PEARL Rundschau

Inhalt

Dezember 1982 Band 3 Nr. 5

PEARL Rundschau

Dezember 1982
Band 3
Nr. 5

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 in 7000 Stuttgart 1, Seidenstraße 36

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.

Vorwort des Vorstands des PEARL-Vereins

Dieses ist das letzte Heft der PEARL-Rundschau als Publikationsorgan des PEARL-Vereins - und gleichzeitig der Beginn einer noch wirkungsvolleren und breiteren Veröffentlichungstätigkeit im Rahmen der Fachzeitschrift "Regelungstechnische Praxis".

Der Abschied von der PEARL-Rundschau mag für viele von Ihnen überraschend kommen.

Überraschend, weil es keineswegs an Material über PEARL zur Veröffentlichung mangelt. Diese Tagung beweist es wieder einmal. Es scheint inzwischen fast so zu sein, daß PEARL diejenige Programmiersprache ist, die die modernsten Softwarehilfsmittel besitzt: Erstellungshilfen verschiedener Komfortgrade, leicht an andere Rechner adaptierbare Compiler, Test- und Verfikationshilfsmittel, Kommunikationsschnittstellen, etc. Auch als Programmiersprache für die Erstellung von Anwenderpaketen - eines der ursprünglichen Entwicklungsziele - wird PEARL jetzt zunehmend eingesetzt. Das heißt, auch diese PEARL-Tagung verspricht fachlich wieder ein Erfolg zu werden und wir möchten dem Programmausschuß und insbesondere Herrn Prof. Hommel zu diesem interessanten Programm beglückwünschen und ihnen für diesen Einsatz danken.

Auch die PEARL-Rundschau als solche war nicht gerade erfolglos. Nachdem sie es in den drei Jahren ihres Bestehens auf 15 Hefte mit zusammen über 750 Seiten und 100 Fachartikeln gebracht hat. Dies ist der besondere Verdienst der Schriftleitung insbesondere von Herrn Dr. Elzer, der sich mit sehr viel persönlichem Engagement für die PEARL-Rundschau eingesetzt hat. Das Redaktionsteam hat es verstanden, die Autoren zu gewinnen, die ja ihre Beiträge ohne Honorar zur Verfügung gestellt haben und sich oft der zusätzlichen Mühe des speziellen Schreibformats und teilweiser Überarbeitung ihrer Arbeiten unterzogen haben. Allen Mitwirkenden, der Schriftleitung, dem Redaktionsteam und den Autoren gebührt deshalb unser besonderer Dank.

Nicht zuletzt soll den Mitarbeitern in der PEARL-Geschäftsstelle und beim VDI-Verlag für ihren Fleiß und ihre Beharrlichkeit bei der Überwindung der tausend kleinen Probleme der Produktion gedankt werden.

Trotz aller Erfolge: Die Auflage der PEARL-Rund schau kam über einige hundert Exemplare nicht hinaus. Damit entstanden einerseits Finanzierungsplobleme, da es nach Wegfall der Förderung schwierig wurde, die Herstellungskosten bei dieser geringen Auflage aufzubringen. Andererseits erreichte die PEARL-Rundschau nur die unmittelbaren PEARL-Interessenten, und es gelang kaum, neue Leserschichten (und damit Anwenderschichten für PEARL) zu gewinnen.

Um daher den Aktionsradius der PEARL-Veröffentlichungen wesentlich zu erweitern, wurde eine Zusammenarbeit mit dem Oldenbourg-Verlag, der eine Reihe von renommierten Fachzeitschriften herausgibt, angestrebt. In den Verhandlungen mit dem Oldenbourg-Verlag ist es nun gelungen, zu einer recht günstigen Vereinbarung zu kommen:

* In die Fachzeitschrift "Regelungstechnische Praxis rtp" (Auflage ca. 9000 Exemplare) wird ab 1.1.1983 eine regelmäßige "PEARL-Spalte" aufgenommen, deren Inhalt vom PEARL-Verein verantwortlich gestaltet wird.
* Gleichzeitig wird ab 1.1.1983 ein Themenbereich "Softwaretechnik" eingeführt, der in jeder Nummer der Zeitschrift "rtp" etwa 14 Druckseiten umfaßt. Für diesen Themenbereich wurde ein eigenes Redaktionsteam gegründet, dessen Mitglieder vom Oldenbourg-Verlag in Abstimmung mit dem PEARL-Verein benannt wurden. Gleichzeitig wurde der Vorsitzende des PEARL-Vereins als Mitglied in den Beirat der Zeitschrift aufgenommen.
* Für die Mitglieder des PEARL-Vereins wurden Sonderkonditionen für den Bezug der Zeitschrift "rtp" vereinbart.

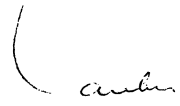
Die Fachzeitschrift "rtp" wird mit den Informationen der PEARL-Spalte und dem Themenbereich "Softwaretechnik" die erste Zeitschrift Deutschlands sein, die sich praxisbezogen der Programmierung in der Automatisierungstechnik widmet.

Man kann also sagen, daß PEARL auch hier wieder bahnbrechend gewirkt hat.

Für den PEARL-Verein, dessen satzungsgemäßes Ziel ja "die Verbreitung von PEARL" ist, bietet diese neue Form der Publikationsmöglichkeit die Chance, den Kreis der anzusprechenden Fachleute mit einem Schlag mehr als zu verzehnfachen!

Wir möchten dem Oldenbourg-Verlag zu seinem unternehmerischen Mut beglückwünschen und der neu gestalteten Fachzeitschrift "rtp" als Nachfolge-Publikationsorgan der PEARL-Rundschau alles Gute - vor allem viele Leser und Abonnenten - wünschen.

Für den Vorstand des PEARL-Vereins



R. Lauber

Vorwort der Tagungsleitung

"PEARL in der Praxis" ist das Motto der 3. PEARL-Tagung, die mit dem dreijährigen Bestehen des PEARL-Vereins zusammenfällt.

Wie bei den letzten Tagungen liegt der Schwerpunkt der Beiträge auf Erfahrungsberichten, die aus verschiedenen Auwendungsbereichen stammen. Diese Berichte sind besonders wertvoll, weil sie Neuanwendern von PEARL gute Hinweise auf zu erwartende Schwierigkeiten geben, aber auch aufzeigen, daß es heute kein Risiko mehr ist, ein Projekt mit PEARL durchzuführen.

Ein weiterer Hinweis auf den Reifegrad von PEARL ist ein eigener Sitzungsabschnitt über Standardbausteine. Die beim Entwurf von PEARL in den Vordergrund gestellte Forderung nach Portabilität von PEARL-Programmen, scheint nun doch langsam Früchte zu tragen.

Ein großer Teil von Anwendern beschäftigt sich derzeit mit der Gestaltung von Mensch-Maschine-Schnittstellen. Ein wichtiges Ergebnis dieser Tagung könnte es sein, daß die bisher unabhängig voneinander arbeitenden Anwender Erfahrungen in diesem Bereich austauschen und vielleicht zu Absprachen im Vorfeld der Standardisierung kommen.

In einem Sitzungsabschnitt über Mikrorechner und verteilte Systeme werden mehr Zukunftsaspekte angesprochen. Jedem Anwender ist klar, daß PEARL in der Zukunft nur dann überleben kann, wenn auch Compiler für Mikrorechnersysteme verfügbar sind. Neben den schon früher vorgestellten Arbeiten wird auch bei dieser Tagung wieder gezeigt, daß solche Entwicklungen laufen. Ein gutes Zeichen für die Lebendigkeit einer Programmiersprache ist die Diskussion über Spracherweiterungen und -ergänzungen, die hier über Sprachmittel zur Programmierung verteilter Systeme geführt werden soll. Nachdem zu dieser Problematik bei verschiedenen Anwendern Überlegungen und sogar Implementierungen vorliegen, dürfte es an der Zeit sein, diese Anstrengungen zu koordinieren, um zu standardisierbaren Lösungen zu kommen.

Zum Schluß möchte ich nicht versäumen, dem Programmausschuß für die Auswahl der Beiträge und die Gestaltung des Tagungsprogramms zu danken.

Im Programmausschuß wirken mit:

Dipl.-Ing. H. Albrecht	Heusch-Boesefeld, Aachen
Prof. Dr. L. Frevert	FH Bielefeld
Prof. Dr. F. Hofmann	Universität Erlangen
Dr. T. Martin	Kernforschungszentrum Karlsruhe
Dr. U. Mayr	OBAG Regensburg
Prof. Dr. H. Rzehak	Hochschule der BW München
Prof. Dr. H.J. Schneider	Universität Erlangen
Prof. Dr. A. Storr	Universität Stuttgart

Nicht zuletzt möchte ich aber Herrn Dipl.-Ing. V. Scheub danken, ohne dessen Mithilfe, insbesondere auch bei der Organisation, die Tagung nicht zustandegekommen wäre.

Hommel

G. Hommel

Anwendung von PEARL in der zentralen Leittechnik

W. Tiedmers, Bremen

0 EINLEITUNG

Bedingt durch Projekte, die von der Aufgabenstellung eine höhere Realzeitsprache erforderlich macht, hat Krupp Atlas Elektronik Bremen (KAE) für die gesamte Rechnerlinie EPR 1000 bereits seit langem PEARL als Standardprogrammsprache für Anwendersoftware eingesetzt. In den leittechnischen Systemen, die in den letzten Jahren geliefert wurden, ist die oberhalb des Betriebssystems angesiedelte Grundsoftware für die ZLT – AWL 1300 genannt – bis auf das Datenverwaltungssystem in PEARL geschrieben. An dem im folgenden beschriebenen und in diesem Jahr in Betrieb genommenen Projekt sollen sowohl einige Erfahrungen der Anwendungsprogrammierer und von KAE eingebrachte betriebssystemnahe Möglichkeiten als einige realisierte Spracherweiterungen in Richtung auf FULL-PEARL beschrieben werden.

