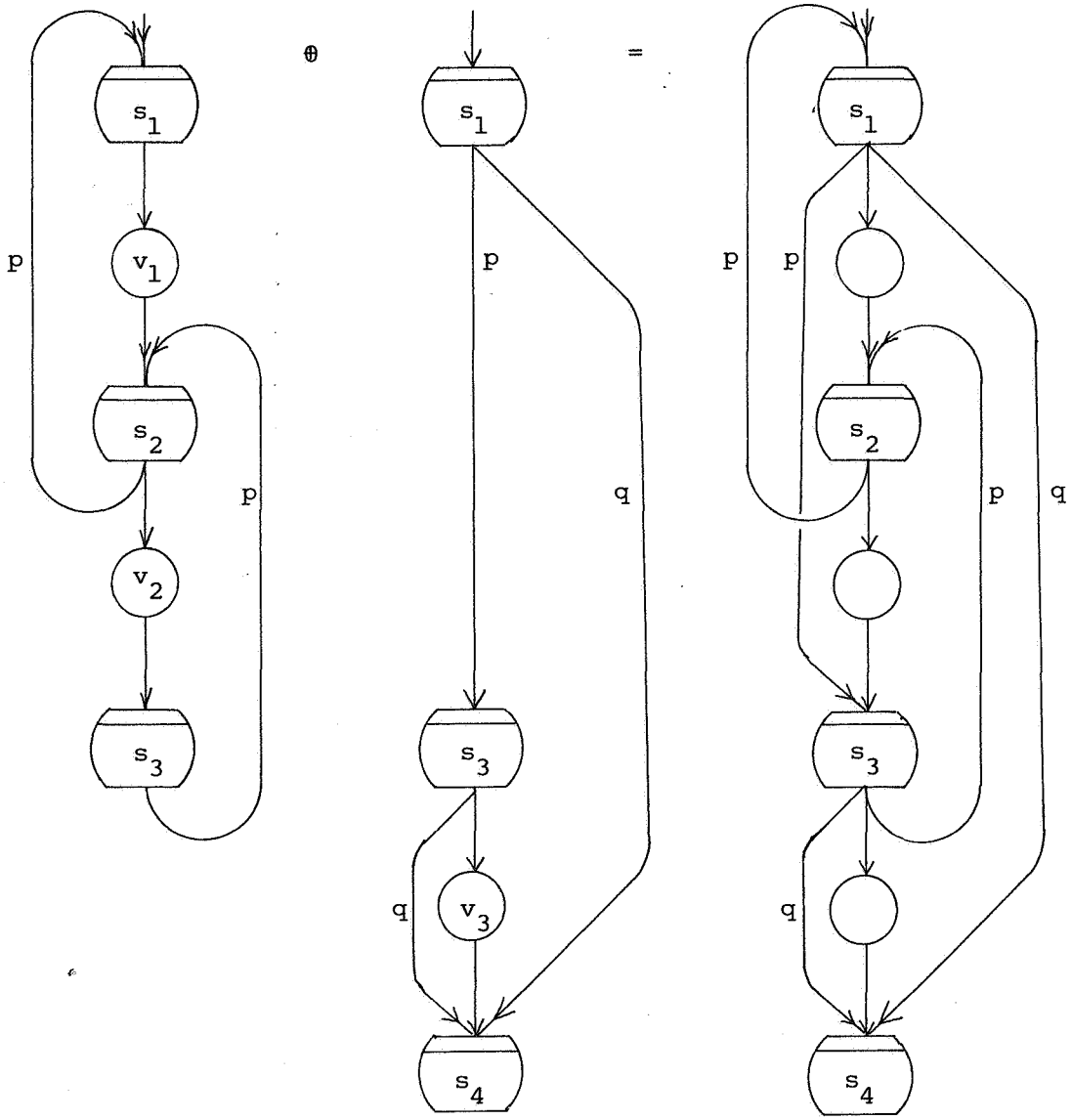
3.3 Optellen van SV-diagrammen

Teneinde complexe SV-diagrammen te kunnen maken is het plezierig de beschikking te hebben over de volgende operatie $\oplus$:

$D = D_1 \oplus D_2$.

