Eine Methode zur schnelleren Entwicklung und übersichtlichen Dokumentation von PEARL-Programmen.

von Prof. Dr. L. Frevert, Bad Salzuflen

Zusammenfassung

Es wird eine Methode beschrieben, mit der aus einem Grobentwurf durch Verfeinerung in kleinen Schritten ein Feinentwurf entwickelt wird, aus dem sich durch Anwendung eines Preprozessors ein kompilierbares Programm ergibt. Die dabei entstehende Dokumentation des Entwurfs-Prozesses ist gleichzeitig eine übersichtliche Dokumentation des Programmes.

Abgrenzung gegen andere Verfahren

Die hier beschriebene Methode wurde aufgrund von sehr guten Erfahrungen entwickelt, die mit der Systemsprache META-S /1/ gemacht worden waren. META-S und die Schulsprache ELAN /2/ enthalten Sprachelemente für die Top-down-Entwicklung von Algorithmen und sind daher sehr geeignet für die schnelle Entwicklung schwieriger Algorithmen. Ursprünglich war nur beabsichtigt, dieses Prinzip für die Entwicklung von PEARL-Algorithmen nutzbar zu machen, wobei ein Nachteil der vorerwähnten Sprachen - hohe Schreibarbeit und schlechte Lesbarkeit längerer Programme - vermieden werden sollte. Es stellte sich jedoch heraus, daß sich auch große Vorteile durch die Anwendung der Methode auf die Beschreibung von Parallel-Arbeit und Problemdaten, sowie bei der Niederschrift von Programm-Spezifikationen ergaben.
Zur Abgrenzung gegenüber anderen Spezifikations- und Entwurfsverfahren /3/ ist zu sagen, daß diese Methode ein Werkzeug für die Programm-Entwicklung ab der Feinspezifikation darstellt, und daß kompilierbare Programme automatisch aus den Entwürfen erzeugt werden. Das dafür benötigte Hilfsmittel ist leicht zu implementieren (der Prototyp des Preprozessors wurde an einem Tage geschrieben). Sie reicht insofern bis in die Test- und Wartungsphase, als nur für Testzwecke benötigte Programmteile durch sehr einfache Maßnahmen aus den Programmen entfernt, bzw. wieder hineingenommen werden können. Die Methode bedient sich keiner speziellen Entwurfs-Sprache, sondern setzt nur PEARL-Kenntnisse voraus. (Anmerkung: da die Methode an sich nicht sprachgebunden ist, wurde sie auch schon für die Entwicklung von Mikroprozessor-Assemblerprogrammen benutzt.)

Aufgaben und Lösungen

Vor die Aufgabe gestellt, ein Programm zu entwerfen und zu schreiben, steht der Entwickler in folgender Situation: er hat eine mehr oder weniger ungenaue Beschreibung dessen, was das Programm leisten soll, und eine ungefähre Vorstellung, wie die Aufgabe zu lösen sein könnte.
Die beste Methode, hier zu einer Lösung zu kommen, besteht darin, die Aufgabe grob in Teilaufgaben zu zerlegen; es sollten nicht mehr als 10-15 sein, damit man den Überblick behält. Die einzelnen Teilaufgaben werden ihrerseits wieder zerlegt; mit der Zerlegung von Teilaufgaben in Teilaufgaben wird so lange fortgefahren, bis alle Teilaufgaben so klein geworden sind, daß sie einzeln gelöst werden können. Wenn dies gelingt, ist damit die ganze Aufgabe gelöst.

Es kann jedoch geschehen, daß man bei dem Versuch, eine Aufgabe durch schrittweise Verfeinerung zu lösen, in eine Sackgasse gerät: für irgendeine Teilaufgabe ist weder eine Lösung noch eine weitere Zerlegung zu finden. Falls die Aufgabe überhaupt lösbar war, wird dies in der Regel dadurch verursacht, daß irgendeine frühere Zerlegung falsch durchgeführt worden ist. Wenn man solche Fehler finden will, ohne ganz von vorn beginnen zu müssen, muß man alle Verfeinerungen genau notiert haben.