1 AUFGABENSTELLUNG

Bei einem Versorgungsunternehmen wurde in den Prozeßbereichen GAS und WASSER die vorhandene Netzwarte ausgebaut.

Unterstützt durch ein zentrales Rechnersystem waren folgende Aufgaben zu erfüllen:

- Echtzeiterfassung der angebotenen Prozeßdaten
- Ausgabe manueller Befehle
- Protokollierung des Prozeßgeschehens
- Prozeßdarstellung durch ein Farbsichtsystem
- Datenaufbereitung für Bilanzierungen und Meßwertverläufe in verschiedenen Zeitrastern
- Verwaltung und Darstellung von Listen.

Neben diesen genannten Aufgaben mußte eine Programm- und Steuerlistenerstellung im Hintergrundbetrieb möglich sein. Programmiersprache für die Anwendersoftware war gemäß der VDEW Empfehlung über „Netzleitsysteme in EVU's" PEARL.

1.1 Schnittstelle zum Prozeß

Die Schnittstelle der zentralen Rechnersysteme (EPR 1300, EPR 1100) zum Prozeß wird gebildet durch:

- 304 Analoge Eingänge 0 - 20 mA
- 800 Digitale Eingänge [Ereignisse]
- 80 Digitale Eingänge [Zählimpulse]
- Diverse Meßwerte + Meldungen über eine Fernwirkanlage und
- 416 Digitale Ausgänge.

Die angeschlossenen 304 analogen Meßwerte werden im Zyklus von 10 s erfaßt und weiterverarbeitet. Zusätzlich hierzu werden 20 aus der Gesamtmenge dynamisch frei wählbare Meßwerte im Zyklus von 1 s erfaßt und verarbeitet, um bei Steuerungsvorgängen eine schnelle Rückkopplung zum Bedienenden zu gewährleisten.

Änderungen der an die digitalen Eingänge angeschlossenen Meldepunkte werden direkt per Interrupt erfaßt und weiterverarbeitet.

Für die Erfassung der Zählimpulse wird ein eigener festprogrammierter Rechner (EPR 1100) mit 160 Zähleingängen vorgesehen. Dieser Rechner wird über eine USV-Anlage betrieben, damit bei einem evtl. Netzspannungsausfall die weitere Zählwerterfassung gewährleistet werden kann.

Über die Fernwirkschnittstelle werden Meßwerte und Meldungen erfaßt. Zusätzlich hierzu werden die über die Fernwirkschnittstelle ausgehenden Befehle vom Rechner erfaßt und kontrolliert.

Die Steuerung des Prozessors erfolgt über 416 digitale Ausgaben, die jeweils direkt per Softwarerangierung auf insgesamt 3 × 3 Taster (AUF/ZU/HALT) reagieren.

1.2 Schnittstelle zum Bedienpersonal

Die Schnittstelle zum Bedienpersonal wird durch insgesamt drei Bedienplätze mit

- Farbmonitoren mit Lichtstift und Cursorführung
- Funktionstastaturen
- Daten-Sichtgeräten und
- Druckern

gebildet.

Die Farbmonitore dienen zur anschaulichen und übersichtlichen Darstellung des Prozesses, Lichtstift und Cursorsteuerung zur Betriebsmittelanwahl bei der Fernsteuerung des Prozesses, die Funktionstastaturen zur Steuerung des Prozesses und zur direkten Bildanwahl. Die Daten-Sichtgeräte sind zur Dialogführung mit dem Rechnersystem eingesetzt. Die Drucker übernehmen die Ausgabe von Ereignismeldungen und Übersichtsprotokollen.

Die Software der beschriebenen Bedienerfunktionen wurde in Teilen in Zusammenarbeit mit dem Auftraggeber erstellt, um diesem sehr genaue Systemkenntnisse zu vermitteln. Mit diesen Kenntnissen können so jederzeit auch größere Änderungen und Erweiterungen des Systems durch den Anwender durchgeführt werden.

2 REALISIERUNG

2.1 Hardwarekonfiguration

Für die Lösung der Aufgabenstellung wurde ein Rechnersystem bestehend aus je einem

- Erfassungsrechner EPR 1300
- Zählwertrechner EPR 1100 und
- Verarbeitungsrechner EPR 1300

eingesetzt (s. Abb. 3-1). Diese Aufteilung wurde u.a. aus folgenden Gründen gewählt:

1) In der Zukunft ist eine 2. Ausbaustufe geplant, so daß bereits jetzt auf Grund der zu erwartenden Meßwert- und Meldungsmengen sowie Verarbeitungsprozeduren eine Trennung von Erfassungs- und Verarbeitungsrechner sinnvoll ist.

2) Der Zählwertrechner ist durch eine getrennte Spannungsversorgung [Notstrom] in der Lage, bei einem Spannungsausfall als autarkes System die Zählwertüberwachung fortzusetzen.
Auch bei einem Systemstop von Erfassungs- und Verarbeitungsrechner gehen keine Zählwertdaten verloren!

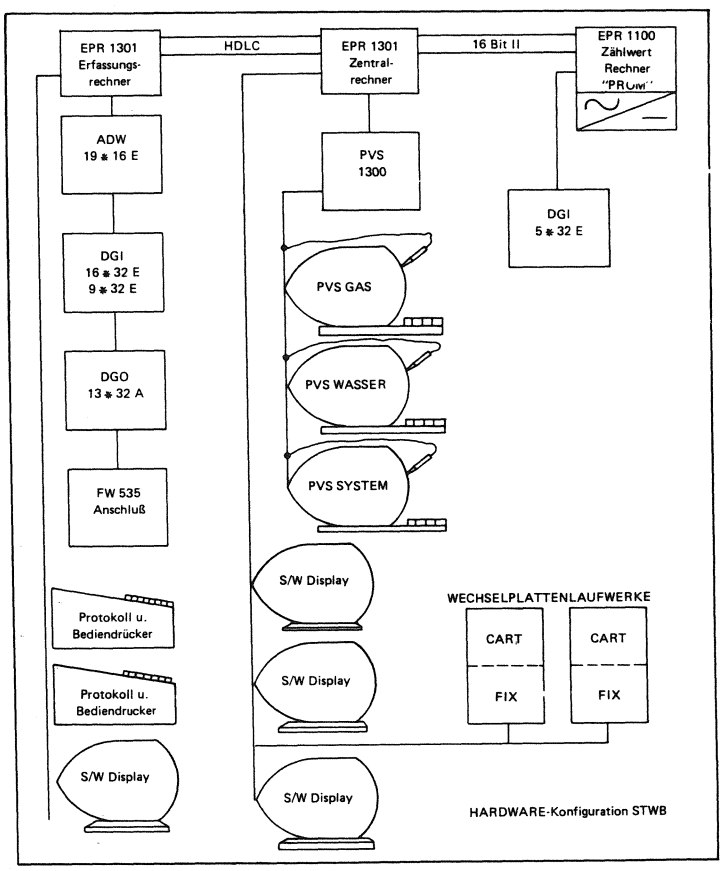


Abb. 2-1: Hardware-Konfiguration

2.2 Systemfunktionen

Die Funktionen des realisierten Systems lassen sich in 6 Gruppen einteilen, siehe auch Abb. 2-2.

- Datenverwaltungssystem (DVS)
- Prozeß-Ein-/Ausgabe-System (PEAS)
- Ereignisverarbeitung (EVA)
- Zentrale Bedienerschnittstelle (ZBS)
- Prozeßvideo-Anwenderpaket (PAP PVS)
- Zählwertrechnerankopplung (ZWR)

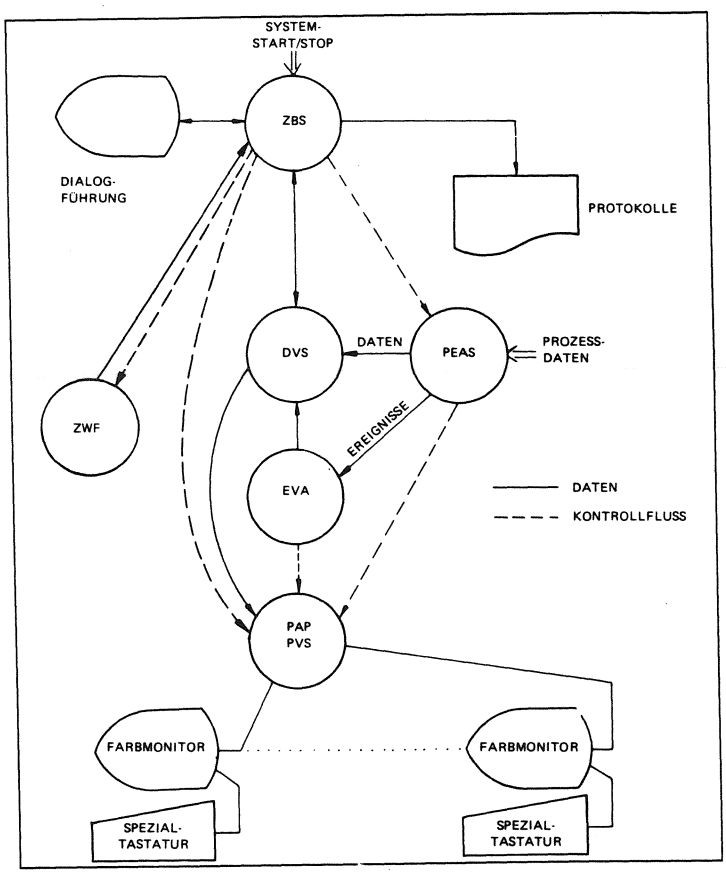


Abb. 2-2: Allgemeine Systemfunktionen mit Prozessvideosystem

2.2.1 DATENVERWALTUNGSSYSTEM (DVS)

Zur Verwaltung der anfallenden Datenmengen dient ein System, das eine Reihe von Funktionen bereitstellt, die speziell zur Verwaltung und Archivierung umfangreicher Datenmengen dienen. Damit kann in einfacher Weise ein gut gegliedertes Datenmodul realisiert werden, das sich leicht erweitern bzw. warten läßt. Zur Einrichtung bzw. Modifikation des Datenmodells sind folgende Dialogfunktionen vorhanden:

- Definition von Datenstrukturen
- Eingabe von Daten
- Modifikation von Daten
- Listen der Datenstrukturen
- Listen der Daten
- Füllen von Datenbereichen mit Konstanten
- Retten und Wiederherstellen der Datenstrukturen
- Retten und Wiederherstellen der Daten

Der Zugriff auf das DVS ist so schnell, daß die Funktion der Prozeß-Ein-/Ausgabe nicht beeinträchtigt wird. Residente (im Halbleiterspeicher gehaltene) Daten und nichtresidente Daten (auf dem Massenspeicher) sind über eine einheitliche Schnittstelle zugreifbar.
Des weiteren verwaltet das DVS auch fifo-organisierte Dateien (fifo = first in – first out, Warteschlangen).

2.2.2 PROZESS-EIN-/AUSGABESYSTEM (PEAS)

Das Prozeß-Ein-/Ausgabe-System übernimmt die Aufgaben

- Decodierung von Telegrammen der Fernwirkanlage
- Unterscheiden von Meldungs-, Meßwert- und Befehlstelegrammen
- Einlesen der analogen Meßpunkte
- Skalierung von Meßwerten (linear oder proportional) und Grenzwertüberwachung

- Einlesen digitaler Eingänge
- Ausgabe digitaler Signale und Befehle
- Melden von Ereignissen an die Ereignisverarbeitung

Wenn eine einlaufende Meldung als Ereignis erkannt wird (z.B. Schalteränderungsmeldung), wird sie (ebenso wie z.B. Grenzwertverletzungen) als Ereignis an die Ereignisverarbeitung weitergeleitet. Dabei wird dem Ereignis die momentane Prozeßrechnerinterne Uhrzeit zugeteilt.

Die Abbildungen 2-3 und 2-4 geben eine Funktionsübersicht über PEAS und die Befehlskontrolle für Fernwirk-Befehle.

Abb. 2-3: Funktionsübersicht PEAS

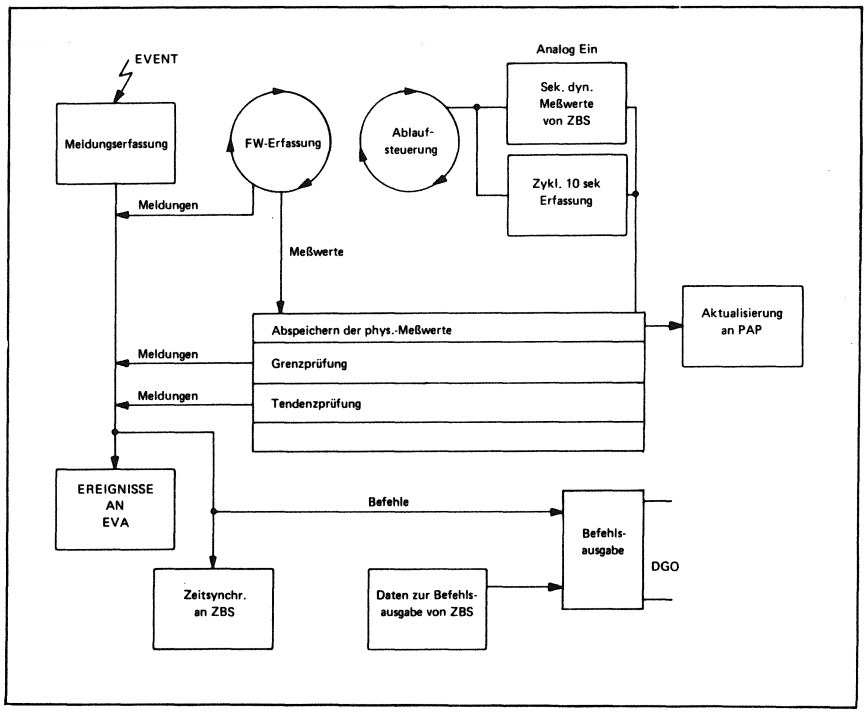


Abb. 2-4: Befehlskontrolle für Fernwirkbefehle

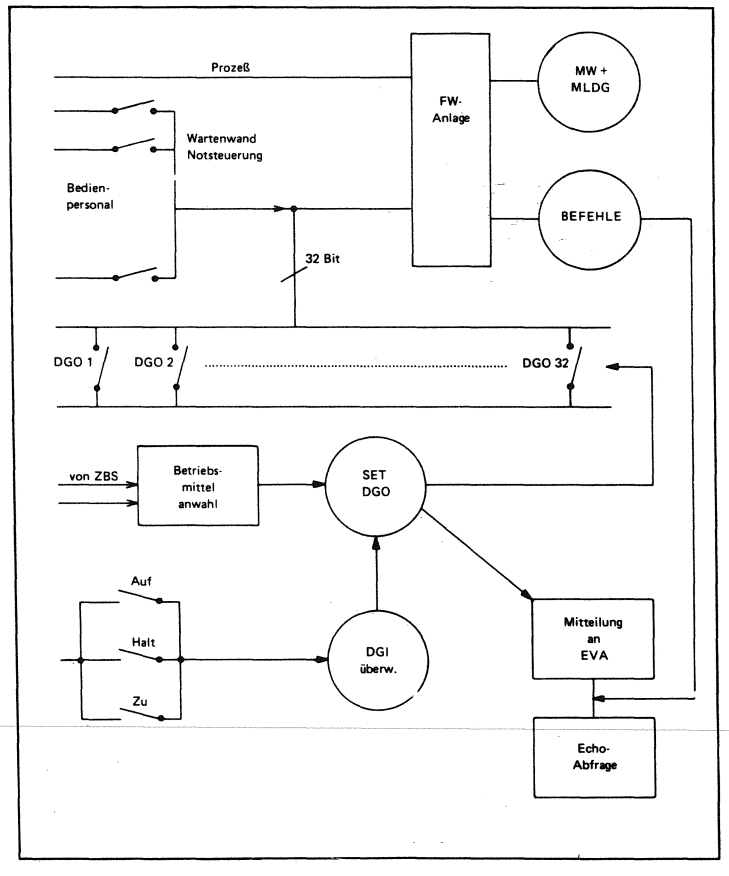
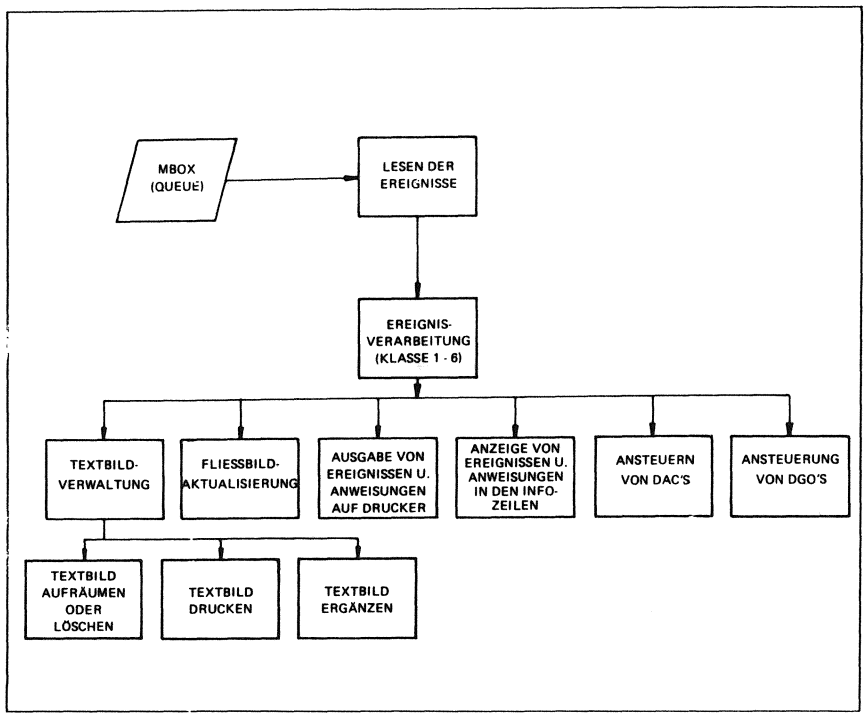


Abb. 2-5: Funktionsübersicht EVA (Ereignis-Verarbeitung)



2.2.3 EREIGNISVERARBEITUNG (EVA)

Alle im Gesamtsystem auftretenden Ereignisse werden an EVA gemeldet. Insgesamt stehen hierfür 6 Ereignisklassen zur Verfügung (Abb. 2-5).

