E S P R E S O -

Ein System zur Erstellung der Spezifikation von Prozeßrechner-Software

Jochen Ludewig
Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik

Inhalt

1. Literaturangaben

Ludewig, J. (1980a):
PCSL - a process control software specification system.
KfK 2874.

Ludewig, J. (1980b):
Zur Erstellung der Spezifikation von Prozeßrechnersoftware.
KfK 3060 (in Vorbereitung).

Watt, D.A., O.L. Madsen (1977):
Extended attribute grammars.
Report no. 10, University of Glasgow, Computing Departement.

2. Einführung

Etwa 1978/79 wurde im IDT die Sprache PCSL entwickelt und ein System zu ihrer Verarbeitung bereitgestellt (Ludewig, 1980a). PCSL ist durch drei Merkmale gekennzeichnet:

- Der vorgesehene Anwendungsbereich ist die Erstellung der Spezifikation für Prozeßrechner-Software.
- Die zugrundeliegenden Konzepte sind anwendungsorientiert, aber restriktiv zugunsten einfacher, klarer Strukturen, wodurch letztlich die Zuverlässigkeit der Programme verbessert wird.
- Die Spezifikationssprache ist abgestimmt auf den sogenannten Generalized Analyzer, der vom ISDOS-Project, University of Michigan, zur Verfügung gestellt worden war. Mit seiner Hilfe können Spezifikationen geprüft, gespeichert und dokumentiert werden.

Bei der Definition von PCSL und ersten Anwendungsversuchen entstand der Wunsch, die Ideen von PCSL in ein völlig neu zu entwickelndes System einzubringen, das von der ISDOS-Software unabhängig ist und dadurch wesentlich mehr Freiheit bietet, die eigenen Vorstellungen zu realisieren. So entstand ESPRESO.

Zunächst wurde ein für die Prozeßrechner-Programmierung geeignetes, anwendungsorientiertes Begriffssystem entworfen und darauf eine Sprache aufgebaut. Das Ergebnis ist die Spezifikationssprache ESPRESO-S. Sie erlaubt eine formale Beschreibung der Daten-, Modul- und Ablaufstrukturen sowie der Datenflüsse, nicht jedoch der arithmetischen und logischen Operationen.

Ähnlich wie in PCSL erfolgt die Spezifikation durch die Beschreibung der Objekte und ihrer Verknüpfungen. Jedes Objekt hat einen (frei wählbaren) Namen und eine Art, die einer vorgegebenen Menge entnommen ist. Der Art bei den Objekten entspricht die Relation bei den Verknüpfungen. Arten und Relationen sind so gewählt, daß praktische Probleme möglichst einfach beschrieben werden können. Die wichtigsten Arten sind:

- der Modul (module), der den Gültigkeitsbereich der darin enthaltenen Objekte darstellt,
- die Prozedur (procedure), deren Ablaufstruktur durch rekursive Verfeinerung in Blöcke (block) definiert ist,
- die Variable (variable), die der Darstellung von Zustandsgrößen dient,
- der Puffer (buffer) zur Aufnahme der Botschaften, die bei der Prozedur-Ausführung gesendet (erzeugt) und empfangen (verbraucht) werden,
- das Betriebsmittel (resource), das bei der Ausführung der Prozeduren nach Bedarf belegt wird,
- der Trigger (trigger) als Sonderform des Puffers, bei dem die Botschaften keine Information enthalten, sondern lediglich Marken wie in Petri-Netzen sind. Trigger dienen auch zur Beschreibung von Uhr-Funktionen: Ihre Marken werden dann nicht durch Lieferungen, sondern durch die Uhr erzeugt.

  Variablen, Puffer, Trigger und Betriebsmittel sind begrifflich zusammengefaßt als Medien.

Das gesamte System wird als Hierarchie (Baum) von Moduln dargestellt. Medien, die einem Modul M zugeordnet sind, können nur in M und den darin eingeschachtelten Moduln verwendet werden. Dagegen ist es möglich, Prozeduren auch außerhalb des Moduls zugänglich zu machen. Dadurch können mit Moduln abstrakte Datenstrukturen beschrieben werden. Bei den Medien lassen sich die Zugriffsrechte noch differenziert einschränken.

Die Verwendung eines Mediums impliziert die Anwendung eines bestimmten Koordinierungsschemas: Variablen erlauben parallelen Zugriff zum Lesen, während das Schreiben exklusiv erfolgt. Puffer und Trigger arbeiten nach dem Hersteller/Verbraucher-Schema, Betriebsmittel nach dem Prinzip des wechselseitigen Ausschlusses. Damit wird eine explizite Beschreibung der Synchronisation beim Zugriff auf Medien überflüssig.