Traditionsgemäß werden Programmabläufe graphisch, z.B. mit Ablaufplänen oder mit Struktoprogrammen dargestellt. Wenn man das auch bei der Aufgabenlösung durch schrittweise Verfeinerung tut, muß man entweder viel zeichnen oder Zwischenschritte weglassen. Beides ist schlecht: wenn man viel zeichnet, vergißt man die guten Ideen, welche man über den Fortgang der Verfeinerung hatte; wenn man Zwischenschritte nicht mitzeichnet, findet man die Stellen nicht, wo man in die Irre gegangen ist. Einzige Abhilfe: nicht zeichnen,

sondern aufschreiben, wie man sich den Programmablauf denkt. Am besten tut man das mit Befehlen, die aus ganzen Sätzen - auch mehreren - bestehen. Wenn solche Befehle mehrmals ausgeführt oder mal das eine, mal das andere getan werden muß, verwendet man Konstruktionen wie bei der Programmierung in einer höheren Sprache: Wiederhole .... Ende, Falls .... dann .... sonst. Mit anderen Worten: man sollte mit Pseudo-Code programmieren. Da man bei der schrittweisen Verfeinerung jeden Pseudo-Code-Befehl durch eine Folge von Pseudo-Code-Befehlen genauer erklären muß, verliert man aber ohne besondere Vorkehrungen sehr schnell die Übersicht, wie Pseudo-Code-Befehle und ihre erklärenden Pseude-Code-Befehlsfolgen zusammengehören. In den Programmiersprachen META-S und ELAN wird dieses Problem dadurch gelöst, daß einfach der Pseude-Code-Befehl noch einmal vor seiner Verfeinerung geschrieben wird. Das ist nicht gut: erstens muß man sehr viel zweimal schreiben, und zweitens ist es mühsam, beim Lesen eines längeren Programmes die Stelle zu suchen, wo genau der gleiche Satz noch einmal vorkommt.

Besser ist es, wenn man jeden Pseudo-Code-Befehl und seine verfeinernde Erklärung mit derselben Kennziffer versieht. Die Kennziffern wählt man so, daß man ihnen ansieht, zu welchem Teil des Gesamtproblems sie gehören: zum Befehl mit der Kennziffer 218F gehören in der Verfeinerung die Befehle mit Kennziffern 218F1, 218F2 usw.. Die Kennziffer schreibt man am besten links vor die Pseudo-Code-Befehle, damit beide dicht zusammen in einer Zeile stehen. Wenn man außerdem die Kennziffern vor den Befehlen etwas einrückt und die vor den zugehörigen Verfeinerungen direkt an den Rand schreibt, lassen sich zusammengehörende Kennziffern auch in längeren Programm-Entwürfen erstaunlich schnell finden, zumal sie ohnehin nicht völlig unsortiert stehen.

Nachdem man das Problem an allen Stellen durch Verfeinerung so präzisiert hat, daß man die Lösung genau kennt, könnte man das Programm in einer Programmiersprache niederschreiben. Das wäre wieder schlecht: Erstens wäre es dann schwierig, Programm und Pseudo-Code zu vergleichen, weil der Pseudo-Code in der Reihenfolge steht, wie man gedacht hat, während die Befehle des Programmes in der Reihenfolge stehen müssen, in der sie ausgeführt werden sollen. Zweitens wird man beim Programmtest feststellen, daß man an einigen Stellen doch falsch gedacht hat; dann ist die Versuchung sehr groß, nur das Programm zu ändern, und der Programm-Entwurf wird als Teil der Programm-Dokumentation unbrauchbar.

