P C S L und E S P R E S O -

zwei Ansätze zur Formalisierung der Prozeßrechner-Softwarespezifikation

Jochen Ludewig
Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik

1. Einleitung

Noch immer fehlen Mittel und Methoden, die es dem Software-Entwickler ermöglichen, ein Programmsystem so zu planen, zu spezifizieren und zu entwerfen, daß seine Annahmen, Folgerungen und Entscheidungen ihm selbst und anderen verständlich und prüfbar sind. Da von solchen Mitteln und Methoden, die nachfolgend einheitlich als "Spezifikationssysteme" bezeichnet werden, eine wesentliche Verbesserung der Software-Qualität zu erwarten ist, wurde im IDT etwa seit 1976 auf diesem Gebiet gearbeitet. Zunächst wurde festgestellt, welche Spezifikationssysteme bisher verfügbar oder vorgeschlagen sind (Ludewig, Streng, 1978) und eines davon installiert, das PSL/PSA-System vom ISDOS-Project an der University of Michigan (Teichroew, Hershey, 1977). Später wurde PCSL entwickelt (Process Control Software Specification Language, Ludewig, 1980a), eine Variante von PSL, die auf die Entwicklung von Prozeßrechner-Software zugeschnitten ist. Schließlich wurden die Konzepte von PCSL, die teilweise mit den durch die ISDOS-Software vorgegebenen Randbedingungen kollidierten, in ein völlig neues System eingebracht, das den Namen ESPRESO (System zur Erstellung der Spezifikation von Prozeßrechner-Software) erhielt (Ludewig, 1980c). Der folgenden Darstellung liegt diese am weitesten entwickelte Form zugrunde; die Unterschiede zwischen ESPRESO und PCSL sind in Abschnitt 9 skizziert.

2. Thesen zur Spezifikation

Jedem Versuch, ein Spezifikationssystem zu entwickeln, liegen - nicht immer bewußte - Annahmen über den Prozeß der Programmentstehung zugrunde. Die wichtigsten waren in diesem Fall:

a) Die Aufgabenstellung liegt zunächst nur in einer unvollständigen, informalen und möglicherweise inkonsistenten Form vor. Aus ihr wird eine präzisere Beschreibung der Anforderungen, die Spezifikation, entwickelt. Damit sind meist Entscheidungen darüber verbunden, wie die Problemlösung beschaffen sein soll. (D.h. die vielbeschworene Trennung zwischen WAS und WIE läßt sich bei den meisten realen Problemstellungen nicht durchhalten.)

b) Die Spezifikation hat während ihrer Entwicklung im allgemeinen Mängel verschiedener Art: Sie ist unvollständig, gibt Anforderungen inkorrekt wieder, enthält Widersprüche und ist nur teilweise formal.

c) Der Spezifikationsprozeß verläuft nicht in einer fest planbaren Reihenfolge wie z.B. erst alle Daten, dann die Ablaufstrukturen, oder Abstraktionsebene i vollständig vor Abstraktionsebene i+1.

d) Die Verständigung zwischen Auftraggeber und Software-Entwickler (oder deren Vertretern) findet auf der Grundlage der Spezifikation statt.

e) Die Akzeptanz eines Spezifikationssystems ist von allen seinen Komponenten (Sprache, Werkzeug, Methode) abhängig.

f) Der Weg, den der Software-Entwickler einschlägt, wird wesentlich durch die Spezifikationssprache beeinflußt.