Die Geräte der Standard- und der Prozeßperipherie können ebenfalls durch Medien beschrieben werden.

Die Ablaufstrukturen umfassen die sequentielle, die parallele und die alternative Ausführung untergeordneter Blöcke und den Prozeduraufruf. Ferner kann jede Prozedur und jeder Block zyklisch ausgeführt werden. In den Prozeduren und Blöcken werden die Zugriffe auf Medien angegeben, z.B. das Lesen einer Variablen oder das Liefern an einen Puffer. Durch die gleichartige Behandlung sequentieller und paralleler Ablaufstrukturen entfällt die Notwendigkeit, schon in einem frühen Stadium der Systementwicklung mit Begriffen wie "Prozeß" oder "Task" zu arbeiten. Jegliche vom Problem her mögliche Parallelität bleibt erhalten, so daß später eine optimale Konfiguration aus Hard- und Software gesucht werden kann.

Da bei der Spezifikation viele Angaben zunächst nur informal gemacht werden können, wurde besonderer Wert darauf gelegt, daß die Sprache auch die Formulierung durch Texte erlaubt und unterstützt. Daher können mit allen Objekten beliebig viele, durch die sogenannten Schlüssel unterschiedene Texte verbunden werden. Darin vorkommende Namen von Objekten werden als Querverweise ausgewertet.

ESPRESO-W ist ein Werkzeug, das die Entwicklung einer Spezifikation in ESPRESO-S unterstützt und dokumentiert. Insbesondere enthält es Funktionen

- zur Konvertierung, d.h. zum Einlesen einer Spezifikation in die sogenannte ESPRESO-Datei, die den abstrakten Gehalt speichert und eine sukzessive Entwicklung erlaubt,
- zur Dekonvertierung, der Rückwandlung aus der ESPRESO-Datei in die externe Form, d.h. nach ESPRESO-S,
- zur konsistenten Ersetzung von Objektnamen,
- zur Prüfung der Spezifikation unter verschiedenen Aspekten und zur Erzeugung entsprechender Reports,
- zur Verwaltung der ESPRESO-Dateien (anlegen, löschen, duplizieren, bringen auf und holen von Band).

Wie bei allen formalen Sprachen stellt sich bei Spezifikationssprachen die Frage nach der Bedeutung. Daher wurde nicht nur die kontextsensitive Syntax von ESPRESO-S, sondern auch die Wirkung ihrer Verarbeitung auf den Inhalt der ESPRESO-Datei formal definiert, und zwar durch eine erweiterte Attribut-Grammatik (Watt, Madsen, 1977). Darüber hinaus sind die Ablaufstrukturen und Koordinierungsmechanismen durch Angabe äquivalenter Programmstrukturen definiert.

Die Definition der Sprache ESPRESO-S ist abgeschlossen. Das Programmsystem ESPRESO-W ist in den wichtigsten Teilen (Konvertierung, Dekonvertierung und Verwaltung der ESPRESO-Datei) programmiert und getestet, kann jedoch zur Zeit wegen Schwierigkeiten mit dem PASCAL/360-Compiler nicht integriert werden. Eine ausführliche Beschreibung des gesamten Systems wird voraussichtlich Ende 1980 verfügbar sein (Ludewig, 1980b).

3. Spezifikation der Paketverteil-Anlage

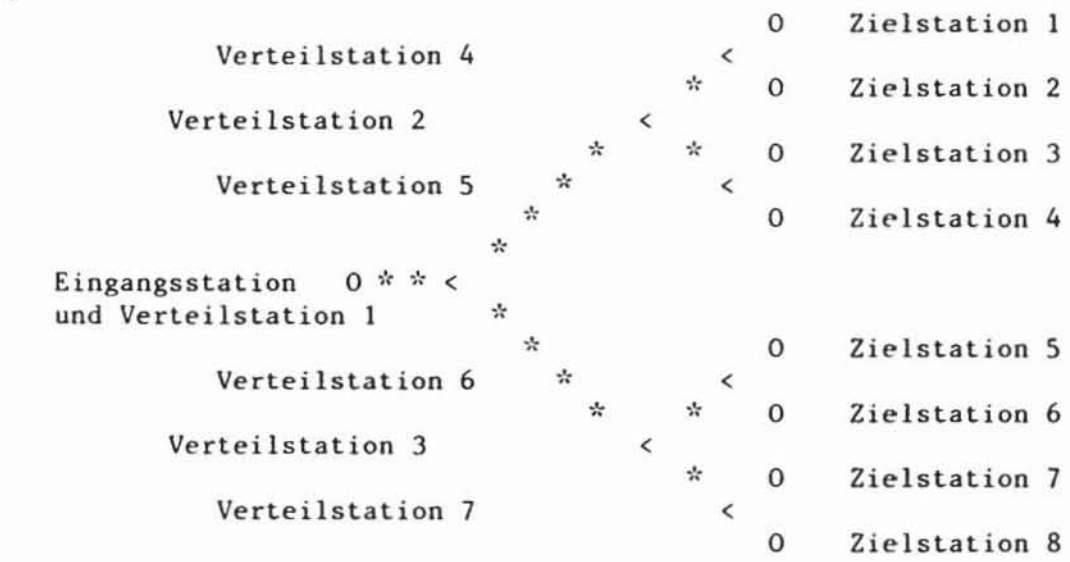
Da ESPRESO die E r s t e l l u n g der Spezifikation unterstützen soll, wird hier nicht nur die resultierende Formulierung gezeigt, sondern der Prozeß ihrer Entstehung verfolgt. Dadurch wird erkennbar, wie sich die Spezifikation formt in Wechselwirkung zwischen Aufgabenstellung und Sprachmitteln. Diese Wechselwirkung ist beabsichtigt und führt zu einer relativ übersichtlichen Formulierung.

