PEARL Rundschau

Inhalt

Vorwort 1

D. Pflügel
Die Bedeutung von PEARL in der Prozeßinformatik 3

K. Nagel, H. Nebel
Didaktische und methodische Prinzipien und Leitlinien für die Kompaktseminar-Reihe „Einführung in die Prozeßdatenverarbeitung" 8

H. Rzehak
PEARL-Kompilersysteme für den Lehr- und Übungsbetrieb an Universitäten 33

H. Brinkkötter, K. Rebensburg
Aufbau und Durchführung einer Anwenderschulung 38

K. Fasold
PEARL als erste Programmiersprache im Grundstudium der Studienrichtung Elektrotechnik (Informationsverarbeitung) 44

P. Holleczek, K. Pelz
PEARL-Kurs und Praktikum an der Universität Erlangen-Nürnberg 51

E. Joho, C. Riewerts
PEARL in der Lehre am Institut für Regelungstechnik und Prozeßautomatisierung 55

Kurzmitteilungen 66

Neue Mitglieder 67

PEARL-Tagung '81 68

November 1981 Band 2 Nr. 4

PEARL Rundschau

November 1981
Band 2
Nr. 4

Der PEARL-Verein e.V. (PEARL-Association) hat das Ziel, die Verbreitung der Realzeitprogrammiersprache PEARL (Process and Experiment Automation Realtime Language) und ihre Anwendung sowie die Einheitlichkeit von PEARL-Programmiersystemen zu fördern.
Sitz des Vereins ist Düsseldorf. Seine Geschäftsstelle befindet sich im VDI-Haus, Graf-Recke-Straße 84, 4000 Düsseldorf 1.

Vorstand des PEARL-Vereins:

Prof. Dr.-Ing. R. Lauber Institut für Regelungstechnik und Prozeßautomatisierung Seidenstraße 36 7000 Stuttgart 1	Vorsitz, zuständig für Technik und Normung
Dipl.-Ing. G. Müller Brown Boveri und Cie. AG Leiter der Vertriebsabteilung Netzführungssysteme, SI/NV 1 Fred-Joachim-Schoeps-Str. 55 6800 Mannheim	stellv. Vorsitz zuständig für Organisation und Finanzen
Dr.-Ing. P. Elzer DORNIER System GmbH Abt. EEA Postfach 1360 7990 Friedrichshafen	zuständig für Öffentlichkeitsarbeit und Marketing

Herstellung durch die VDI-Verlag GmbH, Düsseldorf

Die PEARL-Rundschau ist das Mitteilungsblatt des PEARL-Vereins e.V. Neben Vereinsangelegenheiten werden für alle PEARL-Interessenten Informationen über Erfahrungen, Anwendungsmöglichkeiten und Produkte gegeben.
Preis des Einzelheftes für Nichtmitglieder: DM 15,—, für Studenten DM 5,—. Jahresabonnement DM 75,—.
Zur Veröffentlichung werden sowohl fachliche Abhandlungen über Realzeit-Anwendungen, als auch Kurznachrichten zu Themen des Prozessrechnereinsatzes und der Verwendung höherer Sprachen angenommen, soweit sie für PEARL-Interessenten von Bedeutung sein können.

Es obliegt dem Autor, die Rechte zur Veröffentlichung seines Beitrags in der PEARL-Rundschau sicherzustellen. Der PEARL-Verein erhebt keine Einwände gegen das Kopieren einzelner Beiträge bei Angabe der Quelle.

Beiträge können jederzeit, auch unaufgefordert, an ein Mitglied des Redaktionskollegiums geschickt werden. Autoren werden gebeten, bei der Schriftleitung ein Info-Blatt zur Manuskriptgestaltung anzufordern.

Redaktionskollegium:
Schriftleitung: Dr. P. Elzer
stellvertr. Schriftleitung: Dipl.-Ing. K. Bamberger
Siemens AG
Postfach 211080
7500 Karlsruhe 21

Dr. T. Martin
Kernforschungszentrum Karlsruhe
Projekt PFT
Postfach 3640
7500 Karlsruhe 1

Dipl.-Ing. V. Scheub
Institut für Regelungstechnik
und Prozeßautomatisierung
Seidenstraße 36
7000 Stuttgart 1

Mit dem Namen des Autors gekennzeichnete Beiträge geben nicht unbedingt die Meinung des Schriftleiters, des PEARL-Vereins oder dessen Vorstand wieder.
Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. berechtigt auch ohne besondere Kennzeichnung nicht zur Annahme, daß solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen.

Vorwort

Am 26. 9. 81 sendete das Bayerische Fernsehen in der Reihe "Aus Forschung und Lehre" (an Bayerischen Hochschulen) um 21.30 Uhr im Dritten Programm einen Film über PEARL. Er war von dem zuständigen Redakteur, Herrn Dr. Tilman Steiner in Zusammenarbeit mit dem Institut für Rundfunktechnik in München gestaltet worden.

Ist allein die Tatsache schon sehr erfreulich, daß die nun über 10 Jahre gehende zähe und harte Arbeit der an der Entwicklung von PEARL Beteiligten von einem der großen Medien eines derartigen Beitrages gewürdigt wird, so war die Qualität des Filmes eine zweite sehr angenehme Überraschung. Der Unterzeichnete hat sich deshalb sofort bei Herrn Steiner für die Darstellung bedankt. Es steht zu hoffen, daß der Film im Laufe der Zeit auch von anderen ARD-Anstalten gesendet wird, so daß auch diejenigen Interessenten und Mitglieder, die nicht im südostdeutschen Raum angesiedelt sind, die Möglichkeit haben, einmal zu sehen, wie sich ihre Arbeit "von außen gesehen" darstellt.

Worin besteht aber jetzt der Zusammenhang zwischen dieser Sendung und dem vorliegenden Heft ?

In dem Film wurde klar herausgearbeitet, daß PEARL ein dem technisch schöpferischen Menschen besser angemessenes Ausdrucksmittel ist als andere bisherige Computer-"Sprachen". Dies zu erreichen, war ja eines der Hauptanliegen der PEARL-Entwickler.

Offenbar ist hier ein wesentlicher Fortschritt geglückt, da dieser Aspekt auch in allen Vorträgen der Tagung in Erlangen wieder zur Sprache kam.

Diese Tagung mit dem Thema "PEARL" in Ausbildung, Lehre und Schulung fand am 1. und 2. 10. am Institut für Programmier- und Dialogsprachen, sowie Compiler (IMMD II) der Universtität Erlangen-Nürnberg statt. Die wissenschaftliche Tagungsleitung hatte Herr Prof. Dr. Schneider, der Leiter dieses Instituts, der von Herrn Pelz unterstützt wurde. Der Unterzeichnete möchte sich bei diesen beiden Herren vielmals für ihren Einsatz und die vorzügliche Organisation bedanken.

Besonders eindrucksvoll war die Fülle des Materials, das auf der Tagung vorgestellt wurde. Deswegen ist es auch leider notwendig, den Tagungsband in zwei Teilen zu veröffentlichen, da sonst die Kosten der Herstellung in keinem vernünftigen Verhältnis mehr zum Abgabepreis des Heftes gestanden hätten. Der zweite Teil wird wenige Tage später erscheinen. Wir bitten unsere Leser um Verständnis.

Als ein wesentliches Ergebnis der Tagung stellte sich jedoch heraus, daß ein enger Zusammenhang zwischen der Lehrbarkeit einer Programmiersprache und ihrer Qualität für den Anwender besteht. Besonders interessant erscheint in diesem Zusammenhang der Beitrag von Herrn Prof. Fasold aus Bielefeld.

Das "FORTRAN-Programm", das er als Beispiel vorlegte, überzeugte schließlich auch den Unterzeichneten, daß FORTRAN eigentlich nicht mehr unter die höheren Programmiersprachen gerechnet werden kann. Zumindest ist es in beliebigem Maße mißbrauchbar.

Last, but not least, soll noch auf die PEARL-Jahrestagung im Dezember hingewiesen werden, deren Programm in diesem Heft abgedruckt ist. Sie verspricht, sehr interessant zu werden, da dort wesentliche weitergehende Aspekte der Softwareentwicklung diskutiert werden.

Für die Schriftleitung

Mit freundlichen Grüssen

Dr. P. Elzer

Die Bedeutung von PEARL in der Prozessinformatik

Dr. Dieter Pflügel, St. Georgen

Zusammenfassung. Um einem Studenten der Informatik die Probleme der Prozeßtechnik nahezubringen, benötigt man eine Prozeßbeschreibungssprache. Es zeigt sich, daß PEARL die hierfür geeignetste Sprache ist. Mit PEARL kann die Hardware beschrieben werden, als auch die Lösungsalgorithmen für die Logik der Programme und deren zeitliches Verhalten.

Summary. To teach the problems of process control to students of computer science, we need a process description language. It is shown, that PEARL fulfills all the requirements. PEARL can be used to describe the hardware, the algorithms of programming logic and the time behaviour.

1. Die Bedeutung von höheren Programmiersprachen für den Einsatz von Datenverarbeitungsanlagen

Die Ausbreitung der Datenverarbeitung ist unaufhaltsam. Es gibt kaum noch ein Gebiet, in denen man den Computer nicht in irgendeiner Form wiederfindet. Insbesondere durch die Entwicklung der Mikroprozessoren ist dieser Innovationsschub in der Datenverarbeitung fast unbegrenzt. Parallel zur Verbreiterung des Einsatzspektrums von DV-Anlagen kann man eine Weiterentwicklung bzw. Verbesserung der Programmiersprachen, die zur Realisierung von DV-Problemen herangezogen werden, beobachten. Man kann sogar sagen, daß die Weiterentwicklung der Programmiersprachen eine notwendige Voraussetzung für das Eindringen des Computers in immer weitere Anwendungsbereiche ist. Bis in die Mitte der siebziger Jahre hatten sich auf dem Groß-EDV-Sektor ganz bestimmte Sprachen etabliert, wie FORTRAN,PL/1,ALGOL, APL,COBOL,BASIC u.a.m. Die sogenannten maschinennahen Sprachen, auch Assemblersprachen genannt, wurden dabei immer mehr in den Hintergrund abgedrängt. Ein Hauptgrund war die starke Spezialisierung auf einen bestimmten Maschinentyp. Portabilitätsprobleme spielten eine immer größere Rolle. Jede der oben genannten problemorientierten Sprachen hatte sich ihren Markt erobert, wobei Überschneidungen der Einsatzgebiete natürlich möglich waren, wie z.B. bei FORTRAN,ALGOL,COBOL und PL/1. Mit dem verbesserten Werkzeug 'Programmiersprache' erweiterte sich damit auch automatisch der Problemkreis, der mit diesen Sprachen rationell gelöst werden konnte. Es steht sicher außer Frage, daß ohne FORTRAN und COBOL die Datenverarbeitung ihren heutigen Stand sicher noch nicht erreicht hätte. Mit anderen Worten, je komfortabler eine Sprache ist, um so breiter wird das Einsatzgebiet der Datenverarbeitungsanlagen sein. Eine Grenze der Komplexität der Sprache wird allerdings oft durch die DVA-Systeme selbst diktiert, die z.B. nicht beliebig große Speicher für den Übersetzungsvorgang zur Verfügung stellen können. Ein Ausweg sucht man dann zwar in der sogenannten Cross-Software, die aber nicht immer befriedigende Ergebnisse liefert. Daß höhere Sprachen die Einsatzbereitschaft von DVA-Systemen erhöhen, konnte man ganz extrem beim Einsatz der Mikroprozessoren beobachten. Die anfänglich benutzten primitiven Sprachen wie Hexacodeverschlüsselung bis zum Assembler, grenzten die Einsatzgebiete der Mikros stark ein. Entsprechendes galt für das Programmiererpotential. Nur Spezialisten konnte die Kleinstrechner programmieren. Heute gibt es kaum noch ein Entwicklungssystem, auf dem nicht in den höheren Sprachen wie BASIC,PASCAL, oder FORTRAN programmiert werden kann. Damit ist der Kreis der potentiellen Anwender ganz erheblich angestiegen.

Man kann zusammenfassend sagen: erst höhere Programmiersprachen erlauben eine volle Ausschöpfung der Möglichkeiten des Einsatzes von Datenverarbeitungsanlagen.

2. Auswahl höherer Prozeßsprachen

Bis jetzt wurde von der Datenverarbeitung im allgemeinen gesprochen. Da das Anwendungsspektrum sehr breit ist, ist es sinnvoll, wenn man eine Klassenbildung nach Aufgabengebieten durchführt. Man unterscheidet ganz grob,bekanntlich zwischen der technischwissenschaftlichen, der kommerziellen und der prozeßtechnischen Datenverarbeitung. Eine entsprechende Unterteilung ergibt sich dann auch für die problemorientierten höheren Sprachen. Die beiden erstgenannten Problemklassen werden z.B. durch die Sprachen FORTRAN,ALGOL,PL/1 b.z.w. COBOL,PL/1 versorgt. Für die Prozeßdatenverarbeitung ist die Auswahl an höheren Programmiersprachen bis jetzt noch recht mager. Das Programmieren in maschinennahen Sprachen scheidet aus Kostengründen aus. Das Softwareprodukt wird viel zu teuer und unsicher, eine Wartung ist fast unmöglich. Neben den verschiedenen Echtzeit-FORTRAN-Dialekten hat sich als neu konzipierte Sprache in letzter Zeit PEARL als echte Prozeßsprache hervorgetan. Das Sprachkonzept von PEARL beinhaltet im wesentlichen drei Komponenten. Einmal kann man mit Anweisungen, wie sie von anderen höheren Programmiersprachen bekannt sind, den logischen Ablauf einer Prozeßaufgabe formulieren. Der zeitliche Ablauf b.z.w. Synchronisationen von Teilalgorithmen erfolgt mit speziell entworfenen Sprachelementen. In dem sogenannten Systemteil eines PEARL-Programms kann im Unterschied zu allen anderen Programmiersprachen, die Struktur und die Eigenschaften der zu verwendenden Hardware beschrieben werden. Damit hat der Anwender die Möglichkeit, eine Prozeßaufgabe optimal an die vorhandene Peripherie anzupassen. Ein weiterer Gesichtspunkt von PEARL ist die Möglichkeit, Sicherheitsprobleme der Automationstechnik zu berücksichtigen. Durch die oben aufgezählten Eigenschaften des Sprachkonzeptes ist PEARL sehr gut geeignet, prozeßtechnische Aufgaben im Detail und problemlos zu beschreiben.
Man kann also sagen: von den vorhandenen Prozeßsprachen ist PEARL die am weitesten entwickelte und am besten geeignete Sprache.

3. Die drei Anwendergruppen für die Prozeßtechnik

Die große Vielfalt der Einsatzmöglichkeiten von Datenverarbeitungsanlagen in der Prozeßtechnik bringt es mit sich, daß die Anforderungen an die Kenntnisse zur Lösungsfindung solcher Probleme sehr hoch sind b.z.w. das Wissensspektrum sehr breit sein muß. Hier benötigt man Kenntnisse über die Struktur der DVA's, einen fundierten Überblick über Programmiermethoden und Verfahren, als auch das notwendige Wissen über den Ablauf und das Verhalten des technischen Prozesses selbst.

Diese Tatsache findet einen starken Niederschlag in der Ausbildung von entsprechenden Ingenieuren und Programmierern. Hierbei sind im wesentlichen drei Zielgruppen zu unterscheiden. Die erste Gruppe bringt bereits sehr gute Programmierkenntnisse mit, besitzt aber nur geringe Kenntnisse über den technischen Teil der Prozeßtechnik, das sind die Hardwarekomponenten sowie die Physik des Prozesses. Gruppe zwei ist zu Gruppe eins genau diametral angesiedelt. Programmierkenntnisse sind praktisch kaum vorhanden. Zwischen beiden Gruppen liegt dann die Gruppe drei, die über mäßige Erfahrungen im Programmieren von DVA verfügt. Die erste Gruppe setzt sich im wesentlichen aus Informatikern verschiedener Richtungen zusammen. Hinzu kommen Programmierer aus der Industrie, die sich ihre Programmierkenntnisse an Projekten, z.B. aus der kommerziellen DV angeeignet haben. Zur Gruppe zwei gehört der große Anwenderkreis der Prozeßtechniker der verschiedensten Branchen. Das sind Physiker aus dem Bereich der Kernphysik, Maschinenbauer, die sich mit der Steuerung von Werkzeugmaschinen beschäftigen, das sind Chemiker der chemischen Industrie oder Verfahrenstechniker u.a.m.. Gruppe drei rekrutiert sich aus eins und zwei, wobei die jeweiligen Kenntnisse unterschiedlich tief sind. Sie stellt gewissermaßen den Übergang der ersten zur zweiten Gruppe dar.

4. Gruppe eins als Zielgruppe für die Ausbildung in der Prozeßinformatik

Für die Bearbeitung von prozeßtechnischen Aufgabenstellungen benötigt man das Potential aus den Gruppen eins und zwei. Für die Ausbildung zum Prozeßinformatiker ergeben sich deshalb zwei grundverschiedene Vorgehensweisen. Für Gruppe eins müssen Kenntnisse über die technische Seite eines Prozesses vermittelt werden, sowie die Prozeßhardwarekomponenten der DVA erläutert werden. Der Gruppe zwei müssen dagegen Programmiertechni-

ken beigebracht werden. Dazu gehört auch das Programmieren in geeigneten Programmiersprachen. Die weiteren Ausführungen werden sich jetzt mit Gruppe eins beschäftigen und hier dann wieder speziell die Verhältnisse an der Fachhochschule Furtwangen erläutern.

5. Prozeßinformatik an der FH Furtwangen

Wie wir gesehen haben, ist die Prozeßtechnik nur ein Teilgebiet aus dem heute sehr breit angelegten Arbeitsfeld der Datenverarbeitung. Mit Einführung der Studienrichtung Informatik an den Universitäten und Technischen Hochschulen Mitte der sechziger Jahre wurde zwar der großen Nachfrage nach Fachkräften für die Lösung von Aufgaben aus der Datenverarbeitung Rechnung getragen, es zeigte sich aber sehr schnell, daß der theoretische Teil der Ausbildung den praktischen Teil bei weitem übertraf. Das hatte zur Folge, daß ein Informatiker dieser Prägung doch erhebliche Schwierigkeiten bei der Bewältigung programmiertechnischer Aufgaben hatte.

Die Fachhochschule Furtwangen war seinerzeit die erste Fachhochschule, die den Engpaß der praxisbezogenen Ausbildung im Bereich der Informatik erkannte. Daraus entstand 1971 der Fachbereich Allgemeine Informatik (AI), mit den Schwerpunkten: mathematische Grundlagenfächer wie Algebra(1o) (die in Klammern an gegebenen Zahlen geben die Stundenzahl für Vorlesung b.z.w. Praktikum an), Analysis(12), Wahrscheinlichkeitsrechnung und Statistik(6), numerische Methoden(4+2), Operations Research (4) und den DV-spezifischen Fächern wie Programmiersprachen(12+6), Übersetzerbau(4+2), Betriebssysteme(4+2), Datenorganisation(4+2) mit den jeweiligen Praktikas. Die Hardwarekenntnisse werden mit folgenden Fächern vermittelt: Elektrotechnik(2), Digitaltechnik(2), Elektronikpraktikum(4) und Struktur von DV-Anlagen. Ab Wintersemester 198o/81 wurde die Vorlesung Prozeßinformatik als zweistündige Vorlesung mit einem zweistündigen Praktikum in die Pflichtvorlesungen im 7 Semester mit aufgenommen. Bis dahin wurde die Prozeßtechnik als Wahlvorlesung geführt. Mit der Übernahme der Prozeßinformatik in den Pflichtvorlesungskatalog bestand jetzt allerdings die Notwendigkeit, die Vorlesung vollständig neu zu gestalten und umzustellen. Die Zielgruppe hatte sich weitgehendst in Richtung Gruppe eins verschoben, also sehr gute Ausbildung in den Datenverarbeitungsfächern, nur mangelnde Kenntnisse bezüglich der Hardware und der technischen Seite der Prozeßtechnik.

6. PEARL als didaktische Stütze für die Prozeßinformatik

Wie eingangs schon gesagt wurde, überdeckt die Prozeßtechnik ein sehr breites Spektrum von Anwendungsgebieten. Bei einer zweistündigen Vorlesung bleibt hier nur ein kleiner Spielraum für die Vermittlung von Kenntnissen zur Lösung prozeßtechnischer Probleme übrig. Es war deshalb notwendig, einen didaktischen Weg zu finden, mit dem man die oben geschilderten Schwierigkeiten überbrücken konnte. Es muß versucht werden, sozusagen in der Sprache des Informatikers zu sprechen. Am einfachsten erfolgt dies mittels geeigneter Symbolsprachen, mit denen man das Prozeßgeschehen dann beschreiben kann. Im ersten Ansatz wurde deshalb als Prozeßbeschreibungssprache die in Freiburg entwickelte Sprache BASEX, das ist BASIC für Echtzeitanwendung, versuchsweise genommen. Da die Sprache auf BASIC beruht, war das Erlernen der Sprachelemente, mit denen man den logischen Ablauf eines Algorithmus beschreibt, innerhalb sehr kurzer Zeit möglich. Problematisch wurde es allerdings beim Aufstellen b.z.w. der Realisierung von zeitlichen Abläufen. BASEX gestattet es z.B. nicht, zyklische Zeitaufträge abzusetzen. Das Semaphorkonzept zur Synchronisation von Teilalgorithmen ist nicht realisiert worden. Wegen der BASIC-Eigenschaft der Sprache waren die so entstandenen Programme unstrukturiert und für einen Außenstehenden kaum nachvollziehbar. Als Beschreibungssprache in der Vorlesung kam somit BASEX nicht in Frage. Es wurden deshalb verschiedene andere Sprachen auf ihre Brauchbarkeit hin untersucht. Als Ergebnis dieser Untersuchung zeigte sich, daß PEARL den aufgestellten Forderungen am besten entsprach. Die Forderungen lauteten:

1. Blockorientierte Sprache
2. Moderne Sprachelemente für die algorithmische Beschreibung des logischen Ablaufes eines Programms
3. Sprachelemente für die Beschreibung der zeitlichen Steuerung von Programmteilen
4. Sprachelemente für die Beschreibung des Hardwareaufbaus des Rechensystems und der Peripherie

5. Unterstützung der Idee der strukturierten Programmierung
6. Eignung als Systembeschreibungssprache im Unterricht
7. Möglichkeiten der Zerteilung des Gesamtsystems in getrennt übersetzbare Programmstücke

Die einzelnen Forderungen lassen sich aus den folgenden Bedingungen ableiten.
Schon im ersten Semester werden die Studenten mit der höheren Programmiersprache PASCAL vertraut gemacht. PASCAL wird dann später im Übersetzerbau als Systemsprache verwendet. Damit ist das Erlernen einer weiteren Sprache, die verwandte Sprachelemente wie PASCAL enthält, problemlos. Dies betrifft Punkt 1 und 2 des Anforderungskatalogs. Punkt 3 ist besonders wichtig, da jetzt auf Problemebene das zeitliche Verhalten eines Prozesses formuliert werden kann. Querverbindungen z.B. zu Betriebssystemen können aufgezeigt werden. Punkt 4 folgt aus der Tatsache, daß die Hardwarekenntnisse nicht sehr detailliert sind. Damit ist es möglich, das Verständnis für diese Seite der Prozeßtechnik zu erhöhen. Punkt 5 ermöglicht einen 'sauberen' Programmierstil, dem nicht zu unterschätzende Aufgaben zufallen. Mit 6 ist es möglich benötigte Algorithmen im Unterricht allgemein zu formulieren, die dann später im Praktikum weiter verwendet werden können. Punkt 7 ist besonders wichtig, da zur Durchführung des Prozeßpraktikum von den Studenten verlangt wird, daß sie unter anderem Modifizierungen bestehender Aufgabenlösungen vornehmen sollen, b.z.w. einzelne Module neu schreiben müssen. Mit der Wahl von PEARL als Systembeschreibungssprache im Unterricht und als Programmiersprache im Praktikum kann man folgende Schwerpunkt formulieren, in denen PEARL ein besonders gutes didaktisches Hilfsmittel darstellt.

1. Beschreibung des Hardwareaufbaus eines Prozeßsystems. Dazu gehört einmal die Festlegung der Verbindungen der einzelnen Hardwarekomponenten untereinander sowie die Bestimmung der Informationsflüsse in Richtung und Größe. Zum anderen gehört dazu die Auslegung der Hardware wie z.B. Bitbreite eines Analogdigitalwandlers, Festlegung der benötigten Rechengenauigkeiten, Angleichung der Daten von der externen Darstellung an die interne Darstellung im Rechner u.s.w. Die Vergabe von symbolischen Namen für die zu verarbeitenden Daten wie Interrups, Analogdaten, Einzelbit- und Mehrbitgrößen u.a. macht sich hier besonders positiv bemerkbar.
2. Zerlegung der gesamten Prozeßaufgabe in Teilaufgaben, die teilweise parallel ablaufen können. Hier macht sich das Taskkonzept von PEARL besonders bezahlt. Der Student wird damit gezwungen, den Prozeß genau zu analysieren um z.B. Unabhängigkeiten zu erkennen. Mit der Möglichkeit der Prioritätenvergabe kann dann die Wichtigkeit dieser Teilaufgaben bei Gleichzeitigkeit diskutiert werden.
3. Aufstellung der Lösungsalgorithmen in allgemeingültiger Form. Die in PEARL bekannten Programmstrukturen wie if-then-else, for-while-Schleife und case-Verteiler sowie das Prozedurkonzept genügt im allgemeinen zur problemorientierten Darstellung der Aufgabenlösungen. Entsprechendes gilt für die vorhandenen Datenstrukturen.
4. Aufstellung von Zeitplänen für die Aktivierung der einzelnen Teilalgorithmen und Formulierung mit Hilfe der PEARL-Task-Steueranweisungen.
5. Durchführung des Praktikums. Das Prozeßpraktikum läuft zeitlich parallel zur Vorlesung ab. Diese Konstruktion bringt im allgemeinen immer Schwierigkeiten mit sich, da die benötigten Programmiersprachen noch nicht bekannt sind. Es zeigt sich nun, daß mit PEARL dieses Problem weitgehendst behoben werden kann. Die wichtigsten Sprachelemente werden zu Beginn der Vorlesung kurz vorgestellt und dann im Laufe des Semesters je nach Bedarf erweitert.

7. Verwendetes Prozeßrechensystem

Mit der Umstellung von BASEX auf PEARL wird auch ein anderes Rechensystem verwendet. Zur Zeit wird im Rahmen einer Diplomarbeit die Umstellung durchgeführt, die vor allem für das Praktikum notwendig ist. Als neues System wird eine PDP 11/34 mit folgendem Ausbau verwendet: 256 KB Arbeitsspeicher, eine Magnetbandstation mit 800-1600 bpi, zwei RK07 - Platten mit je 25 MB, sieben Terminals, ein ADC (12 Bit) mit vorgeschaltetem Multiplexer für 16 Eingänge, 4 DAC, eine Digitale-I/O

mit 16 Bit, interruptfähig, ein Drucker ELA 18o. Das PEARL-System läuft unter dem Betriebssystem RSX11M.

8. Schlußbemerkung

Für die Unterrichtung des Faches Prozeßinformatik ist es unbedingt notwendig, eine Prozeßbeschreibungssprache zu benutzen, die die algorithmische Beschreibung von Prozeßabläufen in der dem Informatiker gewohnten Weise erlaubt. Sie muß es weiterhin ermöglichen, Hardwarekonfigurationen im Detail zu beschreiben. Diese Sprache sollte möglichst auch als Programmiersprache für ein Praktikum verwendbar sein. Die Verwendung von PEARL erscheint zur Zeit als die einzig praktikable Lösung. Es ist zu hoffen, daß PEARL in Zukunft auch auf kleineren Systemen implementiert wird.

Dr.Pflügel, Dieter
7742 St.Georgen
Am Sommerrain 4
Tel. o7724 7275

Didaktische und methodische Prinzipien und Leitlinien für die Seminar-Reihe: Systematisches Programmieren mit PEARL

Klaus Nagel, Herbert Nebel, Berlin (West)

Zusammenfassung

Vermittlungsgegenstand in Programmierkursen sollte neben der Programmiersprache der Prozeß der Programmentwicklung und die dabei verwendeten Methoden und Techniken sein.

Es werden Grundprinzipien der Didaktik und der Methodik entwickelt. Darauf aufbauend werden Lernziele wie Eigenschaften von Algorithmen, Problemlösungsmethoden sowie ihre Einordnung in den Programmentwicklungsprozeß konkretisiert.

In einem umfangreichen Beispiel wird die Anwendung der Methoden und Leitlinien gezeigt und verdeutlicht, welche Lernziele und mit welchen Methoden diese Lernziele vermittelt werden können.

Summary

The subject of instruction in programming courses should be, in addition to the programming language itself, the process of program development, and the methods and techniques used for that purpose.

Basic principles of teaching and teaching methods are developed. After this foundation has been laid, further learning objectives are introduced, such as the characteristics of algorithms, problem solving methods, and their place in the process of program development.

A detailed and complex example is used to demonstrate the application of the previously discussed methods and principles, and to clarify which learning objectives and with which teaching methods these objectives can be taught.

1 Einleitung

Innerhalb der elektronischen Datenverarbeitung erlangt die relativ junge Prozeßdatenverarbeitung eine immer stärkere Bedeutung. Die Zeiten, in denen ein Prozeßrechner, übertrieben ausgedrückt, nur nach seiner Zykluszeit beurteilt wurde, sind vorbei.

Die zunehmende Verbreitung solcher Anlagen hat in den letzten Jahren die Definition und Implementierung von geeigneten, problemorientierten Programmiersprachen für diesen Anwendungsbereich verstärkt in den Mittelpunkt des Interesses gerückt. Dahinter verbirgt sich die Motivation, eine spürbare Effizienzsteigerung bei der Lösung und Programmierung von vielfältigen Problemen aus dem Gebiet der Prozeßdatenverarbeitung zu erreichen. Unter Effizienzsteigerung wird nicht nur eine Aufwandsreduzierung (Zeit/Kosten), sondern auch die Erstellung sicherer, wartungsfreundlicher und zuverlässiger Programme verstanden.