Hierbij zijn D_1 en D_2 twee SV-diagrammen. We noteren met $|D|$ de verzameling van schermen van D. In dit geval krijgen we $|D| = |D_1| \cup |D_2|$ waarbij $|D_1| \cap |D_2| \neq \emptyset$ voor de hand ligt. D ontstaat door D_1 en D_2 als het ware op elkaar te plakken.

Voorbeeld



Teneinde dubbelzinnigheden te voorkomen moet aan enkele eisen voldaan zijn om $\oplus$ zinnig toe te kunnen passen.
Noteer met FT(S,D) de verzameling van funktietoetsen $p \in FT$ waarvoor $S \in |D|$ in D een uitgaande pijl bevat. FT(S,D) is dus de kollektie van funktietoetsen, die op S binnen D gedefinieerd zijn.
We eisen nu dat er voor elke $S \in |D_1| \cap |D_2|$ en $p \in FT(S,D_1) \cap FT(S,D_2)$ precies één $S' \in |D_1| \cap |D_2|$ is, zodat S' in D_1 en in D_2 via een met p gelabelde pijl met S verbonden is. Dit betekent dat vanuit S via p slechts één (éénduidig bepaald) scherm bereikt kan worden.
Voorts als $S \in |D_1| \cap |D_2|$ mag S slechts in een van beide diagrammen een uitgaande pijl voor natuurlijke besturing hebben.

3.4 Vermenigvuldigen van SV-diagrammen

Zoals de $\oplus$ een hulpmiddel is om SV-diagrammen te noteren, waarin de funktietoetsen complexe sprongmechanismen binnen een nivo aanduiden, willen we nu een operatie $\otimes_p$ ($p \in FT$) vinden, die behulpzaam is bij het noteren van de andere betekenis van een funktietoets, nl. de sprong naar een lager nivo.
We gaan uit van twee SV-diagrammen D_1 en D_2 met $|D_1| \cap |D_2| = \emptyset$ en een $p \in FT$ waarvoor $p \notin FT(S,D_1)$ voor elke $S \in |D_1|$.
Met $D_1 \otimes_p D_2$ noteren we een diagram dat de volgende werking heeft:
Op scherm S van D_1 heeft intoetsen van p het effect dat D_2 gestart wordt. Na een exit uit D_2 is men terug bij het zelfde scherm S uit D_1. (Tijdens de uitvoering van D_2 kunnen parameters een rol spelen die informatie leveren over S). Het standaard voorbeeld van deze situatie is een Help funktie te bereiken via p.
De volgende notaties en regels liggen nu voor de hand:

$|D_1 \oplus D_2| = |D_1| \cup |D_2|$

$|D_1 \otimes_p D_2| = |D_1| \cup |D_2|$

$FT(S, D_1 \oplus D_2) = FT(S, D_1) \cup FT(S, D_2)$

waarbij $FT(S, D) = \emptyset$ als $S \notin |D|$.

$FT(S, D_1 \otimes_p D_2) =$

$\begin{cases} \text{als } S \in D_1 \text{ dan } FT(S, D_1) \cup \{p\} \\ \text{als } S \in D_2 \text{ dan } FT(S, D_2). \end{cases}$

Het herhaald toepassen van $\oplus$ en $\otimes$ is ook zinvol (hoewel nog enkele gedetailleerde definities in beginsel vereist zijn laten we die hier achterwege omdat de zaak tamelijk voor zich spreekt).

3.5 Voorbeeld

Een systeem van de vorm $((D_1 \,\text{Ø}_p\, D_2) \oplus (D_1 \,\text{Ø}_q\, D_3)) \,\text{Ø}_r\, D_4$ kan corresponderen met de volgende situatie:

D_1 is een hoofdmenu dat een lange conversatie organiseert ten behoeve van (bijvoorbeeld) een medisch diagnostische aktiviteit. Afhankelijk van de door de arts ingevoerde en van patiënt verkregen informatie ondersteunt D_1 de keuze van nieuwe gespreksthema's (hartfunktie/longfunktie etc.) en stelt D_1 de arts zinnige vragen voor.