Besser ist es, wenn man die Programmiersprach-Zeilen und -Abschnitte als letzte Stufe der Verfeinerung zwischen den Pseudo-Code schreibt. Dann braucht man letzteren nur noch als Kommentar zu kennzeichnen und die Entwurfs-Zeilen in eine andere Reihenfolge zu bringen, um ein kompilierbares Programm zu erhalten, in dem der Pseudo-Code als Kommentar steht. Diese Arbeit ist so einfach, daß dafür ein Preprozessor geschrieben worden ist. Um dem seine Arbeit zu erleichtern, werden die Kennziffern mit einem führenden Sonderzeichen geschrieben.

Dieser Preprozessor kann gleichzeitig nach Fehlern suchen, die man beim Schreiben des Programm-Entwurfs gemacht haben könnte: Kennziffern müssen jeweils mindestens zweimal vorkommen und als Kommentar gekennzeichnet sein, Kommentar-Kennzeichen können vergessen oder zuviel notiert worden sein. Bei sehr umfangreichen Programmen erzeugt der Preprozessor auf Wunsch eine Kopie des Programm-Entwurfs, bei der Pseudocodes und Verfeinerungen durch Zeilennummer-Verweise auf den jeweils anderen Partner ergänzt sind.

Ein Beispiel

B i l d 1 zeigt den Anfang des Preprozessor-Programmentwurfs. Es ist ziemlich selbst-erklärend. Bemerkenswert ist, daß auf allen Ebenen der Verfeinerung bereits PEARL-Zeilen in den Pseudo-Code eingestreut sein können.

B i l d 2 zeigt einen Teil des daraus erzeugten PEARL-Programmes, dem in B i l d 1 die Zeile 21 entspricht. Die Zeilen-Numerierung hat sich gründlich geändert, weil der Preprozessor wegen der Reihenfolge der Kennziffern B i l d 1, Zeilen 7, 8, 12, 13, 14 Systemteil, Spezifikationen, Deklarationen und Prozeduren vor der Task eingeordnet hat. Er liest nämlich eine Verfeinerung nur solange monoton ein und gibt sie aus, solange sie aus PEARL-Code besteht. Eine Pseudocode-Anweisung wird ebenfalls eingelesen und ausgegeben; dann aber geht der Preprozessor zur Ausgabe der zugehörigen Verfeinerung über, wobei er darin enthaltene Pseudocode-Anweisungen genauso behandelt; am Ende jeder Verfeinerung wird in der jeweils darüber liegenden Ebene fortgefahren. Da das MODEND am Ende der obersten Verfeinerungsebene steht, kommt es automatisch an den Schluß des Programmes.

```
/* ENTWURF EINES PEARL-PROGRAMMES ZUR UMWANDLUNG EINES NACH TOP-DOWN-
METHODEN GESCHRIEBENEN PEARL-PROGRAMMENTWURFS IN EINE KOMPILIERBARES
PROGRAMM
VERSION 15.6/21.5.81
AUTOR:FREVERT
            */ MODULE PRPRO; /*
  #1        SYSTEMTEIL.
  #2        PROBLEMTEIL.
            */ MODEND; /*