Eine Programmiersprache kann hierzu nur ein, wenn auch mächtiges, Hilfsmittel sein; unerläßlich für die Erreichung dieses Zieles ist jedoch außerdem die Verwendung geeigneter Programmiermethoden und -techniken. Daher

reicht es nicht aus, in Programmierkursen nur die Sprachelemente einer Programmiersprache zu vermitteln und ihre Anwendung mittels kleiner Beispiele zu zeigen, sondern es muß darüberhinaus der Prozeß der Programmentwicklung und die dabei verwendeten Methoden und Techniken vermittelt werden.

Diese Methoden und Techniken sowie die Methoden ihrer Vermittlung stehen in einem engen Zusammenhang. Sie dürfen daher nicht getrennt voneinander betrachtet werden.

2 Grundlegende Prinzipien der Didaktik und der Methodik

Übergeordneter (d.h. didaktischer) Ausgangspunkt sind die

LERNZIELE,

konkreter (d.h. methodischer) Ausgangspunkt ist

E I N ALGORITHMUS,

dessen Konstruktion und Umsetzung in ablauffähige P E A R L- Programme mit geeigneten

METHODEN

zu demonstrieren ist.

Die beim Entwickeln des jeweiligen Algorithmus angewandten Methoden des Algorithmischen Problemlösens sind Teil der Lernziele. Dasselbe gilt für die von Algorithmen unabhängigen Methoden (z.B. die strikte Trennung von Konstruktion eines Algorithmus und der Umsetzung in ein Programm) oder weittragende und immer wiederkehrende Lösungsmethoden (z.B. Rekurrenz-Beziehungen oder Koordinierungsmechanismen).

Als Ergebnis der Lernzielvermittlung verstehen wir einen Qualifizierungsprozeß, indem die Lernenden sich Kenntnisse und Fähigkeiten aneignen, die für den Einzelnen die Bewältigung einer konkreten Arbeit - hier die Lösung von Automatisierungsaufgaben mit Hilfe der Programmiersprache P E A R L - ermöglichen.

Zum Vermitteln der Lernziele haben wir neben dem didaktischen Prinzip der Integrität, d.h. Einheit von Lernzielen, Algorithmus und Methoden, das methodische Prinzip der Induktivität gewählt. Unter Induktivität verstehen wir die exemplarische Vermittlung von ausgewählten Lernzielen anhand einzelner Beispielalgorithmen; von der Praxis ausgehend zu deren theoretischer Fundierung. Die gewünschte Überschaubarkeit der Beispielalgorithmen verbietet uns, diese mit zu vermittelndem Stoff zu überladen. An e i n e m Algorithmus werden nie a l l e Lernziele eingeführt, d.h.: wir benötigen eine Vielfalt von algorithmischen Problemstellungen. Einzelne Methoden, wie z.B. Sortier- oder Suchverfahren oder Kommunikations- oder Kooperationsmechanismen, sind über das konkrete Problem hinaus von Interesse. Diese Methoden müssen verallgemeinert werden.

Die Didaktik und die Methodik sind mit ihren Attributen Integrität bzw. Induktivität miteinander verträglich und bilden eine inhaltliche Einheit.

3 Konkretisierung der Prinzipien und Leitlinien

3.1 Der Begriff A L G O R I T H M U S

Ein Algorithmus ist eine exakte, endliche Beschreibung einer ausführbaren Anweisungsfolge zur Lösung eines gegebenen Problems oder einer Problemklasse in endlicher Zeit.

Einen Algorithmus, der dadurch gekennzeichnet ist, daß alle Anweisungen der Anweisungsfolge streng nacheinander ausgeführt werden, nennen wir

SEQUENTIELLEN Algorithmus.

Ist die Ausführungsreihenfolge einzelner Anweisungsteilfolgen weitgehend unabhängig voneinander, dann sprechen wir von einem

NICHTSEQUENTIELLEN Algorithmus.

Da Anweisungsteilfolgen wieder als sequentielle Algorithmen definiert werden können, sagen wir: ein nichtsequentieller Algorithmus besteht aus einem System von sequentiellen Algorithmen und jeder einzelne sequentielle Algorithmus eines solchen Systems wird nebenläufig (quasi parallel) ausgeführt.

3.2 Eigenschaften von Algorithmen

Je nach Art der Verwendung des Begriffs Algorithmus kann entweder mehr

die Anweisungsfolge selbst (**PROGRAMM**)

oder mehr

die Abarbeitung derselben (**PROZESS**)

im Vordergrund der Betrachtungen stehen.

Wir werden daher im folgenden konstruktionsorientierte und ablauforientierte Eigenschaften von Algorithmen getrennt betrachten.

3.2.1 Konstruktionsorientierte Eigenschaften von Algorithmen

Ein zu lösendes Problem läßt sich durch eine Menge von Objekten mit bestimmten Eigenschaften charakterisieren, auf denen gewisse Operationen (**Funktionen**) definiert sind und zwischen denen bestimmte Beziehungen bestehen (**Prädikate**).

Die Menge aller im jeweiligen Algorithmus vorkommenden Objekte - sie bestehen aus den Eingabe- und Ausgabewerten sowie den weiteren im Algorithmus benutzten Objekten - nennen wir **Objektbereich** des Algorithmus.

Die Gesamtheit bestehend aus dem Objektbereich, den Funktionen und Prädikaten heißt

ALGORITMISCHE BASIS.

Zur Konstruktion von Algorithmen benötigen wir

PROGRAMMVARIABLEN
(VARIABLEN,DATENSTRUKTUREN),

als Platzhalter für die Objekte,und

PROGRAMMOPERATOREN
(ZUWEISUNGEN,KONTROLLSTRUKTUREN),

mit deren Hilfe die Objekte manipuliert werden können.

Generell dienen Programmiersprachen zur Umsetzung von Algorithmen, wenn diese auf einer Rechenanlage ablaufen sollen. In unserem Fall benutzen wir die Programmiersprache P E A R L. Im folgenden geben wir stichwortartig neben allgemeinen (konstruktionsorientierten) Eigenschaften von Algorithmen auch die in P E A R L definierten Datenstrukturierungsmöglichkeiten und Mechanismen zur Ablaufsteuerung bzw. zur Manipulation von Variablen als Konkretisierung an.

Datenstrukturierende Eigenschaften von Algorithmen:

Primitive Strukturen: **BIT**(1), **CHARACTER**(1), **CLOCK**, **DURATION**, **FIXED**(g), **FLOAT**(g).
Referenzen: **REF**.
Homogene Strukturen: Bit- und Zeichenketten und Felder.
Inhomogene Strukturen: **STRUCT**.
Rekursive Strukturen: können mit **STRUCT** und **REF** konstruiert werden.
Abstrakte Strukturen: können mit Hilfe des Modul-Konzepts konstruiert werden.
Dateien (Files): **DATION**.
Typ-Definition: **TYPE**.

Operativ strukturelle Eigenschaften von Algorithmen:

Substitution: := (Wert- und Referenz-Zuweisung) und **CALL**

Sequentielle Operatoren
- Alternative: **IF...THEN...ELSE...FIN.**
- Fallunterscheidung:
  CASE...ALT...OUT...FIN.
- Statische Iteration:
  FOR...FROM...BY...TO...REPEAT...END.
- Dynamische Iteration:
  WHILE...REPEAT...END.
- Sprung: **GOTO.**
- Ausnahmen Behandlung: **SIGNAL: ON, INDUCE.**
- Zeitüberwachung: kann nur bedingt konstruiert werden, z.B. mit Hilfe von **INTERFACE** oder bei Auftragnehmer-/geber-Problemlösungstechnik.

Nichtsequentielle Operatoren
- Synchronisation: **SEMA: REQUEST, RELEASE**
  BOLT: ENTER, LEAVE, RESERVE, FREE.
- Unmittelbare Prozesse: **TASK: ACTIVATE, SUSPEND, RESUME, CONTINUE, TERMINATE.**
- Zeitgesteuerte Prozesse:
  TASK, CLOCK, DURATION:
  time schedule **ACTIVATE,**
  time schedule **CONTINUE,**
  PREVENT.
- Ereignisgesteuerte Prozesse:
  TASK, INTERRUPT:
  event schedule **ACTIVATE,**
  event schedule **CONTINUE,**
  PREVENT, TRIGGER.
- Prioritätsgesteuerte Prozesse:
  TASK, PRIORITY:
  ACTIVATE...PRIO...,
  CONTINUE...PRIO... .

Bei der Umsetzung eines Algorithmus in ein P E A R L-Programm müssen neben den bisher genannten Konkretisierungsmöglichkeiten noch eine Reihe von Besonderheiten der Programmiersprache berücksichtigt werden, damit das Programm ablauffähig wird. Diese Besonderheiten gehören ebenfalls zu den Lernzielen. Als besonders wichtig sind zu nennen:

das Modul-Konzept mit seiner Unterteilung in Systemteil und Problemteil;

das Block-Konzept mit den daraus folgenden Aussagen über Variablen, wie Lebensdauer und Gültigkeit;

die Parameter-Mechanismen, wie "call by reference" und "call by value";

das Ein-/Ausgabe-Konzept, das in P E A R L besonders mächtig und vielfältig ist;

sowie Kenntnis über die in P E A R L definierten Standardfunktionen mit ihrem Rang.

3.2.2 Ablauforientierte Eigenschaften von Algorithmen

Programme beschreiben Algorithmen, die auf einer Rechenanlage ablaufen können. Die Abarbeitung eines sequentiellen Algorithmus, oder anders ausgedrückt: die Inkarnation eines sequentiellen Algorithmus, nennen wir

PROZESS

und die Abarbeitung eines nichtsequentiellen Algorithmus nennen wir ein

SYSTEM von PROZESSEN.

Alle Algorithmen haben, unabhängig davon ob es sich um sequentielle oder nichtsequentielle handelt, einige Eigenschaften gemeinsam:

ENDLICHKEIT der BESCHREIBUNG

Die Beschreibung eines Algorithmus muß einen endlichen Raum einnehmen, damit die durch sie gegebene Problemlösung als Ganzes erfaßt und ausgeführt werden kann.

Gegenbeispiel:

$$a + a^2 + a^3 + \ldots + a^n + \ldots \quad .$$

EFFEKTIVITÄT

Jede einzelne Anweisung zur Lösung eines Problems muß in endlicher Zeit ausführbar sein.

Gegenbeispiel:

$$f(n) = \begin{cases} 1, & \text{wenn es natürliche Zahlen} \\ & a,b,c \text{ mit } a^n+b^n=c^n \text{ gibt} \\ 0, & \text{sonst} \end{cases}$$

n sei eine natürliche Zahl.

DETERMINISMUS

Der Ablauf eines Algorithmus ist eindeutig vorgeschrieben. Bei gleicher Eingabe muß während verschiedener Programmläufe immer eine identische Ausgabe erzeugt werden. Dies ist bei allen sequentiellen und nichtsequentiellen Algorithmen, deren Prozesse ausschließlich ihre eigenen (privaten) Variablen manipulieren, immer der Fall. Bei allen übrigen Algorithmen muß die **DETERMINIERTHEIT** explizit sichergestellt werden.

TERMINIERUNG

Ein Prozeß muß nach Ausführung endlich vieler Anweisungen zu einem Ende kommen. Bei allen nichtsequentiellen Algorithmen, deren Prozesse miteinander konkurrieren, kooperieren oder kommunizieren, muß insbesondere die **VERKLEMMUNGSFREIHEIT** explizit gezeigt werden.

Gegenbeispiel:

$$\sqrt{3} \ .$$

KORREKTHEIT

Bei gültigen Eingabedaten muß eine korrekte Ausgabe erzeugt werden. Es muß explizit nachgewiesen werden, daß ein Algorithmus tatsächlich den Prozeß bzw. das System von Prozessen zur Lösung des gegebenen Problems realisiert.

EFFIZIENZ

Der Aufwand an Ausführungszeit und Speicherplatzbedarf zur Lösung eines Problems sollte minimal sein. Da wenig Ausführungszeit und geringer Speicherplatzbedarf Effizienzkriterien sind, die sich bei konkreten Lösungen bisweilen widersprechen, ist die Realisierung des effizientesten Algorithmus oftmals eine praktische Frage.

3.3 Problemlösungmethoden - Algorithmisches Problemlösen

Unter algorithmischem Problemlösen verstehen wir den Prozeß der systematischen Entwicklung eines Programms. Im Regelfall existiert eine größere Lücke zwischen der Ebene der Problembeschreibung und der Ebene der Problemlösung. Systematische Programmentwicklung bedeutet dann das zielgerichtete Schließen dieser Lücke, wobei die einzelnen Entwicklungsschritte wohlgeordnet sind.

GENAUE BESCHREIBUNG des PROBLEMS

Dem Problemlösen muß immer die genaue Beschreibung des Problems vorausgehen. Die Lösungsstruktur wird grob skizziert und eine Zerlegung des Problems in nebenläufig lösbare Teilprobleme wird vorgenommen.

SPEZIFIKATION

Unter Spezifikation verstehen wir die statische Zerlegung eines Problems in einzelne Moduln und die verbindliche und detailierte Festlegung der Beziehungen zwischen den Moduln. Die Festlegung erfolgt so, daß alle für die Problemlösung wesentlichen Eigenschaften der Moduln beschrieben werden, ohne explizite Festlegung der Algorithmen und der Datenstrukturen.

Die Zerlegung sollte so vorgenommen werden, daß die einzelnen Moduln mit geeigneten Methoden realisierbar sind.

Da große Programmsysteme fast immer von mehreren Programmierern erstellt werden, ist die Spezifikation ein Hilfsmittel bei der Kommunikation zwischen den Programmierern.

Die Spezifikation muß exakt, jedoch nicht zwingend formal sein. Auch verbal lassen sich die Spezifikationen genau beschreiben. Da dies jedoch viel Schreibarbeit bedeuten würde, ist eine Zwischenlösung oft das adäquate Mittel.

KONSTRUKTION und VERIFIKATION

Voraussetzung für die Konstruktion von nichtsequentiellen Algorithmen ist die Klassifizierung der einzelnen Teilprobleme in disjunkte bzw. nicht-disjunkte Prozesse. Desweiteren müssen für die gemeinsamen Objekte und für die Abhängigkeiten zwischen den Prozessen geeignete prozeßspezifische oder objektspezifische Synchronisationsmaßnahmen konstruiert werden.

Der Weg von der Anforderungsdefinition bzw. von der Spezifikation hin zum fertigen Programm geht immer über Zwischenschritte. Ein Verfahren ist das der schrittweisen Verfeinerung.

Dabei wird die Gesamtaufgabe in Teilaufgaben (auch abstrakte Anweisung genannt) zerlegt. Diese Teilaufgaben stellen eine Konkretisierung der Gesamtaufgabe dar. Dieser Vorgang der Verfeinerung wird solange wiederholt, bis die Ebene der Programmiersprache erreicht ist. Die Anzahl der Verfeinerungsschritte hängt dabei von der Komplexität des Gesamtproblems ab.

Das hier für Algorithmen gesagte gilt ebenso für Daten. Bei vielen Problemen ist der Entwurf sinnvoller Datenstrukturen von entscheidender Bedeutung. Ab einer gewissen Komplexität wird man dann auch beim Entwurf geeigneter Datenstrukturen den Weg der schrittweisen Verfeinerung gehen.

Eine andere Technik ist die der Konstruktion von algorithmischen (problemadäquaten) Basen. Hierbei werden, von der vorgegebenen algorithmischen Basis der Programmiersprache ausgehend, solange komplexere Basen konstruiert, bis schließlich eine Basis erreicht ist, die das vorgegebene Problem löst.

In der Praxis wird häufig eine kombinierte Technik verwendet (oszillierende Programmentwicklung), indem die schrittweise Verfeinerung bis zum Erreichen einer gewissen Standard-Problemlösung durchgeführt wird. Solche Standardlösungen sind wohlbekannt und befinden sich häufig in Programmbibliotheken.

Parallel zur Konstruktion muß schlüssig nachgewiesen werden, daß der konstruierte Algorithmus auch korrekt ist. Bei nichtsequentiellen Algorithmen muß insbesondere die Determiniertheit nachgewiesen werden. Korrektheitsnachweise können Plausibilitätsbetrachtungen sein, aber auch mathematische Beweisführungen (induktive Zusicherungen, Fixpunkttheorie), die formal und häufig nicht trivial sind.

Die Erfahrung lehrt, daß auch bei formalen Beweisen bisweilen Fehler gemacht werden und daher trotz scheinbarer Korrektheit eine fehlerfreie Problemlösung nicht sichergestellt ist. Dennoch kann man sagen, daß ein Korrektheitsnachweis ein wichtiges Hilfsmittel ist, die Häufigkeit von logischen Programmfehlern zu minimieren.

IMPLEMENTIERUNG und FALSIFIKATION

Nach der Konstruktion und Verifikation eines Algorithmus kann dieser mit Hilfe der gewählten Programmiersprache in ein ablauffähiges Programm umgesetzt werden. Dabei ist darauf zu achten, daß die Umsetzung sich exakt an dem Algorithmus orientiert. Jede Nebenbei-Optimierung ist zu vermeiden, da dies fatale Folgen haben kann. Zur Erhöhung der Lesbarkeit und damit auch der Wartbarkeit sollte das Programm strukturiert und selbstdokumentierend sein. Dies kann durch eine formatfreie Schreibweise, aussagekräftige Variablen- und Algorithmennamen und durch sinnvolle Kommentare erreicht werden. Ist das Programm codiert, wird mit der Falsifikation der Korrektheitsnachweis für dasselbe abgeschlossen. Bei großen Programmen sollte die Falsifikation über mehrere Stufen laufen. Die einzelnen Module sollten getrennt getestet werden und danach erst in einem Integrationstest das gesamte Programm.

Der Falsifikation liegt der triviale Gedanke zugrunde, daß ein Algorithmus nicht korrekt ist, wenn bei gültigen Eingabedaten ungültige Ausgabewerte erzeugt werden. Wichtig ist hier, auch die Grenzbereiche der erlaub-

ten Eingabemenge als Eingabedaten zu verwenden. Bei Verzweigungen sollten alle Alternativen mindestens einmal durchlaufen werden. Die Tests müssen also wohlüberlegt sein. Wenn auch nur ein einziger Testlauf negativ ausgeht, ist der Algorithmus falsch.

AUFWANDSREDUZIERUNG

Am Ende wird der Aufwand festgestellt, sowohl hinsichtlich der benötigten Rechenzeit, als auch des Speicherplatzbedarfes. Entspricht das Programmverhalten nicht der Problemstellung oder den Erwartungen, so müssen die dafür verantwortlichen Programmteile lokalisiert und optimiert werden. Optimierungen werden erst durchgeführt, wenn die Korrektheit des Programms nachgewiesen wurde, da sonst eine zusätzliche Fehlerquelle im Programm für Verwirrung sorgen kann.

Programm- und Benutzer- DOKUMENTATION

Programme werden oftmals nicht nur von mehreren Personen erstellt, sondern müssen auch nach der Fertigstellung gewartet und gegebenenfalls geändert oder erweitert werden. Dies stellt hohe Anforderungen an die Nachvollziehbarkeit der entwickelten Programme. Um dies mit einem vertretbaren Arbeitsaufwand leisten zu können, muß ein Programm dokumentiert werden.

Haben wir bei der Erstellung des Programms die Methode der schrittweisen Verfeinerung gewählt, so sind die Verfeinerungsschritte selbst ein wichtiger Teil der Dokumentation. Zusammenhänge kann man oft auch durch Graphiken veranschaulichen, wie z.B.: gemeinsam von verschiedenen Prozessen benutzte Datenpuffer oder Beziehungen zwischen den Prozessen, usw. . Die Anforderungsdefinition und die Spezifikation sollten der Dokumentation beigefügt werden.

Die Programmdokumentation ist ein Hilfsmittel für den Programmierer. Ebenso muß eine Dokumentation für den Benutzer des Programms erstellt werden. Dieser ist meistens nicht mit dem Programmierer identisch.

Die Benutzerdokumentation soll in leicht verständlicher Form die Programmfunktionen vollständig und eindeutig beschreiben. Es sollten Hinweise auf Reaktionen bei außergewöhnlichen Ereignissen gegeben und eine Liste der Fehlermeldungen und ihrer Bedeutung beigefügt werden. Auch muß der Aufruf des Programms erklärt sowie über eventuelle Voraussetzungen zur Benutzung desselben berichtet werden.

3.4 Allgemeine Methoden und Leitlinien

Programmiermethoden und -techniken rücken zunehmend - auch im privatwirtschaftlichen Bereich - in den Mittelpunkt des Interesses. Ziel ist es dabei, die Kosten bei der Erstellung und Wartung von Softwareprodukten zu reduzieren. In Zusammenhang damit steht die zunehmende Anwendung höherer Programmiersprachen, die die Lesbarkeit und Sicherheit der Programmprodukte erhöhen.

Doch die Kenntnis von Programmiermethoden und -techniken und auch das richtige Anwenden allein garantiert nicht die erwünschte Effizienzsteigerung bei der Programmentwicklung bzw. die geforderte Sicherheit des Produktes. Die gewünschten Effekte können erst durch das richtige Anwenden der Programmiermethoden und -techniken an der richtigen Stelle innerhalb des Problemlösungsprozesses erreicht werden.

Ein typisches Beispiel - zumindestens für den Ausbildungsbereich : Eine Spezifikation oder ein algorithmischer Entwurf wird erst nach der Implementierung der Problemlösung angefertigt und auch nur, weil es verlangt wurde.

Ein zweites Beispiel: Die Spezifikation wird nur teilweise erstellt und die Programmierer einigen sich darauf, offene Punkte bei Bedarf zu klären. Oder: Der Algorithmische Entwurf wird unvollendet abgebrochen, weil die Problemlösung zumindest im Kopf schon völlig klar erscheint.

Bei den Beispielen muß mit einem erhöhten Entwicklungsaufwand und mit fehleranfälligen Softwareprodukten gerechnet werden. Dies kann nur bei strikter Einhaltung der folgenden Leitlinien verhindert werden:

STRIKTE TRENNUNG VON KONSTRUKTION UND IMPLEMENTIERUNG

Nur durch das unbedingte Einhalten dieser Forderung können die gewünschten Ziele erreicht werden. Entgegen der landläufigen Meinung ('Probieren geht über Studieren') sollte eine Implementierung solange wie möglich vermieden werden. Je intensiver die Logik der Problemlösung analysiert und algorithmisch realisiert wurde, desto weniger Probleme treten bei der Implementierung auf.

EINHEIT VON KONSTRUKTION UND VERIFIKATION

Voraussetzung für die Konstruktion und Verifikation eines Algorithmus ist eine genaue und vollständige Problembeschreibung und, falls das Problem komplex genug ist, eine abgeschlossene Spezifikation.

Mit der Konstruktion einher sollte die Richtigkeit des Algorithmus gezeigt werden. Zumindest muß gezeigt werden, daß der Algorithmus sicher ist, d.h. daß er der Aufgabenstellung und der Spezifikation genügt.

Bei der Konstruktion von nichtsequentiellen Algorithmen ist darauf zu achten, daß die durch die Anforderungsdefinition gegebene Zerlegung des Problems in nebenläufig lösbare Teilaufgaben strikt eingehalten, alle Abhängigkeiten zwischen den einzelnen sequentiellen Algorithmen erkannt und die Verklemmungsfreiheit sichergestellt wird.

Für die sequentiellen Algorithmen gilt - und Teilaufgaben aus dem Problemkreis Prozeßdatenverarbeitung werden im Regelfall iterativ gelöst : Ein korrektes Ergebnis kann nur dann erreicht werden, wenn die benötigten Variablen korrekt initialisiert werden und die Iteration korrekt durchgeführt wird.

EINHEIT VON IMPLEMENTIERUNG UND FALSIFIKATION

Nur ein vollständig entwickelter Algorithmus ist geeignet, implementiert zu werden. Bei der Implementierung darf auf keinen Fall nebenher optimiert werden. Geeignete Testdaten und Teststrategien müssen schon während der Implementierung entwickelt werden. Zielsetzung der Falsifikation muß sein, die Anwesenheit von Fehlern aufzuzeigen und nicht die Richtigkeit des Programms.

KENNTNIS UND ANWENDUNG VON WEITTRAGENDEN UND IMMER WIEDERKEHRENDEN METHODEN

Bekannte Lösungsmethoden sollten bekannt und bei einem zu lösenden Problem erkannt und angewandt werden. Dazu zählen z.B.:

Methoden der Automatisierungstechnik

Koordinierungsmechanismen bei nebenläufigen Teilalgorithmen

Organisation und Verwaltung von Nachrichtenpuffern

Organisation und Verwaltung großer Datenmengen

Sortier- und Suchverfahren

das Teilungs- oder Halbierungsprinzip

Rekurrenz-Beziehungen oder Rekursionsrelationen

Vollständige Fallunterscheidung .

UNTERSCHEIDUNG UND BEHERRSCHUNG DER VIELFALT DER SPRACHEBENEN

Der Problemlösungsprozeß bringt es mit sich, daß das Problem in unterschiedlichen Sprachen beschrieben wird. Alle diese Sprachen müssen verstanden und wohlunterschieden werden. Dies ist insbesondere für die korrekte Umsetzung von einer Sprachebene in die andere notwendig. Die wichtigsten

Sprachebenen, die beherrscht werden müssen, sind im folgenden zusammengefaßt:

Natürliche Sprache

Fachsprache, Formelsprache

Spezifikationssprache,Algorithmische Sprache

Programmiersprache

Bediensprache.

4 Schlußbemerkung

Nur mit entsprechender Qualifikation der Programmierer kann die geplante Kostenreduzierung bei der Programmentwicklung und -wartung erreicht als auch ein sicheres und zuverlässiges Softwareprodukt erzielt werden.

Ausgehend von dem didaktischen Prinzip der Integrität wurden Lernziele und die Methoden ihrer Vermittlung entwickelt, die den geforderten Qualifizierungsprozeß unterstützen.

Anwendungen der Prinzipien und Leitlinien befinden sich im Anhang. Mit Hilfe eines umfangreichen und vollständigen Beispiels werden insbesondere die einzelnen Schritte des algorithmischen Problemlösens gezeigt. Anhand des gegebenen Beispielalgorithmus wird deutlich, welche exemplarischen Lernziele mit geeigneten Methoden vermittelt werden können.

Weitere Anwendungen finden sich in Brinkkötter, H., Nagel, K., Nebel, H., Rebensburg, K.: "Einführung in die Prozeßdatenverarbeitung - Systematisches Programmieren mit P E A R L".

Eine Diskussion über die zugehörigen exemplarischen Lernziele und die Methoden ihrer Vermittlung findet sich in dem Beitrag von Brinkkötter, H., Rebensburg, K.: "Aufbau und Durchführung einer Anwenderschulung - Systematisches Programmieren mit P E A R L / Teil 1".

Anhang:
Anwendung der Prinzipien und Leitlinien

A.1 Aufgabenstellung

Der Zustand einer Anlage soll automatisch überwacht werden. Dazu müssen folgende Aufgaben gelöst werden:

Erfassung des Anlagenzustandes

In einem festen Zyklus f1 werden die Zustandswerte zj (j=1,...,n) an verschiedenen Stellen (s1,...,sn) der Anlage erfaßt. Die einzelnen Zustandswerte sind analoge Größen, die vor ihrer Weiterverarbeitung geeignet konvertiert werden müssen. Zur weiteren Verarbeitung wird der Meßzeitpunkt ti und die Zustandsmenge Zi (i=1,2,3,...) zur Verfügung gestellt.

Überprüfung des Anlagenzustandes

In einem zweiten Zyklus f2 soll geprüft werden, ob die einzelnen Zustandswerte zj der Zustandsmenge Zi innerhalb eines Toleranzbereichs TZ = [zu, zo] liegen. Im Fehlerfall, d.h. Anzahl der fehlerhaften Zustandswerte > 0, wird ein Alarm mit entsprechender Fehlermeldung ausgelöst. Eine Fehlermeldung besteht aus dem Zeitpunkt der Fehlererkennung tk, dem Meßzeitpunkt ti (ti<tk) und einer Menge von Paaren (Meßstellenkennung sj, fehlerhafter Zustandswert zj) (j=1,...,n).

Protokollierung des zeitlichen Verlaufs des Anlagenzustandes

In einem dritten Zyklus f3 wird die Entwicklung des Anlagenzustands protokolliert, indem der Meßzeitpunkt ti und ein Zustandskennwert zki aufgezeichnet werden. Der Zustandskennwert ist definiert als Mittelwert der gültigen Zustandswerte zj. Liegt kein plausibler Zustandswert vor, dann wird ein Standard-Zustandskennwert Szk protokolliert.

Die drei Zykluszeiten werden bei Start der automatischen Überwachung eingegeben. Für die Dauerwerte gilt: $0 < f1 \leq f2 \leq f3$.

Die Problemlösung soll **robust** sein, d.h. es soll auf die beiden folgenden Fehler vernünftig reagiert werden:

Verklemmung der (oder eines Teils der) Prozeßperipherie, z.B. durch einen elektrischen Störimpuls (temporärer Fehler).

Ausfall der (oder eines Teils der) Prozeßperipherie (permanenter Fehler).

Randbedingung

Die Problemlösung soll so konstruiert werden, daß zu einem späteren Zeitpunkt die Automatisierung mit Hilfe eines "verteilten Systems" problemfrei realisiert werden kann.

A.2 Gedanken zur Problemlösung

Um die automatische Überwachung der technischen Anlage zu realisieren, müssen periodisch Zustandswerte erfaßt und geprüft und ein Zustandskennwert protokolliert werden.

Da die periodische Wiederholung der drei Teilaufgaben zu unterschiedlichen Zeiten stattfinden kann ($f1 \leq f2 \leq f3$), ist die nebenläufige Abarbeitung der Teilaufgaben eine adäquate Problemlösung. Zur Lösung der gesamten Aufgabe müssen die Prozesse miteinander kooperieren, d.h. der Meßzeitpunkt ti und die Zustandsmenge Zi des Prozesses Erfassung werden von den beiden anderen Prozessen (Überprüfung und Protokollierung) zur Lösung ihrer Aufgaben benötigt; sie müssen daher diesen zur Verfügung stehen. Um die Integrität der gemeinsamen Daten sicherzustellen muß der Zugriff darauf synchronisiert werden.