Die Spezifikation hält sich streng an die Aufgabenstellung; das bedeutet vor allem, daß die zahlreichen denkbaren, aber in der Aufgabe ausgeschlossenen Defekte der Anlage nicht berücksichtigt werden.

Um dem Leser auch ohne gründliche Anleitung das Verständnis dieser Spezifikationssprache zu erleichtern, wurde ein typischer Abschnitt (3.5.2) mit Erklärungen versehen.

3.1 Verbale Formulierung der Aufgabe

Die Aufgabe kann zunächst durch Text-Objekte in ESPRESO-S aufgenommen werden; sie ist bis auf "redaktionelle" Änderungen unverändert.



Für eine Paketverteilungsanlage, bei der die Pakete auf acht verschiedene Zielstationen gelenkt werden, soll das Steuersystem entwickelt werden. Es wird angenommen, daß innerhalb der Anlage Pakete weder verlorengehen noch hinzukommen. Auch arbeiten sämtliche Komponenten der Anlage stets fehlerfrei, und die Pakete können sich nicht verklemmen. Allerdings muß damit gerechnet werden, daß die Pakete nicht alle gleichschnell durch die Anlage laufen. Das Steuersystem, d.h. der Prozeßrechner, ist schnell im Vergleich zu den Bewegungsvorgängen im technischen Prozeß (Bewegungen der Weichen und Pakete), es werden also durch die Verarbeitungszeiten im Rechner keine Probleme verursacht.
¢ end;

informal Eingangsstation:
¢ Die Eingangsstation besteht aus einem Freigabeorgan mit den Teilen F1 und F2. F2 hält das einlaufende Paket so lange fest, bis das Meldeorgan die Ankunft an das Steuersystem gemeldet hat und dieses mit Hilfe des Leseorgans das Codezeichen aufgenommen hat. Bei fehlendem oder unzulässigem Paketcode wird eine spezielle Zielstation angesteuert (hier: Zielstation 8).

Danach gibt das Steuersystem einen Auftrag an das Freigabeorgan, die Sperre F2 gibt den Weiterlauf für das Paket frei und das Beschleunigungsteil F1 neigt sich. Dadurch gleitet das Paket weiter. Gleichzeitig wird das nächste Paket am Einlaufen gehindert, bis die Sperre F2 wieder eingetreten ist.
Bei der Behandlung des nachfolgenden Pakets ist zu prüfen, ob sich sein Ziel von dem des Vorläufers unterscheidet. In diesem Fall ist die Freigabe seines Weiterlaufs so lange zu verzögern, daß die Lenkorgane zwischen den beiden Paketen sicher umgestellt werden können. Im andern Fall können die Pakete unmittelbar aufeinander folgen.
¢ end;

informal Verteilstation:
¢ Ein- und Ausgangspunkte jeder Verteilstation sind mit Lichtschranken versehen. Diese können aufgrund ihrer technischen Ausführung auch dicht aufeinanderfolgende Pakete mit Sicherheit einzeln erkennen. Die Meldungen werden im Steuersystem zur Laufwegverfolgung jedes einzelnen Pakets ausgewertet. Dadurch kann in Verbindung mit dem bereits ermittelten Ziel der Steuerauftrag für das nächste Lenkorgan ermittelt und ausgegeben werden.
Beim Ausgeben des Steuerauftrages ist darauf zu achten, daß alle Vorläufer die betreffende Verteilstation passiert haben. Die Verteilstation selbst muß frei sein, d.h. zwischen Ein- und Ausgangslichtschranke darf sich kein Paket befinden.
Tritt aufgrund unterschiedlicher Geschwindigkeiten der Fall ein, daß ein Paket, für das die Verteilstation umzustellen ist, den Eingangsmeldepunkt erreicht, bevor der Vorläufer die Station verlassen hat, muß die Umstellung unterbleiben. Er wird zum Falschläufer und bekommt das Ziel seines Vorläufers. Für jeden Falschläufer soll genau eine Meldung mit Soll- und Ist-Ziel ausgegeben werden.
¢ end;

informal Basismaschine:
¢ Die Aufgabe enthält auch zwei Angaben zur Basismaschine:
- Als Konfiguration ist ein einzelner Prozeßrechner vorzusehen.
- Die Programmierung erfolgt in einer höheren Sprache.