================================================================================
#2          */ PROBLEM; /*
  #21       PASSIVE OBJEKTE
  #22       UNTERPROGRAMME
  #23       BEARBEITUNGSTASK
--------------------------------------------------------------------------------
#23         */ MAIN:TASK GLOBAL; /*
  #231      ERÖFFNE DATIONS UND FUEHRE STEUERDIALOG
  #232      LIES ENTWURF AUS STREAM-DATION UND BAUE ADRESSBUCH AUF.
              MELDE DABEI FEHLER
  #233      GIB AUF WUNSCH REFERENZEN-LISTE AUS
  #234      LIES ENTWURF IN RICHTIGER REIHENFOLGE DER ZEILEN UND GIB
              KOMPILIERBARES PROGRAMM AUS (AUF WUNSCH UND WENN DER ENTWURF
               KEINE GROBEN FEHLER ENTHAELT)
  #235      SCHLIESSE DATEIEN UND MELDE FEHLERZAHL.
            */ END; /*
#231        */
             OPEN TASTEN;
             OPEN SCHIRM;
              PUT 'WELCHE DATEI SOLL BEARBEITET WERDEN?' TO SCHIRM;
             GET ENTWURFSNAME FROM TASTEN;
             ENTWURFSDATEINAME:=ENTWURFSNAME)(' ')('-TX';
             ZIELDATEINAME:=ENTWURFSNAME)(' -P ';
             CALL BEFEHLSLESEN('VORUEBERSETZEN? :',OUTPUT);
             CALL BEFEHLSLESEN('REFERENZLISTING? :',  REFERENZLISTING);
             IF REFERENZLISTING THEN
             OPEN DRUCKER;
                PUT 'REFERENZLISTE VON ',ENTWURFSDATEINAME TO DRUCKER BY
                  SKIP,A,A,SKIP;
             FIN;
             OPEN ENTWURF BY IDF(ENTWURFSDATEINAME),OLD; /*
#232
  #232A       BAUE SORTIERTES ADRESSBUCH AUF
  #232B       UNTERSUCHE ADRESSBUCH AUF WIDERSPRUECHE ODER UNVOLLSTAENDIGKEITEN
            UND DRUCKE FEHLER AUS
#232A
  #232A1      BEREITE AUFBAUEN VOR
          */ REPEAT /*
  #232A2      UNTERSUCHE ZEILE FUER ZEILE UND BAUE ADRESSBUCH AUF
  #232A3      BIS DATEIENDE ERREICHT
  #232A4      BIS UEBERLAUFFEHLER PASSIERT
  #232A5      REPEATEND
#232A1
          */
             LETZTEREINTRAG:=0;
             SCHLUESSEL(1):=' ';
             FOR I TO ADRESSBUCHLAENGE  REPEAT
               ZEILENNOTIZ(I,1):=0;
               ZEILENNOTIZ(I,2):=0;
             END;
                      UEBERLAUFENDZEIGER:=0; /*
#232A2
  #232A21     ERHOEHE ZEILENZAEHLER, LIES ZEILE EIN
  #232A22     UNTERSUCHE ZEILE AUF UEBERSCHRIFTANFANG ODER KAPITELANFANG
          %TA24 DRUCKE ZWISCHENERGEBNISSE
  #232A23     IF UEBERSCHRIFTANFANG ODER KAPITELANFANG
  #232A24     THEN MACHE VERMERK IM ADRESSBUCH; FIN
  #232A25     MELDE FEHLER,FALLS ZUVIEL ODER ZUWENIG KOMMENTARANFAENGE
```