Ter ondersteuning van dit proces kan via twee funktietoetsen p en q gebruik gemaakt worden van de eveneens interaktieve processen D_2 en D_3.

D_2 levert achtergrond informatie betreffende wijze waarop de beslissingen en keuzen, die D_1 produceert tot stand komen. Deze achtergrond kan zowel betrekking hebben op algemene statistische gegevens als op historische gegevens betreffende patiënt.

D_3 levert een edit faciliteit waarmee gelijktijdig verschillende dokumenten kunnen worden ontwikkeld; een verslag voor de arts, een voor de patiënt, een verwijzing naar specialist of apotheker etc.

Tenslotte kan elk moment via toets r gebruik gemaakt worden van de Help funktie D_4. Ook dit is een interaktief systeem dat een zeer gebruikersvriendelijk inzicht geeft in de mogelijkheden van de gebruiker (arts) in elke gewenste situatie.

4. IMPLEMENTATIE VAN SV-DIAGRAMMEN, ALGEMEEN

De SV-diagrammen zijn een intuitief aansprekend medium met een redelijk vastgelegde semantiek. De implementatie van de diagrammen is echter geenzins vastgelegd.
Er doen zich hier twee hoofdvarianten voor welke we zullen bespreken na een korte inventarisatie van de verschillende bouwstenen van alle varianten van implementaties.

4.1 Bouwstenen

We nemen aan dat gewerkt wordt tegen de achtergrond van een toestandsruimte met de volgende struktuur:

- globale bestanden, deze worden bij de verwerkingen geraadpleegd en gemuteerd.
- een historie van de dialoog, deze is van toepassing voor de implementatie van faciliteiten als "ga naar vorig scherm". Omdat het per toepassing verschilt hoeveel historie men wil meenemen zijn we hier over niet exact.
- current scherminhoud, op elk moment heeft het scherm een inhoud, welke gelezen kan worden (een bepaald programma leest het scherm mogelijkerwijze slechts nu en dan).
- parameters voor het volgende scherm, als resultaat van een verwerking zien we naast mutaties van globale bestanden en de dialooghistorie ook parameters verschijnen voor het volgende scherm, scherm S_i kan geschreven worden wanneer de bijbehorende parameters gegeven zijn.

De belangrijkste operaties welke in een implementatie een rol spelen zijn de volgende:

Bouw en schrijf scherm S_i	B&S(S_i)
Lees en verwerk scherm S_i	L&VW(S_i)
Verwerk funktietoets p vanuit S_i	VW(p,S_i)
User	User

Hierbij gebruikt B&S(S_i) de parameters om het nieuwe scherm te bouwen, L&VW(S_i) muteert de globale bestanden, de dialooghistorie en genereert nieuwe parameters, VW(p) doet het zelfde maar nu met als invoer een funktietoets welke opgevat wordt als schermvulling.
User is een routine, die de gebruiker de kans geeft op het scherm te schrijven; de bewegingsvrijheid die de gebruiker hierbij heeft hangt vanzelfsprekend af van het current scherm.

De implementaties, welke wij gaan bekijken bevatten naast bovengenoemde basis operaties uitsluitend acties betreffende de boekhouding van scherm indices en besturingsakties om TP-monitor en conversationeel programma goed op elkaar af te stemmen.