Auf die ESPRESO-Spezifikation haben diese Angaben keinen Einfluß.
¢ end.

Damit sind die Text-Objekte Aufgabenstellung, Eingangsstation, Verteilstation und Basismaschine mit jeweils einem Text definiert. Da dieser unmittelbar auf den Doppelpunkt folgt, kann das Wortsymbol text entfallen. Ein Schlüssel fehlt ebenfalls; den Texten ist daher die leere Zeichenreihe als Schlüssel zugeordnet.
In Sonderfällen können auch Text-Objekte ohne Namen verwendet werden (siehe z.B. 3.5.1, "where ...").

3.2 Erste Formulierung des Steuersystems

```
module Paketverteil-Anlage:
comprises

   procedure Steuerung:
   ¢ Steuerung verarbeitet die Signale aus dem technischen Prozeß,
     steuert das Freigabeorgan und erzeugt, wenn notwendig, Fehler-
     meldungen.
   ¢;
   end,

   trigger Eingangsmeldung:
   ¢ wird erhöht, wenn ein Paket in die Eingangsstation einläuft ¢;
   end,

   variable Adresse:
   ¢ verfügbar, wenn die !Eingangsmeldung eingetroffen ist ¢;
   end,

   variable Freigabe:
   ¢ wird vom Rechner nur geschrieben, vom Prozeß "gelesen" ¢;
   values Einlaufstellung, Auslaufstellung;
   end,

   trigger Einlauf-VS1:
   ¢ wird erhöht, wenn Paket in die Verteilstation 1 einläuft ¢;
   end,

   trigger Auslauf-VS1:
   ¢ wird erhöht, wenn Paket links oder rechts aus der Verteilstation 1
     ausläuft ¢;
   text unklar
   ¢ Nach Beschreibung der !Eingangsstation ist es nicht nötig, zwischen
     linkem und rechtem Ausgang zu unterscheiden. Falls die Zusammen-
     fassung der beiden Signale nicht möglich ist, sind zwei zusätzliche
     Blöcke nötig.
   ¢;
   end,

   trigger Einlauf-VS2:
   ...
   ...              (VS2 bis VS7 entsprechend VS1)
   ...
   end Auslauf-VS7

end Paketverteil-Anlage.
```

In dieser Spezifikation wurde bisher nur die - in ESPRESO-S selbstverständliche - Zuordnung der Interrupts zu Triggern vorgenommen. Im übrigen ist die Aufgabenstellung noch unverändert.
Einer der Texte hat den Schlüssel "unklar". Mit einer Report-Funktion von ESPRESO-W können alle Texte mit diesem Schlüssel ausgegeben werden, so daß leicht ein Überblick der Unklarheiten geschaffen werden kann.
"!Eingangsmeldung" im Text zu "Adresse" ist ein Querverweis.

3.3 Modularisierung

Aus der räumlichen Anordnung des Systems folgt direkt die naheliegende Modularisierung:

```
module Paketverteil-Anlage:

comprises
   module Eingangsstation,
   module VS1,
   module VS2,
   ...
   module VS7,

   procedure Steuerung:
   parallel Eingangsbearbeitung
   parallel VS1-Verwaltung
   parallel VS2-Verwaltung
   ...
   parallel VS7-Verwaltung
   end Steuerung

end    Paketverteil-Anlage.
```

Trigger und Variablen, die oben als Teile des Hauptmoduls beschrieben wurden, werden nun den einzelnen Teilen zugeordnet (z.B. kommen Einlauf-VS1 und Auslauf-VS1 in den Modul VS1 und sind damit nur dort verfügbar). In jedem dieser Submoduln gibt es einen Block, der die Steuerung an dem jeweiligen Punkt übernimmt.

Für die Kommunikation zwischen diesen Moduln gibt es zwei Möglichkeiten, die zentrale, bei der ein zusätzlicher Modul oder der Modul "Eingangsstation" als einziger mit den übrigen Moduln kommuniziert, oder die dezentrale, bei der die Moduln untereinander kommunizieren. Die zentrale Lösung hat Nachteile, vor allem im Falle fehllaufender Pakete. Daher wird hier die dezentrale Lösung gewählt.