Bild 1. Anfang des Programm-Entwurfs für den Preprozessor. Pseudo-Code- und PEARL-Anweisungen sind auf allen Verfeinerungs-Stufen gemischt; letztere stehen zwischen "umgekehrten" Kommentarzeichen. Die links vor den Pseudo-Code-Anweisungen stehenden Kennziffern assoziieren jene mit den zugehörigen Verfeinerungen; dort sind die Kennziffern nicht eingerückt.

```
559. #23        */ MAIN%TASK GLOBAL%; /*
560.   #231     EROEFFNE DATIONS UND FUEHRE STEUERDIALOG
561. #231       */
562.             OPEN TASTEN%
563.             OPEN SCHIRM%
564.              PUT 'WELCHE DATEI SOLL BEARBEITET WERDEN?' TO SCHIRM%
565.             GET ENTWURFSNAME FROM TASTEN%
566.             ENTWURFSDATEINAME:=ENTWURFSNAME)(' ')('-TX'%
567.             ZIELDATEINAME:=ENTWURFSNAME)(' -P '%
568.             CALL BEFEHLSLESEN('VORUEBERSETZEN? :',OUTPUT)%
569.             CALL BEFEHLSLESEN('REFERENZLISTING? :',  REFERENZLISTING)%
570.             IF REFERENZLISTING THEN
571.             OPEN DRUCKER%
572.                PUT 'REFERENZLISTE VON ',ENTWURFSDATEINAME TO DRUCKER BY
573.                  SKIP,A,A,SKIP%
574.             FIN%
575.             OPEN ENTWURF BY IDF(ENTWURFSDATEINAME),OLD%; /*
576.   #232    LIES ENTWURF AUS STREAM-DATION UND BAUE ADRESSBUCH AUF.
577.              MELDE DABEI FEHLER
578. #232
579.   #232A       BAUE SORTIERTES ADRESSBUCH AUF
580. #232A
581.   #232A1      BEREITE AUFBAUEN VOR
582. #232A1
583.          */
584.             LETZTEREINTRAG:=0%
585.             SCHLUESSEL(1):=' '%
586.             FOR I TO ADRESSBUCHLAENGE  REPEAT
587.               ZEILENNOTIZ(I,1):=0%
588.               ZEILENNOTIZ(I,2):=0%
589.             END%
590.                        UEBERLAUFENDZEIGER:=0%; /*
591.          */ REPEAT /*
592.   #232A2      UNTERSUCHE ZEILE FUER ZEILE UND BAUE ADRESSBUCH AUF
593. #232A2
594.   #232A21     ERHOEHE ZEILENZAEHLER, LIES ZEILE EIN
595. #232A21       */
596.               CALL EINLESEN% /*
597.
598.   #232A22     UNTERSUCHE ZEILE AUF UEBERSCHRIFTANFANG ODER KAPITELANFANG
599.          %TA24 DRUCKE ZWISCHENERGEBNISSE
600. #232A22       */ CALL UNTERSUCHUNG % /*
601.
602.   #232A23     IF UEBERSCHRIFTANFANG ODER KAPITELANFANG
603. #232A23        */ IF UEBERSCHRIFTANFANG OR KAPITELANFANG /*
604.
605.   #232A24     THEN MACHE VERMERK IM ADRESSBUCH% FIN
606. #232A24       */ THEN /*
607.   #232A240    DIE KOMMENTARZAEHLUNG MUSSTE VOR EINLESEN DER ZEILE 0 GEWESEN
608.             SEIN. DANACH MUSS SIE 0 ODER 1 SEIN.
609. #232A240      */ IF KOMMENTARZAEHLUNG/=1 THEN
610.                IF KOMMENTARZAEHLUNG<1 THEN
611.                    CALL FEHLERMELDUNG('ZUWENIG KOMMENTARANFAENGE VOR ZEILE',
612.                                                    ZEILENR)%
613.                 ELSE
614.                    CALL FEHLERMELDUNG('ZUVIELE KOMMENTARANFAENGE VOR ZEILE',
615.                                       ZEILENR)%
616.                FIN%
617.              KOMMENTARZAEHLUNG:=1%
618.            FIN%
619.            /*
620.
621.   #232A241    IF UEBERSCHRIFTANFANG
622. #232A241      */ IF UEBERSCHRIFTANFANG /*
623.
624.   #232A242      THEN BEHANDLE UEBERSCHRIFT%FIN
625. #232A242
626.            */ THEN /*
```