Beispiel:

Klasse	Ereignisart
1	Steuerkommandos an EVA
2	Standard Software-Meldungen
3	Tastenfunktionen
4	Meßstellen-bezogene-Ereignisse
5	Bilanzwert-bezogene-Ereignisse
6	Binär-Ereignisse

Innerhalb jeder Klasse ist eine Typunterscheidung der Ereignisse möglich.

Beispiel:

Klasse 2 Typ
- 1 = Ablauffehler
- 2 = IOSTAT/SIOSTAT Fehler
- 4 = Arithmetik-Fehler
- 5 = Bolt-Fehler
- 6 = Geräte-Fehler
- 7 = DVS-Fehler

oder

Klasse 6 Typ
- 1 = gekommen
- 2 = gegangen
- 3 = Pseudomeldung – Rückmeldezeit abgelaufen
- 4 = Pseudomeldung: DGO gesetzt
- 5 = Pseudomeldung: FW-DGO gesetzt
- 6 = Quittierung
- 7 = Befehlsecho der FW-Anlage
- 8 = Generalabfrage

Jeder Kombination Klasse/Typ ist eine bestimmte Anzahl verschiedener Reaktionen des Rechner zugeordnet, z.B.:

- Ausdruck auf dem Meldedrucker
- Ausgabe im Ereignistextbild
- Blinken im PVS-Bild

2.2.4 ZENTRALE BEDIENERSCHNITTSTELLE (ZBS)

Die Zentrale Bedienerschnittstelle besteht aus den Teilen

- Dialog G/W/S
- Spooler der Ausgabegeräte und der
- Uhrzeitsynchronisation.

(s. Abb. 2-6).

Dialog G/W/S

Über ein MENUE-Verfahren werden dem Bedienpersonal verschiedene Möglichkeiten angeboten, Aktionen und Daten vom Gesamtsystem abzurufen. Der Dialog kann abgewählt werden, um an seiner Stelle Ereignisprotokolle auf den Bedienerdisplays anzuzeigen.

Spooler

Die Ausgabegeräte DIS1, DIS2, DIS3, TWRW, TWRG sowie ein Drucker (MPR) werden von je einer Spool-Task mit Daten aus Spooler-Queues vom Datenverwaltungssystem versorgt.
Für jedes Gerät existiert eine eigene DVS-Queue.
Per Ersatzgerätestrategie werden defekte Ausgabegeräte im Fehlerfall durch funktionstüchtige Ausgabegeräte ersetzt.

Uhrzeitsynchronisation

Da bei dieser Anwendung der Minuten- und Stundenimpuls über einen DCF 77 Empfänger angeboten wird, findet die Synchronisierung der programminternen Zeit [nicht der Systemzeit des Rechners!] über Digitaleingänge des Erfassungsrechners statt. Der Erfassungsrechner aktiviert die Synchronisierungsaktionen im ZBS-Teil des Verarbeitungsrechners.
Dort wird sekündlich die „UHR"-Task aktiviert. Sie übernimmt die Zeitsteuerung sekundengenau. [Eine höhere Auflösung wäre denkbar, ist jedoch im vorliegenden Anwendungsfall nicht notwendig.]
Bei Ausfall von Minuten- oder Stundenimpuls kann sie autark die Zeitsteuerung übernehmen. Sind Minuten- oder Stundenimpuls wieder da, wird mit dem jeweils nächsten Impuls wieder synchronisiert.
Die Uhrzeittask aktiviert die Task Minutenverarbeitung sowie die Task Stundenverarbeitung.

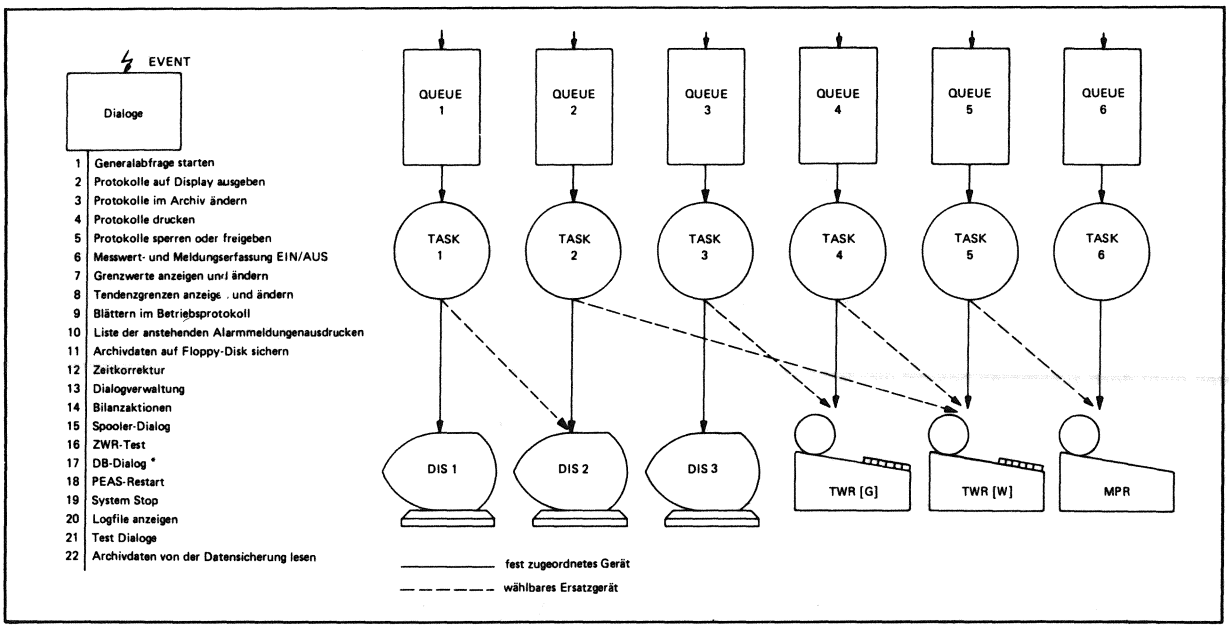


Abb. 2-6: Funktionsübersicht ZBS (Zentrale Bediener-Schnittstelle

2.2.5 PVS-ANWENDERPAKET (PAP)

Das PVS-Anwenderpaket übernimmt die Funktionen

- Dynamische Zuordnung Bediener/Monitor
- Verwaltung der Monitore
- Abbildung gespeicherter Daten auf Monitor unter Einbeziehung aller Darstellungsmöglichkeiten des PVS
- Autonome Bearbeitung von Standard-Tastatur-Funktionen
- Durchreichen von Lichtstift- und Tastaturbefehlen an andere Teilsysteme.

Die Ankopplung des PAP an andere Systemfunktionsbereiche (USER) ist in Abb. 2-7 dargestellt.

2.2.6 ZÄHLWERTRECHNERANKOPPLUNG (ZWR)

Die Zählwertrechnerankopplung beinhaltet die folgenden Funktionen:

- Funktionsüberwachung des ZW-Rechners
- Datenverkehr mit dem Zählwertrechner
- Fehlerbehandlung und Test

Das Programm im Zählwertrechner startet unabhängig vom Leitrechner automatisch beim Einschalten des Zählwertrechners und ist damit sofort in der Lage, Zählwerte zu erfassen und den Dialog mit dem Leitrechner zu führen.

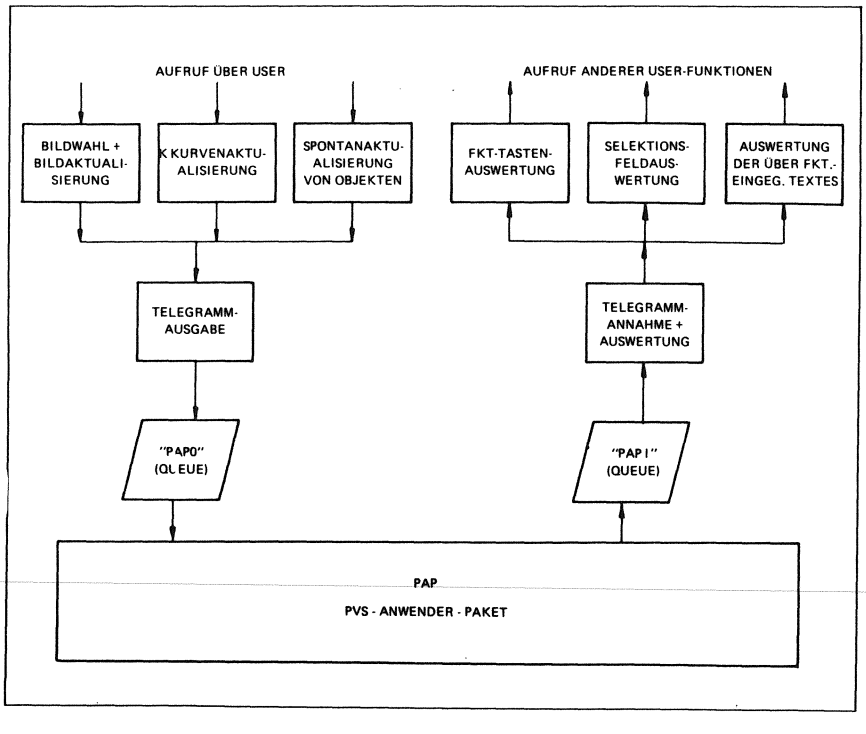


Abb. 2-7: Funktionsübersicht Useraufrufe PAP

3 SCHWIERIGKEITEN UND LÖSUNGSWEGE

Bedingt durch die Leistungsfähigkeit des EPR 1300 und seines Realzeitbetriebssystems MOS wurde eine Vielzahl von Funktionen auf **ein** System geladen. Das bedeutete für den Anwendungsprogrammierer Einschränkungen, die zu lösen waren und erforderte auch im PEARL-Laufzeitpaket effizienzverbessernde Maßnahmen.