Desweiteren soll bei der Problemlösung darauf geachtet werden, daß die drei Prozesse nicht notwendigerweise auf einer Rechenanlage ablaufen, sondern über mehrere verteilt sein können. Eine einfache, aber sinnvolle, Verteilung auf mehrere Rechenanlagen ist die Trennung der Problemlösung in "Prozeßdatenerfassung" und "Prozeßdatenverarbeitung". Der augenscheinliche Nachteil einer solchen Verteilung ist der fehlende gemeinsame Datenbereich.

Eine mögliche Lösung bietet das aus dem Betriebssystembereich bekannte Verfahren der "auftragsorientierten Systeme". In einem solchen System vergibt ein Auftraggeber Aufträge und wartet auf das Ergebnis. Der Auftragnehmer empfängt Aufträge und liefert die gewünschten Ergebnisse. Die für die Kooperation notwendigen Daten (Aufträge und Ergebnisse) sind immer den Empfängern zugeordnet. Sie können dort auch gepuffert werden.

Ein Ausfall der (oder eines Teils der) Prozeßperipherie kann die gesamte automatische Überwachung verklemmen. Der Erfassungsprozeß wird beim Lesen der Zustandswerte blockiert und die beiden anderen Prozesse warten beliebig lange auf die Erledigung ihrer Aufträge. Die Verklemmungssituation kann gelöst werden, wenn die beiden Auftraggeber nur eine endliche Zeit auf die Erledigung ihrer Aufträge warten. Bei einer Zeitüberschreitung können die beiden Prozesse geeignet darauf reagieren.

Zum Starten und Abbrechen der automatischen Überwachung wird ein vierter Prozeß gewählt.

Die Prozeßperipherie kann in einen verklemmungsfreien Anfangszustand gebracht werden; dazu steht eine Systemroutine **NORMALIZE** zur Verfügung. Zum Lesen der aktuellen Systemzeit und des Datums stehen die Systemfunktionen **NOW** und **DATE** zur Verfügung.

A.3 Spezifikation der Problemlösung

Grundeinheit unserer Spezifikationssprache ist der Modul (**MODULE**). Spezifiziert werden alle diejenigen Prozeduren, Datentypen und Konstanten, die der Modul für andere Moduln definiert (**DEFINES**) und alle diejenigen Prozeduren, Datentypen und Konstanten, die

der Modul aus anderen Moduln benutzt (APPLIES).

Die Grundbausteine eines Modul sind Prozeduren (PROC) und Prozesse (PROCESS).

Prozeduren werden spezifiziert, indem die Änderungen, die bei einem Prozeduraufruf entstehen, angegeben werden. Sind Prozeduren parametrisiert, dann wird nebem dem Parametertyp noch eine Übergaberichtung (IN, OUT, TRANS) und ein Zugriffsrecht (VAR, INV) angegeben. Prozeduren können Werte liefern; dies wird durch Angabe eines Typ-Präfix spezifiziert.

Prozesse werden durch Beschreibung ihres Ablaufs spezifiziert.

Die Änderungen und der Ablauf werden verbal beschrieben.

```
MODULE Prozeßdatenerfassung:

DEFINES
  FOR Prozeßdatenverarbeitung:
    Erteile Auftrag,
    Starte zyklische Erfassung,
    Abbruch und erneute Erfassung,
    Abbruch zyklische Erfassung,
    Anlagenzustand,
    Anzahl Zustandswerte.

APPLIES
  FROM Prozeßdatenverarbeitung:

    Melde Ergebnis,
    Auftragszettel,
    Maximale Aufträge.
  FROM SYSTEM:
    NORMALIZE,
    NOW.
TYPE Anlagenzustand.

INV Anzahl Zustandswerte FIXED.

BIT(1)PROC Erteile Auftrag
        (IN INV Auftrag REF Auftragszettel):
  Liefert Erteile Auftrag den Wert '1'B,
  dann wurde der Auftrag angenommen, sonst
  wurde der Auftrag abgelehnt.
ENDPROC Erteile Auftrag.

PROC Starte zyklische Erfassung
          (IN INV Zyklus DURATION):
  Die Prozeßperipherie ist normiert und der
  Erfassungsprozeß, der zyklisch wiederholt
  wird, wurde eingeleitet.
ENDPROC Starte zyklische Erfassung.

PROC Abbruch und erneute Erfassung:
  Alle unerledigten Aufträge sind gestrichen,
  die Prozeßperipherie erneut normiert und
  der noch aktive Erfassungsprozeß abge-
  brochen.
ENDPROC Abbruch und erneute Erfassung.

PROC Abbruch zyklische Erfassung:
  Der Erfassungsprozeß ist abgebrochen und
  die Einplanung ist gestrichen.
ENDPROC Abbruch zyklische Erfassung.

PROCESS Erfassung:
  Holt die aktuelle Systemzeit und die ak-
  tuellen Zustandswerte. Die Zustandswerte
  werden geeignet konvertiert. Falls Auf-
  träge vorliegen werden diese erledigt.
ENDPROCESS Erfassung.

ENDMODULE Prozeßdatenerfassung.


MODULE Prozeßdatenverarbeitung:

DEFINES
  FOR Prozeßdatenerfassung:
    Melde Ergebnis,
    Auftragszettel,
    Maximale Aufträge.


APPLIES
  FROM Prozeßdatenerfassung:
    Erteile Auftrag,
    Starte zyklische Erfassung,
    Abbruch und erneute Erfassung,
    Abbruch zyklische Erfassung,
    Anlagenzustand,
    Anzahl Zustandswerte.
  FROM SYSTEM:
    DATE,
    NOW.

TYPE Auftragszettel.

INV Maximale Aufträge FIXED.
```

```
PROC Melde Ergebnis
        (IN INV Ergebnis REF Anlagenzustand,
         IN VAR Auftrag REF Auftragszettel):
  Das gewünschte Ergebnis ist bei seinem
  Empfänger abgelegt worden und die Auf-
  tragserledigung wurde bestätigt.
ENDPROC Melde Ergebnis.
```

```
PROCESS Überprüfung:
  Holt Anlagenzustand innerhalb einer vorge-
  gebenen Zeit. Ist dies nicht möglich,
  wird die Erfassung abgebrochen und erneut
  gestartet. Tritt der Fehler auch nach
  n-maliger Wiederholung noch auf, wird das
  gesamte Programmsystem abgebrochen. Über-
  prüfung der Zustandswerte. Liegen einzel-
  ne Werte nicht in dem Toleranzbereich wird
  ein Alarm mit entsprechender Meldung aus-
  gelöst.
ENDPROCESS Überprüfung.
```

```
PROCESS Protokollierung:
  Holt Anlagenzustand innerhalb einer vorge-
  gebenen Zeit. Ist dies nicht möglich, wird
  der Auftrag ignoriert und die Protokoll-
  lierung für diesen Zyklus entfällt. Be-
  rechnung des Zustandkennwerts (=Mittelwert
  der gültigen Zustandswerte). Ausgabe von
  Zustandskennwert und Meßzeitpunkt.
ENDPROCESS Protokollierung.
```

```
PROCESS Start und Abbruch der Überwachung:
  Holt gültige Zykluszeiten und startet die
  zur automatischen Überwachung benötigten
  Prozesse und veranlaßt ihre periodische
  Wiederholung.
  Warten auf Bediener-Anforderung.
  Abbruch aller Prozesse und Streichung
  ihrer Einplanungen.
ENDPROCESS Start und Abbruch der Überwachung.

ENDMODULE Prozeßdatenverarbeitung.
```

A.4 Algorithmischer Entwurf der Problemlösung

Zur Konstruktion der einzelnen Algorithmen benutzen wir eine Algorithmische Sprache, die sich semantisch weitgehend an der Sprache P E A R L orientiert, ohne Berücksichtigung der syntaktischen Feinheiten.

A.4.1 Algorithmischer Entwurf des Moduls Prozeßdatenverarbeitung

Zur Lösung dieser Aufgabe müssen der Erfassungsprozeß, die Auftragsverwaltung und die Steuerung des Erfassungsprozesses konstruiert werden.

Konstruktion des Erfassungsprozesses

Es müssen drei Teilprobleme gelöst werden:
- Aktuelle Systemzeit holen
- Erfassung der Zustandswerte mit geeigneter Umformung
- Erledigung von Aufträgen .

Die Zustandswertgeber können, ähnlich wie ein Datenbereich, indiziert angesprochen werden. Zur weiteren Verarbeitung wird ein erfaßter 'Roh'-Zustandswert mit Hilfe eines linearen Anpassungsfaktors in einen entsprechenden 'Fertig'-Zustandswert umgewandelt. Speichern wir nun noch den Meßzeitpunkt und die Zustandswerte in einer Variablen des Typs Anlagenzustand ab, der ebenfalls konstruiert werden muß, dann kommen wir zur folgenden Lösung:

```
TYPE Anlagenzustand
  STRUCT [Meßzeitpunkt CLOCK,
          Zustandswert(1:Anzahl Zustandswerte)
                                        FLOAT]
PROCESS Erfassung:
  Schnappschuß.Meßzeitpunkt := NOW
```

Insbesondere werden dabei die durch einen P E A R L - Compiler bedingten Spracheinschränkungen ignoriert; sie kommen erst in dem Abschnitt A.5 (Implementierung der Problemlösung) zum Tragen.

```
  FOR i TO Anzahl Zustandswerte
  REPEAT
    Roh Zustandswert := Zustandswertgeber(i)
    Schnappschuß.Zustandswert(i):=
         Roh Zustandswert * Anpassungsfaktor
```

```
  END
  WHILE Empfange Auftrag(Auftrag)
  REPEAT
    Melde Ergebnis(Schnappschuß,Auftrag)
  END
ENDPROCESS Erfassung
```

Konstruktion der Auftragsverwaltung

Es müssen die folgenden Teilprobleme gelöst werden:
- Entwurf eines geeigneten Auftragsblocks
- Konstruktion der Zugriffsroutinen.

Der Auftragsblock besteht aus einer Reihe von Auftragszetteln. Die Struktur eines Auftragszettels und die maximale Anzahl der Auftragszettel wird von den Auftraggebern festgelegt.

Wir realisieren den Auftragsblock als Ringpuffer. Ein Ringpuffer besteht aus einer Reihe von leeren Elementen, die von den Auftraggebern (allgemein: Produzenten) gefüllt, und einer Reihe von vollen Elementen, die von dem Auftragnehmer (allgemein: Konsument) geleert werden können.

Auftraggeber und Auftragnehmer greifen auf den Auftragsblock mittels zweier Zeiger Füllzeiger und Leerzeiger zu. Die beiden Zeiger bewegen sich in die gleiche Richtung, ohne sich gegenseitig zu überholen.

Ein korrekter Zugriff wird erreicht, wenn die beiden folgenden Bedingungen beachtet werden:
Der Auftragnehmer kann erst dann einen Auftrag empfangen, wenn dieser erteilt wurde, es muß also gelten:

$$0 \leqslant \text{empfangene Aufträge} < \text{erteilte Aufträge}$$

$$\Downarrow$$

$$\text{erteilte Aufträge} - \text{empfangene Aufträge} > 0.$$

Der Auftraggeber kann nur dann einen Auftrag erteilen, wenn Platz in dem Auftragsblock ist, es muß also gelten:

$$0 \leqslant \text{erteilte Aufträge} < \text{empfangene Aufträge} + \text{Maximale Aufträge}$$

$$\Downarrow$$

$$\text{erteilte Aufträge} - \text{empfangene Aufträge} < \text{Maximale Aufträge}.$$

Nennen wir den Ausdruck 'erteilte Aufträge - empfangene Aufträge' Auftragsstand. Die Variable Auftragsstand ist am Anfang mit dem Wert 0 initialisiert. Sie wird bei jedem Erteile Auftrag um 1 erhöht und bei jedem Erledige Auftrag um 1 verringert. Die beiden Zeiger sollen sich in die gleiche Richtung bewegen, ohne sich zu überholen. Am Anfang sind die beiden Zeiger gleich; sie zeigen auf das erste Pufferelement. Werden beide Zeiger durch dieselbe Funktion

```
Zeiger + 1 MODULO Maximale Aufträge
```

geändert, dann zählen sie dieselbe Reihe von Pufferelementen ab. Daß die beiden Zeiger sich nicht gegenseitig überholen, ist durch die oben genannten Bedingungen sichergestellt.

Da mehrere Prozesse nebenläufig auf den Auftragsblock zugreifen, darf der Zugriff auf den Puffer nur unter wechselseitigen Ausschluß stattfinden; die Zugriffsroutinen müssen also synchronisiert werden. Nun können wir unseren Algorithmus formulieren:

```
Auftragsverwaltung SEMA PRESET(1)
Auftragsblock(0 : Maximale Aufträge - 1)
                              Auftragszettel
Füllzeiger, Leerzeiger, Auftragsstand
                        FIXED INITIAL(0,0,0)

BIT(1) PROC Erteile Auftrag
          (INV Auftrag REF Auftragszettel):
  REQUEST Auftragsverwaltung
  IF Auftragsstand < Maximale Aufträge
  THEN Auftragsblock(Füllzeiger) := Auftrag
       Füllzeiger := (Füllzeiger+1)
                        REM Maximale Aufträge
       Auftragsstand := Auftragsstand + 1
       RELEASE Auftragsverwaltung
       RETURN(Auftrag angenommen)
  ELSE RELEASE Auftragsverwaltung
       RETURN(NOT Auftrag angenommen)
  FIN
ENDPROC Erteile Auftrag

BIT(1) PROC Empfange Auftrag
            (Auftrag REF Auftragszettel):
  REQUEST Auftragsverwaltung
  IF Auftragsstand > 0
```

```
  THEN Auftrag :=Auftragsblock(Leerzeiger)
       Leerzeiger := (Leerzeiger + 1)
                     REM Maximale Aufträge
       Auftragsstand := Auftragsstand - 1
       RELEASE Auftragsverwaltung
       RETURN(Auftrag vorhanden)
  ELSE RELEASE Auftragsverwaltung
       RETURN(NOT Auftrag vorhanden)
  FIN
ENDPROC Empfange Auftrag
```

Konstruktion der Steuerung des Erfassungsprozesses

Die algorithmische Formulierung der beiden Prozeduren Starte zyklische Erfassung und Abbruch der zyklischen Erfassung ist recht einfach:

```
PROC Starte zyklische Erfassung
                    (INV Zyklus DURATION):
  NORMALIZE
  ALL Zyklus ACTIVATE Erfassung
ENDPROC Starte zyklische Erfassung

PROC Abbruch zyklische Erfassung:
  PREVENT Erfassung
  TERMINATE Erfassung
ENDPROC Abbruch zyklische Erfassung
```

Nur müssen wir noch die Prozedur Abbruch und erneute Erfassung konstruieren. Wir müssen dabei, um die Verklemmung zu lösen, auf einen sicheren Zustand zurückkehren - allgemein: eine "error recovery" durchführen:

```
PROC Abbruch und erneute Erfassung:
  Streiche unerledigte Aufträge
  NORMALIZE
  TERMINATE Erfassung
ENDPROC Abbruch und erneute Erfassung
```

Bei der Verfeinerung der abstrakten Anweisung Streiche unerledigte Aufträge müssen wir berücksichtigen, daß dabei auf den Auftragsblock zugegriffen werden muß, d.h. wir müssen synchronisieren. Desweiteren müssen wir berücksichtigen, daß durch die Normierung der Prozeßperipherie der blockierte Erfassungsprozeß befreit werden kann. Die folgende Terminierung kann genau dann, während sich der Prozeß in seinem kritischen Bereich befindet, wirksam werden. Dies hat fatale Folgen, da dadurch die Reihenfolge der Synchronisationsoperationen verletzt wird, was letztlich zu einer nicht lösbaren Verklemmung führt. Dies können wir nur verhindern, indem wir den gesamten Wiederanlaufvorgang zu einem kritischen Bereich machen, d.h. indem wir die Synchronisation auf die gesamte Prozedur ausdehnen. Wir kommen zur endgültigen Lösung:

```
PROC Abbruch und erneute Erfassung:
  REQUEST Auftragsverwaltung
  Auftragsstand:=Füllzeiger:=Leerzeiger:=0
  NORMALIZE
  TERMINATE Erfassung
  RELEASE Auftragsverwaltung
ENDPROC Abbruch und erneute Erfassung
```

A.4.2 Algorithmischer Entwurf des Moduls Prozeßdatenverarbeitung

Zur Lösung dieser Aufgabe müssen die drei Prozesse Start und Abruch der automatischen Überwachung, Überprüfung und Protokollierung und die Ergebnisverwaltung konstruiert werden.

Den Prozeß Start und Abbruch Überwachung können wir direkt hinschreiben:

```
PROCESS Start und Abbruch Überwachung:
   Melde den Start mit Angabe
     von Datum und Uhrzeit
   Hole gültige Zykluszeiten
   Starte zyklische Erfassung(Zyklus1)
   ALL Zyklus2 ACTIVATE Überprüfung
   ALL Zyklus3 ACTIVATE Protokollierung

   WHEN Bediener Anforderung RESUME

   Abbruch zyklische Erfassung
   PREVENT Überprüfung
   TERMINATE Überprüfung
   PREVENT Protokollierung
   TERMINATE Protokollierung
   Melde den Abbruch mit Angabe
     von Datum und Uhrzeit
ENDPROCESS Start und Abbruch Überwachung
```

Zum Start der drei Prozesse benötigen wir die drei Zykluszeiten, die zuerst eingelesen werden. Für die Zykluszeitwerte soll gelten:

$0 < f1 \leq f2 \leq f3$.

Die Bedingung:

$0 < f1$

ist jedoch zu allgemein, denn während eines Zyklus sollte der Erfassungsprozeß auch alle seine Aufgaben lösen können. Wir konkretisieren die Bedingung daher wir folgt:

Mittlere Abarbeitungszeit des Prozesses $\leq f1$.

Zum Einlesen der Zykluszeiten konstruieren wir uns eine Prozedur Hole gültige Zykluszeiten. In der Prozedur werden solange Zeitwerte eingelesen, bis sie die Bedingung erfüllen:

```
PROC Hole gültige Zykluszeiten:
  REPEAT
    Melde, daß Zykluszeitwerte eingegeben
    werden sollen,mit Angabe der Bedingung
    Lese die Zykluszeiten
    IF Zyklus1 ≥ Mittlere Abarbeitungszeit
    THEN IF Zyklus2 ≥ Zyklus1
         THEN IF Zyklus3 ≥ Zyklus2
              THEN RETURN FIN
         FIN
    FIN
  END
ENDPROC Hole gültige Zykluszeiten
```

Konstruktion des Überprüfungsprozesses

Es müssen die folgenden Teilprobleme gelöst werden:

- Hole den Anlagenzustand in vorgegebener Zeit
- Geeignete Reaktion bei einer Zeitüberschreitung
- Überprüfung der Zustandswerte und Alarmmeldung, falls Zustandswerte nicht im Toleranzbereich liegen.

Wir kommen zu einer ersten algorithmischen Formulierung:

```
PROCESS Überprüfung:
  ON Zeitüberschreitung: Geeignete Reaktion
  Hole Anlagenzustand in vorgegebener Zeit
  Prüfe alle Zustandswerte und melde
    die fehlerhaften
ENDPROCESS Überprüfung
```

Das Teilproblem Hole Anlagenzustand in vorgegebener Zeit werden wir erst im Anschluß an die Konstruktion der drei Prozesse lösen. Wir legen aber fest, daß eine Zeitüberschreitung signalisiert wird, falls der Anlagenzustand nicht rechtzeitig gelesen werden kann. Abhängig von der Anzahl der Wiederholungen kann die geeignete Reaktion auf eine Zeitüberschreitung entweder der Abbruch und ein erneuter Start des Erfassungsprozesses oder der Abbruch der gesamten automatischen Überwachung sein:

```
ON Zeitüberschreitung:
  IF Wiederholung < Maximale Wiederholung
  THEN Wiederholung := Wiederholung + 1
       Abbruch und erneute Erfassung
       TERMINATE
  ELSE Melde permanente Störung
       PREVENT Start und Abbruch Überwachung
       CONTINUE Start und Abbruch Überwachung
       TERMINATE
  FIN
```

Fehlerhafte Zustandswerte sind alle die, die außerhalb des vorgegebenen Toleranzbereiches liegen. Die abstrakte Anweisung Prüfe alle Zustandswerte und melde die fehlerhaften kann wie folgt verfeinert werden:

```
FOR i TO Anzahl Zustandswerte
REPEAT
  IF Zustandswert(i) < Untere Toleranz Grenze
     OR
     Zustandswert(i) > Obere Toleranz Grenze
  THEN Melde fehlerhafte Zustandswerte
  FIN
END
```

Fassen wir unseren gesamten Algorithmus zusammen:

```
PROCESS Überprüfung:
  ON Zeitüberschreitung:
    IF Wiederholung < Maximale Wiederholung
```

```
    THEN Wiederholung := Wiederholung + 1
         Abbruch und erneute Erfassung
         TERMINATE
    ELSE Melde permanente Störung
         PREVENT Start und Abbruch Überwachung
         CONTINUE Start und Abbruch Überwachung
         TERMINATE
    FIN

  Hole Anlagenzustand in vorgegebener Zeit
                          (Auftrag, Wartezeit)
  FOR i TO Anzahl Zustandswerte
  REPEAT
    IF Zustandswert(i) < Untere Toleranz Grenze
       OR
       Zustandswert(i) > Obere Toleranz Grenze
    THEN Melde fehlerhaften Zustandswert
    FIN
  END
  Wiederholung := 0
ENDPROCESS Überprüfung
```

Konstruktion des Protokollierungsprozesses

Folgende Teilprobleme müssen gelöst werden:
- Hole Anlagenzustand in vorgegebener Zeit
- Geeignete Reaktion bei einer Zeitüberschreitung
- Berechnung eines Zustandskennwertes mit anschließender Ausgabe.

Die algorithmische Formulierung der Zeitüberschreitung ist sehr einfach, denn bei einer Zeitüberschreitung wird der Auftrag ignoriert und die Protokollierung entfällt:

```
ON Zeitüberschreitung: TERMINATE
```

Zur Bestimmung des Zustandkennwertes muß der Mittelwert der gültigen Zustandswerte berechnet werden.
Zur Berechnung des Mittelwertes:

$$m = 1/n * \sum_{i=1}^{n} z_i$$

muß als erstes die Summe der gültigen Zustandswerte bestimmt werden:

$$summe = (z_1 + z_2 + z_3 + \ldots + z_n).$$

Der nun gefundene Ausdruck für die Summe läßt sich algorithmisch unter Verwendung einer endlichen Wiederholung recht einfach formulieren. Wichtig ist dabei die korrekte Anfangsbelegung der Variablen Summe, da sonst ein falsches Summenergebnis berechnet wird:

```
Summe := 0
FOR i TO n
REPEAT
  Summe := Summe + z_i
END
```

Generell lassen sich Folgen oder Reihen durch endliche Wiederholung lösen. Die verwendete Lösungsmethode wird Rekurrenz-Beziehung oder Rekursions-Relation genannt. Rekurrenz-Beziehungen enthalten immer zwei kennzeichnende Bestandteile, die Anfangsbedingung (Rekursionsbeginn) und die Wiederholungsvorschrift (Rekursionsschritt).

Zur Mittelwertbildung werden nur Zustandswerte benutzt, die innerhalb des Toleranzbereiches liegen. Falls es keinen gültigen Zustandswert gibt, wird ein Standardkennwert ausgegeben. Wir kommen nun zur endgültigen Lösung:

```
PROCESS Protokollierung:
  ON Zeitüberschreitung: TERMINATE

  Hole Anlagenzustand in vorgegebener Zeit
                          (Auftrag, Wartezeit)
  Anzahl := 0
  Summe := 0
  FOR i TO Anzahl Zustandswerte
  REPEAT
    IF Zustandswert(i) ≥ Untere Toleranz Grenze
       AND
       Zustandswert(i) ≤ Obere Toleranz Grenze
    THEN Summe := Summe + Zustandswert(i)
         Anzahl := Anzahl + 1
    FIN
  END
  Zustandskennwert :=
    IF Anzahl == 0 THEN Standardkennwert
                   ELSE Summe / Anzahl FIN
  Melde Zustandskennwert und Meßzeitpunkt
ENDPROCESS Protokollierung
```

Konstruktion von Hole Anlagenzustand in vorgegebener Zeit

Die beiden Auftraggeber bedienen sich der Prozedur Hole Anlagenzustand in vorgegebener Zeit. In der Prozedur wird als erstes ein Auftrag erteilt und dann gewartet, bis der Auftrag erledigt ist. Damit während der Wartezeit auch tatsächlich ein Auftrag erledigt werden kann, sollte die Wartezeit nicht kleiner als die Zykluszeit des Erfassungsprozesses sein (sie sollte aber auch nicht größer als die eigene Zykluszeit sein). Kann der Auftrag nicht in der angegebenen Zeit erledigt werden, wird eine Zeitüberschreitung signalisiert.

Jetzt muß noch geklärt werden, wieviele Aufträge maximal gespeichert werden müssen. Die minimale Anzahl von zwei Aufträgen ist sofort einsichtig, da die beiden Auftraggeber quasi parallel je einen Auftrag erteilen können. Sind die Zykluszeiten der beiden Auftraggeber annähernd gleich, dann kann in einer potentiellen Verklemmungssituation der Überprüfungsprozeß genau einen Auftrag und der zweite Prozeß maximal zwei Aufträge erteilen, bevor die Verklemmung gelöst werden kann.

Es müssen also maximal drei Aufträge gespeichert werden.

Falls ein vierter Auftrag gespeichert werden soll, können wir annehmen, daß ein Systemfehler vorliegt, und die automatische Überwachung abbrechen.

```
PROC Hole Anlagenzustand in vorgegebener Zeit
   (INV Auftrag REF AUFTRAGSZETTEL,
                INV Wartezeit DURATION):
  IF Erteile Auftrag(Auftrag)
  THEN IF Liegt Ergebnis vor
       THEN RETURN FIN
       AFTER Wartezeit RESUME
       IF Liegt Ergebnis vor
       THEN RETURN FIN
       INDUCE Zeitüberschreitung
  ELSE Melde Systemfehler
       PREVENT Start und Abbruch Überwachung
       CONTINUE Start und Abbruch Überwachung
       TERMINATE
  FIN
ENDPROC Hole Anlagenzustand in vorgegebener Zeit
```

Konstruktion der Ergebnisverwaltung

Es müssen ein Datentyp Auftragszettel und zwei Zugriffsroutinen Melde Ergebnis und Liegt Ergebnis vor konstruiert werden.

Der Auftragszettel besteht aus zwei Komponenten. Die eine Komponente enthält einen Verweis, wohin das gewünschte Ergebnis gebracht werden soll, die zweite Komponente enthält ebenfalls einen Verweis, eine Auftragsquittung, mit dessen Hilfe ein Auftraggeber die Erledigung seines Auftrags feststellen kann:

```
TYPE AUFTRAGSZETTEL
   STRUCT [Auftragsquittung REF BIT(1),
           Ergebnis REF ANLAGENZUSTAND]
```

Da auf den einzelnen Auftragszetteln mehrere Prozesse gleichzeitig operieren können, müssen wir die beiden Zugriffsroutinen synchronisieren:

```
Ergebnisverwaltung SEMA PRESET(1)

PROC Melde Ergebnis
          (INV Ergebnis REF ANLAGENZUSTAND,
               Auftrag REF AUFTRAGSZETTEL):
  REQUEST Ergebnisverwaltung
  CONT Auftrag.Ergebnis := Ergebnis
  CONT Auftrag.Auftragsquittung :=
                          Auftrag erledigt
  RELEASE Ergebnisverwaltung
ENDPROC Melde Ergebnis

BIT(1) PROC Liegt Ergebnis vor
           (Auftrag REF AUFTRAGSZETTEL):
  REQUEST Ergebnisverwaltung
  Ergebnis := CONT Auftrag.Auftragsquittung
  RELEASE Ergebnisverwaltung
  RETURN(Ergebnis)
ENDPROC Liegt Ergebnis vor
```

A.5 Implementierung der Problemlösung

Wir implementieren unsere Problemlösung mit Hilfe eines an der TU Berlin entwickelten P E A R L - Programmiersystems für HP 1000 - Rechenanlagen.

Wir nehmen die folgenden Konkretisierungen vor:

- Wir haben 128 Zustandswertgeber und als Zustandswerte werden Spannungswerte erfaßt. Zum Lesen der Zustandswerte benutzen wir einen A/D-Wandler mit 12 Bit-Auflösung und einen Meßbereich von -10V bis +10V.

  Die Zustandswerte werden mit Hilfe der TAKE-Anweisung eingelesen.

  Der Toleranzbereich geht von -5V bis +5V, der Wert des Anpassungsfaktors ist 0,004884 und als Standardkennwert legen wir 0V fest.

- Die globalen Konstanten und Typen entfallen, sie werden lokal in den beiden Moduln vereinbart.

  Die abstrakten Anweisungen Melde... und Lese... werden in äquivalente PUT- und GET-Anweisungen umgesetzt.

- Die drei Systemfunktionen müssen wie folgt importiert werden:

```
ENTRY NORMALIZE GLOBAL(DIRECT)
ENTRY NOW RETURNS(CLOCK) GLOBAL(DIRECT)
ENTRY DATE RETURNS(CHARACTER(8))
                              GLOBAL(DIRECT)
```

- Die Zykluszeiten werden bei Start der Überwachung eingegeben. Als mittlere Abarbeitungszeit legen wir 0,4 Sekunden fest.