In Analogie zum physischen Fluß der Pakete, z.B. von der Verteilstation 1 zur Station 2, gibt es Puffer, durch die die Begleitinformation zu den Paketen übermittelt wird, im Beispiel also zwischen VS1 und VS2:

```
buffer Weitergabe-1-2:
¢ enthält Angaben zu Paketen, die !VS1 verlassen,
  aber !VS2 noch nicht erreicht haben ¢;
produce restricted-to VS1;
consume restricted-to VS2
end Weitergabe-1-2.
```

Dieser Puffer wird global im Modul Paketverteil-Anlage definiert. Durch die Restriktionen ist jedoch der Zugriff beschränkt auf die Moduln VS1 und VS2. Entsprechende Puffer werden zwischen allen direkt verbundenen Verteilstationen definiert, auch zwischen Eingangsstation und VS1.

3.4 Eingangsstation und Verteilstationen

Es folgt nun der wesentliche Teil der Arbeit, die Definition der Eingangsstation und der Verteilstationen.

```
module Eingangsstation:

comprises
   trigger   Eingangsmeldung,
   variable  Adresse,
   variable  Freigabe,

   block      Eingangsbearbeitung:
   while      Automatik;
   started-by Eingangsmeldung;
   produces   Weitergabe-E-1;
   reads      Adresse;
   writes     Freigabe
   end

end Eingangsstation.
```

Die jetzt in diesem Modul definierten Objekte entfallen natürlich im übergeordneten. Es wird hier darauf verzichtet, diese Verschiebungen jeweils im Detail darzustellen. Auch werden hier und nachfolgend weniger Texte zu den Objekten angegeben, als in einer echten Spezifikation, der der begleitende Kommentar fehlt, sinnvoll wäre.

Als Repräsentant der Eingangsstationen wird nachfolgend VS2 beschrieben; VS3 ist völlig entsprechend. VS1 und VS4 bis VS7 sind geringfügig verschieden, VS1 wegen seiner Sonderstellung zur Eingangsstation, die übrigen, da sie keine Begleitinformation weiterreichen, aber eventuell Meldungen erzeugen müssen.

```
module VS2:
text unklar
¢ aus der Aufgabe ist nicht ersichtlich, wie die Umschaltung der Weiche
  erfolgt. Hier wird die Weiche als Variable !Weiche-2 definiert, wie
  schon die Freigabe.
¢;
comprises
   trigger   Einlauf-VS2,
   trigger   Auslauf-VS2,

   block VS2-Verwaltung:
   ¢ überwacht Verteilstation 2, stellt, wenn nötig und möglich, die
     Weiche 2 um, ändert bei Fehlläufern die Begleitinformation, falls
     nötig, und gibt sie weiter an die nächste Station.
   ¢;
   end,

   variable  Weiche-2:
   values    links, rechts
   end

end VS2.
```

3.5 Verfeinerung der Blöcke

3.5.1 Eingangsbearbeitung

Die Struktur des Blocks Eingangsbearbeitung geht aus der Aufgabenstellung hervor, die den Ablauf vorgibt: (1) Paket läuft ein, (2) Paket wird freigegeben und an VS1 weitergemeldet, (3) Eingangsstation wird für das nächste Paket bereitgemacht. Unklar ist nur, wie die Wartezeit auszudrücken ist. Offenbar muß die Dauer bis zur Freigabe des nächsten Pakets einen bestimmten Mindestwert haben, wenn die Adressen unterschiedlich sind. Daraus folgt, daß nach Schritt (1) ein Warteschritt einzufügen ist, dessen Dauer von der Adreßsequenz und davon abhängt, wieviel Zeit bereits seit der letzten Freigabe vergangen ist.

```
block       Eingangsbearbeitung:
while       Automatik;
started-by Eingangsmeldung;
sequential
   Adresse-lesen:
   reads    Adresse, Zeit;
   updates  Vorgänger-Adresse, Vorgänger-Startzeit;
   writes   Wartezeit;
   produces Weitergabe-E-1;
   end Adresse-lesen

then
   Freigeben:
   delay  Wartezeit of sec;
   (* dies ist eine in ESPRESO-S vorgesehene implizite Beschreibung
      eines Triggers, der den Block nach einer Wartezeit startet.     *)
   writes Freigabe where ¢ := Auslaufstellung ¢;
   end Freigeben

then
   Vorbereiten:
   text unklar
   ¢ Die Aufgabe läßt offen, ob die Eingangsstation nach vorgegebener
     Zeit oder nach Quittierung des Pakets durch VS1 wieder in die
     Ausgangstellung gehen soll. Hier wird der (sinnvolle) zweite Fall
     angenommen.
   ¢;
   started-by Quittung-1-E;
   writes     Freigabe where ¢ := Einlaufstellung ¢
   end Vorbereiten

end Eingangsbearbeitung;


module Paketverteil-Anlage:
comprises
   trigger Quittung-1-E:
   ¢ bestätigt den Einlauf in !VS1 und beendet damit den Wartezustand
     in der !Eingangsbearbeitung ¢;
   produce restricted-to VS1;
   consume restricted-to Eingangsstation;
   end Quittung-1-E
end    Paketverteil-Anlage.
```