Aus diesen Thesen folgen einige Forderungen an ein Spezifikationssystem:

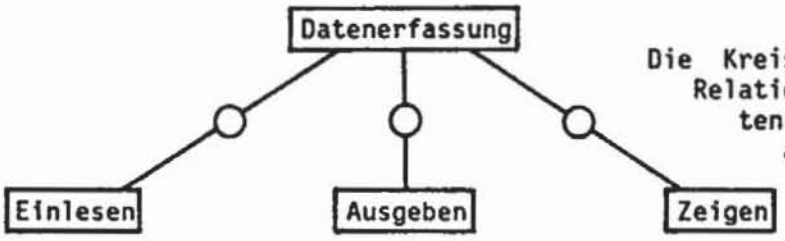
- Es muß informale und unvollständige Spezifikationen akzeptieren,
- ihre schrittweise Vervollständigung unterstützen,
- dabei die Reihenfolge der Präzisierung offenlassen,
- bei der Suche nach Mängeln helfen,
- die Kommunikation und Dokumentation unterstützen,
- bequem zu handhaben und leicht verständlich sein,
- den Anwender zu Lösungsansätzen führen, die sich leicht in gut strukturierte Programme umsetzen lassen.

Hinzu kommt hier die spezielle Ausrichtung auf Prozeßrechner-Software.

3. Das Beschreibungsschema

Die Beschreibung des zu entwickelnden Programms einschließlich seiner Schnittstellen zur Umgebung (Bediener, technischer Prozeß, andere Programme) erfolgt durch einen bipartiten Graphen, dessen Knoten die Objekte und ihre Verknüpfungen sind. Jedem Objekt ist eine Art, jeder Verknüpfung eine Relation zugeordnet; die Mengen der Arten und der Relationen sind vorgegeben. Nur die Objekte sind durch Namen identifiziert.

Beispiel: "Datenerfassung" bestehe aus der parallelen Ausführung von "Einlesen", "Ausgeben" und "Zeigen" (siehe 7). Alle Objekte haben die Art "Block".



Die Kreise sind Verknüpfungen der Relation "parallel in". Die Kanten nach oben zeigen jeweils auf den "Vater", die anderen auf den "Sohn".

Die Relation bestimmt, wieviele Kanten von der Verknüpfung ausgehen, durch welche Selektoren sie unterschieden sind und welche Arten die Objekte am Ende der Kanten haben können. Im Beispiel oben könnte der Vater auch die Art Prozedur haben, die Söhne müssen Blöcke sein. Objekte können jeweils mit beliebig vielen Verknüpfungen verbunden sein.

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,
- der Trigger (trigger) als Sonderform des Puffers, bei dem die Botschaften keine Information enthalten,
- das Betriebsmittel (resource), das bei der Ausführung der Prozeduren nach Bedarf belegt wird.

Variablen, Puffer, Trigger und Betriebsmittel sind begrifflich zusammengefaßt als Medien. Sie werden in Abschnitt 5 näher beschrieben.

Die wichtigsten Relationen erlauben die Beschreibung

- der Zuordnung von Objekten der meisten Arten (einschließlich Modul) zu Moduln, wodurch ihr Gültigkeitsbereich definiert wird,
- der Ablauforganisation von Prozeduren und Blöcken (parallele, sequentielle oder alternative Ausführung untergeordneter Blöcke, Prozeduraufruf, siehe 4),
- der Datenflüsse und der Betriebsmittelverwaltung und damit - implizit - der notwendigen Koordination simultaner Zugriffe auf Medien (siehe 5).

4. Ablaufstrukturen

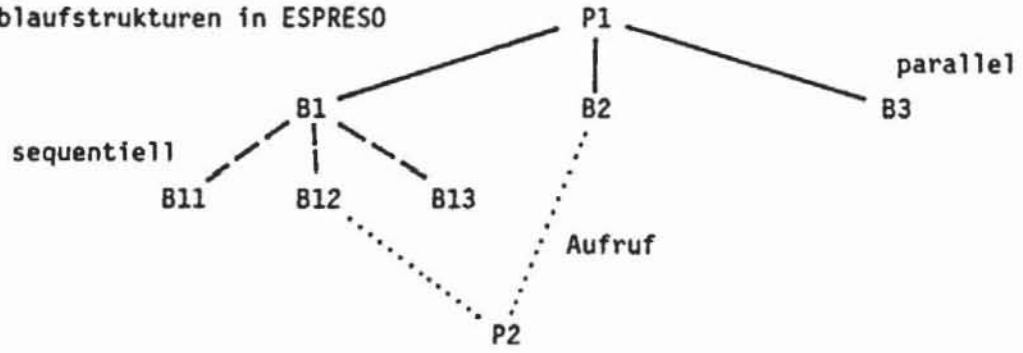
Die Spezifikation soll die Möglichkeiten der Implementierung nicht unnötig einschränken. Daher sind Spezifikationssprachen meist nicht-prozedural (Leavenworth, Sammet, 1974), d.h. sie legen nicht fest, in welcher Reihenfolge die Systemkomponenten ablaufen. Da auch nicht verlangt wird, daß die Spezifikation operationell, also ausführbar ist, kann sie auf statische Aspekte konzentriert werden. Dies gilt aber nicht, wo - wie bei den Prozeßrechner-Programmen - die Dynamik selbst in den Anforderungen eine wesentliche Rolle spielt.