  Die Wartezeit wird gleich Zyklus1 und die Anzahl der Wiederholungen gleich 3 gesetzt.

```
MODULE (prozessdatenerfassung);

SYSTEM;

  zustandswertgeber(1:128) : ANALOG_INPUT(0:7)*(0:15) <- ;

PROBLEM;

  SPECIFY
    zustandswertgeber () DATION IN BASIC,
    NORMALIZE ENTRY GLOBAL(DIRECT),
    NOW ENTRY RETURNS(CLOCK) GLOBAL(DIRECT),
    melde_ergebnis ENTRY(ANLAGENZUSTAND IDENTICAL,
                         AUFTRAGSZETTEL IDENTICAL)
                            GLOBAL(prozessdatenverarbeitung);

  LENGTH FIXED(15);
  LENGTH FLOAT(23);


  TYPE ANLAGENZUSTAND
    STRUCT [messzeitpunkt CLOCK, zustandswert(1:128) FLOAT];

  TYPE AUFTRAGSZETTEL
    STRUCT [auftragsquittung REF BIT(1), ergebnis REF ANLAGENZUSTAND];
```

```
/****          Auftrags - Verwaltung          ****/
/*-----------------------------------------------*/
DECLARE
  auftrags_verwaltung SEMA PRESET(1),
  maximale_auftraege INV FIXED INITIAL(3),
  auftragsblock(0:2) AUFTRAGSZETTEL,
  (fuellzeiger,leerzeiger,auftragsstand) FIXED INITIAL(0,0,0);

erteile_auftrag: PROCEDURE(auftrag INV AUFTRAGSZETTEL IDENTICAL)
                                    RETURNS(BIT(1)) REENT GLOBAL;
  REQUEST auftrags_verwaltung;
  IF auftragsstand < maximale_auftraege
  THEN auftragsblock(fuellzeiger) := auftrag;
       fuellzeiger := (fuellzeiger+1) REM maximale_auftraege;
       auftragsstand := auftragsstand + 1;
       RELEASE auftrags_verwaltung;
       RETURN('1'B) /*Auftrag angenommen*/;
  ELSE RELEASE auftrags_verwaltung;
       RETURN('0'B) /*NOT Auftrag angenommen*/;
  FIN;
END /*erteile_auftrag*/;

empfange_auftrag: PROCEDURE(auftrag AUFTRAGSZETTEL IDENTICAL)
                                          RETURNS(BIT(1));
  REQUEST auftrags_verwaltung;
  IF auftragsstand > 0
  THEN auftrag := auftragsblock(leerzeiger);
       leerzeiger := (leerzeiger+1) REM maximale_auftraege;
       auftragsstand := auftragsstand - 1;
       RELEASE auftrags_verwaltung;
       RETURN('1'B) /*Auftrag vorhanden*/;
  ELSE RELEASE auftrags_verwaltung;
       RETURN('0'B) /*NOT Auftrag vorhanden*/;
  FIN;
END /*empfange_auftrag*/;
/*===============================================*/

/****   Steuerung des Erfassungsprozesses   ****/
/*-----------------------------------------------*/


starte_zyklische_erfassung: PROCEDURE(zyklus INV DURATION) GLOBAL;
  CALL NORMALIZE;
  ALL zyklus ACTIVATE erfassung;
END /*starte_zyklische_erfassung*/;

abbruch_und_erneute_erfassung: PROCEDURE GLOBAL;
  REQUEST auftrags_verwaltung;
  auftragsstand := fuellzeiger := leerzeiger := 0;
  CALL NORMALIZE;
  TERMINATE erfassung;
  RELEASE auftrags_verwaltung;
END /*abbruch_und_erneute_erfassung*/;
```

```
abbruch_zyklische_erfassung: PROCEDURE GLOBAL;
  PREVENT erfassung;
  TERMINATE erfassung;
END /*abbruch_zyklische_erfassung*/;

/*=============================================*/


  /****          Erfassungsprozess          ****/
  /*---------------------------------------------*/

  erfassung: TASK;
    DECLARE
      schnappschuss ANLAGENZUSTAND,
      auftrag AUFTRAGSZETTEL,
      anzahl_zustandswertgeber INV FIXED INITIAL(128),
      roh_zustandswert FIXED,
      anpassungsfaktor INV FLOAT INITIAL(0.004884);

    schnappschuss.messzeitpunkt := NOW;
    FOR i TO anzahl_zustandswertgeber
    REPEAT
      TAKE roh_zustandswert FROM zustandswertgeber(i);
      schnappschuss.zustandswert(i) := TOFLOAT roh_zustandswert
                                          * anpassungsfaktor;
    END;
    WHILE empfange_auftrag(auftrag)
    REPEAT
      CALL melde_ergebnis(schnappschuss,auftrag);
    END;
  END /*erfassung*/;


  /*=============================================*/


MODEND /*prozessdatenerfassung*/;




MODULE (prozessdatenverarbeitung);

SYSTEM;

  konsole : DISPLAY(1);
  report: DISPLAY(2);
  zeitueberschreitung: EXCEPTION(33);
  bediener_anforderung: EVENT(6);

PROBLEM;
```

```
SPECIFY
  konsole DATION INOUT ALPHIC CONTROL(ALL),
  report DATION OUT ALPHIC CONTROL(ALL),
  zeitueberschreitung SIGNAL,
  bediener_anforderung INTERRUPT,
  NOW ENTRY RETURNS(CLOCK) GLOBAL(DIRECT),
  DATE ENTRY RETURNS(CHARACTER(8)) GLOBAL(DIRECT),
  erteile_auftrag ENTRY(AUFTRAGSZETTEL IDENTICAL) RETURNS(BIT(1))
                              REENT GLOBAL(prozessdatenerfassung),
  starte_zyklische_erfassung ENTRY(DURATION)
                              GLOBAL(prozessdatenerfassung),
  abbruch_und_erneute_erfassung ENTRY
                              GLOBAL(prozessdatenerfassung),
  abbruch_zyklische_erfassung ENTRY
                              GLOBAL(prozessdatenerfassung);

LENGTH FIXED(15);
LENGTH FLOAT(23);

TYPE ANLAGENZUSTAND
  STRUCT [messzeitpunkt CLOCK, zustandswert(1:128) FLOAT];

TYPE AUFTRAGSZETTEL
  STRUCT [auftragsquittung REF BIT(1), ergebnis REF ANLAGENZUSTAND];

/****          Ergebnis - Verwaltung          ****/
/*-----------------------------------------------*/


DECLARE
  ergebnis_verwaltung SEMA PRESET(1);


melde_ergebnis: PROCEDURE
           (ergebnis INV ANLAGENZUSTAND IDENTICAL,
                auftrag AUFTRAGSZETTEL IDENTICAL) GLOBAL;
  REQUEST ergebnis_verwaltung;
  CONT auftrag.ergebnis := ergebnis;
  CONT auftrag.auftragsquittung := '1'B /*Auftrag erledigt*/;
  RELEASE ergebnis_verwaltung;
END /*melde_ergebnis*/;

liegt_ergebnis_vor: PROCEDURE
     (auftrag INV AUFTRAGSZETTEL IDENTICAL)
                              RETURNS(BIT(1)) REENT;
  DECLARE erledigt BIT(1);
  REQUEST ergebnis_verwaltung;
  erledigt := CONT auftrag.auftragsquittung;
  RELEASE ergebnis_verwaltung;
  RETURN(erledigt);
END /*liegt_ergebnis_vor*/;

/*===============================================*/
```

```
/****   Verwaltungsobjekte und -prozedur    ****/
/*-----------------------------------------------*/

DECLARE
  anzahl_zustandswerte INV FIXED INITIAL(128),
  (untere_toleranz_grenze,obere_toleranz_grenze)
                              INV FLOAT INITIAL(-5.0,5.0);

hole_anlagenzustand_in_vorgegebener_zeit:
  PROCEDURE(auftrag AUFTRAGSZETTEL IDENTICAL,
               wartezeit INV DURATION ) REENT;

  IF erteile_auftrag(auftrag)
  THEN IF liegt_ergebnis_vor(auftrag)
       THEN RETURN; FIN;
       AFTER wartezeit RESUME;
       IF liegt_ergebnis_vor(auftrag)
       THEN RETURN; FIN;
       INDUCE zeitueberschreitung;
  ELSE PUT '*** SYSTEM-FEHLER ***'
         TO konsole BY A,SKIP;
       PREVENT start_und_abbruch_ueberwachung;
       CONTINUE start_und_abbruch_ueberwachung;
       TERMINATE;
  FIN;
END /*hole_anlagenzustand_in_vorgegebener_zeit*/;

/*===============================================*/



/****         Ueberpruefungsprozess          ****/
/*-----------------------------------------------*/

DECLARE
  wiederholung FIXED INITIAL(0),
  maximale_wiederholung INV FIXED INITIAL(3),
  wartezeit1 DURATION;

ueberpruefung: TASK;
  DECLARE
    schnappschuss ANLAGENZUSTAND,
    auftrag AUFTRAGSZETTEL,
    gueltiger_wert INV FLOAT INITIAL(0.0),
    (quittung, erste_fehlermeldung) BIT(1) INITIAL('0'B,'1'B);

  ON zeitueberschreitung:
    IF wiederholung < maximale_wiederholung
    THEN wiederholung := wiederholung + 1;
         CALL abbruch_und_erneute_erfassung;
         TERMINATE;
    ELSE PUT 'Permanente Stoerung der Prozessperipherie'
           TO konsole BY A,SKIP;
```

```
        PREVENT start_und_abbruch_ueberwachung;
        CONTINUE start_und_abbruch_ueberwachung;
        TERMINATE;
    FIN;

  auftrag.auftragsquittung := quittung;
  auftrag.ergebnis := schnappschuss;
  CALL hole_anlagenzustand_in_vorgegebener_zeit
                                    (auftrag,wartezeit1);
  FOR i TO anzahl_zustandswerte
  REPEAT
    IF schnappschuss.zustandswert(i)< untere_toleranz_grenze
       OR
       schnappschuss.zustandswert(i) >obere_toleranz_grenze
    THEN IF erste_fehlermeldung
         THEN erste_fehlermeldung := '0'B /*falsch*/;

              PUT '  ***  A L A R M  ***  ',
                  'Messzeitpunkt: ',schnappschuss.messzeitpunkt,
                  'Fehler erkannt: ',NOW,
                  'Fehlerhafte Zustandswerte: '
                TO konsole BY A,SKIP,(2)(A,T(10,1),SKIP),A,SKIP(2);
          FIN;
          PUT 'Zustandswertgeber: ', i,
              'Zustandswert: ',schnappschuss.zustandswert(i)
            TO konsole BY A,F(3),X(3),A,F(9,6),SKIP;
    FIN;
  END;
  wiederholung := 0;
END /*ueberpruefung*/;

/*=============================================*/

/****      Protokollierungsprozessesses      ****/
/*---------------------------------------------*/
DECLARE
  wartezeit2 DURATION;

protokollierung: TASK;

  DECLARE
    quittung BIT(1) INITIAL('0'B),
    schnappschuss ANLAGENZUSTAND,
    anzahl FIXED,
    (summe,zustands_kennwert) FLOAT,
    standard_kennwert INV FLOAT INITIAL(0.0),
    auftrag AUFTRAGSZETTEL;

  ON zeitueberschreitung: TERMINATE;

  auftrag.ergebnis := schnappschuss;
  CALL hole_anlagenzustand_in_vorgegebener_zeit
                                    (auftrag,wartezeit);
```

```
    anzahl := 0;
    summe := 0.0;
    FOR i TO anzahl_zustandswerte
    REPEAT
      IF schnappschuss.zustandswert(i)  = untere_toleranz_grenze
         AND
         schnappschuss.zustandswert(i)  = obere_toleranz_grenze
      THEN summe := summe + schnappschuss.zustandswert(i);
           anzahl := anzahl + 1;
      FIN;
    END;
    zustands_kennwert :=
           IF anzahl == 0 THEN standard_kennwert
                          ELSE summe / anzahl FIN;
    PUT schnappschuss.messzeitpunkt, zustands_kennwert
      TO report BY T(10,1),X(3),F(9,6),SKIP;
  END /*protokollierung*/;
  /*=============================================*/

  /****  Start und Abbruch der Ueberwachung   ****/
  /*---------------------------------------------*/

  start_und_abbruch_ueberwachung: TASK;

    DECLARE zyklus(1:3) DURATION;

    hole_gueltige_zykluszeiten: PROCEDURE;
      REPEAT
        PUT 'Bitte Zykluszeiten in folgender Reihenfolge eingeben:',
            '1. Erfassungszyklus (f1)',
            '2. Ueberwachungszyklus (f2)',
            '3. Protokollierungszyklus (f3)',
            'es gilt: 0.4 SEC <= f1 <= f2 <= f3'
          TO konsole BY (5)(A,SKIP);
        GET zyklus(1:3) FROM konsole BY (3)(D(24,3)),SKIP;
        IF zyklus(1) >= 0.4 SEC
        THEN IF zyklus(2) >= zyklus(1)
             THEN IF zyklus(3) >= zyklus(2)
                  THEN RETURN; FIN;
             FIN;
        FIN;
      END;
    END /*hole_gueltige_zykluszeiten*/;

    PUT 'Start der Ueberwachung am: ',DATE,' um: ', NOW
      TO konsole BY (3)A,T(10,1),SKIP;
    CALL hole_gueltige_zykluszeiten;
    CALL starte_zyklische_erfassung(zyklus(1));
    wartezeit1 := wartezeit2 := zyklus(1);
    ALL zyklus(2) ACTIVATE ueberpruefung;
    ALL zyklus(3) ACTIVATE protokollierung;

    WHEN bediener_anforderung RESUME;
```

```
      CALL abbruch_zyklische_erfassung;
      PREVENT ueberpruefung;
      TERMINATE ueberpruefung;
      PREVENT protokollierung;
      TERMINATE protokollierung;
      PUT 'Abbruch der Ueberwachung am: ',DATE,' um: ', NOW
        TO konsole BY (3)A,T(10,1),SKIP;
    END /*start_und_abbruch_ueberwachung*/;


    /*=============================================*/


MODEND /*prozessdatenverarbeitung*/;
```

Literaturhinweise

Bauer, F.L., Wössner, H.: Algorithmische Sprache und Programmentwicklung. Springer Verlag, Berlin, Heidelberg, New York, 1981.

Brinch Hansen, P.: The Architecture of Concurrent Programs. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1977.

Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1976.

Klaus Nagel, Herbert Nebel

Fachbereichsübergreifender Forschungsschwerpunkt Prozeßdatenverarbeitung im Rechnerverbund (FSP-PV) der Technischen Universität Berlin

Postanschrift: Technische Universität Berlin
Prozeßrechnerverbund-Zentrale
Sekr.: HE 3 / Einsteinufer 19
1000 Berlin 10
Tel.: (030) 314 51 54

PEARL-Kompiliersysteme für den Lehr- und Übungsbetrieb an Universitäten

Prof. Dr. H. Rzehak, Hochschule der Bundeswehr München

1. Der Bereich "Programmieren von Realzeitsystemen" im Lehrbetrieb einer Universität

Der Bereich "Realzeitsysteme" taucht an unseren Universitäten sowohl in den Studienplänen der Informatik als auch der Elektrotechnik auf. Trotz der Verschiedenheit der Studienpläne an den einzelnen Universitäten kann man davon ausgehen, daß im Studium der Elektrotechnik die Erstellung von Anwendungsprogrammen für die Prozeßsteuerung stärker im Vordergrund steht, während in der Informatik der Bereich der Grundsoftware sowie der Software-Erstellungsmethoden und -Hilfsmittel größere Beachtung findet. Trotz dieser etwas unterschiedlichen Sicht auf die Systeme kann im allgemeinen von den folgenden Voraussetzungen ausgegangen werden:

1.) Die Studenten haben bei der Beschäftigung mit Realzeitsystemen bereits allgemeine Programmierkenntnisse.

2.) Hauptzweck ist nicht das Erlernen einer Programmiersprache, sondern allgemeine Programmiermethodik und Systemkenntnis.

3.) Es gilt, die besonderen Probleme beim Testen von Realzeitprogrammen kennenzulernen und entsprechendes Methodenwissen zu erarbeiten.

4.) In den Diplomarbeiten sollen einzelne Studenten die Möglichkeit haben, Probleme zu bearbeiten, für die noch keine gängigen Lösungen bekannt sind.

Unter Beachtung dieser Randbedingungen dient eine höhere Programmiersprache in den Vorlesungen zur prägnanten Formulierung von Beispielen und wird in den Übungen zur Lösung von Aufgaben verwendet. Hier sollte den Studenten die Möglichkeit gegeben werden, am Rechner selbst zu arbeiten und die entworfenen Programme zu testen. In Studien- und Diplomarbeiten wird eine höhere Programmiersprache zum Teil wegen der bequemen Programmierung verwendet, zum Teil soll darauf aufsetzend an neuen Methoden zur Programmentwicklung oder neuen Systemkonzepten gearbeitet werden.

2. Forderungen an eine Programmiersprache für Realzeitsysteme im Lehrbetrieb

Hierzu gehören natürlich alle guten Eigenschaften einer höheren Programmiersprache wie leicht erlernbar, selbstdokumentierend, wenig fehlerträchtig und andere. Speziell für den Bereich der Lehre treten einige Gesichtspunkte in den Vordergrund, auf die hier näher eingegangen werden soll.

Die Programmiersprache ist zunächst Darstellungshilfsmittel zur Formulierung von Beispielen. Sind zum Verständnis der Beispiele erst lange Erklärungen erforderlich, so verlieren sie ihren didaktischen Wert. Typische Probleme sollen sich daher in wenigen Programmzeilen formulieren lassen, wobei ein Student mit elementaren Programmierkenntnissen eine zutreffende intuitive Vorstellung über die Bedeutung gewinnen sollte, ohne daß es weiterer Erläuterungen bedarf. Die Sprache muß daher entsprechend mächtige Anweisungen insbesondere für typische Realzeitfunktionen enthalten, die weitgehend selbsterklärend sind. Die Darstellung sollte sich an dem orientieren, was in modernen Programmiersprachen üblich ist.

Im Übungsbetrieb soll der Student selbständig Aufgaben lösen, wobei die Zeit hierfür - vor allem Testzeit am Rechner - knapp bemessen ist. Die Aufgabenstellung soll nicht allzu realitätsfern sein. Damit der Student zum Kern des Problems vordringt und ihm die Lösung als Erfolgserlebnis gelingt, müssen ihm fertig programmierte Lösungen der Teilbereiche vorgegeben werden, die nicht zum Kern des Problemes gehören, oder die triviale Routinearbeit darstellen. Man kann für diesen Zweck geeignete Unterprogramme (Prozeduren) bereitstellen, jedoch kann man hiermit nicht alle Bedürfnisse abdecken. Z.B. lassen sich keine vom ganzen Programm aus zugreifbaren Datenobjekte einrichten

und gegebenenfalls vorbesetzen; auch sind die resultierenden Problemlösungen häufig nicht gerade glücklich strukturiert. Wünschenswert ist es daher, wenn Teile des Programmes vorübersetzbar sind, die allgemeinere Schnittstellen haben und nicht an das Schema von Prozedurvereinbarungen und -aufruf gebunden sind.

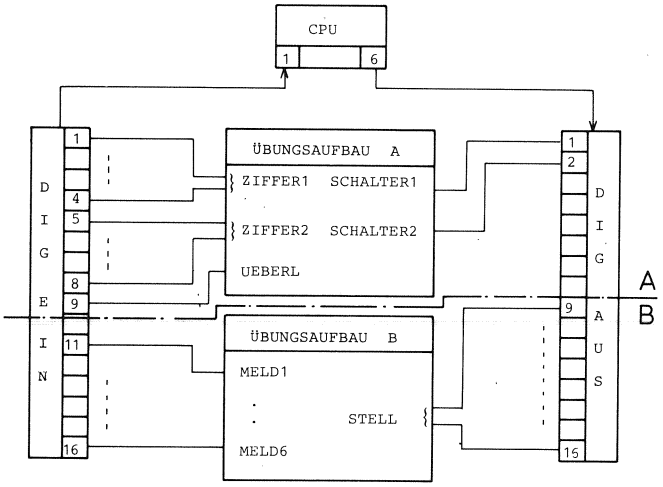
Betreute Übungen, in denen die Studenten durch einen Mitarbeiter angeleitet werden, können aus personellen und stundenplantechnischen Gesichtspunkten nur in begrenztem Umfang angeboten werden. Dabei ist ein Mitarbeiter durchaus in der Lage, mehrere Übungsgruppen parallel zu betreuen. Die Rechensysteme lassen bei entsprechendem Ausbau, der bei den heutigen Preisen für Hauptspeicher in Halbleiter-Technologie auch kostengünstig ist, das parallele Betreiben mehrerer Übungsprozesse grundsätzlich zu. Für einen störungsfreien Betrieb müssen diese allerdings auch bei Programmfehlern seiteneffektfrei sein, denn bei Übungen ist naturgemäß ein Großteil der Programmläufe fehlerhaft. Die Betriebssysteme sehen in der Regel gewisse Schutzmechanismen (z.B. Speicherschutz) vor; nicht geschützt ist gewöhnlich jedoch der unbeabsichtigte Zugriff auf die Prozessperipherie. Wichtig ist, daß Fehler, die zu einem Systemzusammenbruch oder zu einer nicht gewollten gegenseitigen Beeinflussung der Übungsprozesse führen können, möglichst schon bei der Kompilation erkannt werden, mindestens jedoch im Laufzeitsystem abgefangen werden können. Dies ist zuerst eine Forderung an die Sprache, die so aufgebaut sein muß, daß diese Fehler möglichst statisch, d.h. zur Kompilationszeit, erkennbar sind. Eine mögliche gegenseitige Störung ist insbesondere die falsche Adressierung der Prozessperipherie. Da Zugriffe auf vorhandene Peripherie gewöhnlich als berechtigt angesehen werden, können sich die Übungsprozesse gegenseitig stören, indem sie auf Peripherieendstellen anderer Übungsprozesse zugreifen.

Es bleibt nun zu klären, wie PEARL diese drei Hauptforderungen erfüllt. In PEARL sind alle Konzepte moderner Programmiersprachen realisiert, insbesondere sind ausreichende Hilfsmittel zur Abstraktion enthalten (z.B. Typ-Deklaration, Referenzen, Interfaces). Die Elemente zur Beschreibung der Realzeiteigenschaften und der Ein- und Ausgabe von Prozessignalen sind sehr problemnah gestaltet. Ohne im einzelnen weiter darauf einzugehen, kann gesagt werden, daß es in dieser Hinsicht sicherlich keine gleichwertige andere Programmiersprache gibt. PEARL eignet sich in hervorragender Weise zur Formulierung von Beispielen.

Die zweite Forderung wird durch das Modulkonzept von PEARL erfüllt. Man kann diejenigen Programmbausteine, die man dem Studenten als Hilfsmittel zur Verfügung stellen möchte, in vorübersetzte Modulen packen. Der Zugriff auf Objekte dieser Modulen kann in geeigneter Weise beschränkt werden (GLOBAL-Attribut). Gleichzeitig kann der Student die Wirkung der Programmbausteine an Hand des Quellentextes nachvollziehen. Die Möglichkeit zur Gestaltung der Modulschnittstellen muß notwendigerweise durch die Sprache eingeschränkt werden (in PEARL durch die Objekte, die das GLOBAL-Attribut tragen können), eine Beschränkung auf klare, übersichtliche Schnittstellen ist jedoch auch aus didaktischen Gründen notwendig.

Dem dritten Punkt schließlich kommt PEARL in hervorragender Weise entgegen. Jeder Benutzer programmiert seine eigene PEARL-Welt, die gegenüber den PEARL-Welten der anderen Benutzer abgeschlossen ist. Da alle Objekte dieser PEARL-Welt deklariert werden müssen, so ist eine weitgehend statische Prüfung auf Zulässigkeit der Benutzung möglich. Dies gilt auch für die Endstellen der Prozessperipherie. In Bild 1 ist dies schematisch dargestellt. Die zu jedem Übungsbetrieb gehörenden Prozessendstellen werden im SYSTEM-Teil mit einem Bezeichner versehen, unter dem sie im PROBLEM-Teil aufgerufen werden können. Das Kompilierer-System kann nun feststellen, ob auf Geräte zugegriffen wird, die im Systemteil benannt sind.

Da im SYSTEM-Teil die benutzten Geräte aufgeführt sein müssen, so beschränkt sich die Kontrolle über eine mögliche Störung der Übungsprozesse durch falsche Geräteadressierung auf die Kontrolle des SYSTEM-Teils. Dieser kann zunächst als vorübersetzter MODULE zur Verfügung gestellt werden. Eine Überprüfung durch den Betreuer ist jedoch auch leicht durchführbar, wenn der Student den SYSTEM-Teil selbst erstellen soll. Grundsätzlich ist es auch möglich, beim Laden der Programme automatisch zu prüfen, ob die gleichzeitig zu betreibenden PEARL-Welten verträglich sind, also z.B. nicht auf identische Klemmen der Prozessperipherie zugreifen.




```
MODULE; */ PEARL-Welt A */
SYSTEM;
  DIGEIN → CPU*1;
  DIGAUS ← CPU*6;
  ZIFFER1 : → DIGEIN*(1:4);
  ZIFFER2 : → DIGEIN*(5:8);
  UEBERL  : → DIGEIN*9;
  SCHALTER1: ← DIGAUS*1;
  SCHALTER2: ← DIGAUS*2;
PROBLEM; */ Algorithmischer Teil
            für PEARL-Welt A */
  :
MODEND;
```

```
MODULE; */ PEARL-Welt B */
SYSTEM;
  DIGEIN → CPU*1;
  DIGAUS ← CPU*6;
  MELD1  : → DIGEIN*11;
  :
  MELD6  : → DIGEIN*16;
  STELL  : ← DIGAUS*(9:16);
PROBLEM; */ Algorithmischer Teil
            für PEARL-Welt B */
  :
MODEND;
```

Bild 1. Definition von zwei PEARL-Welten in der Prozessperipherie

3. Spezielle Forderungen aus dem Übungsbetrieb an ein PEARL-Kompilierer-System

Ein Kompiliersystem für den Übungsbetrieb muß leicht handhabbar sein. Der Student soll nicht durch Tätigkeiten und Überlegungen abgelenkt werden, die mit dem Lernziel einer Übung nichts zu tun haben. Auch darf die Bedienung des Kompilierers keine Kenntnisse (z.B. über das Betriebssystem) voraussetzen, die der Student noch nicht haben kann. Zum Bedienen sollen möglichst wenig Kommandos notwendig sein, die außerdem möglichst keine Parameter enthalten, mit denen der Student nichts anzufangen weiß. Leider sind die meisten verfügbaren Kompiliersysteme wenig bedienungsfreundlich. Es fällt auf, daß vor allem für das Binden auch bei wenig komplexen Programmen eine Vielzahl von Bindeanweisungen erstellt werden muß, die zudem betriebssystemspezifische Parameter enthalten. An einem System waren für ein Programm, bestehend aus einem Modul mit 5 Prozeduren und 9 Tasks, insgesamt 60 (!) Bindeanweisungen nötig. Andere Systeme schneiden hier nicht viel besser ab. Dem Verfasser sind nur zwei rühmliche Ausnahmen bekannt, von denen eines allerdings nicht den vollen Sprachumfang von Basic PEARL verarbeitet und deshalb gewissermaßen außer Konkurrenz läuft. Hier ist Abhilfe dringend geboten, indem die Bindeanweisungen beim Kompilationsvorgang automatisch miterzeugt werden. Erleichterung bringt es, wenn das Betriebssystem die Definition von Kommandoprozeduren durch den Benutzer zuläßt. Dann kann man wenigstens Standardfälle durch geeignete Kommandoprozeduren abdecken. Es wäre aber auch hier Sache des Herstellers, die Bedienung des Kompiliersystems entsprechend zu vereinfachen.

Ist die Bedienung vielfach noch nicht befriedigend, so kommt hinzu, daß die Bearbeitung der Quellenprogramme bis zum Entstehen eines geladenen lauffähigen Programmes häufig noch so lang dauert, daß der Übungsbetrieb erheblich beeinträchtigt wird. Auch hier stört meist weniger die Übersetzungszeit, als vielmehr die Zeit für das Binden. Insgesamt entsteht der Eindruck, daß der Kompilierer noch nicht optimal mit dem Betriebssystem und den Standard-Hilfsprogrammen wie z.B. dem Binder abgestimmt ist.

Da jede Programmkorrektur ein neuerliches Übersetzen und Binden erforderlich macht, ist es gerade in Anbetracht der geschilderten Mängel erforderlich, daß möglichst viele Fehler bereits beim Übersetzen erkannt werden. Der Übersetzungsvorgang soll dabei nicht beim ersten erkannten Fehler unterbrochen werden, er ist vielmehr, wenn möglich, fortzusetzen, damit etwaige weitere Fehler nach Möglichkeit bereits beim ersten Übersetzungsversuch erkannt werden. Fehlermeldungen müssen differenziert mit Bezug auf den Quellentext ausgegeben werden. Hier ist ein befriedigender Stand der Technik erreicht. Kompilierer, die Fehlermeldungen noch in verschlüsselter Form ausgegeben haben, werden enstprechend umgerüstet. Wenn auch Verbesserungen im Einzelfall wünschenswert sind, so befriedigt doch im allgemeinen die Zahl der erkannten Fehler.

Leider werden die Möglichkeiten des SYSTEM-Teils von der Mehrzahl der Kompilierer nicht voll genutzt. Zunächst ist bei einigen Herstellern eine einzelne Klemme bzw. eine Folge von Klemmen der Prozessperipherie nicht als DATION identifizierbar. Wenn z.B. nur eine gesamte Baugruppe der Prozessperipherie als DATION eingeführt werden kann, so ist es im Programm nicht möglich, auf einzelne Klemmen unabhängig voneinander zuzugreifen. Da man im Übungsbetrieb die einzelnen Übungsprozesse getrennt und unabhängig voneinander betreiben können muß, führt dies zu einer unnötig

großen Prozessperipherie, indem jedem Übungsprozess seine eigene Digitaleingabe, Analogeingabe, Digitalausgabe, usw. zugeordnet werden muß. Belegt ist aber tatsächlich nur ein bescheidener Teil der möglichen Anschlüsse.