Für den Anwendungsprogrammierer besteht die Einschränkung durch den 64 K-Adreßraum. Diese kann vielfach umgangen werden, in dem viele Prozeduren NONRESIDENT gehalten werden. Hier muß jedoch bei der Programmierung darauf geachtet werden, daß im eigentlichen Betriebsfall diese Prozeduren durch Parallelitäten nicht doch zusammen benutzt werden.

Solche Fälle können durch die Schaffung weiterer Teilsysteme (in der Terminologie des MOS-Betriebssystems USER genannt) gelöst werden, da die für jedes Teilsystem notwendigen Daten zentral über das Datenverwaltungssystem (DVS) verwaltet werden.

Eine weitere Schwierigkeit wurde durch die Einführung des sogenannten „Segmentbereiches" gelöst: Die Datenübermittlung von einem Teilsystem zum anderen über das sogenannte „MAILBOX"-Verfahren – wie es in PEARL definiert ist – erforderte einen nicht mehr zu vertretenden Speicherraum im einzelnen Teilsystem. Ebenso war die residente Datenhaltung von Relationen kaum effizient.

Durch Segmentbereiche, es handelt sich hierbei um jeweils maximal 64 K große Datenbereiche, wurde dieses Adreßraumproblem beseitigt.
Ein solcher **Daten**bereich existiert als Modul

(QDVGD MODUL GLOBAL (Ø))

in jedem PEARL-Teilsystem des EPR 1300.

Die Adreßraumbelastung beschränkt sich auf Dope-Vektoren, die notwendig sind, um Arrays zu beschreiben. Der Platzbedarf eines solchen Vektors ist äußerst gering (für ein dreidimensionales Feld = 9 Worte). Ein weiterer Vorteil des beim EPR 1300 geschaffenen Segmentraumes liegt in der Möglichkeit, auch Teilsysteme aus anderen Sprachbereichen (META-S, Fortran) einem PEARL-System aufzuschalten.

Eine weitere Schwierigkeit des Datenaustausches wurde durch Einführung interner Dations vom Typ ALL beseitigt. Dations vom Typ ALL akzeptieren alle primitiven Datentypen und Strukturen als Transfer-Elemente. Der Programmierer hat selbst darauf zu achten, daß er in Objekte passenden Typs einliest. Es findet keine Prüfung statt.

4 SCHLUSSBEMERKUNG

Die Programmiersprache PEARL wird bei KAE genutzt, wofür sie konzipiert wurde, als

Process and Experiment Automation Real-Time Language.

Das aufgezeigte Projekt ist nur eines von vielen, daß im Hause KAE unter den Einsatz der Programmiersprache PEARL abgewickelt wurde. Die Erfahrungen mit dieser Sprache sind bei KAE mitlerweile so gewachsen, daß sie sich zur Standard-Programmsprache bei Softwareprojekten entwickelt hat.

Aus diesem Grund wird PEARL auch in Zukunft bei KAE gepflegt und eingesetzt werden.

Test der PEARL-Anwender-Programmbibliothek an verschiedenen Prozeßrechnern

Dipl.-Inform **Th. Roehrich**, Stuttgart

Zusammenfassung.

Im Rahmen dieses Beitrages wird ueber Portabilitaetserfahrungen berichtet, die im Zusammenhang mit der Erprobung der PEARL-Anwender-Programmbibliothek gewonnen wurden. Dabei wird speziell dargelegt, welche Syntaxanpassungen bei der Implementierung des Basis-Moduls Analogdatenerfassung auf verschiedenen Prozessrechnern notwendig sind.

Es zeigte sich, dass trotz weitestgehender Einhaltung von Basic- PEARL und unveraenderter Aufgabenstellung bei der Implementierung auf verschiedenen Zielrechnern einige syntaktische Aenderungen im PEARL-Problemteil notwendig waren.

Schluesselworte.

PEARL, Programmbibliothek, Portabilitaet, Analogwerterfassung.

Abstract.

The purpose of this part is a report about our experience in the portability in connection to testing of PEARL User Library. The main emphasis is to show which corrections of the syntax have to be made for the implementation of the basis-module Analog-Signal-Input in different processcomputers. It turned out that in spite of sticking strictly to Basic PEARL and the same problem, some corrections of the syntax in the PEARL-problempart had to be made to implement the modules in different target computers.

Keywords.

PEARL, User Library, Portability, Analog-Signal-Input

1. EINFUEHRUNG

Wie im vorangehenden Beitrag /1/ berichtet, wurde an der Abteilung Stromerzeugung und Automatisierungstechnik (IVD) der Universitaet Stuttgart eine Pearl-Anwender-Programmbibliothek (PAP) auf der Grundlage von Basic-PEARL entwickelt. Die Zielsetzung war dabei, moeglichst universell einsetzbare, getestete (Basic-) PEARL Programmbausteine auf Prozedur-, Task- und Modulebene fuer den Anwender zur Verfuegung zu stellen.

Durch die Programmierung der PEARL-Anwender-Programmbibliothek in Basic-PEARL mit nur geringen Ergaenzungen aus Full-PEARL ist die Portabilitaet der erstellten Programmbibliothek weitgehend gewaehrleistet. Dennoch erforderliche Syntaxaenderungen bei der Uebertragung von PAP-Prozeduren, -Tasks oder -Modulen auf andere PEARL-Systeme soll im folgenden anhand der Uebertragung des PAP-Moduls "Analogdatenerfassung" auf verschiedene Prozessrechner gezeigt werden.

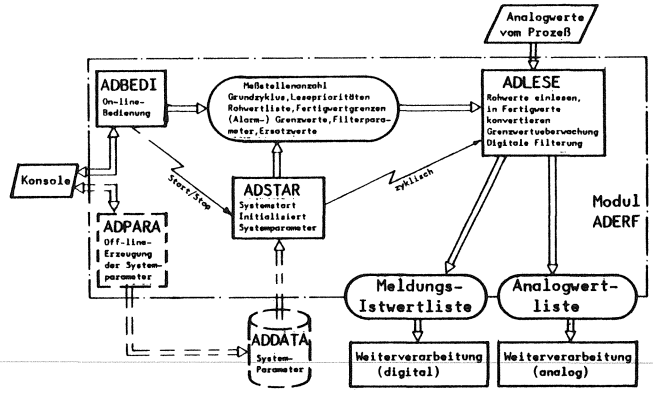


Bild 1: Aufbau des PAP Modols Analogdatenerfassung

2. DER ANALOGDATENERFASSUNGSMODUL

In B I L D 1 ist eine Uebersicht ueber den Aufbau des Analogdatenerfassungs-Moduls ADERF gegeben. Die Tasks sind dabei als Rechtecke, die globalen Daten des Moduls als langgezogene Kreise dargestellt. Der Zugriff der Tasks zu den globalen Daten wird durch gerichtete Pfeile symbolysiert. Der Kontrollfluss wird durch zusaetzliche 'Blitz'-Pfeile von aktivierenden zu aktivierten Tasks veranschaulicht.
Die in B I L D 1 gestrichelt dargestellte Task 'ADPARA' sowie die Datei 'ADDATA' dienen zur alternativen variablen Initialisierung des Moduls. Bei dieser Variante werden - im Gegensatz zur festen Initialisierung, bei der die notwendigen Parameter fest programmiert sind - die notwendigen Initialisierungen beim Start von der Datei ADDATA eingelesen. Diese Datei kann im vorbereitenden oder im Hintergrundbetrieb mit Hilfe der Task ADPARA erstellt oder veraendert werden. Eine genauere Beschreibung des Moduls Analogdatenerfassung ist in /4/ gegeben.

3. PORTABILITAETSERFAHRUNGEN

In diesem Abschnitt wird gezeigt, welche Aenderungen im Problemteil des PAP-Moduls Analogdatenerfassung bei der Implementierung auf verschiedene Prozessrechner, notwendig waren.

Bei der Uebertragung wurde von der Version des Moduls ADERF mit fester Initialisierung (B I L D 1) ausgegangen, die auf dem Prozessrechner Siemens 330 (mit Siemens PEARL-System) entwickelt wurde. Dieses Programm wurde von verschiedenen Mitarbeitern nacheinander auf die Prozessrechner EPR 1300 der Fa. Krupp Atlas Elektronik, PDP11/34 der Fa. DEC und AEG 80-20 uebertragen. Da bei den Rechnern EPR 1300 und PDP11/34 keine Prozessperipherie zur Verfuegung stand, wurde fuer diese Rechner der Modul dahingehend abgeaendert, dass die Messwerte aus einer Pheripherspeicherdatei gelesen werden. Zusaetzlich wurde bei allen Versionen eine Ausgabetask ergaenzt, die es ermoeglicht, die erfassten Werte als Protokoll auf einem Drucker auszugegeben.