Bild 2. Teil des aus dem Programm-Entwurf erzeugten Programmes. Pseudo-Code und Kennziffern bilden die Kommentare. Zeile 599 bezieht sich auf ein nur zum Test benötigtes Programmstück, das jetzt vom Preprozessor hinter dem MODEND eingeordnet worden ist; Rückänderung des % in # würde es in das Programm zurückholen.

In der Verfeinerungs-Hierarchie der Task erfolgt der Abstieg über die Kennziffern 23-231, dann die Rückkehr nach 232, erneuter Abstieg 232-232A-232A1, Rückkehr nach 232A2, Abstieg 232A2-232A21, Rückkehr nach 232A22, usw.. Einen besonderen Hinweis verdient Zeile 599 aus B i l d 2. Dort ist ein Programmstück aus der Testfassung des Programmes dadurch entfernt worden, daß in der Kennziffer das führende # in ein % geändert wurde. Würde diese Änderung im Entwurf rückgängig gemacht, würde der Preprozessor die Zeile als Pseudocode-Anweisung erkennen und die zugehörige Verfeinerung hinter ihr einordnen, während sie jetzt hinter dem MODEND stehen bleibt.

Erfahrungen

Bisher wurden der Preprozesser (ca. 1500 Zeilen) und ein Programm zur Steuerung einer Modelleisenbahn (10 Moduln mit insgesamt 20 Tasks und ca. 3000 Zeilen) mit dieser Methode entwickelt, die bei dem letzten Projekt auch für die Abfassung der Spezifikationen verwendet wurde. Dabei wurden folgende Erfahrungen gemacht.

* Die Programm-Entwicklung geht erstaunlich rasch; bei schwierigen Programmstücken und bei Änderungen in der Wartungsphase dürfte die Zeitersparnis gegenüber Entwerfen mit Struktogrammen über 50 % betragen.

* Unterbrechungen bei der Arbeit wirken sich bei weitem nicht so störend aus, wie sonst beim Programmieren, weil der Gedankenfluß bis zum Zeitpunkt der Unterbrechung lückenlos dokumentiert ist.

* Schwere Programmfehler sind fast nur da, wo Programmteile nicht mit schrittweiser Verfeinerung, sondern sofort in der Programmiersprache entwickelt wurden.

* Die Programm-Entwürfe dokumentieren die Denkprozesse des Entwicklers. Das hilft außerordentlich bei der Suche nach logischen Fehlern. Sie werden häufig bei nochmaligem Durchlesen gefunden.

* Man kann alle Teile des Programmes - Parallelarbeit, Daten - übersichtlich und gut gegliedert beschreiben; B i l d 3 zeigt einen Ausschnitt aus der Datenbeschreibung des Preprozessors, B i l d 4 einen Teil der Tasking-Beschreibung der Bahnsteuerung.

* Die Programme sind auch nach längerer Zeit noch leicht zu ändern, da sie sofort wieder verständlich sind.

```
377. #21
378.    #211     DATIONS
379.    #212     PROBLEMDATEN
380.    #213     HILFSDATEN
381.
382. #212
383.    #2121    KONSTANTEN
384.    #2122    ADRESSBUCH
385.    #2123    STACK
386.
387.
388. #2122
389.    #21221       SCHLUESSEL-NOTIERUNGEN
390.    #21222       NOTIERUNG DER PROGRAMMZEILEN, IN DENEN SCHLUESSEL VORKOMMEN
391.                 (JEWEILS GLEICHER INDEX WIE 21221)
392.    #21223       VERKETTUNG DER SCHLUESSEL IN ALPHABETISCH AUFSTEIGENDER REIHEN-
393.                  FOLGE
394.    #21224       GEMEINSAME ZEIGER, HILFSDATEN
395.    #21225       NOTIERUNG DER ZEILEN-EIGENSCHAFTEN, UM 2. DURCHGANG ZU
396.                 BESCHLEUNIGEN
397.
398.
399. #21221      */ DCL SCHLUESSEL (300) CHAR(10),
400.                     SCHLUESSELLAENGE INV FIXED INIT(10); /*
401.
402. #21222
403.    #212221   JEDER SCHLUESSEL KOMMT I. A. JE EINMAL VOR EINER UEBERSCHRIFT
404.              UND VOR EINEM KAPITELANFANG VOR. EINGETRAGEN WERDEN DIE JEWEILIGEN
405.              ZEILENNUMMERN.
406.    #212222   UEBERLAUFTEIL FUER EINTRAG MEHR ALS 2-MAL VORKOMMENDER SCHLUESSEL;
407.              IN DIESEM FALL WIRD IN 212221 EIN VERWEIS AUF DEN UEBERLAUFTEIL
408.               EINGETRAGEN (NEGATIVES VORZEICHEN). DER UEBERLAUTEIL ENTHAELT
409.              U.U. VERKETTUNGEN ZU WEITEREN EINTRAGUNGEN. KETTENENDE IST 0
```