Sind einzelne Klemmen der Prozessperipherie als DATION identifizierbar, so kann auch bei einem Feld von DATION's geprüft werden, ob der Zugriff berechtigt ist. Diese Bedingungen erfüllen die Kompilierer im allgemeinen. Für den Übungsbetrieb ist es ferner erforderlich, daß mehrere PEARL-Programmsysteme ("PEARL-Welten") gleichzeitig im Rechner betrieben werden können. Für die Mehrzahl der Kompilierer ist angegeben, daß dies möglich ist. Ob der störungsfreie Betrieb gewährleistet ist, wenn zwar nicht gleiche Peripherie-Endstellen, aber gleiche Baugruppen in mehreren PEARL-Welten benutzt werden, ist unklar. Es ist auch unklar, inwieweit Prüfungen durch das System vorgenommen werden, um die Benutzung der gleichen Endstellen aus verschiedenen PEARL-Welten heraus zu verhindern. Eine Prüfung durch das System wäre jedoch grundsätzlich möglich.

Testhilfen spielen im Übungsbetrieb eine zweifache Rolle. Sie sollen die schnelle Lokalisierung von Fehlern ermöglichen; gleichzeitig soll der Student den Umgang mit fortschrittlichen Testhilfen kennenlernen. Zunächst bietet PEARL mit dem SIGNAL-Konzept eine Möglichkeit zur Behandlung von Unregelmäßigkeiten des Programmlaufs an. Sicherlich sollte die Programmierung von SIGNAL-Reaktionen in die Übungsaufgaben mit aufgenommen werden; als allgemeine Testhilfe eignen sich die Signale weniger. Die notwendige Programmierungsarbeit ist zu umfangreich, da das System nur den Fehler registriert, die eigentliche Testhilfe jedoch in das Programm integriert werden muß. Man überlege nur, wie schwer es ist, die Anweisung möglichst genau zu identifizieren, bei der ein arithmetischer Alarm (Overflow-SIGNAL) auftritt.

Das SIGNAL-Konzept von PEARL kann in die Testhilfen mit einbezogen werden. Für den Übungsbetrieb wird eine Testversion der erzeugten Programme benötigt, bei der die Systemreaktion auf Signale in einer Meldung auf einem geeigneten Protokollgerät besteht, aus der die Anweisung hervorgeht, die das Signal verursacht hat. Die Meldung hat im Klartext unter Bezug auf Marken und Anweisungsnummern des Quellenprogrammes zu erfolgen. In der Regel bieten die Kompiliersysteme diese Möglichkeit, wenn auch die Meldungen teilweise erst entschlüsselt werden müssen. Im übrigen werden in der Normalversion bei manchen Kompilierern überhaupt keine Signale erzeugt oder die Systemreaktion ist leer. Beides befriedigt nicht, da zum mindesten eine benutzerspezifische SIGNAL-Reaktion angebbar sein müßte und durch die Systemreaktion die Auswirkung auf andere Programme verhindert werden muß.

Notwendige Testhilfe ist eine Überwachung des Programmlaufes mit der Möglichkeit, Werte von Variablen bei gewissen Kontrollereignissen oder auch im Fehlerfall beim Abbruch des Programmes auszugeben. Die Werte sind selbstverständlich unter Bezug auf die Variablen-Bezeichner des Quellenprogrammes auszugeben. Da solche Überwachungen des Programmlaufes sehr zeit- und ausgabeintensiv sind, müssen sie gezielt einsetzbar sein, d.h. es muß verschiedene Ebenen der Programmüberwachung geben und diese muß während des Programmlaufes ein- und ausschaltbar sein. Als Ebenen kommen Meldungen nach

- jeder Anweisung
- jeder Programmverzweigung
- jedem Prozeduraufruf
- jedem Start bzw. Ende einer TASK

in Frage. Das Ein- und Ausschalten kann durch entsprechende Kommentarzeilen im Programm oder auch interaktiv durch entsprechende Eingaben über die Bedienkonsole erfolgen. Ansätze für derartige Testhilfen sind teilweise vorhanden, teilweise sind entsprechende Testhilfen angekündigt. Die derzeit angebotenen Möglichkeiten befriedigen jedoch bei weitem nicht.

Da durch die beschriebene Überwachung des Programmlaufes das Zeitverhalten der Programme erheblich gestört wird, sind sie in Realzeitprogrammen nur begrenzt einsetzbar. Wünschenswert wäre das Aufzeichnen einer Programmhistorie im Hauptspeicher, aus der ersichtlich ist, welches die letzten ausgeführten Anweisungen jeder TASK sind, ob gewisse im Programm definierte Kontrollereignisse eingetreten sind und welche Werte einzelne Variablen dabei hatten. Um den Speicherbedarf in Grenzen zu halten, kann man die Listen als Ringpuffer organisieren, aus denen dann die neuesten Eintragungen ersichtlich sind. Eine derartige Testhilfe wird jedoch von keinem Hersteller angeboten.

4. Experimentieren mit neuen Systemkonzepten auf der Basis von PEARL

Will man die Möglichkeit schaffen, daß Studenten im Rahmen von Diplomarbeiten an der

Fortentwicklung einer Programmiersprache arbeiten können - sei es, daß neue Sprachkonstrukte untersucht werden, oder daß Zielsysteme mit neuartigen Strukturmerkmalen verwendet werden sollen - so ist ein experimentelles Kompiliersystem erforderlich, das leicht modifiziert werden kann. Nun kann man ein kompiliererzeugendes System, das zudem den Bereich einer Realzeitprogrammiersprache abdeckt, nicht gerade fertig kaufen. Hat man nicht das Personal und möchte man auch aus andern Gründen kein derartiges System selbst entwickeln, muß man nach anderen praktikablen Lösungen suchen.

Stellt man einmal die unverzichtbaren Eigenschaften eines experimentellen Kompiliersystems zusammen, so ergibt sich:

- Der Kompilierer muß in einer höheren Programmiersprache geschrieben sein, weil sonst der Programmieraufwand bei Änderungen zu groß wird.

- Wiederkehrende gleichartige Aufgaben (z.B. Syntaxanalyse, Aufbau des Namensbuches) müssen ohne Programmierungsaufwand anpassbar sein.

- Es muß eine geeignete interne Schnittstelle (Zwischendarstellung) existieren, die für die Kodeerzeugung für verschiedene Zielsysteme verwendbar ist.

- Der Kompilierer muß modulartig strukturiert sein, damit erforderliche Anpassungen überschaubar bleiben.

- Es muß eine ausreichende Dokumentation und die Quellenform des Kompilierers verfügbar sein.

Freundlicherweise stellte ein Systemhaus Quellenprogramme und Dokumentation seines Kompiliersystems, das die geforderten Eigenschaften weitgehend aufweist, für experimentelle Arbeiten zur Verfügung. Einschränkungen des Sprachumfangs fallen hier nicht so stark ins Gewicht, da zudem zwei Besonderheiten weitere Experimente unterstützen: Das Kompiliersystem ist so ausgelegt, daß

- für den Zielrechner ein Betriebssystem mitgeneriert wird, das nur die für das Programm erforderlichen Funktionen enthält. Als Zielrechner kommen daher auch Mikroprozessoren ohne entsprechendes eigenes Betriebssystem in Frage.

- Programme für ein lokales Rechnernetz erzeugt werden können, ohne daß die Kommunikation zwischen den Prozessoren explizit programmiert werden muß.

Es soll hier nicht auf Einzelheiten des Systems eingegangen werden. Nach einer Anpassung der Quellenprogramme an die Rechensysteme, die dem Verfasser zur Verfügung stehen, kann zunächst im Rahmen von Diplomarbeiten der verarbeitbare Sprachumfang erweitert werden, oder andere Verbesserungen des Kompiliersystems durchgeführt werden. Ein für die Forschung interessanter Themenbereich, der in Zukunft bearbeitet werden soll, ist die Programmierung von hierarchischen lokalen Rechnernetzen in PEARL unter Berücksichtigung der Fehlerunempfindlichkeit bzw. Überlebensfähigkeit solcher Systeme.

Diplomarbeiten in diesem Bereich stellen nicht zuletzt wegen der aufwendigen Einarbeitung in das System besondere Anforderungen an den Studenten. Andererseits können in hervorragender Weise für die spätere Berufspraxis relevante Erfahrungen gesammelt werden. Studenten, die bisher an diesem System gearbeitet haben, bestätigen, daß das erworbene Wissen und die gesammelten Erfahrungen den hohen Arbeitseinsatz aufwiegen.

Aufbau und Durchführung einer Anwenderschulung Systematisches Programmieren mit PEARL Teil 1

Dipl.-Inform. Herbert Brinkkötter, Dr.-Ing. Klaus Rebensburg, Berlin (West)

Zusammenfassung:

In diesem Beitrag wird über den Aufbau einer an der TU-Berlin entwickelten PEARL - Schulung, die dazugehörigen Schulungsunterlagen, sowie über die bei der Durchführung gesammelten Erfahrungen berichtet. Konzeption dieser einwöchigen Schulung, Lernziele, praktischer Übungsteil und die Dokumentation werden vorgestellt.
Verschiedene Stufen der Entwicklung und Erprobung führten zur Heraushebung einer als unbedingt notwendig erachteten Menge von Lernzielen und Inhalten bei vorgegebener Zielgruppe und vorgegebenen zeitlichen Rahmenbedingungen.

Summary:

This paper describes the contents of a PEARL - Course, documentation and experiences. Concept of this One-Week-Course, objectives, practical part and documentation is presented.
Different phases of development and practice led to the emphasis of absolutely necessary objectives and contents in the context of the given target group and time frame.

1. Historie

Am 1. 7. 1978 begann ein bei der Prozeßrechnerverbundzentrale (PRZ) der TU - Berlin durchgeführtes und vom BMFT gefördertes Forschungsprojekt. Im Rahmen dieses Vorhabens wird die Programmiersprache PEARL(Process and Experiment Automation Realtime Language) auf verschiedenen Rechnern implementiert und bei mehreren Instituten eingeführt.
Durch die Einführung dieser höheren Programmiersprache versprechen sich die Anwender eine Effizienzsteigerung und eine Kostenreduzierung bei der Lösung von wissenschaftlichen Problemen in unterschiedlichen Fachgebieten. Für die TU - Anwender bedeutet das u.a. eine Konzentration der Kräfte auf die inhaltlichen wissenschaftlichen und technischen Problemstellungen.
Auf diesem Hintergrund entstand schon 1979 für die Schulung der Anwender im PRZ ein Skript "Systematisches Programmieren mit PEARL", das für 14-tägige Schulungen verwandt wurde. Der sehr heterogene Anwenderkreis in diesen Kursen - außer Informatik- und Elektrotechnik - Studenten waren die in Industrieanwendungen und Forschung tätigen Institute zu berücksichtigen - führte zu einer Umorientierung des Schulungskonzepts. Dieses wurde u.a. extern Anfang 1981 bei einer einwöchigen Kompaktschulung beim Institut für Rundfunktechnik in München erprobt. Der PEARL - Verein erwarb Rechte an der dort verwendeten Schulungsunterlage und beauftragte die TU - Berlin, daraus eine für die Schulungen des PEARL - Vereins verbindliche Lehr- und Lernkonzeption zu erstellen. Diese wurde, inclusive der dazugehörigen Unterlagen termingerecht fertiggestellt und ist seit Juli 1981 Grundlage der vom PEARL - Verein in Zusammenarbeit mit dem VDI - Bildungswerk veranstalteten PEARL-Kurse.

Konzeption und Inhalte wurden von den Herren H. Brinkkötter, K. Nagel, H. Nebel, K. Rebensburg von der **TU** - Berlin unter Mitwirkung der Herren K. Pelz, Uni Erlangen, D. Pflügel, **FHS** Furtwangen und T. Röhrich, Uni Stuttgart, erarbeitet.
In den folgenden Ausführungen berichten wir über die mit externen (nichtuniversitären) Anwendern gemachten Erfahrungen.

2. Zielgruppen

Die Konzeption zielte ursprünglich auf die Umschulung von **FORTRAN** - Kennern. Zu den Kursen meldeten sich jedoch gleichermassen:

- Anfänger der Programmierung und Fortgeschrittene
- Assembler und **FORTRAN** - Experten
- Programmierer, Ingenieure und Abteilungsleiter

Aufgrund dieser Erfahrungen berücksichtigt die Schulung des **PEARL** - Vereins nun diese Gruppen.

3. Aufbau der Schulung "Systematisches Programmieren mit PEARL" Teil 1:

Dauer: 5 Tage Kompaktkurs (Teil 1)
40 Unterrichtseinheiten (U-Einheiten) (30 Zeitstunden)

Form: Definierte Mischung aus:

Vortrag: 20 U-Einheiten (15.5 Zeitstunden)

Tutorium und praktische Übungen am Prozeßrechner:

15 U-Einheiten (11 Zeitstunden)

Diskussion: 5 U-Einheiten (3.5 Zeitstunden)

Durchführung: 4(mind.3) Dozenten, die, ständig anwesend, Vorträge, Tutorium und praktische Übungen am Rechner durchführen.

Teilnehmer: gesamt max. 18, in Gruppen je 2 bzw 3 am Rechner

Ausstattung: 7 Terminals an einer Rechenanlage mittlerer Größe zur **PEARL** - Programmentwicklung.

Ort: Auf der Anlage eines Anwenders, hier am **IRT** München, an der **TU** - Berlin, am **IVD** Stuttgart.

Kursplan :

1. Tag 10:00 Begrüßung: Vorstellung der Teilnehmer und Dozenten. Klärung der Frage: Welche Erwartungen haben Teilnehmer und Dozenten? Welche Probleme hinsichtlich Automatisierung und Softwareentwicklung (Spezifikation, Entwurf, Implementierung) haben die Teilnehmer?

10:30 **Vortrag:** 'Vom Problem zum Algorithmus' 1.

13:00 Mittagspause

14:30 **Vortrag:** 'Vom Problem zum Algorithmus' 2.

16:00 **Praktikum/Tutorium:** Beispiel 1 der Schulungsunterlage wird von den Teilnehmern am Terminal eingegeben, getestet und modifiziert. Teilnehmer lernen den Dialog mit dem Rechner und machen erste praktische **PEARL** - Erfahrungen.

18:00 -

2. Tag 9:30 Vortrag: 'Problemgerechte Konstruktion von Algorithmen und Daten' 1.

11:00 Pause

11:30 **Praktikum/Tutorium:** Fortsetzung der Aufgabe vom 1. Tag, Diskussion mit den einzelnen Gruppen am Terminal.
Neue Aufgabe: Multiplikation zweier natürlicher Zahlen; als Operationen stehen nur == und + zur Verfügung

13:00 Mittagspause

14:30 **Vortrag:** 'Problemgerechte Konstruktion von Algorithmen und Daten' 2.
17:30 -

3. Tag 9:30 **Diskussion:** Vertiefung **PEARL**, Zusammenfassung der bisher erreichten Lernziele. "Automatisierungsprobleme der Zielgruppe".
Ist **PEARL** eine Sprache der Planer? Einbettung von Assembler, warum, wie?
PEARL,FORTRAN,ASSEMBLER: Vorzüge, Nachteile,
PEARL -) **ADA**, Verbreitung von **PEARL**.
11:30 Pause
12:00 **Tutorium:** Wie liest man die **PEARL** - Grammatik?
13:00 Mittagspause
14:30 **Tutorium/Praktikum:** Beispiel 2 der Schulungsunterlage, Teilnehmer bekommen die Programme als Datei zur Verfügung und programmieren das Layout der Ausgabe selbst.
17:30 -

4. Tag 9:30 **Vortrag:** 'Nebenläufige Algorithmen' 1.
11:00 Pause
11:30 **Vortrag:** 'Nebenläufige Algorithmen' 2.
13:00 Mittagspause
14:30 **Praktikum/Tutorium:** Aufgabe 2 (Nachzügler), Programmierung eines kleinen Beispiels (Autorennen) mit nebenläufigen Tasks, ohne Engpässe (Evtl Prozeß-I/O).
Diskussion der Problemstellung mit allen Gruppen, Einzeldiskussion in kleinen Gruppen während der Arbeit am Terminal.
18:00 -

5. Tag 9:30 **Vortrag:** 'Koordinierung nebenläufiger Algorithmen' 1.
11:00 Pause
11:30 **Vortrag:** 'Koordinierung nebenläufiger Algorithmen' 2.
Lösung eines Problems mit Nebenläufigkeit und Synchronisation (Semaphoren). Hinweise, wie man die Aufgabe "Autorennen" mit Engpaß (Semaphoren) lösen kann.
13:00 Mittagspause
14:00 **Vortrag, Diskussion:** Vertiefung und Abschlußdiskussion. Wurden die Erwartungen der Teilnehmer erfüllt? Perspektiven der **PEARL** - Anwendung, Tagungen, Literatur.
15:00 Ende des Kurses

4. Lernziele

Lernziele bzgl. Programmiermethoden und -techniken und Lernziele der Sprache PEARL:

Kapitel 1: Programmiermethoden und -techniken:

- Entwurf von Algorithmen
- Schrittweise Verfeinerung mit unterschiedlichen Formulierungsebenen
- Unterstützung der Programmdokumentation

Kapitel 1: Lernziele **PEARL**

. Variablen, Konstanten	INV, INITIAL
. Vereinbarungen	DECLARE, SPECIFY
. Einfache Datentypen	FIXED, CHARACTER, BIT, FLOAT
. Wertzuweisung	:=
. Einfache Operatoren	+ - * / **
. Verzweigungen	IF .. THEN .. ELSE .. FIN; CASE .. ALT .. OUT .. END;
. Bedingte Wiederholung von Teilalgorithmen	WHILE .. REPEAT .. END;
. Ein/Ausgabebefehle	GET .. FROM .. , PUT .. TO ..
. Formatangaben	A, F, SKIP
. Vereinbarungen von Datenstationen	SPECIFY .. DATION .. ALPHIC;
. Genereller Aufbau eines PEARL-Programms Modulkonzept	MODULE; SYSTEM; .. ; PROBLEM; ... TASK; .. END; ...

Kapitel 2: Programmiermethoden und -techniken

- Verwaltung komplexer Daten und großer Datenmengen
- Strukturierung durch Einführung benannter Teilalgorithmen

Kapitel 2: Lernziele PEARL

. Verwendung von Dateien	OPEN...BY IDF...OLD,CLOSE, CREATED, DIM, DATION
. Typ-Vereinbarung	TYPE, STRUCT
. Vereinbarung von Operatoren	OPERATOR
. Kombination von Bedingungen für Wiederholungen von Teilalgorithmen	FOR...WHILE...REPEAT...END
. Benannte Teilalgorithmen	PROCEDURE, RETURN, RETURNS
. Aufruf von Prozeduren	CALL
. Parameterübergabe bei Funktionen und Prozeduren	IDENTICAL
. Umwandlung von Buchstaben zu Zahlenwerten	TOFIXED, CHARACTER

Kapitel 3: Programmiermethoden und -techniken

- Nebenläufige Algorithmen
- Zeitliche Steuerung von Teilvorgängen
- Schrittweise Verfeinerung (Vertiefung)
- Konstruktion von Daten und ihren Zugriffsalgorithmen
- Prozeß - Schnittstellen
- Buchstabengrafik

Kapitel 3: Lernziele PEARL

. Nebenläufige Prozesse	TASK
. Task-Steueranweisungen	ACTIVATE, TERMINATE, RESUME, CONTINUE, PREVENT, SUSPEND
. Bedingungen bei Task-Einplanungsanweisungen	WHEN, AT, AFTER, EVERY, ALL, DURING, UNTIL
. Datentypen	CLOCK, DURATION
. Zeitkonstanten	HRS, MIN, SEC
. Prozeß - Ein-/Ausgabe	TAKE .. FROM..; SEND .. TO ..;

Kapitel 4: Programmiermethoden und -techniken

- Zerlegung in nebenläufige Teilalgorithmen
- Ermittlung der Schnittstellen und geeignete Koordinierung von Teilalgorithmen
- Konstruktion von gemeinsam benutzten Daten
- Konstruktion komplexer Systemteile
- Summierung durch endliche Iteration
- Bestimmung von Minimum/Maximum einer Reihe

Kapitel 4: Lernziele PEARL

. Vertiefung der Prozeß - Ein-/Ausgabe	BASIC; TAKE .. FROM ..; SEND .. TO ..
. Binäre Ein-/Ausgabe	ALL; READ .. FROM ..; WRITE .. TO ..
. Vereinbarung von SEMA - Variablen	SEMA .. PRESET
. Synchronisation nebenläufiger Algorithmen	REQUEST .. ; RELEASE ..;
Operatoren	SQRT, **, CAT, TOCHAR

5. Dokumentation zur Schulung

Für die einwöchige Kompaktschulung erhält der Teilnehmer folgende Materialien:

1 Stundenplan,

1 Handbuch "Systematisches Programmieren mit **PEARL**" ca. 250 Seiten, das die vollständige Ausarbeitung des Kurses sowie u.a. im Anhang eine für diesen Kurs erarbeitete syntaktische Beschreibung der Sprache enthält,

1 **PEARL** - Kurzbeschreibung, ca 5 Seiten, bestehend aus einer Kurzversion der **PEARL**-Syntax, **PEARL**-Standardoperatoren, und **PEARL**-Schlüsselwörtern (**PEARL**-Reference-Card),

1 Kurzbeschreibung der Rechnerbedienung für den Kurs, ca 4 Seiten, enthält die Beschreibung der notwendigsten Kommandos zum Editieren und zur Bedienung des **PEARL**- Compiliersystems. Diese Kurzbeschreibung liegt bisher für die Rechner **HP1000, HP3000, PDP 11** vor.

Die einschlägige **PEARL** - Literatur zur Ansicht während des Kurses.

Zusätzliche Materialien des Dozenten:

1 Grundlagenpapier zum Kurs (ca 70 Seiten) [1]
1 kommentierter Stundenplan
1 Satz Overhead - Folien (ca 200 Stück),

6. Gedanken zur Grundkonzeption und praktischen Durchführung

Die grundlegenden Prinzipien der Didaktik, Methodik und Leitlinien des Kurses sind in [1] ausführlich dargelegt. Der Anhang von [1] enthält eine exemplarische Anwendung der Prinzipien und Leitlinien.
Methodik und Leitlinien sind anwendbar auf beliebige Zielgruppen und sind zunächst nicht an einen bestimmten zeitlichen Rahmen fixiert.
Für die konkrete Gestaltung dieser PEARL - Kurse wurden daher zusätzlich folgende Gesichtspunkte berücksichtigt:

. Die Forderungen der Teilnehmer, in kurzer Zeit möglichst viel zu lernen (Anreise, Nebenkosten), führen zur Entscheidung der Durchführung als Kompaktkurs.

. Die Vorkenntnisse und Voraussetzungen der Teilnehmer erfordern eine sorgfältige Dosierung von Informationsmenge und Schwierigkeitsgrad.

. Die Trockenheit des Stoffes erfordert zusätzliche Motivationshilfen, die in Form von praktischen Erfolgserlebnissen am Rechner, Diskussion und individueller Betreuung in kleinen Gruppen den Kurs begleiten.

. Die Heraushebung der algorithmischen Schreibweise bei der Konstruktion erlaubt die relativ leichte Implementierung in anderen Programmiersprachen (PASCAL, ADA,
. FORTRAN). Es wird, insbesondere am Anfang des Kurses, auf leichte Überschaubarkeit des gesamten Lösungswegs geachtet.
. Nach der konkreten Programmierung in PEARL folgt stets die Verallgemeinerung, bezogen auf die Sprache.

. **Der vorgegebene zeitliche Rahmen führt zur Konzentration auf die "wichtigsten" Lernziele.** Es müssen Schwerpunkte gesetzt werden, es wird bewußt auf Vollständigkeit verzichtet. Wichtige, in [1] genannte Bausteine des Konzepts werden herausgehoben, einige können nicht vertieft werden, einige werden reduziert - sie bleiben einem PEARL-Kurs Teil 2 vorbehalten:

Beispiele für nicht vertiefte Lernziele (Grund s.o.):

. Formale Spezifikation, es wurde eine einfachere verbale Zwischenlösung gewählt,
. Korrektheitsüberprüfungen (Plausibilitätskontrolle, Beweise),
. Sicherheit der Algorithmen,
. Große Programme, Programmsysteme.
. Optimierung,
. Erstellen einer Bedienungsanleitung für Programme

Beispiele für Einschränkungen:

. Vielfalt der algorithmischen Problemstellungen,
. dargebotener Sprachumfang, z.B. Vielfalt des DATION-Konzepts

Beispiele für Heraushebungen, Vertiefungen:

. Der Prozeß der Programmentwicklung,
. die genaue Beschreibung des Problems,
. Methode der schrittweisen Verfeinerung,
. Dokumentierung des Lösungswegs auf den verschiedenen Formulierungsebenen,
. Verklemmungsfreiheit
. Strikte Trennung von Konstruktion und Implementierung
. Grundlegende Bausteine (Sprachelemente) zum Problemlösen.

Strikte Vermeidung des folgenden Themas:

. "exotische" Feinheiten der Sprache PEARL.

7. Erfahrungen

Lob:

. Unsere Erfahrungen zeigen, daß mit einer Einheit aus Kurs, Tutorium und integrierten praktischen Übungen am Prozeßrechner der Lehrstoff am besten vermittelt wird. Eigene Artikulierung und Versuche motivieren die Teilnehmer zusätzlich, bauen die Schwellenängste zum eigenen Tun ab. Für eine praxisorientierte Ausbildung sind praktische Übungen am Prozeßrechner we-

sentlicher Bestandteil. Dieses bestätigten sämtliche Teilnehmer.

. Aufbau, zeitliche Einteilung und Schwierigkeitsgrad wurden allgemein positiv bewertet.

. Die Schulungsunterlage wurde sehr positiv bewertet.

. Die intensive Betreuung durch 4 Dozenten fand breite Zustimmung

. Die Diskussion am 3. Tag hat die Teilnehmer stark motiviert.

. Die induktive Vorgehensweise vom Beispiel zur Verallgemeinerung wurde gut akzeptiert.

. Die Teilnehmer hatten keine nennenswerten Schwierigkeiten mit der Rechnerbedienung.

. Jedes behandelte PEARL-Programm ist so abgeschlossen, daß es auf der Rechenanlage lauffähig ist.

Kritik:

. Das Lesen einer PEARL - Grammatik bleibt trotz großer Bemühungen der Kurs-Entwickler ein Problem. Die im Kurs verwendete Darstellung ist zwar relativ leicht auch von Anfängern lesbar, aber dennoch ein umfangreiches Werk.

. Die PEARL - Kurzdarstellung ist verbesserungswürdig (PEARL- Reference Card). Die Verwendung von Syntaxdiagrammen wird geprüft.

. Modellprozesse wären eine sinnvolle Ergänzung der Übungen.

. Der PEARL - Übungsbetrieb kann durch schnellere PEARL Compilier - Systeme effektiver werden. Die HP3000 - Anlage war mit 7 PEARL - Teilnehmern (zu) gut ausgelastet.

. Kursunterlagen sollten vor dem Kurs verschickt werden.

. Die Begeisterung der Teilnehmer schränkte die Freizeit der Dozenten etwas ein; nur mit sanfter Gewalt ließen sich die Herren abends und in den Mittagspausen von den Terminals lösen.

Perspektiven:

Kurse der oben beschriebenen Art werden 1982 zu folgenden Terminen vom PEARL-Verein in Zusammenarbeit mit dem VDI - Bidungswerk angeboten werden:

15. - 19. 3. 1982 (7. Woche) Stuttgart (IVD)
21. - 25. 6. 1982 (25. Woche) Berlin (TU)
6. - 10. 9. 1982 (36. Woche) Stuttgart (IVD)
8. - 12.11. 1982 (45. Woche) München (IRT)

8. Literaturhinweise

[1] N a g e l, Klaus, N e b e l, Herbert: Didaktische und Methodische Prinzipien und Leitlinien für die Seminarreihe: Systematisches Programmieren mit PEARL. Beitrag zur Tagung "PEARL in Lehre, Ausbildung und Schulung. Erlangen Okt 1981

PEARL als erste Programmiersprache im Grundstudium der Studienrichtung Elektrotechnik (Informationsverarbeitung)

von Prof. K. Fasold, Bielefeld

Zusammenfassung

Im Studiengang Elektrotechnik, Studienrichtung Informationsverarbeitung, der Fachhochschule Bielefeld wurden im Grundstudium die Programmiersprachen FORTAN und PEARL eingesetzt. In einem analysierenden Bericht und anhand einiger der vorliegenden Erfahrungen wird gezeigt, daß für die genannte Studienrichtung die Programmiersprache einen leistungsfähigen Einstieg in die Programmierung technischer Prozesse bietet. Die Unterschiede in der Fähigkeit, komplexe Programme mittleren Umfangs auch nach kurzer Lernzeit konstruktiv zu beherrschen, sind beachtlich.

Schlüsselwörter: PEARL, Grundausbildung

1. Studienordnung Informationsverarbeitung

1.1. Fächer des Grund- und Hauptstudiums

Das Studium in der Studienrichtung Informationsverarbeitung gliedert sich gemäß der (auf der Grundlage der Prüfungsordnung vom 21.10.76 erlassenen) Studienordnung an der Fachhochschule Bielefeld in ein Grundstudium und ein Hauptstudium. Die Fächer des Grundstudiums sind klassische Fächer der Fachrichtung Elektrotechnik: Mathematik, Physik, Grundgebiete der Elektrotechnik, Meßtechnik, Grundlagen der ADV / Programmiersprachen, Bauelemente. Das Fachstudium ist dagegen sehr stark auf die Anwendung automatischer Datenverarbeitung ausgerichtet. Fächer des Hauptstudiums sind: technischer Aufbau von Datenverarbeitungsgeräten, Betriebssoftware von Datenverarbeitungsgeräten, Datennetze und Datenfernübertragung, Steuerungs- und Regelungstechnik, Betriebswirtschaft und Operations Research u.a. Dazu kommen Wahlpflichtfächer, die aus einem Katalog von 20 Datenverarbeitungsanwendungsfächern wie Geometrische Datenverarbeitung, Dokumentation und Datenbanken u.a. auszuwählen sind.