Daher muß eine Spezifikationssprache für diesen Bereich Möglichkeiten zur Beschreibung der Abläufe bieten. In ESPRESO werden die ausführbaren Teile der Programme durch Prozeduren und Blöcke beschrieben. Die Ausführung einer Prozedur oder eines Blocks kann bestehen aus

- der parallelen Ausführung untergeordneter Blöcke,
- der sequentiellen Ausführung untergeordneter Blöcke,
- der alternativen Ausführung eines von mehreren untergeordneten Blöcken,
- der Ausführung (genau) einer Prozedur.

Durch Strukturierungsregeln ist sichergestellt, daß die Blöcke Bäume bilden, deren Wurzeln auch Prozeduren sein können. (In der fertigen Spezifikation müssen es Prozeduren sein.) Jeder Block gehört nur einem Baum an.

Beispiel: Ablaufstrukturen in ESPRESO



Die wiederholte Ausführung einer Prozedur oder eines Blocks wird durch die Angabe einer Wiederholbedingung beschrieben.
Bei paralleler Ausführung ist die Synchronisation am Schluß impliziert. (Im Beispiel oben endet P1, wenn B1, B2 und B3 beendet sind.)

Die formale Gleichbehandlung sequentieller und paralleler Abläufe, wie sie ähnlich schon in RSL (Alford, 1977) zu finden ist, trägt bei zur Erhaltung der im Problem grundsätzlich vorhandenen Parallelität, wodurch die Spezifikation einfacher wird und die Möglichkeiten der Implementierung weniger beschneidet (McKeag, Milligan, 1980).

5. Datenflüsse und Betriebsmittelverwaltung

Für Prozeßrechner-Programme ist es typisch, daß die Kommunikation und Koordination verschiedener Komponenten untereinander und mit der Umgebung einen wesentlichen oder sogar den zentralen Teil der Probleme schaffen. Daher wurde bei der Entwicklung von PCSL und ESPRESO nach Möglichkeiten gesucht, dem Anwender Sprachmittel zu geben, die

in den praktisch vorkommenden Aufgabenstellungen einfach zu gebrauchen sind, eine klare Bedeutung haben und Fehler vermeiden helfen. Grundlage des gewählten Konzepts ist die Beobachtung, daß drei charakteristische Situationen immer wieder auftreten:

- Statische Information (Zustandsgrößen, Betriebsparameter, Schalterstellungen, Anzeigelampen) wird vom Programm gelesen und geschrieben.
- Dynamische Information (Botschaften, Meßgrößen, Meldungen) fließt durch das System und wird dabei transformiert.
- Logische Betriebsmittel (Datenstrukturen oder Teile davon, Prozeduren, die nicht wiedereintrittsfest sind) und reale Betriebsmittel (Speicherplätze, Bandgeräte usw.) werden vorübergehend belegt, d.h. für andere Zugriffe gesperrt.