Bei der Implementierung des Moduls auf den verschiedenen Prozessrechnern erwiesen sich folgende Aenderungen im Problemteil als notwendig:

a) Modulname

Bei Siemens 330: 'MODULE;'
bei AEG 80-20: 'MODULE;'
bei EPR1300: 'MODULE name;'
bei PDP11: 'MODULE(name);'

b) Name der bei Programmstart vom Betriebssystem zu startenden PEARL-Task

Bei den Prozessrechners Siemens 330 und AEG 80-20 ist keine Namenskonvention fuer PEARL-Tasks vorhanden. Bei diesen Rechnern wird jede PEARL-Task in eine vom Betriebssystem verwaltete Task ueberfuehrt. Beim Programmstart wird dann explizit die Starttask gestartet.
Bei den Versionen fuer die Rechner EPR1300 und PDP11 war es notwendig, die Starttask ADBEDI in MAIN umzubenennen. Die Verwendung dieses Standardnamens fuer die Starttask ist notwendig, da bei diesen PEARL- Implementierungen das gesamte PEARL-Programm in eine vom Betriebssystem verwaltete Einheit umgesetzt wird und beim Programmstart bekannt sein muss, welche interne PEARL-Task vom Betriebssystem gestartet werden soll.

c) Einlesen und Verarbeitung der Bedienzeile in Task ADBEDI

Die on-line Bedienung des Moduls ADERF ist so konzipiert, dass Task ADBEDI eine Bedienzeile vom Bediengeraet einliest, analysiert und dann den gewuenschten Befehl ausfuehrt. Danach wird die Befehlsausfuehrung quittiert und Task ADBEDI er-

wartet erneut Bedieneingaben. Eingelesen werden maximal 60 Einzelcharacter, die mit Hilfe einer Prozedur analysiert und zu Schluesselworten zusammengebaut werden.

Bei der Version fuer den Siemens 330 Rechner konnte die Eingabe wie konstruiert mit Hilfe des Statements:
GET ZEILE(1:60) FROM TERMINAL BY (60)A(1);
vorgenommen werden. Zudem wird dabei das Bediengeraet nicht durch die auf Eingabe wartende Task ADBEDI blockiert, sondern vom gleichen Bediengeraet aus koennen gleichzeitig andere Tasks bedient werden.

Bei der Uebertragung auf den Prozessrechner AEG 80-20 musste auf die Eingabe von Einzelcharacter verzichtet werden, da bei der Implementierung der character-verarbeitenden Prozedur LEX zur Analyse des Bedienstrings und Aufbau der Schluesselworte ein Fehler bei der Characterverarbeitung des verwendeten PEARL-Systems festgestellt wurde. Bei dieser Version wurden deshalb die einzugebenden Schluesselworte als CHAR(6) Variablen, getrennt durch Blank eingelesen, so dass die Analyse des Bedienstrings und der Aufbau der Schluesselworte komplett entfallen konnte.

Bei der Version fuer den Prozessrechner EPR1300 musste die Bedienzeile als eine CHAR(60) Variable ohne Formatangabe eingelesen werden und danach in ein Feld (60) CHAR(1) umgespeichert werden. Dies war notwendig, da sonst bei weniger als 60 eingegebenen Zeichen ein Fehler auftrat. Zudem musste eine Reaktion auf ein I/O Errorsignal im Programm vorgesehen werden, das auftritt, falls der leere Bedienstring, d.h. nur (CR) eingegeben wird. Desweiteren ist bei dieser PEARL Implementierung das Bediengeraet bei der auf Eingabe wartende Task ADBEDI fuer andere Bedienungen blockiert.

Bei der Version fuer den Rechner PDP11 musste ebenso wie beim EPR1300 die Bedienzeile in eine CHAR(60) Variable eingelesen und in ein Feld von (60) CHAR(1) umgespeichert werden.

d) Eroeffnung der Datenwege zur Standardperipherie

Bei den Versionen fuer die Prozessrechner Siemens 330 und AEG 80-20 ist das Eroeffnen der Datenwege zur Standardperipherie, d.h. zum Drucker und Terminal nicht erforderlich.
Hingegen ist bei der Version fuer den Prozessrechner EPR 1300 ein einmaliges Eroeffnen der Datenkanaele zu den Standardperipheriegeraeten Drucker und Terminal erforderlich.
Bei der Version fuer den Rechner PDP 11 ist ein Eroeffnen und Schliessen der Datenwege zum Drucker und Terminal in jeder - diese Geraete benutzende - Task erforderlich.

e) Holen der aktuellen Uhrzeit vom System

Bei der Version fuer den Siemens 330 Rechner wird die im Modul notwendige aktuelle Erfassungszeit mit Hilfe der vom System zur Verfuegung gestellten Funktion NOW ermittelt. Vor Verwendung der Funktion NOW muss diese spezifiziert werden.
Bei der Version fuer den AEG 80-20 Rechner wird die aktuelle Uhrzeit ebenfalls mit NOW ermittelt, wobei die Spezifikation der Systemfunktion NOW vor Verwendung entfallen kann.
Bei der Version fuer den Prozessrechner EPR1300 muss die aktuelle Uhrzeit mit der Funktion DAYTIME ermittelt werden. Zusaetzlich kann die Spezifikation dieser Systemfunktion vor deren Verwendung entfallen.
Bei der PDP11 Version erfolgt die Ermittlung der aktuellen Uhrzeit mit der Systemfunktion NOW. NOW muss vor Verwendung spezifiziert werden.

f) Sonstiges

Ausser den Aenderungen a) bis e) musste bei der Version fuer den PDP 11 Rechner zusaetzlich eine Funktion GETCHAR fuer die Character-Selektion '.CHAR(i)' geschrieben werden, da bei der PDP11-PEARL Implementierung bei obiger Character-Selektion 'i' keine Variable,

sondern nur eine Konstante sein darf.

Zudem wurde bei der Version fuer den AEG 80-20 Rechner die Deklaration von Prozeduren die nur in einer Task benutzt werden nicht auf Modul- sondern auf Taskebene vorgenommen. Dies war notwendig, um das Programm an die vorhandene Speicheraufteilung der Maschine (Laufbereiche) anzupassen.

In T A F E L 1 sind die oben beschriebenen notwendigen Aenderungen bzw. Abweichungen der vier Versionen unter Angabe der 'Verwendung' (Basis-PEARL, Full-PEARL, Zusatzfunktion) zusammengestellt.

Speicherplatzbedarf

Der Speicherplatzbedarf fuer den ladefaehigen Objektcode ohne Daten des Modul ADERF war

bei Siemens 330:
alle gebundene Tasks 1) 20k Worte

bei AEG 80-20:
alle gebundene Tasks 1) 3) 22k bytes

bei KAE 1300:
fuer kompl. Programm 2) 25k Worte

bei PDP 11/34:
fuer kompl. Programm 2) 30k Worte
incl. Errortask

1) Tasks koennen z.T in dieselben Laufbereiche geladen werden.
2) Speicherplatz fuer Daten abgezogen
3) Laufzeitroutinen z.T. in System

Umstellungsaufwand

Die Anpassungen des PAP-Moduls ADERF wurde durch, auf dem jeweiligen Zielrechner, erfahrene PEARL- Programmierer durchgefuehrt.

Der Aufwand fuer die Umstellung des Moduls ADERF, d.h. die Anpassung des Systemteils und obige Aenderungen im Problemteil war dabei folgender:
fuer KAE 1300 etwa halber Tag
fuer PDP 11/34 etwa ein Tag
fuer AEG 80-20 etwa ein Tag

Nicht enthalten sind die Zeiten die benoetigt wurden um den Quellcode physikalisch auf den jeweiligen Zielrechner zu transferieren. Zudem wurde von den Kollegen uebereinstimmend erklaert, dass die obigen Zeiten nur notwendig waren, da die Abweichungen im einzelnen erst ermittelt werden mussten. Fuer die da-

Änderung / Version	a) Modulname	b) Task 'MAIN'	c) d) alphanum. E/A mit Standardperipherie	e) holen von Uhrzeit	f) Character-verarbeitung
Siemens 330	MODULE;	-	·kein open ·Einleseformat (60)A(1)	·NOW ·muß spezifiziert werden.	-
AEG 80-20	MODULE;	-	·kein open ·Einlesen ganz abgeaendert.	NOW	Fehler entdeckt in akt. PEARL-Version
EPR 1300	MODULE: 'name'; 1) 2)	erforderlich	· open erforderlich ·I/O Signalreaktion vorsehen ·Einleseformat A(60)	DAYTIME	-
PDP11	MODULE: ('name'); 1) 3)	erforderlich	· open erforderlich ·Einleseformat A(60)	·NOW ·muß spezifiziert werden	.CHAR(i) mit i:= Konstante
Verwendung (Grund)	1) nicht Basic-PEARL 2) name kann entfallen bei nur einem Modul 3) Full-PEARL	wegen Implementierung	Implementierung der Dations	Systemfunktion	nicht Basic-PEARL

Tabelle 1:
Notwendige Änderungen im PEARL-Problemteil

nach ausgefuehrten Anpassungen anderer PAP-Module sind die oben angegebenen Zeiten etwa zu halbieren.