Bild 3. Ausschnitt aus dem Entwurf des Preprozessors: Beschreibung des Adressbuches.

```
181. #U3      FUER JEDEN FAHRWEG GIBT ES JE ZWEI TASKS, DIE DAFUER SORGEN, DASS
182.          DER FAHRWEG FORTGESETZT WIRD, BZW., DASS ER TEILWEISE WIEDER
183.          FREIGEGEBEN WIRD.     EINE TASK, DIE IHREN FAHRWEG NICHT FORTSETZEN
184.          KANN, WARTET AUF EINEM REQUEST IN DER PROZEDUR AUFVERSUCHWARTEN
185.              (RELEASE IN PROZEDUR VERSUCHMACHENLASSEN, TEIL DER FREIGABE).
186.          FUER JEDEN SENSOR EXISTIERT EINE TASK, IN DER AUF DIE SENSORMELDUNG
187.          GEWARTET UND DIE MELDUNG BEARBEITET WIRD.
188.
189.          DA DIE BEARBEITUNG EINER SENSORMELDUNG UND DIE FAHRWEGFORTSETZUNG
190.          BEIDE DEN ZUGORT DES FAHRWEGES NEU EINTRAGEN, WERDEN SIE VORSICHTS-
191.          HALBER DURCH DEN SEMAPHOR KRITISCH1 BZW. KRITISCH2 SEQUENTIALISIERT.
192.
193.          AUSSERDEM EXISITERT FUER JEDEN FAHRWEG EINE UEBERWACHUNGSTASK,
194.          DIE EINE FEHLERMELDUNG VERURSACHT, WENN LAENGER ALS 30 SEC
195.          AUF EINE FAELLIGE SENSORMELDUNG GEWARTET WORDEN IST.
196.
197.          ALTERNATIV KOENNTE FUER JEDE DER FUNKTIONEN ANFORDERN, WEICHENSTELLEN
198.          USW. EINE EIGENE TASK GESCHRIEBEN WERDEN. DIESE TASKING-STRUKTUR
199.          WUERDE JEDOCH FUER AUF- UND ABBAU DER FAHRWEGE INSGESAMT 6 TASKS
200.          ERFORDERN.
201.
202.    #U31  FORTSETZUNGSTASKS FUER BEIDE FAHRWEGE
203.    #U32  FREIGABETASKS FUER BEIDE FAHRWEGE
204.    #U33  TASKS ZUR INTERRUPT-BEARBEITUNG
205.    #U34  UEBERWACHUNGSTASKS FUER BEIDE FAHRWEGE
206.
207. #U31
208.             */  WEG1:TASK PRIO 2;
209.                 CALL FORTSETZUNGSARBEIT(1);
210.          /*
211.    %UT61    TEST-SIMULATION
212.          */
213.               END;
214.
215.               WEG2:TASK PRIO 2;
216.                 CALL FORTSETZUNGSARBEIT(2);
217.          /*
218.    %UT62    TEST-SIMULATION
219.          */
220.               END; /*
221.    #U311 GEMEINSAME BEARBEITUNGSPROZEDUR FORTSETZUNGARBEIT
```

Bild 4. Ausschnitt aus dem Programmentwurf für die Steuerung einer Modellbahn: Beschreibung des Tasking.

* Nur für den Test benötigte Programmteile lassen sich fast vollautomatisch aus den Programmen entfernen und später bei Wartungsarbeiten wieder in die Programme einordnen.

* Der Rückwärts-Bezug vom kompilierbaren Programm zum Programm-Entwurf ist wegen der Kennziffern sehr leicht.

* Die Entwürfe konnten so geschrieben werden, daß kritische Abschnitte aus einer einzigen Pseudocode-Anweisung bestanden. Dadurch ist es möglich, Operationen auf denselben Semaphor eng benachbart niederzuschreiben. Mögliche Verklemmungen werden so leicht erkannt.