In allen drei Fällen ist eine Koordination der Zugriffe notwendig. Auf Code-Niveau gibt es dafür primitive Sprachmittel wie Semaphore, deren Gebrauch aber schwierig ist und zu schwer erkennbaren Fehlern führt. Daher werden hier durch komplexe Operationen Koordination und Datentransfer zusammen beschrieben. Folgende Relationen gibt es für diesen Zweck:

Relation	anwendbar auf Medien der Art	impliziert die Koordination der Zugriffe nach
lesen und schreiben	Variable	"2nd reader/writer problem" (Courtois et al., 1971)
herstellen und verbrauchen	Puffer, Trigger	Hersteller/Verbraucher-Schema
belegen und freigeben	Betriebsmittel	Prinzip des wechselseitigen Ausschlusses

Da die üblichen Programmiersprachen keine Puffer und Trigger oder entsprechende Konstruktionen enthalten, muß sich der Anwender zunächst an den Gebrauch gewöhnen; er wird jedoch bald feststellen, daß die gängigen Probleme mit diesen Mitteln äußerst einfach und klar beschrieben werden können. Wo immer Information weitergereicht und verarbeitet wird (z.B. Meßdaten oder Meldungen), ist ein Puffer der Kommunikationskanal. In der Prozeßdatenverarbeitung entsteht auf diese Weise oft eine Kette von Prozessen, die verbunden sind durch Puffer. Nur einer der Prozesse muß explizit von außen angestoßen werden (in der Regel durch eine Uhr) alle anderen werden durch die Puffer gesteuert. Sie laufen zyklisch ab und werden vorübergehend blockiert, wenn sie aus einem leeren Puffer abnehmen oder an einen vollen liefern wollen.

Die Trigger befördern nur Marken, mit denen wie in Petri-Netzen keine Information verbunden ist. Sie ersetzen die in anderen Spezifikations- und Programmiersprachen zu findenden Ereignisse, deren Wirkung oft unklar bleibt.

Trigger können (einmalig oder zyklisch) selbst Marken generieren, gesteuert durch die Uhr. Damit werden Wartezeiten, Abfragezyklen usw. beschrieben.

Für Puffer und Trigger können als Charakteristika spezifiziert werden

- die Kapazität (Zahl der Plätze),
- die Ausgabestrategie (FIFO, nach Priorität oder beliebig),
- das Verhalten bei Überlauf
  (Blockieren des liefernden Prozesses oder Verlust der Lieferung).

"critical regions" werden beschrieben durch die Belegung der Betriebsmittel, und zwar stets für die Dauer einer Prozedur- oder Blockausführung. Auf diese Weise sind Klammerungsfehler (vergessene Freigabe o.ä.) automatisch ausgeschlossen.

Das Konzept der Variablen und Puffer ist angelehnt an das der "pools" und "channels" in MASCOT (Jackson, Harte, 1976).

6. Die Spezifikationssprache

Für die Beschreibungskonzepte, wie sie zum Teil oben skizziert sind, wurde die Spezifikationssprache ESPRESO-S entwickelt. Dabei waren die wichtigsten Ziele

- gute Erlernbarkeit und Verständlichkeit,
- Flexibilität im Sinne der Forderungen, wie sie unter 2 genannt sind,
- einfache Implementierung eines Systems, das ESPRESO-S verarbeitet (siehe 8).

Entsprechend der letzten Forderung wurde ESPRESO-S so definiert, daß eine Spezifikation in einem einzigen Durchlauf analysiert werden kann, wobei vom Rechtskontext nur ein Grundsymbol benötigt wird (1-Pass-Analysierbarkeit, LL-1, vgl. Eckert, 1980).

Erlernbarkeit und Verständlichkeit werden gefördert durch ein einfaches Grundschema der Syntax: Jedes Objekt wird durch eine Sektion beschrieben, die entweder nur den Sektionskopf oder auch -rumpf und -schwanz enthält. Beispiele sind etwa

```
variable Datum.
```

```
procedure Datenerfassung:
parallel block Einlesen
parallel block Ausgeben
parallel block Zeigen
end Datenerfassung.        (* vgl. 3 *)
```

Im rechten Beispiel enthält der Rumpf drei weitere Sektionen, die ihrerseits wieder Rumpf und Schwanz enthalten könnten; die Syntax ist also rekursiv. Dadurch ist es möglich, Hierarchien in der Aufschreibung unmittelbar durch die Struktur darzustellen.