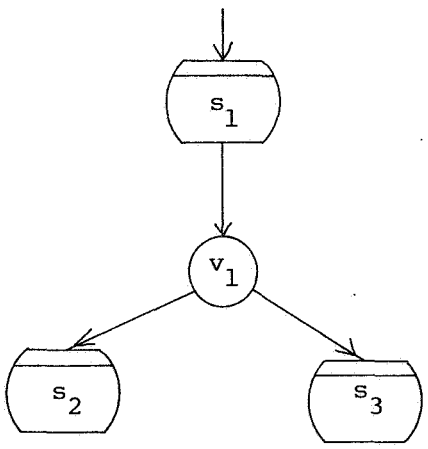
4.2 Schermformaat en verwerkingsformaat

De programma's, die de SV-diagrammen implementeren, hebben een modulaire struktuur. Hier dienen zich nu twee varianten aan:

S-formaat: één module per scherm (in het SV-diagram)

V-formaat: één module per verwerking (in het SV-diagram).

Om het verschil tussen beide formaten te zien, beschouwen wij het volgende SV-diagram.



In het S formaat zien we hier (in iets vereenvoudigde vorm):

```
S-CONVERSATION-D =
begin new t := 1
      initialiseer parameters en historie
  S1: B&S(S1)
      call S-C-USER (on S1)
      L&VW(S1)
      t := nieuwe schermindex
      case of t do goto St od
  S2: B&S(S2)
      goto eind
  S3: B&S(S3)
      goto eind
eind:
end
```

De betekenis van call S-C-USER wordt in 4.3 besproken.

De instruktie t := nieuwe schermindex staat voor: bepaal de index t van nieuw scherm, op basis van de alle componenten van de toestand behalve current-scherm (dit is immers net gelezen en verwerkt).

In het V-formaat krijgen we:

```
V-CONVERSATION-D =
begin new t := 1
      initialiseer parameters en historie
      B&S(S1)
  V1: call V-C-USER
      L&VW(S1)
      t := nieuwe schermindex
      case of t do B&S(St) od
      goto Vt
  V2: goto eind
  V3: goto eind
eind:
end
```

We zien dat in het V-formaat de V_i met een call V-C-USER beginnen en tot taak hebben om een nieuw scherm op te bouwen. Afhankelijk van de faciliteiten van de monitor en de host language zal men kiezen voor S- of V-implementaties.
Beide formaten kunnen overigens naar elkaar vertaald worden.

4.3 Conversationeel en pseudo-conversationeel

4.3.1 In een conversationele implementatie stellen we dat het monitorprogramma het hoofdprogramma is. De gebruiker aan de terminal wordt gezien als een niet-deterministische subroutine van het monitor-programma. Terwijl het monitor-programma CONVERSATION een call naar C-USER pleegt wacht CONVERSATION aktief totdat C-USER control aan CONVERSATION retourneert.

Een bruikbare pseudocode voor C-USER is de volgende:

```
procedure C-USER
      in: current scherm
     out: bewerkt scherm and (enter or p∈FT)
begin  bewerk scherm middels keyboard door (herhaald) gebruik van toetsen
       ongelijk enter of p∈FT. Beïndig de schermbewerking met enter of
       een p∈FT.
end
```

De body van C-USER staat elke uitvoering ervan toe die binnen de erin gestelde kriteria blijft. De programma's S-C-USER en V-C-USER uit 4.2 zijn identiek met C-USER.

4.3.2 Bij een pseudo-conversationele implementatie willen we vermijden dat het monitorprogramma aktief moet wachten op de terminal. We kiezen in de implementatie voor een coroutine mechanisme, die in feite een programma inversie realiseert van het monitorprogramma naar de reeks van uitvoeringen van C-USER. (Zie 4 voor inversie). Zo krijgen we P-CONVERSATION-D en P-USER-D onderling gekoppeld als coroutines. Het programma P-USER-D hangt nu ook van het SV-diagram af waar het bij hoort.

In pseudocode:

```
Coroutine (S,V)-P-USER-D
    begin resume (S,V)-P-CONVERSATION-D
          restart
          while (S niet eindpunt in D)
          do call C-USER
             resume (S,V)-P-CONVERSATION-D
             restart
          od
    end
```

Hier is S-P-USER de 'user' ten behoeve van een pseudo-conversationele S-implementatie en V-P-USER de 'user' ten behoeve van een pseudo-conversationele V-implementatie.