4. SCHLUSSBETRACHTUNG

Wie oben gezeigt, sind bei der Uebertragung von PEARL-Programmen auf andere Zielrechner trotz weitestgehender Einhaltung von Basic-PEARL und identischer Aufgabenstellung Aenderungen im Problemteil notwendig. Der Aenderungsaufwand ist jedoch alles in allem gering und betrifft in der Hauptsache die alphanumerische E/A c) d), die Systemfunktionen fuer die Ermittlung von Uhrzeit und Datum e), die Character-Verarbeitung c) f) und die Namenskonvention fuer die Start-Task a).

SCHRIFTTUM

/1/ Welfonder E.,Th. Röhrich:
Uebersicht ueber die PEARL-Anwender-Programmbibliothek
Vortrag bei PEARL-Tagung '82

/2/ Welfonder E.,Th. Roehrich,H. Sternad:
Aufbau und Erprobung einer modular strukturierten portablen Basic-PEARL Programmbibliothek
PEARL Rundschau, Bd.1, Nr.3 Nov.1980

/3/ Welfonder E.,Th. Roehrich:
Meldungs- und Protokolliersystem der PEARL-Anwender Programmbibliothek
PEARL Rundschau, Bd.2, Nr.6 Dez.1980

/4/ Th. Roehrich,E. Welfonder,M. Alt:
Kurzbeschreibung der PEARL-Anwender Programmbibliothek

ANSCHRIFT DES AUTORS

Roehrich, Thomas

Abteilung Stromerzeugung und
Automatisierungstechnik am
Institut fuer Verfahrenstechnik
und Dampfkesselwesen
Pfaffenwaldring 9
7000 Stuttgart 80

Tel. 0711/685-6203

Ein PEARL-Kommunikationsprogramm für den Einsatz von Sichtgeräten

Dipl.-Ing. **M. Harrer**

Kurzfassung

Es wird ein PEARL-Programm vorgestellt, das in Verbindung mit einem modernen Hilfsmittel, einem semigrafischen Prozeßvideosystem (PVS), die Führung eines Prozesses gestattet. Die Aufgabe des Programms ist es, die Komponenten Prozeß, Operateur und Prozeßvideosystem so miteinander zu verbinden, daß eine komfortable, den anthropotechnischen Anforderungen entsprechende Kommunikation zwischen Operateur und Prozeß ermöglicht wird.

Schlüsselworte: Leitwarte, Prozeßführung, Prozeßvideosystem, Mensch-Maschine Schnittstelle

Summary

A PEARL-Program is presented which permits process control when used in combination with a modern semigraphic process video system (PVS) The function of this program is to link the components process, operator and process video system in such a manner as to enable a comfortable man - machine interface which fulfills anthropotechnical requirements.

Key words: Control room, process control, process video system, man-machine interface

Der **Mensch als Operateur** in der Leitwarte hat **bei der Prozeßführung** drei Aufgaben zu erfüllen:

- **Überwachen des Prozesses**, um den Anlagenwirkungsgrad zu beurteilen oder Störungen bereits in der Anbahnungsphase zu erkennen,
- **Eingreifen (Bedienen)** im Normalbetrieb, um **so Laständerungen oder** Strukturumschaltungen **vorzunehmen; Eingreifen** im Störfall, **um negative Auswirkungen** der Störung **zu minimieren,**
- **Klären der Störungsursache**, damit diese beseitigt werden kann [1].

Wird für diese Aufgaben in der Warte ein Bildschirmsystem eingesetzt, so ist es mit dem Ziel zu konzipieren, den Operateur bei den obengenannten Aufgaben wirkungsvoll zu unterstützen.

Betrachtet man die Zusammenhänge für das System Mensch - Warte - Prozeß, so gelangt man zu der in B i l d 1 gezeigten Darstellung.

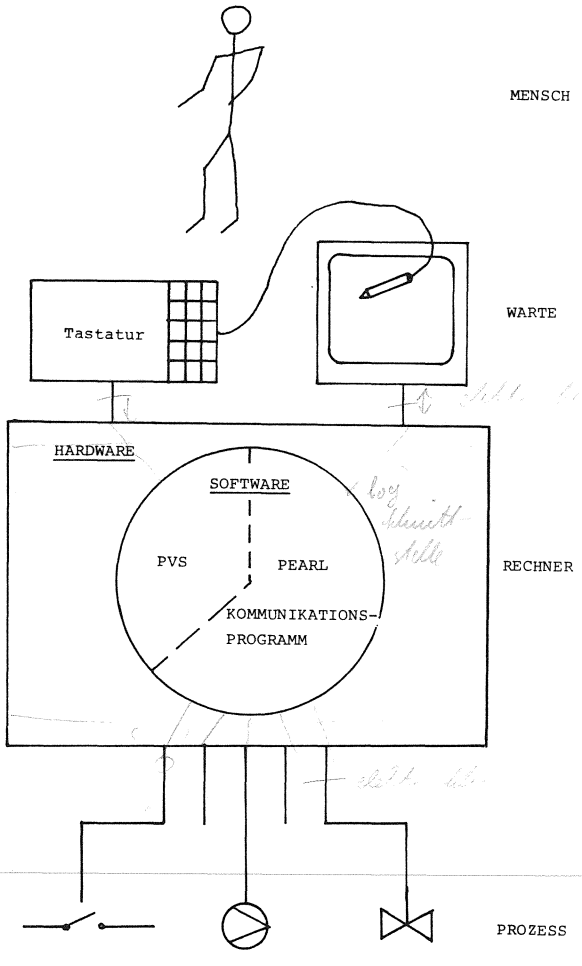


BILD 1: AUFBAU EINES MENSCH-MASCHINE-SYSTEMS

Prozeß

Der Prozeß liefert über Analog-/Digital-Eingabekarten analoge und binäre Signale an den Rechner und erwartet Stellsignale für Regler, Pumpen, Schrittmotoren usw.

Operateur

Für die Prozeßüberwachung benötigt der Operateur Informationen unterschiedlichen Detaillierungsgrades aus der Anlage. Ihm muß deshalb die Möglichkeit gegeben sein, einfach und schnell die gewünschten Prozeßdaten auf den Bildschirm heranzuholen, um sich ein gedankliches Modell über den augenblicklichen Anlagenzustand aufzubauen und ständig zu aktualisieren. Auf außergewöhnliche Zustände hat der Rechner besonders hinzuweisen (Farbe, Blinken) und auf Wunsch zusätzliche Informationen wie Kurven oder Vorschläge für die Störungsbeseitigung bereitzustellen.

Greift der Operateur in den Prozeß ein, so unterteilt sich dieser Vorgang in die Anwahl des zu betätigenden Aggregats und den eigentlichen Schaltvorgang.

Prozeßvideosystem

Die von den Herstellern gelieferten Bildschirmsysteme sind im allgemeinen black box Systeme, d.h. Veränderungen können daran nicht vorgenommen werden. Die softwaremäßige Ankopplung geschieht über mehrere Datenkanäle, die meist unidirektional ausgelegt sind. Bei dem PVS 1300 von Krupp Atlas können z.B. drei Datenwege für Befehlseingabe, Statusmeldungen und interaktive Rückmeldungen (Lichtgriffeleingabe, Dialoge) konfiguriert werden [2]. Alle Bildmanipulationen erfolgen über den Befehlseingabekanal; einige Beispiele für PVS-Befehle zeigt
T a b e l l e 1.

PVS-Befehl	Bedeutung
LB37;	Lade Bild 37
BE281;	Schalte Blinken in Feld 281 ein
ZT201,'PEARL';	Weise dem Feld 201 den Text 'PEARL' zu
ZM4,7;	Weise dem Feld 7 den Makroinhalt 4 zu
FH5,2;	Ändere im Feld 5 die Hintergrundfarbe auf Rot
UK4,*,126;	Füge zu Kurve 4 den Amplitudenwert 126

T a b e l l e 1: Beispiele für PVS-Befehle

Nach einer Anwahl mit Lichtgriffel in einem Bild werden die Koordinaten verschlüsselt ausgegeben. Desweiteren sendet das PVS über Tastatur eingegebene Dialogtexte an das übergeordnete Kommunikationsprogramm.