Die Flexibilität ist gewährleistet durch die (bei Programmiersprachen nicht übliche) Möglichkeit, frühere Angaben zu wiederholen und zu ergänzen. Im Beispiel könnte also

die Sektion "Datenerfassung" mit gleichem oder anderem Inhalt des Rumpfs (aber natürlich ohne Widersprüche) an anderer Stelle der Spezifikation erneut erscheinen.

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.
Für die Aufnahme auch solcher Texte in die Spezifikation, die nicht an spezielle Prozeduren, Blöcke, Medien usw. gebunden sind, gibt es in ESPRESO-S die Art "Text-Objekt". Diese Objekte können höchstens durch Querverweise in den Texten mit anderen Objekten formal verbunden sein. Das folgende Beispiel zeigt ein Text-Objekt mit zwei Texten; der zweite enthält den Querverweis auf ein Objekt KONV.

```
informal Termine:
text Absprachen ¢ Lieferung bis ...            ¢;
text Planung    ¢ Ende XII/80 soll der Modul !KONV installiert sein ¢
end Termine.
```

Eines der Ziele bei der Entwicklung von ESPRESO war, die Spezifikationssprache weitestgehend vollständig formal zu definieren. Daher wurde sie mit einer Erweiterten Attribut-Grammatik (Watt, Madsen, 1977) beschrieben. Auf diese Weise war es möglich, nicht nur die kontextfreie und, mittels der Attribute, die kontextsensitive Syntax anzugeben, sondern es konnte ohne erheblichen Mehraufwand auch dargestellt werden, wie sich die Eingabe einer Spezifikation auf den Inhalt der ESPRESO-Datei (siehe 8) auswirkt. Bei der Implementierung gab es dadurch nur Zweifelsfälle, wo die Grammatik Fehler enthielt.

Die Bedeutung der Spezifikation im Sinne des Benutzers, der ja ein operationelles Programm entwickeln will, ist beschrieben durch eine Abbildung von ESPRESO-S in die (hypothetische) Programmiersprache E-PASCAL. Der bei der Abbildung entstehende Code ist naturgemäß unvollständig, da alle arithmetischen und logischen Ausdrücke fehlen. Die Ablaufstrukturen sind jedoch vorhanden. Sie enthalten Bausteine, deren Außenschnittstellen auf die spezifizierten Medienzugriffe beschränkt sind.

7. Ein Beispiel für ESPRESO-S

Ein größeres Beispiel für ESPRESO-S ist in (Ludewig, 1980c) enthalten. Hier wird ein Datenerfassungssystem skizziert, das in (Ludewig, 1980b) schon mit PCSL beschrieben wurde.

```
procedure Datenerfassung:
¢ liest die Daten eines (hier nicht näher beschriebenen) technischen
  Prozesses alle 3 s ein, dazu die Eichfaktoren sowie die Zeit. Die
  Daten werden in physikalische Einheiten umgerechnet, gefiltert und
  archiviert. Auf Knopfdruck zeigt ein Bildschirm die neuesten Werte. ¢
end Datenerfassung.
```

Der Text zwischen den Cent-Zeichen hat den leeren Schlüssel; das Wortsymbol text kann nach dem Doppelpunkt entfallen. Die informalen Angaben werden wie folgt formalisiert:

```
reads Prozeßdaten, Eichfaktoren, Zeit;
started-by Meßzyklus, Knopf;
produces Archiv-Daten;
writes Bildschirm where ¢ neueste Archiv-Daten jeweils nach Knopfdruck ¢
```