De betekenis van het resume/restart mechanisme is als volgt: we gaan uit van twee gekoppelde coroutines:
bijv. S-P-CONVERSATION-D en S-P-USER-D waarvan S-P-USER-D 'de leiding' heeft, dat wil zeggen als eerste opgestart wordt.
Bij de resume S-P-CONVERSATION-D instruktie geeft S-P-USER-D besturing aan S-P-CONVERSATION-D die bij de eerste restart in de tekst van zijn body begint. Wanneer een der beide programma's nu via een resume in het andere programma besturing terugkrijgt dan start hij weer in de eerste restart die volgt op het punt waar voor het laatst de executie door een resume onderbroken werd.
In het monitorprogramma zullen we nu niet aantreffen call (S,V)-C-USER-D, doch resume (S,V)-P-USER-D
restart

4.4 De gebruikte pseudocode

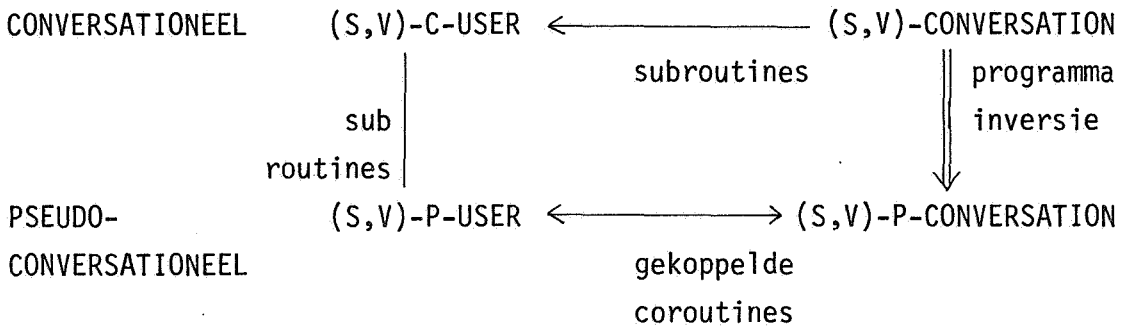
We gebruiken een ALGOL-achtige notatie met coroutines. We zullen zeer frequent de goto gebruiken in de verwachting dat hierdoor leesbaarheid noch korrektheid wordt geschaad. Opgemerkt zij dat formeel de korrektheid van de pseudocodes aan te tonen in willekeurig welke programmanotatie een niet eenvoudige zaak is.

Frequent zullen we bovendien de volgende notatie voor de guarded command gebruiken:

```
[ on konditie 1 do aktie 1 od
|
| on konditie 2 do aktie 2 od
|
| on konditie 3 do aktie 3 od
]
```

Tenslotte gebruiken we regelmatig de konstruktie:
case of index do aktie (index) od.
Van de vier hoofdvarianten uit onderstaande tabel zullen we in hoofdstuk 5, conversationeel voor het S-formaat en pseudo-conversationeel voor het V-formaat nader uitwerken.

Hoe onze pseudocodes het best in COBOL omgezet kunnen worden is een probleem dat we hier niet nader bespreken. (Zie hiervoor [1]). Tenslotte staan in onderstaand diagram de onderlinge relaties van de 4 hoofdvarianten aangegeven.



5. PSEUDOCODE VOOR CONVERSATIONELE S-FORMAAT IMPLEMENTATIES.

We gaan uit van een eenvoudig SV-diagram, met behulp waarvan de pseudocode goed geïllustreerd kan worden.

$$D^* = (D \mathbin{\text{Ø}}_{q_1} D_1) \oplus \ldots\ldots \oplus (D \mathbin{\text{Ø}}_{q_k} D_k)$$

$FT(D) = \{p_1, \ldots p_m\}$
$FT(D_i) = \emptyset$
$|D| = \{S_1, \ldots\ldots, S_n\}$ en $|D_i| = \{S_1^i, \ldots\ldots, S_{n(i)}^i\}$