1.2. Formen der Lehrveranstaltungen

Die Fachhochschule kennt im Lande Nordrhein-Westfalen drei Grundformen der Lehrveranstaltungen in den technischen Fachbereichen, nämlich Vorlesung, Seminar und Praktikum, wozu in unserer Studienordnung etwa folgendes ausgeführt ist:

Die V o r l e s u n g dient der zusammenhängenden Darstellung eines Lehrstoffes sowie der Vermittlung von Fakten und Methoden durch den Lehrenden. Der Lehrende trägt vor und beantwortet Fragen. Die Studenten verhalten sich überwiegend rezeptiv und stellen Informationsfragen.

Das S e m i n a r dient der Erarbeitung von Fakten, Erkenntnissen, komplexen Problemstellungen im Wechsel von Vortrag und Diskussion. Der Lehrende leitet die Veranstaltung und führt die Diskussion. Die Studenten erarbeiten Beiträge und diskutieren die Beiträge*.

* Die Wirklichkeit sieht allerdings häufig so aus, daß die Professoren Übungen veranstalten, in denen sie Aufgabenlösungen vortragen, die eine Anwendung des Vorlesungsstoffes auf die Praxis darstellen.

Tabelle 1. Lehrveranstaltungsform und -umfang einiger Fächer der Studienrichtung Informationsverarbeitung an der Fachhochschule Bielefeld

Bezeichnung des Faches	1. Sem. V S P	2. Sem. V S P	3. Sem. V S P	4. Sem. V S P	5. Sem. V S P	6. Sem. V S P
Grundlagen der ADV / Programmiersprachen		2 - 2	2 2 2			
Prozeßdatenverarbeitung				2 1 1	1 1 2	
Prozeßmeßtechnik				1 1 -	1 - 2	
Betriebswirtschaft und Operations Research						2 2 -

Das P r a k t i k u m dient dem Erwerb und der Vertiefung von Kenntnissen durch Bearbeitung praktischer, experimenteller Aufgaben. Der Lehrende leitet die Studenten an und überwacht die Veranstaltung. Die Studenten führen praktische Arbeiten und Versuche durch.

Durch einen Studienverlaufsplan werden den einzelnen Fächern und Studienhalbjahren Lehrveranstaltungen der Form Vorlesung, Seminar und/oder Praktikum zugeschrieben. Eine inhaltliche Festlegung der Lehrveranstaltungen ist durch "Stoffpläne" erfolgt, die jedoch glücklicherweise unverbindlich sind. Einige der "Stundentafeln" sind in Tabelle 1 gegeben.

2. Didaktik des Faches Grundlagen der ADV / Programmiersprachen

2.1. Eingangsniveau

Die Studenten des Fachhochschulstudienganges Informationsverarbeitung an der Fachhochschule Bielefeld sind nahezu ausschließlich Absolventen der Fachoberschule Technik - Fachrichtung Elektrotechnik - und nahezu ausschließlich männlich. Sie haben beim Eintritt in die Fachhochschule in der Mehrzahl noch keine Datenverarbeitungsausbildung gehabt. Allerdings hat ein merklicher Teil der Studenten an einem Hobbycomputer Programmiererfahrung, häufig in BASIC, erworben. Dieser Anteil liegt im laufenden Studienjahr bei etwa 12,5 % und scheint gegenüber den vergangenen Jahren zugenommen zu haben.

Insgesamt scheinen bei Beginn der Lehrveranstaltung das Interesse für praktische Datenverarbeitung und die Fähigkeit zur konkreten technischen Problemlösung sehr ausgeprägt zu sein, während die Fähigkeit zu Abstraktion und Systematisierung und Lösen komplexer Probleme sich noch entwickeln muß.

Bezüglich ihrer speziellen Vorkenntnisse, insbesondere ihrer Programmiererfahrung, sind die Studenten außerordentlich heterogen.

2.2. Lernsituation

Das propädeutische Fach Grundlagen der ADV / Programmiersprachen ist das einzige Datenverarbeitungsfach des Grundstudiums und zeitlich mit 11,5 % an dessen Lehrangebot beteiligt. Es wird regelrecht erdrückt von den "großen" Fächern Mathematik, Physik und Elektrotechnik. Der Lehrende hat daher in seiner Veranstaltung zu berücksichtigen, daß die Teilnehmer in ihrer Mehrheit überhaupt noch nicht in der Lage sind, sich diesem Fach, das sie als Nebenfach empfinden, zuzuwenden.

Das Fach Grundlagen der ADV / Programmiersprachen wird an der Fachhochschule Bielefeld zweisemestrig dem 2. und 3. Fachsemester angeboten. Eine Besonderheit des Bielefelder Studienverlaufsplanes besteht darin, daß das erste Halbjahr ausschließlich mit Vorlesung und Praktikum bestritten wird. Es ist in diesem Kreis wohl nicht erforderlich zu begründen, daß die Veranstaltungsform Vorlesung ebensowenig geeignet ist, Grundfertigkeiten im Programmie-

ren zu vermitteln, wie diese Veranstaltungsform die Beherrschung des kleinen 1 x 1 vermitteln könnte. In Bielefeld ist es also der Veranstaltungsform Praktikum überlassen, die Grundfertigkeiten in der Konstruktion von Programmen zu entwickeln.

Die hier geschilderte Lernsituation an der Fachhochschule Bielefeld verlangt in ganz besonderem Maß eine Anfangsprogrammiersprache und ein Programmiersystem, die den Lernenden mit geringem Zeitaufwand und mit wenig Anleitung zur sicheren Beherrschung einfacher Programmiertechniken verhelfen. Damit sind auch für das Anfängerselbststudium geeignete Unterlagen gemeint - wo zweifellos noch immer die größte Schwäche von PEARL liegt.

2.3. Lernziele

Das Fach Grundlagen der ADV / Programmiersprachen ist Grundlage nahezu aller Fächer des Hauptstudiums. Daher müßte es Hardware und Software behandeln. Es hat sich jedoch bewährt, den Inhalt des Faches auf die Softwaregrundlagen zu beschränken. Die Lernziele liegen dann allgemein gesprochen in Dingen wie der Entwicklung eines guten Programmierstils, im Erlernen einiger grundlegender Algorithmen, in Fertigkeiten in der Handhabung des Programmiersystems, in der Fähigkeit zu ingenieurmäßiger Konstruktion von Programmen, und vor allem in der Entwicklung der Fähigkeit, "datenverarbeitungsmäßig zu denken".

Aber auch dies zum Stichwort Lernziele: Ein außerordentlich wichtiges Lehrziel besteht darin, von Anfang an Augenmaß für die Komplexität einer Automatisierungsaufgabe zu wecken und Software als technischen Gegenstand zu begreifen, der nach professionellen Gesichtspunkten zu konstruieren ist.

3. Instrumentelle Möglichkeiten

3.1. Datenverarbeitung im Fachbereich Elektrotechnik der Fachhochschule Bielefeld

Der Fachbereich Elektrotechnik der Fachhochschule Bielefeld ist aus der Staatlichen Ingenieurschule für Maschinenwesen Bielefeld hervorgegangen. Bereits im Jahre 1965 wurde dort - damals in einer Ingenieurschule ganz ungewöhnlich - eine Datenverarbeitungsanlage Zuse Z 25 mit 6 Programmierplätzen und mit on-line betriebenem Zeichentisch Zuse Z 64 in Betrieb genommen. Es folgte dann nach jeweils vier Jahren etwa ein neues System: Siemens 302 mit Lochstreifenumgebung, Siemens 305 mit Platte und Prozeßelement, KAE Doppelrechnersystem. Parallel dazu erschlossen wir dem Fachbereich die Rechenkapazität des Großrechners in der Universität, wobei es mir allerdings noch immer nicht gelungen ist, eine RJE-Station ins Haus zu bekommen. Die Zahl der Datenverarbeitungskollegen wuchs seit 1972 um fünf Professoren und drei Ingenieure. Daneben erhielten einige Laboratorien Kleinrechner für begrenzte Automatisierungsaufgaben und Mikrorechnerprogrammierung. Insgesamt steht heute dem Fachbereich Elektrotechnik eine ganz beachtliche Ausbildungs- und Forschungskapazität auf dem Gebiet Datenverarbeitung zur Verfügung.

3.2. Erfahrungen mit FORTRAN IV / Siemens 305

Es war von Beginn an unser Ziel, im Fachbereich selbst nur eine kleinere Datenverarbeitungsanlage für die Laborautomatisierung zu installieren und daneben für die "große" Datenverarbeitung der übrigen Anwender das Hochschulrechenzentrum der Universität verfügbar zu machen. Dies war keineswegs eine vorgeschobene Absicht, um einen unabhängigen Universalrechner örtlich verfügbar zu machen, sondern entsprach unseren Vorstellungen von den Schwerpunkten der Lehre in einer Studienrichtung Informationsverarbeitung. Im Ergebnis des demokratischen Beschaffungsprozesses für

unseren ersten echten Prozeßrechner war es dann allerdings so, daß wir einen relativ leistungsfähigen Universalrechner mit sehr magerer Prozeßperipherie erwarben.

Dem Stand der damaligen Softwaretechnologie entsprechend habe ich vorzugsweise in der Assemblersprache PROSA 300 programmieren lassen. Das Betriebssystem war für unsere einfachen, im Grundstudium eingesetzten Prozeßexperimente (Ampelsteuerung, Heizungsmischregelung, Impulszähler, einfache Kennlinien u.s.w.) erfreulich schlicht, und in dieser Periode entstanden einige sehr gute Arbeiten, zumal der Programmieraufwand in PROSA durch die Vereinbarung von Makrobefehlen recht gering gehalten werden konnte.

Als unsere Prozeßrechenstelle die Bausteine der Prozeßprogrammierung auf FORTRAN-Bibliotheksunterprogramme übernahm, ließ ich von da an vorzugsweise in FORTRAN programmieren, konnte mich aber mit den Ergebnissen nicht in gleicher Weise wie mit den PROSA-Programmen zufrieden geben. Die den Studenten erteilte Auflage, FORTRAN-Programm in der üblichen Weise "wohlstrukturiert" zu gestalten, brachte nur mäßigen Erfolg. Regelmäßig sind Lösungen vorgelegt worden, die erst in FORTRAN aufgeschrieben und dann mit einem Struktogramm beschrieben wurden, das dann um eine geeignete Symbolik zur Darstellung von Sprüngen erweitert worden war.

Zu unserem weiteren Unglück installierte uns Siemens einen sehr komfortablen FORTRAN-IV-Compiler, und einige der Benutzer aus unserem Fachbereich beglückten uns damit, daß sie Programmpakete wie CSMP und ECAP auf unsere Anlage brachten, statt mit ihren Projekten den Unirechner zu benutzen. Das brachte uns eine gewaltige Überlastung der Anlage, und ich kann mich erinnern, daß ich einige Male den Rechnerraum mit körperlicher Gewalt geräumt habe, um meinen Studenten zu ermöglichen, ihre Laboraufgaben mit den einfachen Prozeßexperimenten zu fahren.

Dies veranlaßte offensichtlich den Fachbereich, einen größeren Prozeßrechner zu beantragen, der in Parallelarbeit als Vielzweckgerät genutzt werden sollte: Laborautomatisierung, Programmentwicklung mit time sharing und Stapelverarbeitung. Eigentlich sollte das Gerät auch noch RJE-Station zum Unirechner werden. Jedoch ist dieser Plan auf unbestimmt Zeit zurückgestellt. Im übrigen aber haben wir das System nach einem Beschaffungsprozeß voller Dramatik schließlich zum größten Teil bekommen.

3.3. Möglichkeiten mit PEARL

Mit dem neuen System, das zunächst neben der Siemens 305 lief und nach und nach eine gewisse Zuverlässigkeit erreicht hat, stand uns nun mit PEARL zum ersten Mal eine blockorientierte Sprache im Hause zur Verfügung. Da die Fächer Prozeßdatenverarbeitung und Betriebssysteme und weitere Fächer des Hauptstudiums Echtzeitverarbeitung beinhalten, liegt es nahe, bereits im Grundstudium die Elemente der Programmierung nebenläufiger Prozesse zu lehren. Damit ergibt sich eigentlich zwangsläufig die Entscheidung, allen Bedenken zum Trotz PEARL als erste Programmiersprache im Grundstudium der Informationsverarbeitung einzuführen. Der Kurs Sommersemester 80 - Wintersemester 80/81 lief erstmals in dieser Form.

4. Erreichbarkeit der Lernziele im Vergleich FORTRAN / PEARL

4.1. Ausgangslage

Der im Sommersemester 1979 begonnene Jahreskurs im Fach Grundlagen der ADV / Programmiersprachen baute noch voll auf der Siemens 305 und auf FORTRAN auf. Lediglich ein Praktikumstermin wurde am KAE-System, das zu diesem Zeitpunkt noch sehr unstabil und auch unzulänglich dokumentiert war, abgehalten. Zum Prüfungsumfang gehörten die

Programmiersprachen FORTRAN und PROSA. Bei der Prüfung im Frühjahr 1980 lag die Bestandenquote bei 50 %.

Der im Sommersemester 1980 begonnene Jahreskurs wurde voll auf die KAE-Rechner abgestellt. Als Programmiersprachen standen FORTRAN 77 und PEARL zur Verfügung. ich entschied mich für den Einsatz von PEARL. Eine weitere Programmiersprache wurde mit Rücksicht auf die außerordentlich schwierige Praktikumssituation nicht hinzugenommen: Im Sommer 1980 konnte keine Praktikumssitzung ordnungsgemäß beendet werden, da das System noch zu häufigen Ausfällen neigte. Ein besonders gravierender Zwischenfall muß hier noch berichtet werden: In der Sommerpause 1980 hatte ich auf der Grundlage der mit von der Prozeßrechenstelle der Fachhochschule Bielefeld dafür zur Verfügung gestellten KAE-PEARL-Dokumentation und unter Verwendung der deutschen Begriffe nach Werum und Windauer [1]eine Syntaxdokumentation in Form von Syntaxdiagrammen erarbeitet und war dabei, ein kleines Lehrbuch unter Verwendung der Begriffe und der frisch gezeichneten Diagramme zusammenzustellen, als ich erfahren mußte, daß das zugrunde gelegte KAE-Dokument völlig veraltet war, so daß meine Arbeit in den Papierkorb wandern mußte. Leider hat sich die Situation bis heute nicht gebessert. KAE hat eine Anfrage wegen Informationen bis heute nicht beantwortet. Der PEARL-Verein hat noch keine Liste deutscher Begriffe herausgegeben. Es bleibt festzuhalten, daß ich seitdem die PEARL-Kurse mit äußerst dürftigen Arbeitsunterlagen durchführe. Es spricht für die Eignung von PEARL als Anfangsprogrammiersprache, daß bei der Prüfung im Frühjahr dieses Jahres, die auch die Echtzeitelemente der Sprache umfaßte, die Bestandenquote 98 % betrug, obwohl ich die Leistungsanforderungen keineswegs herabgesetzt hatte.

4.2. Strukturierter Entwurf

Ebenso wie in den Kursen unter Verwendung von FORTRAN verlange ich in den PEARL-Kursen von den Teilnehmern wohlstrukturierte Entwürfe. Allerdings verzichte ich bei PEARL auf die Vorlage von Struktogrammen und verlange nur eine durch Einrücken sorgfältig hierarchisch gegliederte Niederschrift des Programms. Anders als bei FORTRAN haben wir bei PEARL vollen Erfolg. Bild 1 zeigt dagegen ein FORTRAN-Beispiel, wie es heute auf meinen Tisch kam.

4.3. Beschreiben von Algorithmen

In den FORTRAN-orientierten Kursen habe ich zur Darstellung von Algorithmen gewöhnlich eine Pascal-Fassung entwickelt, die dann nach gewissen formalen Regeln auf FORTRAN umgeschrieben wurde. Selbstverständlich können wir in einer PEARL-Umgebung den Umweg über Pascal entbehren.

Es braucht in diesem Kreise wohl nicht unterstrichen zu werden, daß PEARL ganz vorzüglich zur Beschreibung von Algorithmen geeignet ist. Fraglich ist eigentlich nur, ob wir durch Einsatz von PEARL eine Hemmschwelle aufbauen, die unsere Studenten hindert, mit Pascal oder Algol formulierte Algorithmen zu übernehmen.

Diese Frage dürfte sich bereits in den nächsten Monaten beantworten, wenn wir als weitere Programmiersprachen FORTRAN, Algol und Pascal in den Kurs Grundlagen der ADV / Programmiersprachen aufnehmen werden.

4.4. Lesbarkeit der Programme

Das schon vorgestellte Bild 1 stellt einen Ausschnitt aus einer Arbeit eines Studenten im 6. Fachsemester dar, der eine ganz beachtliche Leistung vollbracht und sich selbständig die Benutzung einiger Programmpakete des Unirechners erarbeitet hat. Allerdings hatte er bereits BASIC-Vorkenntnisse, als er zu uns kam.

```
      INTEGER H,SB,DB,HZ(60),SBD(60),DBD(60),HU,HO,SBO,SBU,DBO,DBU
      INTEGER DTH,DTB
      COMMON NH,HU,HO,DTH,HZ(60),NB,DBU,DBO,DTB,DBD(60),SBU,SBO,SBD(60)
      READ(10,20)NH,HU,HO,DTH,NB,DBU,DBO,SBU,SBO,DTB
20    FORMAT(1X,10I3)
      K=0
11    K=K+1
      READ(10,21,END=30,ERR=39)H,DB,SB
21    FORMAT(1X,3I3)
      HZ(K)=H
      SBD(K)=SB
      DBD(K)=DB
      GOTO11
39    K=K-1
      BACKSPACE 10
40    K=K+1
41    READ(10,50,END=30,ERR=60)H
50    FORMAT(1X,I3)
      HZ(K)=H
      GOTO40
60    READ(10,70,END=30)DB,SB
70    FORMAT(4X,2I3)
      SBD(K)=SB
      DBD(K)=DB
      K=K+1
      GOTO60
30    M=K-1
      CALL ZEITK(TH,TD,TS)
      STOP
      END
```

Bild 1. Teil eines studentischen FORTRAN-Programms

Es handelt sich eigentlich um ein ganz schlichtes Eingabeprogramm. Sie werden kaum länger als eine Minute brauchen, um es in seinem prinzipiellen Ablauf zu enträtseln. Dennoch - ich bin froh, daß mir solche trickreichen Programme in Zukunft seltener vorgelegt werden dürften.

4.5. Fehlerhäufigkeit

Zur Frage der Fehlerhäufigkeit können keine qualifizierten Angaben gemacht werden. Im allgemeinen dürfte aber die Lösung von Praktikumsaufgaben mit PEARL besser gelingen. Jedenfalls scheint es bei PEARL sehr viel seltener vorzukommen, daß ein Student wegen eines von ihm eingebauten Programmfehlers einen Assistenten zu Hilfe ruft.

4.6. Konstruieren oder Basteln

Zu dem Thema Konstrukteur u. Bastler sind schon ganze Bücher geschrieben worden. Hier wäre nur zu berichten, daß es uns nicht gut gelingt, die routinierten BASIC-Programmierer zur gegliederten und leserorientierten Gestaltung ihrer Entwürfe und zur Zerlegung in für sich verifizierbare Programmteile zu überreden.

Als außerordentlich erfreulich kann ich berichten, daß es mir mit PEARL anders als mit FORTRAN mühelos gelungen ist, eine komplexe Aufgabe mit verketteten Datenstrukturen und mehrfachen Rekursionen im Praktikum mit zum Teil ganz beachtlichen Ergebnissen bearbeiten zu lassen.

4.7. Weiterführende Fächer

Zum Schluß möchte ich noch erwähnen, daß die Echtzeitsprachelemente in PEARL sehr gut bereits in der Grundlagenveranstaltung eingeführt werden können. Das bringt ohne Zweifel eine ganz außerordentliche Vereinfachung für die weiterführenden Fächer Prozeßdatenverarbeitung und Betriebssysteme.

Am anderen Ende des Spektrums der softwareorientierten Fächer liegt das Fach Betriebswirtschaft und Operations Research. In ihm werden Programmpakete des Unirechenzentrums benutzt. Dabei spielt die Anfangsprogrammiersprache keine große Rolle.

5. Schlußbemerkung

Nach dem bisherigen Erfahrungsstand werden wir wohl bei PEARL als erster Programmiersprache für die Studienrichtung Informationsverarbeitung bleiben. Ein neuer, über Basis-PEARL hinausgehender Compiler ist angekündigt. Mit ihm sollten alle wesentlichen Sprachmittel einer modernen, universellen Programmiersprache verfügbar werden.

Literatur

[1] Werum, W.; Windauer, H.: Process and experiment automation realtime language: PEARL; Braunschweig 1978

Anschrift des Autors:

Prof. Klaus Fasold
Wildhagen 48, 4800 Bielefeld 1
Tel. (0521) 103579

PEARL-Kurs und Praktikum an der Universität Erlangen-Nürnberg

Dr. P. Holleczek, Nürnberg, K. Pelz, Erlangen

Einleitung

Es ist Absicht dieses Artikels, über zwei Lehrveranstaltungen zu berichten, die seit 1975 im Rahmen der PEARL-Ausbildung an der Universität Erlangen-Nürnberg stattfinden. Neben der Darstellung von Form und Inhalt der beiden Veranstaltungen soll auch eine Erfahrungsauswertung geboten werden.

Introduction

It is the intention of this paper to report on two lectures concerning the PEARL training at the University of Erlangen, FRG. The experiences are presented together with the mode and the main topics of the lectures.

1. Veranstaltungsform

Bei den beiden Veranstaltungen handelt es sich um die "PEARL-Vorlesung" und das "Prozeßrechnerpraktikum". Während die Vorlesung "Einführung in die höhere Realzeit-Programmiersprache PEARL" eindeutig auf die Programmiersprache PEARL ausgerichtet ist, besteht im Prozeßrechnerpraktikum prinzipiell auch die Möglichkeit, Lösungen in einer anderen Programmiersprache vorzunehmen. Davon wurde bisher jedoch noch kein Gebrauch gemacht. Deswegen wird das Prozeßrechnerpraktikum in diesem Zusammenhang der PEARL-Ausbildung zugerechnet.

Die beiden Lehrveranstaltungen finden jährlich statt, wobei die PEARL-Vorlesung im Wintersemester, das Prozeßrechnerpraktikum im Sommersemester abgehalten wird. Beide Veranstaltungen sind inter-fakultativ, sowohl was die Zuhörerschaft, als auch was Veranstalter angeht. Die meisten Hörer sind Informatiker und Physiker, in geringerer Zahl z.B. auch Elektrotechniker. Veranstalter war anfangs das Physikalische Institut [1] , inzwischen sind auch die Informatik-Lehrstühle für Programmiersprachen und Betriebssysteme, sowie das Regionale Rechenzentrum beteiligt. Apparativ (d.h. durch Rechner, Apparaturen, etc.) werden die Veranstaltungen durch das Physikalische Institut unterstützt, personell durch die anderen Institutionen.

Beide Veranstaltungen werden als "Vorlesung + Übung" angekündigt, wobei das Schwergewicht naturgemäß den "Übungen" zuzurechnen ist. Der Besuch ist freiwillig. Die PEARL-Vorlesung ist als Wahlpflicht-Fach zur Diplom-Vorprüfung für Informatiker zugelassen.

2. Inhalt der Veranstaltungen

Die 1975-1980 zugrunde liegende Implementation war ASME-PEARL [2] , eine der Ur-Formen von PEARL; seit 1981 wird BASIC-PEARL [3,4] angeboten. Das Begleitmaterial sind die Hersteller-Sprachbeschreibung und ein Manuskript. Es ist erwähnenswert, daß von der ASME-Implementation als Prozeß-Schnittstelle ein CAMAC-System unterstützt wurde und zur Kommunikation zwischen Programmen und Bediener(n) graphische Ein-/Ausgabe zur Verfügung stand. Inhaltsschwerpunkte der Vorlesung sind asynchrone Ereignisse, Nebenläufigkeit, Wiedereintritts-Invarianz, Parameterübergabe-Mechanismen, Zusammenwirken zwischen Systemteil, zugehörigen Vereinbarungen und ausführbaren Anweisungen, Beseitigen historisch und lokal bedingter Scheuklappen (Batch- und Teilnehmerbetrieb am CYBER-

Rechner, BS-orientiertes "Prozeß"-Verständnis, ...). Das Prozeßrechnerpraktikum baut inhaltlich auf der PEARL-Vorlesung auf. Da sich immer viele Teilnehmer melden, die diese Voraussetzung nicht erfüllten, werden seit 1980 die ersten zwei Tage der Lehrveranstaltung für einen PEARL-Steilkurs benutzt, der Hörern ohne PEARL-Kenntnisse einen Einstieg erlaubt. Der im Steilkurs gebotene Sprachumfang beschränkt sich somit auf das Unerlässliche; alles weitere erarbeiten sich die Teilnehmer sowohl durch Programmierpraxis als auch Konsultation der Betreuer oder PEARL-kundigen Studienkollegen. Weitere Termine im Vorlesungsteil des Praktikums dienen der Konkretisierung der Aufgabenstellung und der Behandlung programmiertechnischer Themen. Die "Prozesse" des Praktikums entstammen zum größten Teil der physikalischen Experimentiertechnik, sind aber so allgemeinverständlich, daß von den Teilnehmern kein technologisches Wissen vorausgesetzt werden muß. Bisher wurden z.B. bearbeitet:

- Aufnahme und graphische Darstellung der Kennlinien von Transistoren
- Steuerung eines Experiments zur Messung der Reichweite von α-Teilchen (mit graphischer Darstellung)
- Messung und graphische Darstellung der Zeitstatistik von Zerfallsprozessen
- Steuerung eines Experiments zur Aufnahme optischer Spektren

  und
- Kopplung eines Prozeßrechners mit einem Großrechner über eine asynchrone Dialog-Schnittstelle.

Seit 1980 wird angesichts der steigenden Bedeutung verteilter Rechner-Systeme nicht nur in der Prozeßrechentechnik immer ein Versuch zur Rechnerkopplung angeboten.

Jedes Jahr stehen (mindestens) zwei Versuche zur Auswahl. Ihr Umfang ist so bemessen, daß ein Versuch immer von zwei Teilnehmern vollständig bearbeitet werden kann, was ein Gefühl der eigenen Leistung vermitteln hilft.

Die bei der Lösung der Aufgaben zu verwendenden Sprachkonstrukte sind neben den arithmetischen Elementen

- Prozeß-E/A
- Bit-/Zeichen-Kettenverarbeitung
- Parallelarbeit verschiedener Tasks (z.B. Prozeßsteuer-/Bedien-Task)
- Semaphor-Operationen (z.B. für Datenübergabe zwischen Tasks)
- Interrupt- und Signalreaktionen.

Häufig auftretende Teilaufgaben sind Realisierungen von Puffer-Mechanismen. Bei der Bearbeitung der Aufgaben hat sich ein bestimmtes Schema herausgebildet. Zuerst werden Programme geschrieben, die einen Test sämtlicher Hardwarefunktionen der angesprochenen Peripherie erlauben. Auf diese Weise lernen die Praktikanten die einschlägigen Befehlssequenzen zum Ansteuern der Prozeßperipherie kennen und können zudem den Prozeß, sobald er vollständig an den Rechner angeschlossen ist, schrittweise "fahren". Im Anschluß daran wird entsprechend der Aufgabenstellung das endgültige Prozeßprogramm erstellt und getestet. Zum Abschluß wird eine Dokumentation verlangt, die neben dem Quell-Listing eine Beschreibung der Aufgabenstellung, des erstellten Programms und einen "Erfahrungsbericht" enthalten soll.

3. Erfahrungen

Die Erfahrungen lassen sich weiter gliedern in Lehrbarkeit der Sprache, Dokumentation, Eignung von Rechner bzw. Compiliersystemen und Veranstaltungsform.

- Lehrbarkeit der Sprachen ASME- /BASIC-PEARL:

Zunächst einige Anmerkungen zu Sprachumfang und Sprach-Elementen: Der Übergang von ASME-1-PEARL auf BASIC-PEARL bedingte den Verlust der graphischen Ein-/Ausgabe. Dies wiederum führte bei den Teilnehmern zu einer fühlbaren Einbuße an Übungsbereitschaft. Auch bereitet es erhebliche Schwierigkeiten, den Sinn des STRUCT-Attributes zu schildern, wenn Strukturen weder in E-/A-Anweisungen, noch in Zuweisungen angesprochen werden können (Typ-Anweisung und Zeiger für Verkettungen existieren ohnehin nicht).

Demgegenüber soll auf die problemlose Akzeptierung des Datenstations-Konzeptes hingewiesen werden. Dies geht so weit, daß nach kurzer Praxis schon die noch vorhandenen Entwurfs-Mängel erkannt werden:

Dank der klaren Unterteilung in Kommunikations-Klassen und der im wesentlichen konsequenten Modellbildung sind die Voraussetzung zur Stoff-Vermittlung erheblich besser geworden.

Auf dem Sektor "Nebenläufigkeit" hat es sich bewährt, deutlich zwischen den 4 Task-Operationen, die Taskzustands-Veränderungen herbeiführen und Einplanungen (von Task-Operationen) zu unterscheiden. Durch die Reduzierung der Task-Zustände auf ein Minimum, kann den Teilnehmern eine präzise Kenntnis der Semantik auf einfacherem Weg vermittelt werden.

Auch der Themenkreis der systematischen Erstellung von Prozeß-Programmen sollte angesprochen werden: Bei einem Hörerkreis, der im wesentlichen aus Teilnehmern mit wenig oder gar keiner Programmierpraxis besteht, ist es ohnehin schwer und zeitaufwenig, ausreichendes Verständnis für so wichtige Spracheigenschaften wie Vereinbarungspflicht, Typ-Reinheit, Armut an Default- und Implizit-Regelungen, ... zu wecken (Verführungseffekt).

Da überdies das Denktraining beim Erlernen moderner Programm-Erstellungsmethoden in eine ganz andere Richtung geht, als beim Erlernen und Einüben einer Programmiersprache, lehnen wir die Integration beider Lernvorgänge in eine Veranstaltung mit der Begründung ab, daß sonst eine Überlastung der Hörer unvermeidlich wäre.