3.5.2 Eine repräsentative Verteilstation

Der schwierigste Teil der Aufgabe ist die Definition der Verteilstationen. In diesen laufen jeweils zwei durch den physischen Prozeß gekoppelte Prozesse ab, der Einlauf und der Auslauf der Pakete. Asynchron werden außerdem Pakete gemeldet.

Hier wird zunächst eine Spezifikation entwickelt, wie sie ohne lange Vorüberlegungen entsteht. Eine elegantere Lösung folgt unter 3.7.

Offensichtlich sind für die Ein- und Ausgangsmeldungen zwei parallele Prozesse erforderlich. Weniger leicht ist die Einordnung der Ankündigungen neuer Pakete, die von VS1 weitergereicht werden. Nach einiger Überlegung wird klar, daß ein spezieller Prozeß für diesen Zweck streng vom Auslaufprozeß kontrolliert würde, so daß es einfacher ist, ihn gleich dort zu integrieren.
Für die Buchführung über ein- und auslaufende Pakete sind Variablen und Trigger nötig, deren Inhalt zu jedem Zeitpunkt konsistent sein muß. Daher werden sie logisch zusammengefaßt im Betriebsmittel "Buchführung", dessen Belegung den exklusiven Zugriff sichert.

Die Beschreibung des Blocks VS2-Verwaltung ist mit zahlreichen Kommentaren versehen, die die Funktion der einzelnen Sprachelemente erklären sollen. Sie sind im Sinne der Spezifikation nicht relevant und werden bei einer Verarbeitung durch ESPRESO-W ignoriert, also anders als die Texte auch nicht gespeichert.

Das Prinzip sei zur besseren Übersicht nachfolgend kurz erläutert: Ankommende Pakete (Puffer Weitergabe-1-2) werden, falls die Weiche zuvor leer wird, vom Block VS2-Auslauf registriert und in akt-Paket-2 an VS2-Einlauf gemeldet. Andernfalls holt VS2-Einlauf selbst die Information aus Weitergabe-1-2.

```
block VS2-Verwaltung:

parallel                          (* VS2-Verwaltung ist durch
                                     parallele Blöcke verfeinert.      *)
   VS2-Einlauf:                   (* VS2-Einlauf ist ein eingeschachtelter
                                     Block. Das Wortsymbol block vor dem
                                     Namen ist redundant und kann daher
                                     entfallen.                        *)
   while      Automatik;          (* Der Block wird zyklisch aus-
                                     geführt, solange die Variable
                                     Automatik den Wert true hat.      *)
   started-by Einlauf-VS2;        (* Die Ausführung beginnt, wenn
                                     Einlauf-VS2 eingetroffen ist.     *)
   occupies   Buchführung-2;      (* VS2-Einlauf belegt, wenn es
                                     gestartet ist, Buchführung-2
                                     und verhindert dadurch, daß ein
                                     anderer Block (d.h. VS2-Auslauf)
                                     gleichzeitig auf die Medien
                                     Pakete-in-Weiche-2, akt-Paket-2
                                     und Weiche-2-frei zugreift.       *)
```

```
sequential                          (* VS2-Einlauf ist durch
                                       sequentielle Blöcke verfeinert. *)
   VS2-E1:
   produces Pakete-in-Weiche-2; (* zählt die Pakete in der Weiche, *)
   tests    akt-Paket-2;       (* prüft, ob Weiche schon umge-    *)
   writes   Weiche-2-frei      (* stellt, notiert Ergebnis.       *)
      where ¢ := true, falls !akt-Paket-2 nicht leer ¢

   controlled-by Weiche-2-frei; (* steuert die Verzweigung        *)

   either key true for
      VS2-E11:  end             (* keine Aktion notwendig         *)

   or key false for
      VS2-E12:
      consumes Weitergabe-1-2;     (* holt nächste Anmeldung      *)
      reads    Weichenstellung-2;  (* stellt Stand der Weiche fest *)
      produces akt-Paket-2
         where ¢ das Soll-Ziel wird unverändert übernommen,
                 das Ist-Ziel richtet sich nach !Weichenstellung-2.
                 Sind die beiden unterschiedlich, so handelt es
                 sich um einen Fehlläufer.
               ¢;
      end VS2-E12
   end VS2-E1


then
   VS2-E2:                          (* Weitergabe der Begleitinfor-
                                       mation an die nächste Station  *)
   controlled-by Weichenstellung-2;

   either key links for
      VS2-E2-links:
      consumes akt-Paket-2;
      produces Weitergabe-2-4;
      end VS2-E2-links

   or key rechts for
      VS2-E2-rechts:
      consumes akt-Paket-2;
      produces Weitergabe-2-5;
      end VS2-E2-rechts
   end VS2-E2

end VS2-Einlauf
```

```
parallel
   VS2-Auslauf:
   while      Automatik;              (* zyklisch wie VS2-Einlauf     *)
   started-by Auslauf-VS2;
   occupies   Buchführung-2;          (* symmetrisch zu VS2-Einlauf   *)

   consumes Pakete-in-Weiche-2;       (* senkt den Zähler             *)
   tests    Pakete-in-Weiche-2;
   writes   Weiche-2-leer where ¢ := true, falls leer ¢;

   controlled-by Weiche-2-leer;
   either key true for
      VS2-A1:                         (* Ist die Weiche leer, so wird *)
      consumes Weitergabe-1-2;        (* das nächste Paket erwartet,  *)
      produces akt-Paket-2;           (* notiert und die Weiche ggf.  *)
      writes   Weiche-2,              (* gestellt.          Die neue  *)
               Weichenstellung-2;     (* Position wird gespeichert.   *)
      end VS2-A1
   or key false for
      VS2-A2:  end                    (* Umschalten nicht möglich     *)
   end VS2-Auslauf

end VS2-Verwaltung.
```