De funktie H(j,l) = t levert de index $t \in \{1, \ldots, n\}$ van het scherm in D dat men bereikt wanneer in scherm $S_j \in |D|$ funktietoets $p_l \in FT(D)$ wordt gebruikt.
We beschrijven hier S-CONVERSATION-D^*, die S-C-USER als subroutine gebruikt.
Per component $(D, D_1, .., D_k)$ van D^* bevat dit programma een procedure die de conversatie binnen die componenten implementeert.
De struktuur van het programma wordt dan als volgt:

```
S-CONVERSATION-D* =
begin procedure S-CONV-D
          body S-CONV-D
      procedure S-CONV-D1
          body S-CONV-D1

      procedure S-CONV-Dk
          body S-CONV-Dk
      initialiseer parameter en historie
      call S-CONV-D
end
```

Hier zien we een module per component van het diagram D^*.
De procedures S-CONV-D_i zijn wat eenvoudiger dan procedure S-CONV-D omdat de D_i geen funktietoetsen gebruiken (deze vereenvoudiging dient louter de expositie en is geenszins een algemene beperking). Per scherm S_j^i bevat S-CONV-D_i een (sub)module, MS_j^i, ditmaal geplaatst achter een label.

```
procedure S-CONV-D_i
begin     new t := 1
     S_1^i : MS_1^i
     ========
     ========

   S_n(i)^i: MS_n(i)^i
   STOP_i:
end
```

```
MS_j^i ≡
  B&S(S_j^i)
  if   S_j^i eindpunt in D_i
  then goto STOP_i
  else call S-C-USER (on S_j^i)
       L&VW(S_j^i)
       t := nieuwe schermindex
       case of t do goto S_t^i od
  fi
```

(Met new t := 1 deklareren we een lokale integer variabele die tevens op 1 geïnitialiseerd wordt).

De procedure S-CONV-D heeft dezelfde struktuur als S-CONV-D$_i$ maar de modulen MS$_j^i$ zijn aanmerkelijk ingewikkelder omdat diverse funktietoetsen (naast enter) door S-C-USER kunnen worden geretourneerd.

```
procedure S-CONV-D
begin     new t := 1
   S_1 :   MS_1
   ========
   ========

   S_n :   MS_n
 STOP :
end
```

Het deel MS_i is nu als volgt:

```
MS_i ≡ B&S(S_i)
      if S_i eindpunt in D
      then goto STOP
      else call S-C-USER (on S_i)
           [on p = p_1
            |   do case of 1
            |        do VW(p_1, S_i)
            |           goto S_H(i,1 )
            |        od
            |   od
            □
            on p = q_h
            |   do case of h do call S-CONV-D_h od
            |      goto S_i
            |   od
            □
            on enter
            |   do L&VW(S_i)
            |      t := nieuwe schermindex
            |      case of t do goto S_t od
            |   od
           ]
      fi
```

Dit voltooit de pseudocode voor het S-CONV-D* in het gekozen vereenvoudigde voorbeeld. Opgemerkt moet worden dat dit slechts één mogelijkheid is uit vele.

6. PSEUDOCODE VOOR PSEUDO-CONVERSATIONELE V-FORMAAT IMPLEMENTATIES.

We gaan uit van hetzelfde SV-diagram als in sektie 5. Nu beschrijven we echter een coroutine V-P-CONVERSATION-D* die tesamen met in 4.3.2 beschreven coroutine V-P-USER de gewenste implementatie levert. De struktuur van V-P-CONVERSATION-D* op het hoogste nivo is als volgt:

```
coroutine V-P-CONVERSATION-D*
begin
    procedure V-P-CONV-D(t),    integer t ∈ [1,n]    , name t
        body V-P-CONV-D
    procedure V-P-CONV-D1(t),   integer t ∈ [1,n(1)], name t
        body V-P-CONV-D1
        ----------
        ----------
        ----------

    procedure V-P-CONV-Dk(t),   integer t ∈ [1,n(k)], name t
        body V-P-CONV-Dk

    new t := 1
    restart
    initialiseer parameters en historie
    B&S(S1)
    t := 1
    call V-P-CONV-D(t)
end
```

Hierbij valt het volgende op te merken.