An der Verwendung von "read" für die Prozeßdaten, die Eichfaktoren und die Zeit ist sichtbar, daß es sich um Variablen handelt. Sie werden beliebig oft gelesen und - davon unabhängig - überschrieben: die Prozeßdaten durch Änderungen im Prozeß, die Zeit durch die Hardware-Uhr und die Eichfaktoren durch andere Programme. Die Archivdaten stehen dagegen in einem Puffer ("produce"): Jedes Schreiben erzeugt einen zusätzlichen Datensatz, und es droht ein Puffer-Überlauf, wenn nicht von Zeit zu Zeit Platz geschaffen wird durch einen anderen, abnehmenden Prozeß (der konkret beispielsweise durch einen Operator gebildet werden kann, der ein volles Band gegen ein leeres tauscht, oder von einem Prozeß, der alle 24 Stunden die Daten, die sieben Tage alt sind, löscht).
Der Text zur writes-Angabe ist an den Medien-Zugriff gebunden. Dies ist eine spezielle Form des Text-Objekts: Sein Name ist durch den Kontext bestimmt, der Schlüssel ist leer. Es dient zur Angabe der Annahmen und Zusicherungen, die mit dem Datentransfer verbunden sind.

Damit ist aber die Dynamik noch nicht beschrieben. Dies geschieht durch Verfeinerung der Prozedur. Die Zugriffe auf Medien wandern in die untergeordneten Blöcke:

```
procedure Datenerfassung:
parallel
   block Einlesen:
   while DE-eingeschaltet;
   reads Prozeßdaten, Eichfaktoren, Zeit;
   started-by Meßzyklus;
   produces Rohwerte
   end Einlesen
parallel
   block Ausgeben:
   while DE-eingeschaltet;
   consumes Rohwerte;
   produces Archiv-Daten;
   writes aktuelle-Werte
   end Ausgeben
parallel
   block Zeigen:
   while DE-eingeschaltet;
   reads aktuelle-Werte;
   started-by Knopf;
   writes Bildschirm where ¢ neueste Archiv-Daten jeweils nach Knopfdruck ¢
   end Zeigen
end Datenerfassung.
```

Durch Angabe einer Wiederholbedingung ("DE-eingeschaltet", eine Variable des Typs boolean) ist impliziert, daß die Blöcke jeweils zyklisch ablaufen. (Wäre stattdessen die übergeordnete Prozedur zyklisch, so könnte keiner der Blöcke öfter als die anderen ausgeführt werden.)

Hinzugekommen ist ein Puffer für Rohwerte. Der Block "Ausgeben" wird über diesen Puffer gesteuert. "Zeigen" ist dagegen über die Variable "aktuelle-Werte" mit "Einlesen" verbunden, denn beide Prozesse laufen völlig entkoppelt voneinander ab.

Die Spezifikation muß jetzt vervollständigt werden durch Angabe der Modulstruktur und Definition der Medien.

```
module Datenerfassungssystem:
comprises
   procedure Datenerfassung,

   module Einlese-Modul: comprises block Einlesen end,
   module Ausgabe-Modul: comprises block Ausgeben end,
   module Display-Modul: comprises block Zeigen   end,

   buffer Rohwerte:
   of-type Meßwerte-Block;
   capacity 2;
   produce restricted-to Einlese-Modul;
   consume restricted-to Ausgabe-Modul
   end Rohwerte
end Datenerfassungssystem.
```

In diesem Beispiel fehlen noch die Beschreibungen der übrigen Medien, Typen usw., außerdem Texte mit erklärenden Hinweisen, die normalerweise zuerst entstehen.

8. Das Spezifikationswerkzeug

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).

Die Arbeit an einer Spezifikation beginnt mit dem Anlegen einer neuen ESPRESO-Datei unter einem frei gewählten Namen. In der Folgezeit werden alle Informationen, sobald sie anfallen, in ESPRESO-S formuliert und eingegeben. Dabei sind Wiederholungen unschädlich.
Soll die bereits eingegebene Information modifiziert werden, so werden die vom Benutzer gewählten Teile dekonvertiert und in der ESPRESO-Datei automatisch gelöscht. Er editiert die Ausgabe (ohne Unterstützung durch ESPRESO-W) und konvertiert wieder. Das Löschen beim Dekonvertieren beugt der Entstehung von Widersprüchen vor. Auch beim Konvertieren wird der gewandelte Teil der Spezifikation in der Eingabe gelöscht (nach automatischem Anlegen einer Sicherheitskopie). Dadurch bleiben nur die aufgrund von Fehlern nicht verarbeiteten Angaben übrig.