3.5.3 Die anderen Verteilstationen

Der Modul VS1 unterscheidet sich, wenn man analoge Namen für die Objekte voraussetzt, nur durch Erzeugung der Quittung in VS1-E1:

```
block VS1-E1:
produces Pakete-in-Weiche-1, Quittung-1-E;
tests    akt-Paket-1;
...
```

Weiter analog zu VS2.

VS4-Einlauf bis VS7-Einlauf unterscheiden sich am Ende von VS2-Einlauf:

```
block VS4-E2:
consumes akt-Paket-4;
writes   Fehlläufer-in-4;

controlled-by Fehlläufer-in-4;
either key true for
   Meldung-aus-VS4:                   (* Hier fehlt noch die Angabe,     *)
   produces Meldungen;                (* woraus die Meldung erzeugt wird *)
   end
or key false for
   VS4-E2-OK: (* alles klar, kein Fehlläufer *) end
end VS4-E2.
```

3.5.4 Die Erzeugung von Meldungen

Es fehlt noch ein Prozeß, der den Meldungspuffer abbaut:

```
module Paketverteil-Anlage:
comprises

   module Meldungserzeugung:
   comprises
      block Meldungen-drucken:
      consumes  Meldungen;
      produces  Druckausgabe;
      end,

      buffer Druckausgabe:        (* Dieser Puffer wird scheinbar    *)
      interface;                  (* nur beliefert, da der Abnehmer, *)
      capacity 1;                 (* d.h. hier ein Hardware-System,  *)
      of-type Zeile;              (* nicht beschrieben ist. Daher    *)
      end                         (* ist "Druckausgabe" als "inter-  *)
                                  (* face" gekennzeichnet.           *)
   end Meldungserzeugung,


   procedure Steuerung:
   parallel Meldungen-drucken
   end Steuerung,

   buffer Meldungen:
   consume restricted-to Meldungserzeugung;
   capacity 10;
   of-type Zeile;
   end

end Paketverteil-Anlage.
```