(*) S_1 is het startscherm van D, dit zou opgebouwd moeten worden door de eraan voorafgaande verwerking;
daar deze er niet is wordt deze bouwopdracht verplaatst naar de initialisatie in de body van V-P-CONVERSATION-D*.

(**) De parameter t van de procedures V-P-CONV-D(t) en V-P-CONV-D_i(t) dienen om aan te geven in welk scherm gestart moet worden. Dit kan elk scherm zijn in het geval van terugkeer uit een subdialoog. (De S-struktuur is hier iets eenvoudiger).

(***) De eerste aktie van de body is een restart welke de eerste ingang is voor een call van V-P-CONVERSATION-D* vanuit V-P-USER (die als eerste begint).

De struktuur van de procedures V-P-CONV-D(t) en V-P-CONV-D_i(t) wordt nu gedetailleerd weergegeven. Hierbij beginnen we met de V-P-CONV-D_i(t) die (op grond van de struktuur van D^*) het eenvoudigst zijn.

```
procedure V-P-CONV-D_i(t), integer t ∈ [1,n(i)], name t.
  begin
  BEGIN_i: resume V-P-USER
           restart
           case of t do goto V_t od
      V_1: MV^i_1
           ═══════
   V_n(i): MV^i_n(i)
   EIND_i: if S_t eindpunt in D_i
           then goto STOP_i
           else goto BEGIN_i
           fi
   STOP_i:
end
```

De module MV^i_j per scherm zijn als volgt:

$MV^i_j \equiv$

```
    L&VW(S^i_j)
    t := nieuwe schermindex
    case of t do B&S(S^i_t) od
    goto EIND_i
```

Tenslotte moeten we de procedure V-P-CONV-D(t) beschrijven, daarbij zien we een struktuur die volstrekt overeenkomt met bovenstaande voor V-P-CONV-D_i met dien verstande dat na de restart nu ook de mogelijkheid van een 'return na funktietoets' moet worden opgevangen. Uit de keuze van D^* volgt dat dit de mogelijke funktietoetsen zijn:

q_h: sprong naar subdialoog D_h

p_l: sprong naar scherm $S_{H(i,l)}$ binnen D (vanuit scherm S_i).

enter: ga verder in D middels 'natuurlijke besturing'.

Zo komen we uit op:

```
procedure V-P-CONV-D(t), integer t∈ [1,n], name t.
begin
 BEGIN: resume V-P-USER
        restart
        [on p = p1
           do case of t,1
              do VW (p1, St)
                 goto VH(t,1)
              od
           od
         □
        on p = qm
           do case of m:
              do new integer t' := 1
                 call V-P-CONV-Dm(t')
                 case of t
                   do B&S(St)
                      goto EIND
                   od
              od
           od
         □
        on enter
           do case of t do goto Vt od
           od
        ]
   V1: MV1
       ═══════
       ═══════
       ═══════

   Vn: MVn
 EIND: if St eindpunt in D
       then goto STOP
       else goto BEGIN
       fi
 STOP:
 end
```

waarbij MV_j als volgt is:

$MV_j \equiv$
  L&V(S_j)
  t := nieuwe schermindex
  case of t do B&S(S_t) od
  goto EIND

Hiermee is de pseudocode voor V-P-CONVERSATION-D* volledig beschreven.

LITERATUUR

[1.] VOLMAC kursusmateriaal over conversationeel programmeren.

[2.] H. Jansen, Jackson structured programming in een on-line omgeving.
Informatie jaargang 24, nr. 6 (blz. 341 - 348).

[3.] H. Janssens, Gestruktureerde conversaties.
Informatie jaargang 24, nr. 3 (blz. 152 - 160).

[4.] M. Jackson, Principles of program design, Prentice-Hall International, 1983.