ESPRESO-W ist so konzipiert, daß es auf einem Kleinrechner implementiert werden kann.

9. Unterschiede von PCSL gegenüber ESPRESO-S

PCSL ist älter als ESPRESO-S und enthält daher noch nicht alle hier beschriebenen Konzepte, insbesondere nicht das der Moduln. Daher können keine Gültigkeitsbereiche definiert werden. Die Wortsymbole sind überwiegend unterschiedlich in den beiden Sprachen. Der wichtigste Nachteil von PCSL ist jedoch die nicht rekursive Syntax; dadurch wird die Beschreibung hierarchischer Strukturen umständlich und weniger übersichtlich. Inkonsistenzen, die in ESPRESO-S durch die Grammatik ausgeschlossen sind, lassen sich in PCSL nicht abfangen. Dazu wären spezielle Prüfmittel erforderlich, die bisher nicht existieren.

Eine genauere Darstellung von PCSL ist in (Ludewig, 1980a) enthalten.

10. Stand der Arbeit (Oktober 1980)

Zur Arbeit mit PCSL ist der sogenannte Generalized Analyzer erforderlich, ein umfangreiches Programmsystem (ca. 55000 Zeilen FORTRAN IV), das im IDT seit 1979 verfügbar ist. Die neueste Version der Sprache trägt die Bezeichnung PCSL.3.

Die Definition der Sprache ESPRESO-S ist abgeschlossen (Ludewig, 1980c). Das Programmsystem ESPRESO-W ist in den wichtigsten Teilen (Konvertierung, Dekonvertierung und Verwaltung der ESPRESO-Dateien) nach dem Prinzip der abstrakten Datenstrukturen entworfen, codiert und getestet, kann jedoch aufgrund von Schwierigkeiten mit dem PASCAL/360-Compiler nicht wie ursprünglich geplant auf dem IBM-Großrechner installiert werden. Daher wird zur Zeit eine Implementierung auf SIEMENS 330 vorgenommen, die Anfang 1981 zur Verfügung stehen soll.

Gegenwärtig wird im IDT die Spezifikation eines Reaktorschutzrechnersystems in ESPRESO-S vorgenommen.

11. Literaturangaben

Alford, M. (1977):
A requirements engineering methodology
for real-time processing requirements.
IEEE Trans. Software Eng., SE-3, 60-69.

Courtois, P.J., F. Heymans, D.L. Parnas (1971):
Concurrent control with 'readers' and 'writers'.
Commun. ACM, 14, 667-668.

Eckert, K. (1980):
Implementierung eines Spezifikationssystems für Prozeßrechner-Software.
Diplomarbeit, Universität Karlsruhe.

Jackson, K., H.F. Harte (1976):
The achievement of well-structured software in real-time applications.
Proc. of the IFAC/IFIP workshop on real-time programming,
Rocquencourt, Juni 1976. pp.229-238.

Ludewig, J., W. Streng (1978):
Überblick und Vergleich verschiedener Mittel
für die Spezifikation und den Entwurf von Software.
KfK 2506.

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

Ludewig, J. (1980b):
Process control software specification in PCSL.
in Haase, V.(ed.): IFAC/IFIP Workshop on requirement specification.
Graz, 14.-16. April 1980. Pergamon Press, erscheint etwa Dezember 1980.

Ludewig, J. (1980c):
Zur Erstellung der Spezifikation von Prozeßrechnersoftware.
Dissertation, TU München; erscheint ebenfalls als KfK 3060.

McKeag, R.M., P. Milligan (1980):
An experiment in parallel program design.
Software Practice and Experience, 10, 687-693.

Teichroew, D., E.A. Hershey III (1977):
PSL/PSA: a computer-aided technique for structured
documentation and analysis of information processing systems.
IEEE Trans. Software Eng., SE-3, 41-48.

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