* Das Top-down-Verweis-System ist zur übersichtlichen Gliederung aller verbalen Beschreibungen, z.B. auch für Grob- und Fein-Spezifikationen, gut geeignet; der Preprozessor ermöglicht deren Prüfung auf formale Vollständigkeit.

* Fein-Spezifikationen und Programm-Entwurf lassen sich zu einem einzigen Dokument integrieren; dadurch läßt sich das Nicht-Übereinstimmen nach dem Programm-Test leichter vermeiden.

* Last not least: der Lernaufwand für die Formalien der Methode ist so gering, daß sich ihre Einführung auf freiwilliger Basis durchführen läßt.

Diesen Vorteilen stehen nur zwei Nachteile gegenüber:

* Nach Programmänderungen ist zusätzlich zur Kompilation usw. ein Preprozessor-Lauf erforderlich.

* Möglichkeiten zu lokaler Optimierung sind im Entwurfs-Dokument schwer zu erkennen, weil nacheinander auszuführende PEARL-Anweisungen in diesem oft weit voneinander entfernt stehen.

Schlußbemerkungen

Es dürfte wesentlich zur Schnelligkeit und zum Erfolg kreativer Arbeit beitragen, wenn kein Zwang zur Ausarbeitung unwichtiger Details oder zu umständlichen Darstellungsmethoden besteht. Insofern ist es nicht

überraschend, daß eine Papier- und -Bleistift-Methode, bei der zunächst nur das Wichtigste möglichst schnell notiert zu werden braucht, gute Ergebnisse liefert.

In den Programmentwürfen sind dieselben Sachverhalte mehrfach beschrieben: in verschiedenen Verfeinerungsstufen des Pseudocodes, zuletzt in der Programmiersprache. Angesichts dieser Redundanzen ist es wiederum kein Wunder, daß die Programme leichter verstanden und Fehler leichter gefunden werden.

Anhang: Technische Daten der Preprozessor-Implementation

Der Preprozessor wurde unter Verwendung von Bais-PEARL entwickelt. Auf der Krupp-Atlas EPR-1300 belegt er insgesamt ca. 8 K 16-Bit-Worte, von denen knapp 4 K für ein Adreßbuch benötigt werden, das durch folgende (leicht änderbare) technische Daten bestimmt wird:

Maximale Zeilenlänge des Inputs:	80 Zeichen
Maximale Zeilenzahl des Inputs:	3000 Zeilen
Maximale Kennzifferlänge:	10 Zeichen
Maximale Anzahl d. Kennziffern:	300
Maximale Verfeinerungstiefe:	20

Er benötigte auf der EPR-1300 zur Erzeugung eines kompilierbaren Programmes aus seinem eigenen Entwurf ca. 6 Minuten. Diese Zeit konnte durch Übergang von Einzelzeichen-Verarbeitung auf Substrings auf die Hälfte gesenkt werden. Die Bearbeitungszeit hängt wesentlich vom E/A-System und vom verwendeten Massenspeicher ab, sie dürfte auch auf anderen Systemen etwa in der Größenordnung der für eine Kompilation benötigten Zeit liegen.
Von den 1500 Zeilen des Preprozessors enthalten ca. 300 die hierarchisch gegliederte Spezifikation, ca. 180 Zeilen Pseudocode und ca. 600 Zeilen PEARL-Code. Eine für einen Bootstrap zusammengestrichene PEARL-Version würde ca. 400 Zeilen lang sein.
Komplette Listings von Preprozessor-Entwurf und kompilierbarem Programm werden auf Wunsch zugeschickt.

Literatur

/1/ META Sprachbeschreibung: Krupp-Atlas-Elektronik, Bremen

/2/ Rainer Hahn, Peter Stock: ELAN Handbuch; Akademische Verlagsgesellschaft Wiesbaden (1979)

/3/ G.Hommel (Hrsg.): Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage; Kernforschungszentrum Karlsruhe GmbH, KfK-PDV 186 (1980)