3.6 Variablen, Puffer, Trigger und Betriebsmittel

Die in 3.5.2 genannten Objekte werden, soweit sie noch nicht definiert sind, nachfolgend beschrieben. Sie sind lokal zu VS2. Die Objekte in den anderen Moduln ergeben sich analog.

```
module VS2:
comprises

   resource Buchführung-2:
   ¢ sichert die Konsistenz von !Pakete-in-Weiche-2, !Weitergabe-1-2
     und !akt-Paket-2 ¢;
   end,
```

```
    buffer akt-Paket-2:
    ¢ zur Speicherung der Information über Pakete, die in !VS2
      angemeldet sind ¢;
    capacity 1;
    of-type Begleitinformation
    end,

    variable gleiche-Richtung-2:
    of-type boolean;
    text ¢ true, wenn die Richtung des einlaufenden Pakets für diese
            Weiche mit der des Vorgängers übereinstimmt. ¢
    end,

    variable Weiche-2:
    ¢ repräsentiert die physische Weiche ¢;
    interface;
    values links, rechts;
    end,

    variable Weichenstellung-2:
    ¢ zur Speicherung des Weichenstandes ¢;
    values links, rechts;
    end,

    type Begleitinformation:
    structure-of Soll-Ziel, Ist-Ziel
    end,

    type Soll-Ziel:
    of-type Zielstationen
    end,

    type Ist-Ziel:
    of-type Zielstationen
    end,

    type Zielstationen:
    values ZS1, ZS2, ZS3, ZS4, ZS5, ZS6, ZS7, ZS8
    end,

    variable Weiche-2-frei:
    of-type boolean;
    text ¢ true, wenn die Weiche laut Buchführung frei ist. ¢;
    end,

    trigger Pakete-in-Weiche-2:
    ¢ Zähler für die Pakete, die von !VS2-Einlauf registriert,
      von !VS2-Auslauf aber noch nicht abgebucht sind ¢;
    end

end VS2.
```

3.7 Eine Vereinfachung der Block-Struktur

Betrachtet man die entstehende Struktur im Hinblick auf den Grenzfall, daß die Weiche zwischen zwei Paketen nur sehr kurz frei wird, so fällt auf, daß die Umschaltung erfolgt, während das Paket bereits in die Weiche einläuft. Sie kann also ebensogut in allen Fällen von der Eingangsverwaltung vorgenommen werden, da nach Aufgabenstellung der Rechenprozeß keine nennenswerte Zeit beansprucht. Diese Lösung hat eine erheblich einfachere Struktur, da der Puffer akt-Paket-2 durch eine Variable ersetzt ist und die Auslaufverwaltung nur noch Pakete abbucht. Das macht auch das abstrakte Betriebsmittel "Buchführung" überflüssig.

```
block VS2-Verwaltung:

parallel
   VS2-Einlauf:
   started-by Einlauf-VS2;
   consumes   Weitergabe-1-2;
   writes     akt-Paket-2;                    (* ist jetzt Variable *)
   produces   Pakete-in-Weiche-2;

   sequential
      VS2-Umschalt-Test:
      inhibits Pakete-in-Weiche-2;
      tests    Pakete-in-Weiche-2;
      writes   Weiche-2-frei
      end

   then
      VS2-Umschaltung:
      controlled-by Weiche-2-frei;

      either key true for
         VS2-Weiche-stellen:
         reads  akt-Paket-2;
         writes Weiche-2, Weichenstellung-2
         text Optimierung ¢ durch weitere Verfeinerung kann hier
                           verhindert werden, daß eine Ausgabe
                           erfolgt, wenn die Stellung der Weiche
                           nicht verändert wird.                ¢;
         end VS2-Weiche-stellen

      or key false for
         VS2-Weiche-belegt: (* keine Aktion möglich *) end

      end VS2-Umschaltung
```

```
   then
      VS2-Weitergabe:
      controlled-by Weichenstellung-2;

      either key links for
         VS2-Weitergabe-links:
         reads    akt-Paket-2;
         produces Weitergabe-2-4;
         end VS2-Weitergabe-links

      or key rechts for
         VS2-Weitergabe-rechts:
         reads    akt-Paket-2;
         produces Weitergabe-2-5;
         end VS2-Weitergabe-rechts
      end VS2-Weitergabe

   end VS2-Einlauf

parallel
   VS2-Auslauf:
   started-by Auslauf-VS2;
   consumes   Pakete-in-Weiche-2;
   end VS2-Auslauf

end VS2-Verwaltung.
```

Fügt man die oben skizzierten Teile zusammen und ergänzt das fehlende sinngemäß, so entsteht eine vollständige, weitgehend formale Beschreibung des Systems, die als Spezifikation bezeichnet werden kann. Zweifellos sind bereits Entwurfsentscheidungen getroffen (z.B. hinsichtlich der Strukturierung). Dennoch lag keine vollständige Spezifikation vor; diese konnte erst auf der jetzt erreichten Detaillierungsebene von einigen Unklarheiten befreit werden.
Es bestätigte sich damit, daß die Auffassung, Spezifikation und Entwurf könnten sequentiell durchgeführt werden, praktisch nicht aufrecht erhalten werden kann. Vielmehr handelt es sich um zwei miteinander verzahnte Tätigkeiten.

Das Beispiel läßt auch einige ungelöste Probleme erkennen. Dazu gehört die Beschreibung gleicher oder ähnlicher Systemkomponenten, hier z.B. die der verschiedenen Verteilstationen. Es wird sinnvoll sein, die Sprache zu erweitern, so daß die Spezifikationen soweit wie möglich zusammengefaßt werden können.