Als konstruktive Beiträge, in dieser Richtung seien 2 Punkte genannt:

- Als Begleitmaterial zur Vorlesung wird ein Manuskript verteilt, das gleichzeitig als Leitfaden für den Vortrag dient. Es ist im Vergleich zu Vorlesungs-Mitschriften äußerst kanpp gehalten und soll für die Teilnehmer eine optische Unterstützung sein, die es erlaubt, während des Vortrags eigene Erläuterungen und Ergänzungen einzufügen.
- Zum Thema "Umgang mit Tasks und Semaphoren" kann auf die regelmäßig wiederkehrenden Fragen "Wie viele Tasks für welches Problem?", "Für welche Zustände welche Objekte Semaphore mit welcher Vorbesetzung?", usw. keine Antwort mit Rezeptcharakter gegeben werden.

  Wir schlagen aber als pragmatisches Vorgehen die Bearbeitung immer wieder auftretender Teil-Probleme vor, deren Schwierigkeitsgrad langsam steigen sollte und für die Lösungen zur Reproduktion, Diskussion und Variierung verfügbar sein sollten.
  Beispiele wären
  Gegenseitiger Ausschluß
  Erzeuger/Verbraucher auf Puffer m.L = 1
  " " " " " = n

- Dokumentation

Die Hersteller-Sprachbeschreibungen sind als Lehrbuch nicht geeignet, als Dokument jedoch unentbehrlich. Das macht zusätzliche Leitfäden notwendig.

Eine Gebrauchsanweisung für die Compiler-Bedienung war bei der ASME-Implementation nicht nötig, da sie beispiellos einfach war.
Bei BASIC-PEARL ist die vorhandene Beschreibung sowohl unzureichend als auch zu kompliziert.

Während der Prozeß (=Experiment) leicht erklärt werden kann, stößt das Verständnis der Funktionsweise von Prozeßperipherie und der angeschlossenen Steuerungen auf gewisse Schwierigkeiten. Eine Beschreibung als "black box" (was bewirkt welches Bitmuster) käme dem Studenten sicher am meisten entgegen.

- Eignung von Rechner und Compiliersystem

Bis 1980 stand der ASME-Compiler auf einer Siemens 306 mit Kartenleser und Schnelldrucker zur Verfügung, inzwischen wird ein BASIC-PEARL-Compiler auf einer R30 eingesetzt. Während auf der 306 die langen Übersetzungszeiten (bedingt durch den in Fortran geschriebenen Compiler) den Ablauf der Übungen merklich verzögerten, waren zum Ausgleich die Bedienung des Compilers und der restlichen Maschine denkbar einfach: Es gab ein Kommando zum Starten des Compilers mit anschließendem Bindelauf, ein Kommando zum Starten des PEARL-Laders, ein Kommando zum Laden des Programms und ein Kommando zum Starten des Programms.

Bei der inzwischen eingesetzten R30 sind die Übersetzungszeiten hinreichend schnell; ihnen stehen jedoch mehrere separate Bindeläufe und

eine besonders für Neulinge umständliche Bedienung gegenüber. Dadurch ist die Zeit zur Erzeugung eines ablauffähigen Programms fast genau so lang, wie bei der älteren 306-Implementation. Auch im "Durchsatz" bei der Programmerstellung ist die alte Technologie nicht leicht zu schlagen. Was an der 306 mit zwei Kartenlochern bewältigt werden konnte, bedarf an der neuen Maschine trotz des komfortablen Editors sicher einer größeren Anzahl von Terminals.

Das Testen der entwickelten Programme stellte mit dem ASME-Compiliersystem der Siemens 306 kein Problem dar, da ein interaktiver Test auf Sprachebene möglich war.

- Veranstaltungsform

Das eigentliche Hindernis in der Veranstaltung (insbesondere des Prozeßrechnerpraktikums) sind die langen Programmerstellungszeiten. Für einen ausführlichen Programmtest bleibt oft nur wenig Zeit, so daß die dabei unweigerlich auftretenden Probleme erst gar nicht entdeckt werden. Entsprechend kritisch hängt die "Qualität" der Veranstaltung von der Teilnehmerzahl ab. Selbst bei 10 Teilnehmern reicht ein Nachmittag pro Woche für den Praktikumsbetrieb nicht mehr aus, was wiederum die Betreuung zu einem (zeitlichen) Problem macht.

4. Zusammenfassung

Die Lehrveranstaltungen PEARL-Vorlesung und Prozeßrechnerpraktikum haben an der FAU bereits eine gewisse Tradition. Man kann also bereits gewisse Aussagen über Wert und Unwert des eingeschlagenen Weges machen. Die Durchführung der Veranstaltungen kann sicher (z.B. durch schnellere Programmerstellung und durch bessere Dokumentation) verbessert werden. Die Veranstaltungen, insbesondere das Prozeßrechnerpraktikum, vermitteln aber wertvolle Praxis im Umgang mit Prozeßrechnern. Sie bereiten, obwohl aufwendig für Teilnehmer und Betreuer, allen Beteiligten einen gewissen Spaß. Nicht zuletzt deswegen, weil mit Hilfe von PEARL ein relativ rascher Fortschritt im Erfahren der Problematik und die vollständige Lösung abgeschlossener Probleme im Rahmen einer Lehrveranstaltung möglich ist. Damit konnte eine Lücke im hiesigen Veranstaltungs-Angebot geschlossen werden, deren Beseitigung sonst nur mit einem unverhältnismäßigen Zeitaufwand hätte erreicht werden können.

5. Literatur

[1] H o l l e c z e k , Peter: Einsatz der Siemens 3Ø6 im Rahmen eines Prozeßrechner-Praktikums, Jahresbericht 1979/8Ø des Physikalischen Instituts der Universität Erlangen-Nürnberg, Februar 1981

[2] Programmieranleitung für das ASME-1-PEARL-Subset, PDV-Bericht KFK PDV 1ØØ, Ges. f. Kernforschung, Karlsruhe, Nov. 1976

[3] Beschreibung PC3O/PEARL 3ØØ, Programm-Service Siemens Systeme 3ØØ, Produkt-Nr. P711ØØ-D3Ø1O-X-A3-35, 3O.7.8O

[4] DIN 66253, Teil 1, Informationsverarbeitung, Programmiersprache PEARL Basic-PEARL, Beuth-Verlag, 1978

Dr. Peter Holleczek
Regionales Rechenzentrum Erlangen

Martensstr. 1
8520 Erlangen

Kuno Pelz
Institut für Mathematische Maschinen und Datenverarbeitung (Informatik)
Lehrstuhl für Programmier- und Dialogsprachen sowie ihre Compiler

Martensstr. 3
8520 Erlangen

PEARL in der Lehre am Institut für Regelungstechnik und Prozeßautomatisierung der Universität Stuttgart

von E. Joho und Ch. Riewerts

1. Zusammenfassung

Am Institut für Regelungstechnik und Prozeßautomatisierung der Universität Stuttgart sind seit 1975 insgesamt etwa 1000 Studenten der Elektrotechnik in der Prozeßprogrammiersprache PEARL ausgebildet worden.

Es wird den Studenten keine eigene Vorlesung über PEARL angeboten, sondern die Kenntnisse über die Programmierung mit PEARL werden vermittelt im Rahmen der Vorlesung über Prozeßautomatisierung sowie dem zugehörigen Praktikum. Die Erfahrung zeigt, daß diese Ausbildung (insgesamt ca. 5 Stunden Vorlesung, 10 Stunden Seminar-Übungen, 8 Stunden Praktikum) ausreicht, um Studenten der Elektrotechnik soweit auszubilden, daß sie kleinere Aufgabenstellungen in PEARL programmieren können.

Wir werden auch in Zukunft PEARL verstärkt in die Ausbildung der Ingenieure integrieren, wobei die folgenden drei Punkte von besonderer Bedeutung sind:

1) PEARL zeichnet sich aus durch übersichtlichen Sprachaufbau und ein anwenderfreundliches E/A- und Tasking-Konzept. Nur dadurch ist es möglich, im zeitlich begrenzten Rahmen des Lehrangebots die Verfahren der Realzeitverarbeitung in einem Prozeßautomatisierungssystem zu lehren.

2) Die Anwendung von PEARL trägt wesentlich dazu bei, daß die Programmierung strukturiert, modular und hinreichend dokumentiert abgefaßt wird. Ein zusätzliches Hilfsmittel ist das an unserem Institut entwickelte Entwurfswerkzeug EPOS, das ein strukturiertes Vorgehen beim Entwurf von Prozeßautomatisierungssystemen erzwingt [2] [3].

3) Neben der theoretischen Ausbildung in PEARL wird den Studenten im Rahmen von Studien- und Diplomarbeiten sowie in einem Praktikum die Möglichkeit zur praktischen Anwendung geboten. Die Grundlage dafür bieten eine Reihe von Modellprozessen, die an unseren Prozeßrechner AEG 80-20/4 angeschlossen sind.

2. Einführung

Die Lehre, Ausbildung und Schulung von PEARL am Institut für Regelungstechnik und Prozeßautomatisierung (IRP) stützen sich auf drei Säulen:

- auf die angebotenen Lehrveranstaltungen,
- auf Studien- und Diplomarbeiten und
- auf die praktische Arbeit im Labor.

Im folgenden Vortrag werden diese drei Themenkreise behandelt und anhand einiger Beispiele näher erläutert.

Im ersten Teil wird darüber berichtet, wie der PEARL-betreffende Lehrstoff vermittelt wird, welche Unterlagen den Studenten zur Verfügung gestellt werden und welche Kenntnisse sie zur Diplomprüfung mitbringen müssen. Anhand der ausgegebenen Themenstellungen für Studien- und Diplomarbeiten kann ermessen werden, auf welchen Arbeitsgebieten das Institut tätig ist. Den Schwerpunkt dieses Vortrags bildet die exemplarische Beschreibung einiger Modellprozesse. Diese modellhaften Nachbildungen technischer Prozesse sind die Grundlage unserer praktischen Ausbildung. Nur sie können in Verbindung mit unserem Prozeßrechner illustrieren, durch welche Möglichkeiten sich PEARL im praktischen Einsatz auszeichnet.

3. Lehrveranstaltungen

3.1 Gesamtumfang

Einen ersten Überblick über PEARL sowie erste praktische Erfahrungen in seiner Anwendung werden den Studenten durch verschiedene Lehrveranstaltungen geboten. Als Voraussetzung bringen die Studenten Grundkenntnisse mit, die sie sich in der Vorlesung "Datenverarbeitung" (5. Fachsemester) angeeignet haben, und auf denen wir aufbauen können.

Aus der folgenden Auflistung kann entnommen werden, welchen zeitmäßigen Anteil PEARL bei unseren Lehrveranstaltungen einnimmt, und an welchen Zuhörerkreis sich unser Angebot richtet:

- Vorlesung "Prozeßautomatisierung I"
  (wöchentlich 1 Stunde)
  PEARL-Anteil: 4 Stunden
  6. Fachsemester (ca. 120 Hörer)
- Seminar zu "Prozeßautomatisierung I"
  (wöchentlich 1 Stunde)
  PEARL-Anteil: 9 Stunden
  6. Fachsemester (ca. 120 Hörer)
- Seminar zu "Prozeßautomatisierung II"
  (wöchentlich 1 Stunde)
  PEARL-Anteil: 2 Stunden
  7. Fachsemester (ca. 60 Hörer)
- Praktikum "Regelungstechnik und Prozeßautomatisierung" (5 Versuche)
  PEARL-Anteil: 1 ganztägiger Versuch
  (ca. 6-8 Stunden)
  7. Fachsemester (ca. 60 Teilnehmer)

Eine Vertiefung des Fachwissens und der praktischen Anwendung dieser Prozeßprogrammiersprache kann durch Studien- und Diplomarbeiten erworben werden, die sich größtenteils mit den in Abschnitt 5.2 aufgeführten Modellprozessen befassen.

3.2 Vorlesung

Hier werden folgende Informationen über PEARL vermittelt:

- Entstehungsgeschichte von PEARL
- PEARL im Vergleich zu anderen höheren Prozeßprogrammiersprachen und Assemblersprachen
- derzeitig verfügbare PEARL-Systeme
- Einführung in die Spracheigenschaften von PEARL
  - EA-Konzept
  - Echtzeiteigenschaften
  - Synchronisierungsmöglichkeiten
  - Blockstruktur.

An Literatur wird - neben dem Vorlesungsmanuskript[1] - ein ungefähr 50-seitiger Umdruck zu PEARL sowie eine Sammlung von kleineren Anwendungsbeispielen zur Verfügung gestellt. Studenten, die ein über den Rahmen der Vorlesung und des Seminars hinausgehendes Interesse an PEARL finden, können sich in der Institutsbibliothek mit den gängigen PEARL-Fachbüchern [6][7] und den Sprachbeschreibungen verschiedener Firmenimplementationen beschäftigen.

3.3 Seminar und Prüfung

Im Seminar zu "Prozeßautomatisierung I" werden ausschnittsweise Eigenschaften von BASIC-PEARL behandelt und an kleineren Fallbeispielen vertieft. Grundlage hierzu sind der bereits im vorigen Abschnitt erwähnte Umdruck und die Aufgabensammlung. Die Schwerpunkte liegen dabei in der Darstellung

- des Systemteilaufbaus
- der Prozeßdatenein-/ausgabe
- des "Tasking" und den dabei möglichen Synchronisierungen
- der Algorithmik.

Im Seminar zu "Prozeßautomatisierung II" werden im Zusammenhang der Themengebiete "strukturierte Programmierung" und "Petri-Netze" Beispiele in PEARL vorgestellt.

Die zu den beiden Vorlesungen und Seminaren stattfindenden zweistündigen Hauptdiplomprüfungen enthalten jeweils eine Aufgabe, die sich mit PEARL beschäftigt, wobei der Anteil der PEARL-Aufgabe zur Prüfung in "Prozeßautomatisierung I" ca. 50 % beträgt.

In den Prüfungen wird von den Studenten verlangt, eine sinnvolle Umsetzung der Automatisierungsaufgaben in eine Taskstruktur vorzunehmen, sie müssen die Synchronisierungsmechanismen in PEARL beherrschen und den in einem Flußdiagramm dargestellten Kontrollfluß in PEARL-Anweisungen übersetzen.

3.4 Praktikumsversuch "Heizungsautomatisierung"

Für die praktische Ausbildung der Studenten der Elektrotechnik wird vom IRP im Rahmen der Regelungstechnik und Prozeßautomatisierung ein Fachpraktikum angeboten.

Einer der fünf Versuche hat zur Aufgabe, ein Automatisierungsprogramm für eine Heizungsanlage in BASIC-PEARL zu erstellen. Die Bearbeitung dieser Aufgabe erfolgt in mehreren Schritten:

- Analyse der Heizungsanlage
- Festlegung des Automatisierungsprogramms
- Spezifikation des Automatisierungsprogramms
- Codieren, Übersetzen und Test des Programms
- Inbetriebnahme des Prozeßautomatisierungssystems.

Versuchsvorbereitung

Die Problemstellung ist vereinfacht dargestellt und den Praktikanten aus der Vorlesung "Prozeßautomatisierung I" bekannt. Sie erhalten mindestens eine Woche für die Versuchsvorbereitung Zeit, in der sie die Tätigkeiten vor der eigentlichen Programmierung durchzuführen haben. Dazu gehört die Analyse der Heizungsanlage, die Aufteilung der Gesamtaufgabe in Teilaufgaben und der Entwurf einer Programmstruktur bezüglich Tastaufteilung, E/A-Konzept und Synchronisierungsvorschläge.

Hierzu erhalten sie einen 10-seitigen Umdruck mit den notwendigen Erklärungen und einem Wirkschaltbild der automatisierten Zentralheizungsanlage, siehe Bild 1.

Versuchsdurchführung

Zur Versuchsdurchführung stehen den Studenten 6-7 Stunden zur Verfügung, in denen unter Anleitung einer Hilfskraft die Praktikanten am Prozeßrechner die entworfene Programmstruktur durch ein PEARL-Programm realisieren.

Das Programm hat folgende Aufgaben zu lösen: (B i l d 1).

- Regelung der Raumtemperatur im Raum 1 (zyklische Erfassung der analogen Spannung

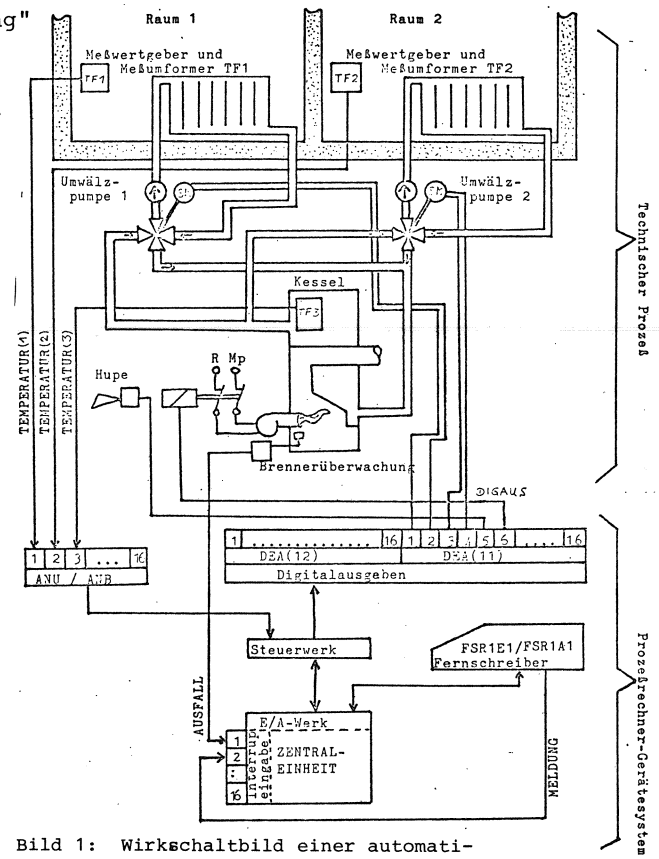


Bild 1: Wirkschaltbild einer automatisierten Zentralheizungsanlage

  (TEMPERATUR (1), die der Meßwertgeber und Meßumformer TF1 liefert, Anwendung eines geeigneten PID-Algorithmus, Ausgabe des Stellsignals). Als Stellglied dient ein Stellmotor SM, der an ein Mischerventil gekoppelt ist. Mit diesem Ventil wird heißes Kesselwasser mit kälterem, aus dem Heizkörper zurückströmenden Wasser gemischt und so die Heizkörpertemperatur eingestellt.

- Regelung der Raumtemperatur im Raum 2 (in gleicher Weise wie für Raum 1).
- Regelung der Wassertemperatur im Heizkessel (zyklische Erfassung der analogen Spannung TEMPERATUR (3), Anwendung eines Zweipunkt-Regelalgorithmus, Ausgabe der Stellgröße als binäres Spannungssignal). Als Stellglied dient ein Schütz, mit dem der Ölbrenner ein- und ausgeschaltet wird.
- Alarmgabe bei Brennerstörung. Hierzu soll ein von einer Brennerüberwachungseinrichtung geliefertes Signal AUSFALL erfaßt und eine Warnhupe betätigt werden (Ausgabe des binären Signals WARNUNG). Gleichzeitig soll

auf der Schreibmaschine eine Störungsmeldung ausgedruckt werden.

- Alarmgabe bei Störung der Analogeingabe. Hierzu wird ein internes Signal ausgewertet und ein Störungsprotokoll ausgegeben.

Von den Studenten sind nur Teile dieser Aufgabenstellung als Task oder Prozedur zu programmieren, damit der Zeitaufwand nicht zu groß wird.

Zum Testen des erstellten Gesamtprogramms steht ein kleiner Modellprozeß zur Verfügung. Dementsprechend ergeben sich kleine, technisch bedingte Abweichungen gegenüber der im Wirkschaltbild dargestellten Anlage:

- Die Wärmeerzeugung erfolgt elektrisch. Ein Widerstand heizt ein Kühlblech im Innern des Kessels bis auf eine maximale Temperatur von ca. 100°C auf.
- Der Wärmetransport erfolgt nicht mit Hilfe von Wasser sondern durch Luft. Dabei wird die Konvektion durch einen unter dem Kessel angebrachten Lüfter unterstützt.
- Die Mischer werden durch zwei Schieber ersetzt, die die Luftzufuhr zu den beiden Räumen steuern. Die beiden Schieber werden durch Elektromotoren angetrieben.
- Die durch eine Fotozelle überwachte Flamme des Brenners wird durch eine von einem Fototransistor überwachte LED ersetzt.

Der Versuch wird abschließend von den Praktikanten protokolliert. Dabei sollen tiefergehende Fragen zur Nachbearbeitung beantwortet werden wie z.B. der zeitliche Ablauf der verschiedenen Tasks und eine Erläuterung des Regelverhaltens.

4. Diplom- und Studienarbeiten

4.1 Gesamtumfang

Im Durchschnitt werden vom IRP pro Semester 35 Diplom- und Studienarbeiten ausgegeben und betreut.

Betrachtet man den Zeitraum von Anfang 75 bis Ende 1980, so ergibt sich die Tatsache, daß sich von den über 400 Arbeiten jede 9. mit der Programmiersprache PEARL beschäftigt. Eine Aufteilung über die Zeit ist dem Bild 2 zu entnehmen:

Ab 1975/1976 wurde der ASME-PEARL-Compiler mit Laufzeitsystem an realen Modellprozessen erprobt. Die Eignung von PEARL zur Automatisierung von unterschiedlichen technischen Prozessen konnte dadurch nachgewiesen werden.

1978 mit dem DIN-Entwurf von BASIC-PEARL und der Auslieferung eines AEG-PEARL-Compilers an das IRP begann man, weitere Modellprozesse zu projektieren und an die AEG 80-20/4 anzuschliessen. Die Wartung der Programme und das Umstellen von der AEG 60-50 auf die 80-20 erklärt die hohe Zahl der Semesterarbeiten 1979.

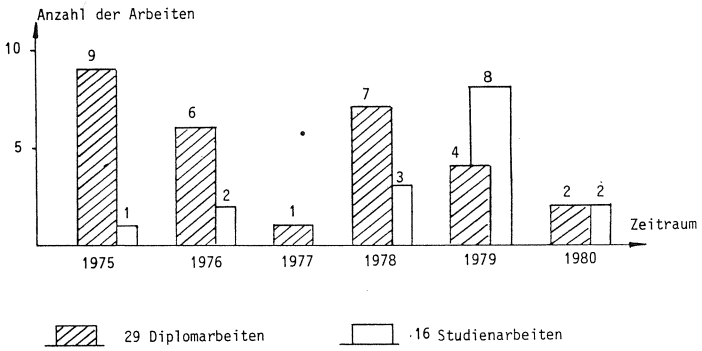


Bild 2: Übersicht der Diplom- und Studienarbeiten bezgl. PEARL

4.2 Themenkreise

Die ca. 45 Arbeiten, die bisher am IRP über PEARL von den Studenten angefertigt wurden, lassen sich in zwei grobe Themenkreise aufteilen: PEARL in der Anwendung und PEARL als eigenes Untersuchungsobjekt.

PEARL in der Anwendung

Hierzu zählen zum einen die Softwareentwürfe, die mit Hilfe des Entwurfs- und Spezifikationssystems EPOS angefertigt wurden. Dabei ist auf unterster Entwurfsebene der CODE-Part benutzt worden, wo - noch in einer Entwurfsphase - PEARL-Anweisungen angegeben werden können.

Ebenfalls sind hier hinzuzurechnen die Arbeiten, die sich mit der Implementierung unterschiedlichster Automatisierungsaufgaben in PEARL beschäftigen: Durchführung von Plausibilitätskontrollen, Optimierung der Wegberechnung bei einer Stückverfolgung, digitale Filterung, Benutzerdialog, usw.

Hervorzuheben sind die Arbeiten, die den Anschluß der im Institut aufgebauten Modellprozesse an den Prozeßrechner betreffen. Diese Aufgabenstellung ist oft verteilt auf mehrere Studien- und Diplomarbeiten, sodaß diese Programme ausgetestet und verifiziert werden können.

PEARL als eigener Untersuchungsschwerpunkt

Zu diesem Themenkreis zählen die Sprachvergleiche, die die Studenten anhand von Beispielen durchführen, um die besonderen Eigenschaften von PEARL herauszuarbeiten. Verglichen wurden u.a. Assemblersprachen und Prozeß-FORTRAN jeweils mit PEARL. Insbesondere war es unser Anliegen, verschiedene PEARL-Implementierungen untereinander zu vergleichen, so z.B. AEG 80-20-PEARL mit BASIC-PEARL, um bei der derzeitigen Marktsituation den Überblick zu behalten.

Außerdem sind für unterschiedliche Beispiele aus der Automatisierungstechnik Untersuchungen gemacht worden bezüglich der Portabilität, der Wartbarkeit und der strukturierten Programmierung mit PEARL.

5. Labortätigkeit

Dem Institut steht zur praktischen Anwendung von PEARL ein voll ausgebauter Prozeßrechner AEG 80-20/4 zur Verfügung. Hinzu kommt die Möglichkeit, im Rahmen der bewilligten Mittel und unter Zuhilfenahme der eigenen Werkstatt Modellprozesse im Labor aufzubauen. Diese Modellprozesse sind modellhafte Nachbildungen realer technischer Prozesse und bilden die Voraussetzung für den Einsatz einer Realzeitprogrammiersprache.

5.1 Prozeßrechner

Die Gerätekonfiguration des Prozeßrechners AEG 80-20/4 zeigt das Bild 3. Es muß dabei erwähnt werden, daß der Plotter vom PEARL-System aus nicht angesprochen werden kann.

(B i l d 3)

5.2 Modellprozesse

Zur Zeit sind am IRP die folgenden Modellprozesse, für die ein Programm in PEARL geschrieben wurde, verfügbar:

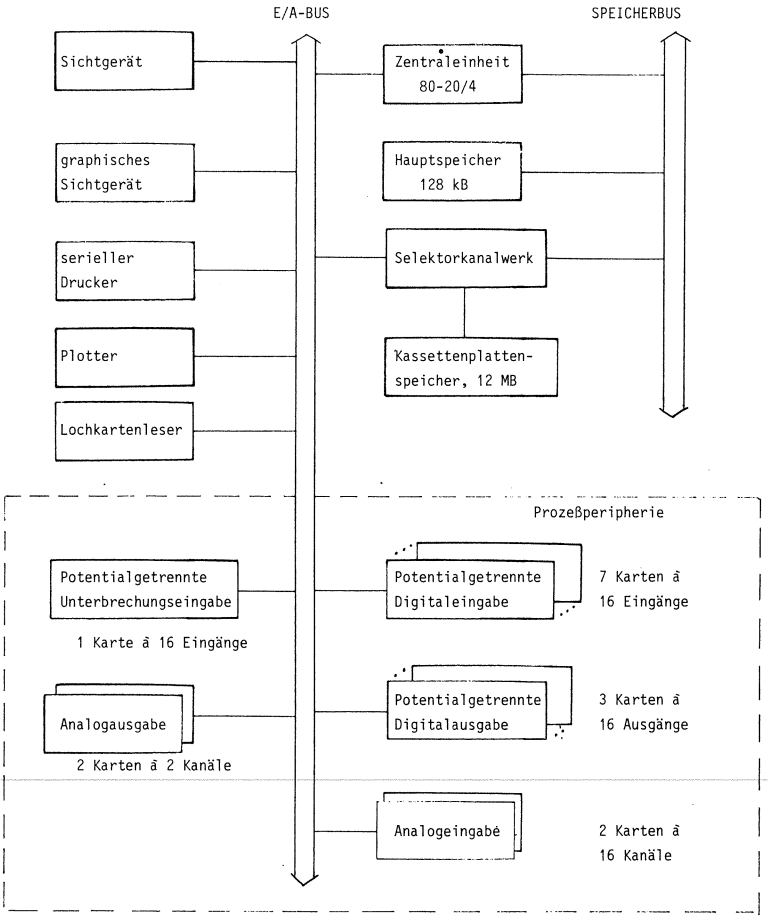


Bild 3: Gerätekonfiguration des Prozeßrechners AEG 80-20/4 am IRP

- Betriebsüberwachung
- Heizungsautomatisierung
- Stückgutlager
- Schachspiel
- Paketverteilanlage

Neben dem schon beschriebenen Automatisierungssystem "Heizungsautomatisierung" (siehe Kap. 3.4) werden in den beiden folgenden Abschnitten die Modellprozesse "Betriebsüberwachung" und "Stückgutlager" vorgestellt.

Das Automatisierungssystem "Paketverteilanlage" war in letzter Zeit Gegenstand zahlreicher Veröffentlichungen. Eine umfassende Beschreibung kann dem PDV-Bericht 186 Teil I entnommen werden [5].

5.2.1 Modellprozeß "Betriebsüberwachung"

1) Zielsetzung und Aufgabenstellung

In einem technischen Prozeß, der vorwiegend durch kontinuierliche Vorgänge gekennzeichnet ist, sollen analoge und binäre Prozeßgrößen erfaßt und überwacht werden.

Nach dem zyklischen Einlesen im stationären Betriebszustand sind folgende Teilaufgaben zu lösen:

- Anpassung der Rohwerte in Fertigwerte
- Überprüfung auf obere und untere Grenzwerte
- bei Grenzwertüberschreitung Ausgabe eines Störungsprotokolls
- regelmäßige Ausgabe eines Betriebsprotokolls.

Als Rechnerbetriebsart ist für diese Überwachungsaufgabe der on-line open-loop Betrieb vorgesehen.

2) Beschreibung des technischen Prozesses

Der technische Prozeß ist in Form einer Kaffeemaschine realisiert (siehe dazu Bild 4).

(B i l d 4)

Die Anlage besteht im wesentlichen aus einem Wasser- und einem Dampfkessel, denen in unregelmäßigen Abständen Wasser bzw. Dampf verschiedener Menge entnommen werden. Die Sollaustrittstemperatur kann dabei vorgegeben werden. Der Wasserzulauf in die Kessel erfolgt über separate, schwimmergesteuerte Magnetventile und einer Pumpe mit Druckregulierventil. Beide Kessel verfügen über Heizspiralen, die vom Netz gespeist werden. Je nach Druck bzw. Temperatur werden sie über Relais ein- und ausgeschaltet. Beide Kessel verfügen über einen sogenannten

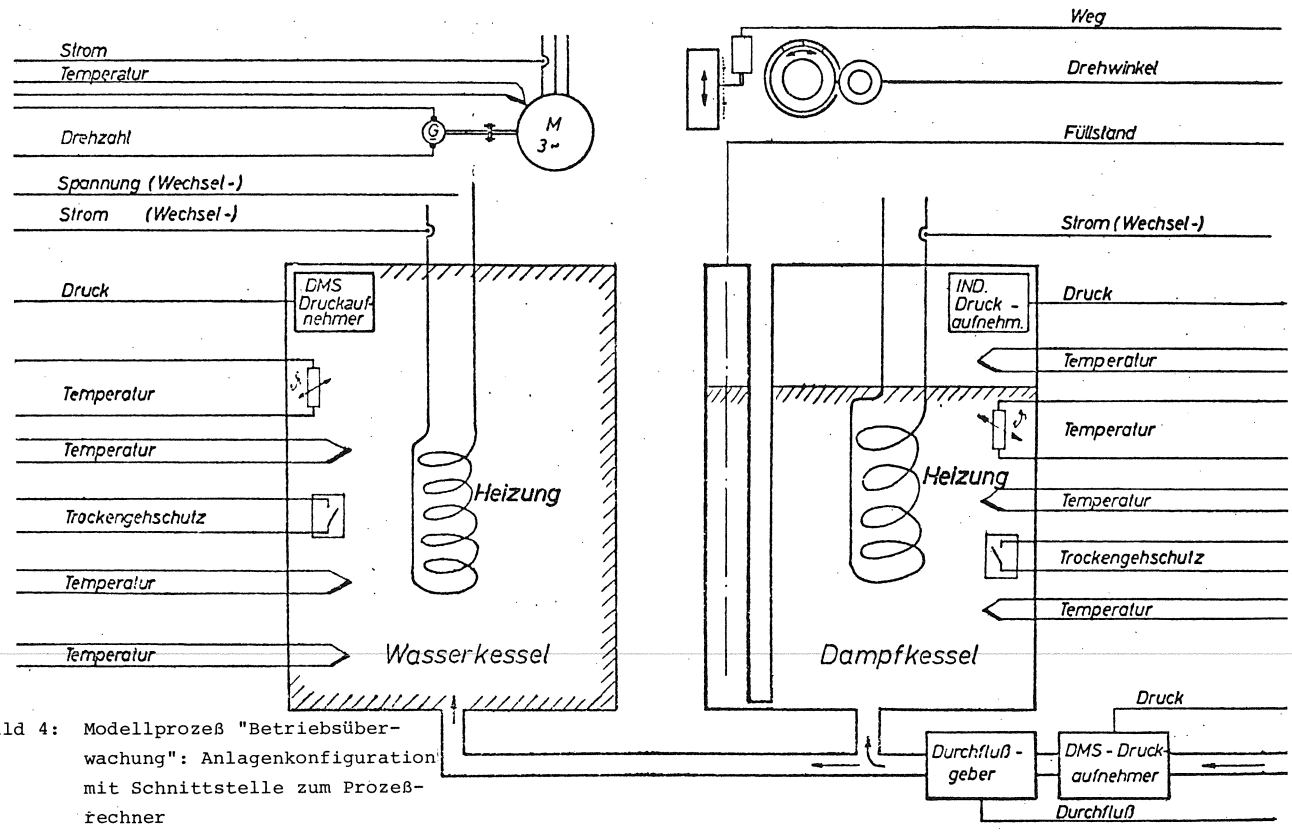


Bild 4: Modellprozeß "Betriebsüberwachung": Anlagenkonfiguration mit Schnittstelle zum Prozeßrechner

Trockengehschutz, der verhindert, daß bei zu geringem Wasserstand die Heizungen eingeschaltet werden. Über verschiedene Ein/Aus-Schalter wird Dampf bzw. Wasser bestimmter Menge entnommen, der Wasserzulauf zu den Kesseln geöffnet und der Pumpenmotor eingeschaltet.

Neben diesen existieren weitere Anlagenteile, auf die nicht näher eingegangen wird: Endschalter, Überstromschutzschalter, Lichtschranken, Drehstrommotor, ...

3) Anschluß des technischen Prozesses an den Prozeßrechner

Wie aus Bild 4 ersichtlich, werden folgende Analoggrößen gemessen und dem Rechner übermittelt:

- Temperatur im Dampfkessel (^{o}C)
- Temperatur im Wasserkessel (^{o}C)
- Wasserkesseldruck (atue)
- Dampfkesseldruck (atue)
- Leitungsdruck (atue)
- Durchfluß (1/min)
- Heizstrom Wasserkessel (A)
- Heizstrom Dampfkessel (A)
- Füllstand (cm)
- Mahlgrad (Winkelstellung)

Insgesamt sind 26 Analogwerte zum Rechner geführt.

Von den digitalen Eingabepunkten werden vom Modellprozeß "Betriebsüberwachung" 42 belegt. Dazu zählen die verschiedenen Schalterstellungen, die Überstrom- und Trockengehschütze, Motorrelais sowie die 8-bit-breite digitale Durchflußmessung und eine 10-bit-breite Zeitmessung für die Laufzeit der Kaffeemühle.

Außerdem werden vom Prozeß 7 Interrupteingänge benutzt, um u.a. das Ein- und Ausschalten der Heizungen zu melden.

4) Beschreibung des Programmsystems

Zur Erfüllung der oben beschriebenen Aufgaben existiert ein in ASME-PEARL geschriebenes Programm, das einen Umfang von 2494 Quellzeilen aufweist. Bild 5 zeigt eine Übersicht über das Programmsystem, wobei den vier Tasks folgende Aufgaben zukommen:

Task DATEN: Eingabe der Meßstellendaten mit Überprüfung der Kartenreihenfolge.

Bei falscher Reihenfolge erfolgt ein Ausschrieb. Einplanung der zyklisch zu startenden Tasks: EINGABE und PROTOKOLL.

Task EINGABE: Erfassung der Analog- und Binärgrößen.
Umrechnung der Rohwerte in Fertigwerte.
Grenzwertprüfung der Fertigwerte.
Prüfung der Binärwerte auf richtigen Zustand.

Task STOER: Protokollierung der Grenzwertverletzungen bei Analoggrößen.
Protokollierung falscher Binärzustände.

Task PROTOKOLL: Protokollierung der Fertigwerte der Analoggrößen.

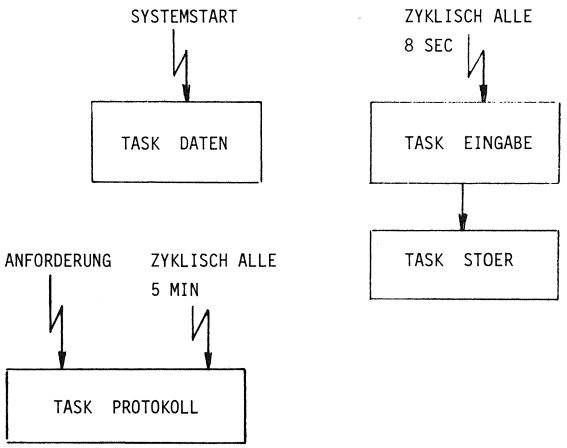


Bild 5: Programmsystem zur Automatisierung der "Betriebsüberwachung"

Dieses Programm wurde vor 2 Jahren auf AEG-PEARL übertragen, um neben der AEG 60-50 auch den Rechner AEG 80-20 an den Modellprozeß anschließen zu können. Dabei wurde aus Übersichtlichkeitsgründen und zur Testvereinfachung das 1-Modul-Konzept verlassen und eine Neuimplementierung vorgenommen. Hierbei lagen die Hauptschwierigkeiten bei der unterschiedlichen Wortlänge der zwei Prozeßrechner (60-50: 24 bit/80-20: 16 bit), bei der statischen Speicherverwaltung der 80-20 und dem kleinen Common-Bereich von 4 KB.

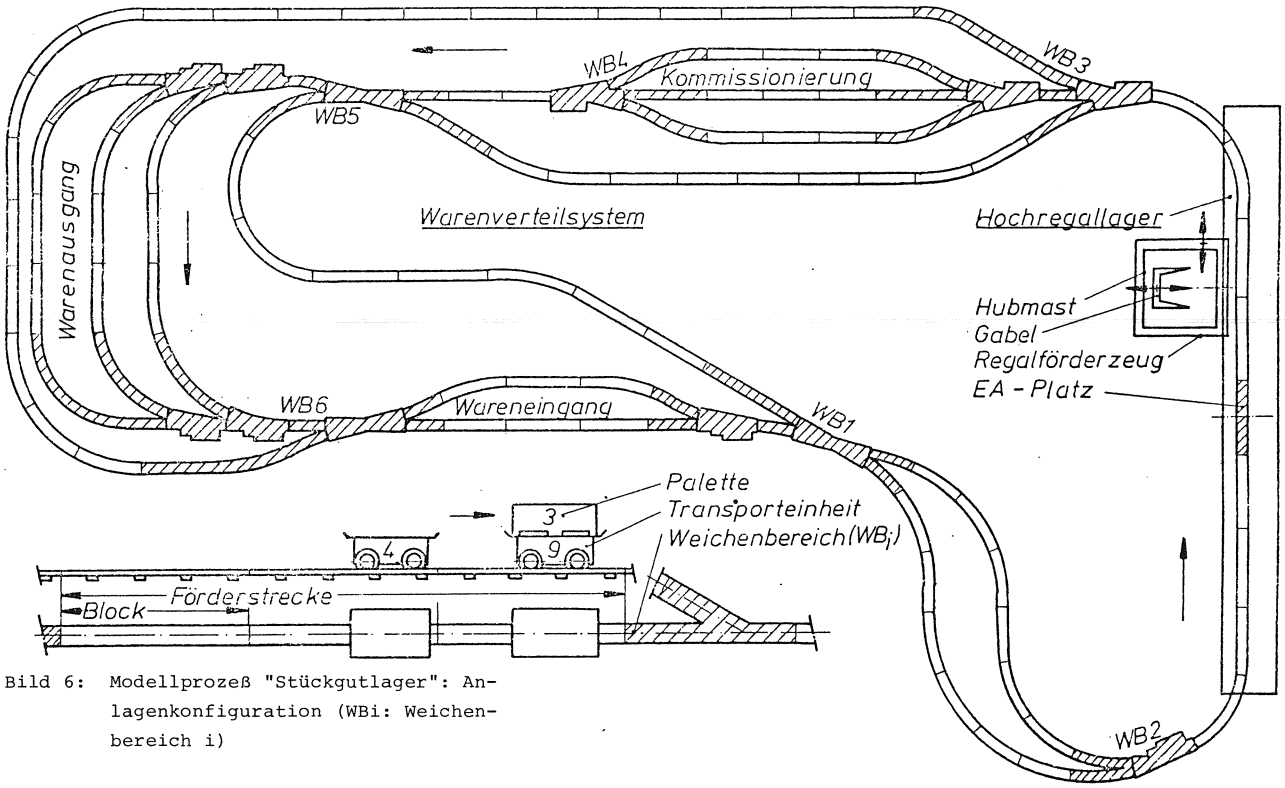


Bild 6: Modellprozeß "Stückgutlager": Anlagenkonfiguration (WBi: Weichenbereich i)

5.2.2 Modellprozeß "Stückgutlager"

1) Zielsetzung und Aufgabenstellung

In einem modellhaft aufgebauten Stückgutlager sollen Lieferaufträge durch die automatische Steuerung von Förderzeugen in einem Warenverteilsystem und eines Regalförderzeuges in einem Hochregallager bearbeitet werden.

Manuelle Eingriffe in das Geschehen im technischen Prozeß sollen sich dabei im wesentlichen auf die Eingabe der Lieferaufträge über ein Dateneingabegerät und die Kommissionierung beschränken.

Dabei sind die folgenden Teilaufgaben zu lösen:

- Erfassung, Steuerung und Überwachung der Förderzeugbewegungen
- Erfassung, Steuerung und Überwachung der Regalförderzeugbewegungen
- Eingriffe des Bedienpersonals über ein Dialogsystem in das Prozeßgeschehen.

2) Beschreibung des technischen Prozesses

Der gesamte technische Prozeß ist in zwei Anlagenteile gliederbar, dem

- Warenverteilsystem

und

- Hochregallager.

Einen Überblick der Anlagenkonfiguration ist aus Bild 6 zu entnehmen.

(B i l d 6)

Warenverteilsystem:

Es besteht aus 16 Förderabschnitten (Förderstrecken) und 6 Weichenbereichen mit insgesamt 9 stellbaren Weichen. Die Förderabschnitte sind in sich nochmals in (Förder-)-Blöcke aufgeteilt, die jeweils nur von einem Förderzeug belegt werden können. Untereinander sind die Förderblöcke über eine Auffahrsicherung (Blocksicherung) gekoppelt, die eine Kollision der Förderzeuge verhindert.

Eine Ausnahme bildet unter den Förderabschnitten die Strecke vom Weichenbereich 2 zum EA-Platz, die aus nur einem belegbaren Förderblock besteht, der entgegen der Fahrtrichtung nicht gegen Auffahrten gesichert ist. Jeder Förderabschnitt hat einen Ausfahrförderblock (Förderblock, der in Fahrtrichtung direkt vor dem sich anschließenden Weichenbereich liegt), von dem durch Anlegen eines Prozeßsignals mit entsprech-

endem Spannungspegel eine Förderzeugausfahrt initiiert werden kann. Der EA-Platz ist zudem mit einer Positioniereinrichtung ausgerüstet, durch die die Förderzeuge in die für einen Entnahme-bzw. Absetzvorgang durch das Regalförderzeug erforderliche genaue Halteposition gebracht werden.

Alle belegbaren Förderblöcke und die Weichenbereiche sind über die Blocksicherung auf den Zustand "belegt" oder "unbelegt" überprüfbar. Die Förderzeuge in diesem Warenverteilsystem, das mittels Modelleisenbahnschienen und - weichen realisiert ist, sind umgebaute Modelleisenbahnlokomotiven, die mit einer Ladefläche versehen sind.

Durch Zusammenfassung einzelner Förderstrecken lassen sich die für ein Lagersystem typischen, durch ihre Funktion bestimmten Bereiche simulieren:

- Wareneingang
- Warenausgang
- Kommissionierbereich

Hochregallager:

Es besteht aus einem Lagergerüst mit insgesamt 99 Stellplätzen für die Paletten und dem Regalförderzeug.

Das Regalförderzeug besteht aus einem horizontal verfahrbaren Hubmast, innerhalb dessen eine vertikal verfahrbare Hebeplattform mit einer Gabel zum Absetzen und Aufnehmen von Paletten angebracht ist.

Das Regalförderzeug ist mit unterlagerten Steuerungen für die Positionierung der Hebeplattform, sowie für die Hub- und Schubbewegungen der Gabel ausgerüstet.

3) Anschluß an den technischen Prozeß

- Digitaleingabe
  99 Eingabepunkte zur Belegungsanzeige der Förderblöcke
  1 Eingabepunkt zur Anzeige des Ende einer Förderzeugpositionierung am EA-Punkt
  5 Eingabepunkte zur Anzeige des Regalförderzeugstatus
- Digitalausgabe
  16 Ausgabepunkte für die Ausfahrbefehle
  9 Ausgabepunkte für die Weichensteuerung
  8 Ausgabepunkte für die Positionierungskodierung des Regalförderzeugs
  4 Ausgabepunkte für die Steuerung des Regalförderzeugs
- Interrupte
  1 Interrupt als Sammelanzeige für einen HW-Fehler bei der Regalförderzeugsteuerung

Sämtliche Anschlüsse des technischen Prozesses an die Ein-, Ausgabeeinheiten des Prozessrechners wurden über Reed-Relais ausgeführt.

4) Beschreibung des Programmsystems

Das PEARL-Programmsystem zur Automatisierung des Modellprozesses "Stückgutlager" hat eine statische Länge von ca. 60 KByte und ist in 4 Moduln mit insgesamt 8 Tasks gegliedert. Das Programmsystem verfügt über 5 global und 30 auf Taskebene vereinbarten Prozeduren.

Im Folgenden ist eine Übersicht über die Aufgabenverteilung auf die 8 Tasks aufgelistet. Der kontrollflußmäßige Zusammenhang der Tasks innerhalb des Programmsystems ist in vereinfachter Form in Bild 7 angegeben.

Aus Gründen der Übersichtlichkeit wurde auf eine Darstellung der Datenverbindungen verzichtet.

Task DIALOG:	Ermöglicht dem Bedienpersonal durch Anwahl von Bedienfunktionen (insgesamt 16) den Eingriff in das Prozeßgeschehen.
Task INITIALISIERUNG:	Stellt einen definierten Ausgangszustand im Warenverteilsystem und im Hochregallager her.
Task EINPLANUNG:	Übernimmt die Eingabe von Lieferaufträgen und vergibt dann aufgrund der augenblicklichen

- Lieferauftragssituation
- Warenverteilsystembelegung
- Hochregallagerbelegung

die Bestimmungsorte für Förderzeuge und Paletten

Task AUFTRAGSAUSFÜHRUNG: Ermittelt aufgrund der Daten aus Task EINPLANUNG und der aktuellen Belegung im Warenverteilsystem und im Hochregallager die Streckenführung der Förderzeuge sowie die Ein-, Auslagerungsposition von Paletten.

Task WVS-STEUERUNG: Übernimmt die Stellsignalausgabe zur Förderzeugsteuerung und überwacht gleichzeitig die angestossenen Förderzeugfahrten.

Wird zum ersten Mal über Task DIALOG gestartet, danach im Einsekundenzyklus aktiviert.

Ermittelt den jeweils aktuellen Belegungszustand im Warenverteilsystem.

Task-HRL-STEUERUNG: Wird von Task WHS-STEUERUNG aktiviert, sobald sich ein Förderzeug auf den EA-Platz zubewegt. Übernimmt dann die Stellsignalausgabe und eine Zeitüberwachung der Regalförderzeugsteuerung. Bei Überschreiten einer Zeitgrenze für ein Lagerspiel wird HRL-FEHLER aktiviert.

Ermittelt den jeweils aktuellen Belegungszustand im Hochregallager.

Task HRL-FEHLER: Meldet die aufgetretenen Fehler bei der Regalförderzeugsteuerung.

Kann zusätzlich durch HRL-FEHLERINTERRUPT aktiviert werden.

Task ABSCHALTEN: Steuert das Warenverteilsystem und das Hochregallager in einen definierten Ruhezustand.

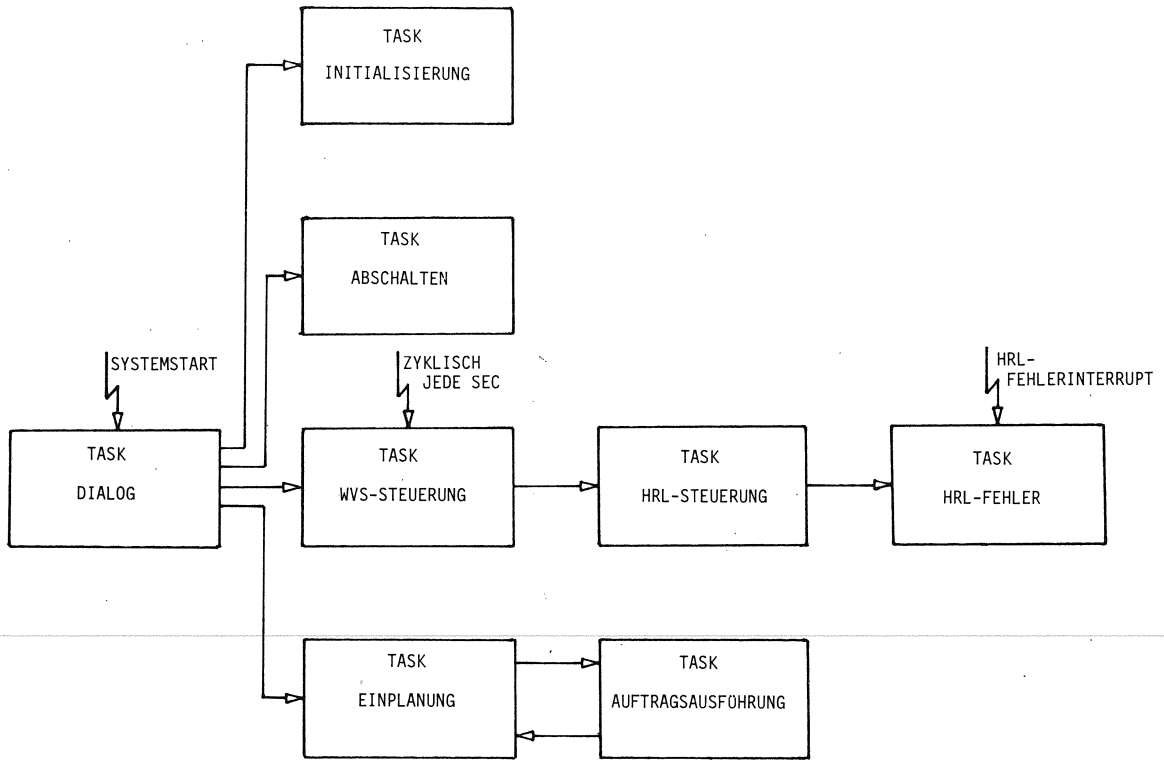


Bild 7: Programmsystem zur Automatisierung des "Stückgutlagers"

6. Ausblick

Seit einigen Jahren beschäftigen wir uns intensiv mit dem Themenkreis "Entwurfs- und Spezifikationssysteme für die Prozeßautomatisierung". Diese Problematik steht in engem Zusammenhang mit dem Einsatz einer höheren Programmiersprache für die Realzeitdatenverarbeitung. Das am Institut entwickelte entwurfsunterstützende und Prozeßorientierte Spezifikationssystem ist so konzipiert, daß es möglich sein wird, aus einer EPOS-Spezifikation automatisch ein PEARL-Programm zu generieren [4] .

7. Literaturverzeichnis

[1] L a u b e r, R.: Prozeßautomatisierung I. Springer Verlag, Berlin, Heidelberg, New York 1976.

[2] L a u b e r, R.: Modelle zur Beschreibung des Entwurfs von Prozeßautomatisierungssystemen. Regelungstechnik 27 (1979) Heft 12.

[3] B i e w a l d, J., Göhner, P., Lauber, R. und Schelling, H.: Das Software-Werkzeug EPOS zur Unterstützung der Ingenieurtätigkeiten beim Entwurf und bei der Wartung von Prozeßautomatisierungssystemen. Regelungstechnik 28 (1980) Heft 1.

[4] J o h o, E. und Jovalekić, S.: Rechnergestützte Umsetzung von EPOS-Spezifikationen in PEARL-Programme. Fachtagung Prozeßrechner 1981, München, März 1981.

[5] H o m m e l, G.: Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage. KfK Karlsruhe, PDV-Bericht 186, August 1980.

[6] W e r u m, W. und Windauer, H.: PEARL Vieweg Verlag, 1978.

[7] K a p p a t s c h, A., Mittendorf, H. und Rieder, P.: PEARL. Oldenbourg Verlag, München, Wien 1979.

Anschrift der Autoren

E. Joho, C. Riewerts
IRP, Universität Stuttgart
Seidenstrasse 36

7000 Stuttgart - 1

Tel. 0711/ 2073 - 2542

Kurzmitteilungen

Das 1 000 000 000 000 $ Mißverständnis

Ein Beispiel, wie durch die Anwendung einer unsicheren Sprache, bei der Programmierung des zu steuernden Prozesses ein großer Schaden entstand, wurde in den ACM SIGSOFT, Software Engineering Notes, Vol 4 No 4 Page 6 wiedergegeben.
Die Programmierung der Steuerung der ersten amerikanischen Venus-Sonde erfolgte in FORTRAN. Diese, aus heutiger Sicht veraltete Programmiersprache besitzt die Eigenschaft, daß die fehlerhafte Vertauschung eines einzigen Zeichens, eine formell richtige, neue Anweisung ergibt.

Der in den Software Engineering Notes geschilderte Fall sieht wie folgt aus:
Der Verlust der Sonde war zurückzuführen auf das versehentliche Austauschen einer Anweisung der Form

```
      DO  3  I = 1.3
```

durch eine Anweisung der Form

```
      DO  3  I = 1,3
```

was dazu führte, daß die Venus-Sonde Millionen von Kilometern am Ziel vorbeiflog.

Was nützt ausgefeiltes Programm, Design Methodik und 'Software Engineering' und das Bemühen um zuverlässige Hardware, so fragt J. Horning , wenn ein einziger Schreibfehler, der von Compiler nicht erkannt werden kann, die korrekte Funktion eines Programms zerstört.

Angesichts der Tatsache, daß heute bewährte PEARL-Implementationen zur Verfügung stehen, sollte es sich ein Anwender eigentlich nicht mehr leisten können, wichtige Prozesse mit Rechnern zu automatisieren, die in FORTRAN programmiert sind.

Neue Mitglieder

71. Dipl.-Wi.-Ing. Karl-Johannes Voßloh
Flemingweg 1
4300 Essen 1

72. Dr. Sandro Bologna
CNEN CSN-Casaccia
S.P. Anguillarese km. 1.300
00060 ROMA - Italien

73. Prof. Dr. Ing. K.W. Pleßmann
Lehrgebiet f. Verfahren der Prozeßdatenverarbeitung
u. Prozeßführung der RWTH Aachen
Postfach, 5100 Aachen

74. Dipl.-Ing. Anselm Triestram
Grenzstraße 89
4630 Bochum 6

75. Gerd Schairer
Untere Aue 2
7150 Backnang

PEARL-Tagung '81

am 7.– 8.Dezember 1981
in Düsseldorf
VDI – Haus
Graf-Recke-Str. 84

Motto:
PEARL–Praxis und Zukunft

PROGRAMMAUSSCHUSS

Dr. Blask	BMFT
Prof. Dr. F. Hofmann	Universität Erlangen
Dipl.-Math. G. Koch	BBC
Dr. H.-P. Leppin	Krupp Atlas Elektr.
Dr. U. Mayr	OBAG
Dr. Munk	BMVg
Prof. Dr. R. Lauber	Universität Stuttgart
Dr. H. Mittendorf	Siemens AG
Prof. Dr. H.-J. Schneider	Universität Erlangen
Dr. K. Rebensburg	Techn. Universität Berlin
Prof. Dr. H. Rzehak	Hochschule der BW München

WISSENSCHAFTLICHE TAGUNGSLEITUNG

Prof. Dr. H. Rzehak
Hochschule der Bundeswehr
Fachbereich Informatik
Werner-Heisenberg-Weg 39
8014 Neubiberg
Tel.: 089/ 6004 - 3397

TAGUNGSBEITRAG

Voller Beitrag	190,00 DM
GI und GMR - Mitglieder	150,00 DM
PEARL-Vereins-Mitglieder	90,00 DM
Studenten	20,00 DM

ZIMMERNACHWEIS

Verkehrsverein Düsseldorf
Konrad-Adenauer-Platz 12
D-4000 Düsseldorf
Tel.: 0211/ 350 505

ANMELDUNGEN

PEARL - Tagung '81
Tagungsleitung
Institut für Regelungstechnik
und Prozeßautomatisierung
Seidenstraße 36
D-7000 Stuttgart 1

WICHTIG

Aus Platzgründen ist die Teilnehmerzahl auf 150 begrenzt. Die Berücksichtigung erfolgt in der Reihenfolge der Anmeldung.

Tagungsprogramm:

MONTAG, 7.12.1981

11^{00} BEGRÜSSUNG
R. Lauber, H. Rzehak

11^{15} QUELLBEZOGENE TESTSYSTEME
Sitzungsleitung: R. Lauber

K. Mangold, AEG Konstanz:
Ein quellbezogenes Testsystem für PEARL auf einem Prozeßrechner

P. Brunner, K. Meffert, H. Windauer, Werum Lüneburg: PEARL-Testsysteme

P. Heine, IITB Karlruhe:
PETSY - ein PEARL Test System

13^{15} MITTAGSPAUSE

14^{30} INTEGRATIONSPROBLEME
Sitzungsleitung: H. Steusloff

K. Lucas, GPP München: Konsistenzprüfung beim Integrieren von Basis-PEARL-Moduln

P. Puhr-Westerheide, GRS München:
Die statische Analyse von PEARL Moduln mit dem PEARL-Analysator

B. Boysen, Krupp-Atlas-Elektronik:
PEARL auf einer 16-Bit Rechenanlage mit Mehrbenutzersystem

16^{30} KAFFEEPAUSE

16^{45} MENSCH-MASCHINE SCHNITTSTELLE
Sitzungsleitung: U. Mayr

E. Welfonder, Th. Röhrich, Universität Stuttgart: Meldungs- und Protokolliersystem der PEARL Anwender Programmbibliothek

H. Weber, Battelle-Inst. Frankfurt:
Ein modulares, universelles Dialogsystem programmiert in PEARL

18^{00} ENDE

20^{00} MITGLIEDERVERSAMMLUNG DES PEARL-VEREINS

DIENSTAG, 8.12.1981

9^{00} ANWENDUNGSSYSTEME
Sitzungsleitung: H.-P. Leppin

H. Steusloff, IITB Karlsruhe:
Einsatz von PEARL auf verteilten Rechnersystemen in der Stahlindustrie

H. Albrecht, Heusch-Boesefeldt, Aachen:
Anwendung von PEARL in der Verkehrstechnik

W. Gerth, Universität Hannover:
Basis-PEARL auf dem MC 68000 Design-Module

11^{00} KAFFEEPAUSE

11^{15} SOFTWARE-ENGINEERING
Sitzungsleitung: H.-J. Schneider

L. Frevert, Fachhochschule Bielefeld:
Top-down-Spezifikation-Entwicklung und Dokumentation von PEARL Moduln

V. Scheub, Universität Stuttgart:
Generierung von PEARL Programmen aus einer EPOS-Spezifikation

12^{30} MITTAGSPAUSE

14^{00} SPRACHNORMUNG UND SPRACHKONZEPTE
Sitzungsleitung: H. Rzehak

S. Heilbrunner, HSBw München:
Nutzen einer Sprachnorm

B. Eichenauer, GPP München:
Sprachkonzepte in PEARL und Ada

15^{30} ABSCHLUSSDISKUSSION

16^{00} ENDE