Kommunikationsprogramm

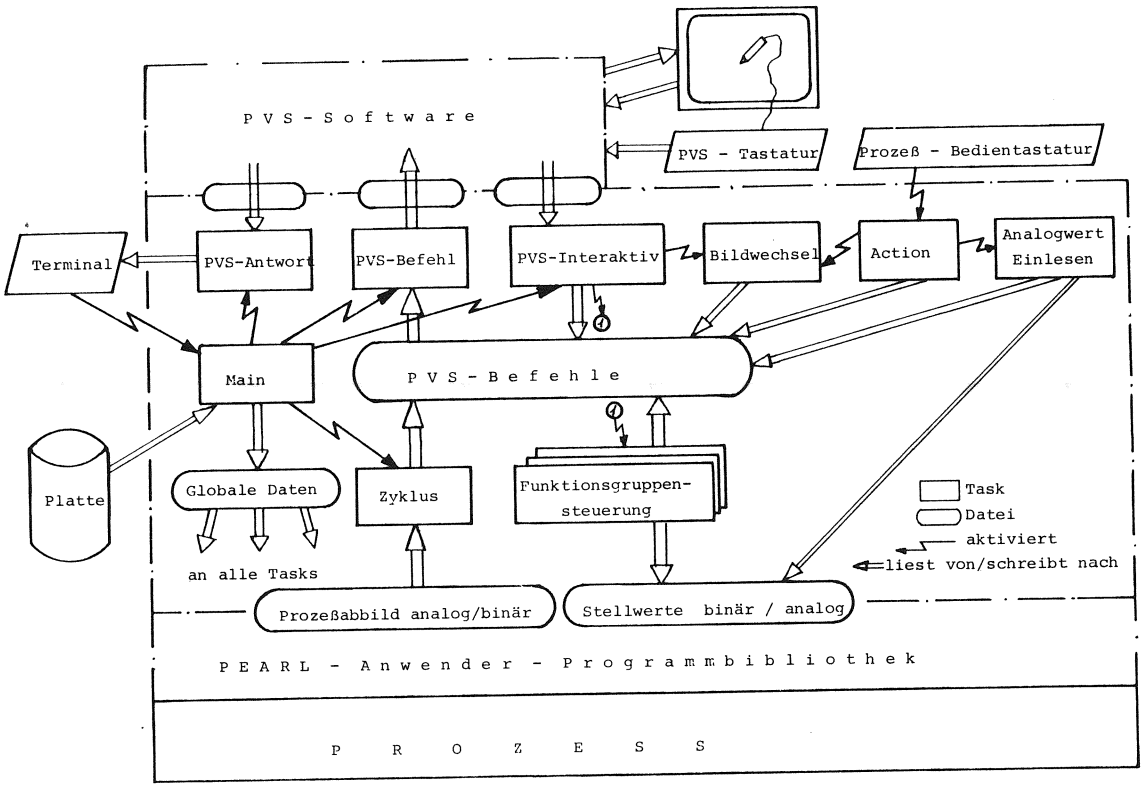
Die Koordinierung, Weiterleitung, Umsetzung und Auswertung der zwischen den vorgenannten Komponenten auszutauschenden Daten obliegt dem PEARL - Kommunikationsprogramm. Es "versteht" die jeweilige Befehlssprache der Komponenten und besitzt die "Intelligenz", mächtige Befehle zu decodieren und in Einzelanweisungen für den Befehlsempfänger aufzuschlüsseln.

Im einzelnen gliedern sich die Aufgaben des Kommunikationsprogramms in zwei Gruppen:

1. Zyklische Aufgaben
   - Meßwerterfassung
   - Grenzwertbildung
   - Update der angezeigten Bilder
     -- Datum/Uhrzeit
     -- Analogwerte (Balkendarstellung)
     -- Binärwerte
     -- Textausgaben
     -- Kurvendarstellung

2. Azyklische Aufgaben
   - Prozeßanforderungen
   - Anforderungen des Operateurs
     -- Bildwechsel
     -- Lichtgriffeleingaben
     -- Dialoge
   - PVS-Anforderungen

Zur Bearbeitung obengenannter Aufgaben stehen in dem ausgeführten Kommunikationsprogramm in sich geschlossene Programmsegmente zur Verfügung. Für die Datenerfassung und die Stellsignalausgabe werden Module aus der PEARL-Anwenderprogrammbibliothek verwendet [3,4]. Damit der Synchronisierungsaufwand klein blieb, wurden Teilaufgaben, die nicht unbedingt eine parallele Bearbeitung verlangen, in Prozeduren behandelt, während nebenläufige Algorithmen in Tasks abgearbeitet werden. Insbesondere Schnittstellen sind



B i l d 2: Kontrollfluß für die Hauptfunktionen

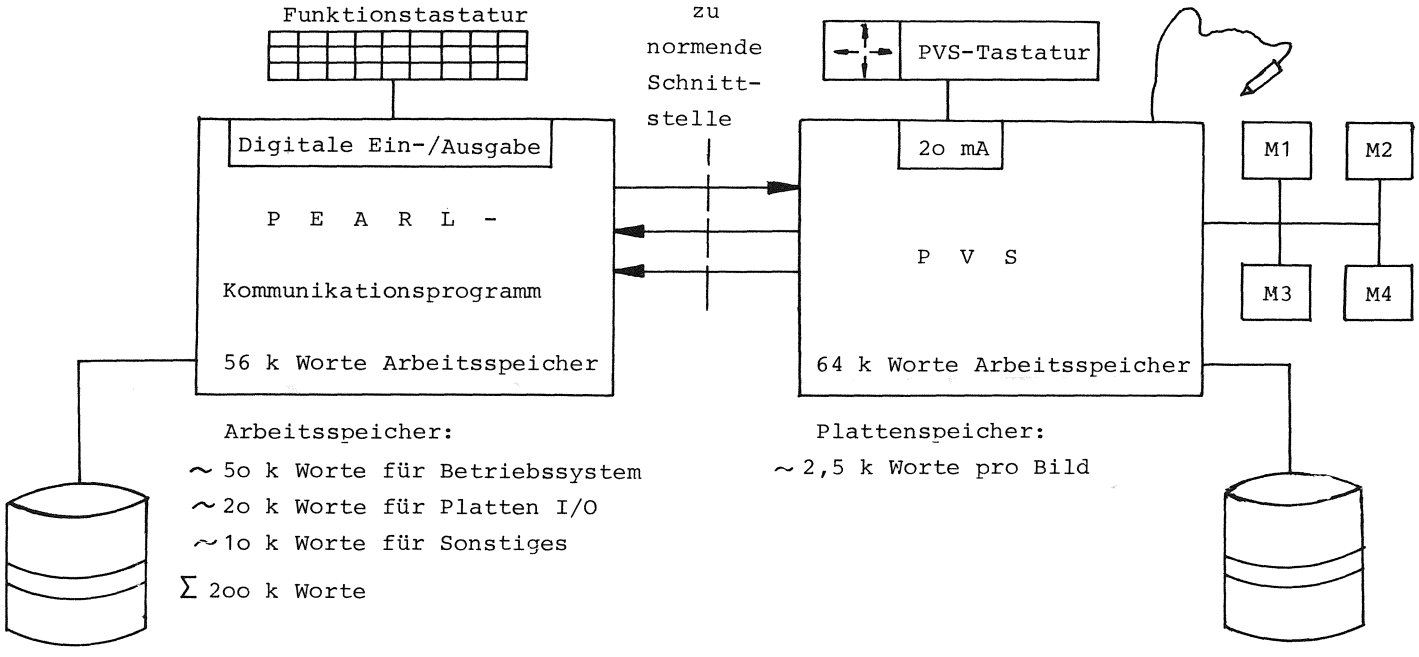
durch Tasks abgedeckt, sodaß eintreffende Informationen quasiparallel verarbeitet werden können. Den Kontrollfluß für die Hauptfunktionen zeigt B i l d 2.

Es lag nahe, zeitkritische Programmteile wie Datenerfassung usw. resident im Speicher zu halten und seltener ablaufende Hintergrundprogramme auf Peripherspeicher auszulagern. Für den Arbeitsspeicher erfolgt die Verwaltung dynamisch, d.h. das Betriebssystem weist den nichtresidenten Tasks und Prozeduren Arbeitsspeicher zu.

Bei der Konzeption des Datenflusses wurde unter anderem auf eine gute Testbarkeit des Programms geachtet. So bewährte sich die Möglichkeit, den gesamten Datenverkehr zwischen Kommunikationsprogramm und PVS auf Platte mitzuschreiben und im Fehlerfall zu analysieren.

Das Kommunikationsprogramm ist auf einem KAE EPR 1300 Rechner implementiert. Unter einem Multiuser - Betriebssystem laufen PVS- und PEARL - Software in zwei Benutzerbereichen; der Datenaustausch zwischen den Benutzern geschieht über Mail-Boxes, die von PEARL aus als Alphic Dation mit GET/PUT-Anweisungen angesprochen werden. Als Bedienmittel für das Bildschirmsystem stehen eine Funktionstastatur sowie Lichtgriffel oder Cursor mit alphanumerischer Tastatur zur Verfügung (B i l d 3).

Das Kommunikationsprogramm wurde mit dem Gedanken entworfen, ein breites Aufgabenspektrum in einer Kraftwerks-Bildschirmwarte abzudecken und gleichzeitig für Erweiterungen offen zu sein. Durch die Verwendung von PEARL als Programmiersprache ist das Programm portabel, doch ist es nur in Verbindung mit der KAE/PVS-Software einsetzbar. Die Anpassung an die Bildschirmsoftware von anderen Herstellern wäre mit einigem Aufwand realisierbar. Eine Normierung der Semigrafikschnittstelle, die auch Teile der Konstruktionsphase umfassen soll und eine Standardisierung der Informationsanzeige auf dem Bildschirm, wie sie vom VDI/VDE durchgeführt wird [5] , kann die Wiederverwendbarkeit von Rechnersoftware ermöglichen und wesentlich zur Verbesserung der Wirtschaftlichkeit von Bildschirmsystemen beitragen.



B i l d 3: Hardwareausstattung

Schrifttum

[1] Friedewald, W.; Charwat, H.: Gestaltung von Grafikbildern für Farbsichtgeräte in Prozeßwarten, RTP 1/1979

[2] Beschreibung Prozeßvideosystem PVS 1300 Krupp-Atlas Elektronik, Bremen

[3] Welfonder, E.; Röhrich, T.: Übersicht über die PEARL-Anwender-Programmbibliothek, Vortrag PEARL-Tagung 1982, Düsseldorf

[4] Röhrich, T.: Test der PEARL-Anwender-Programmbibliothek an verschiedenen Prozeßrechnern, Vortrag PEARL-Tagung 1982, Düsseldorf

[5] Prutz, G.: Darstellung von Regelungs- und Steuerungsinformationen auf Bildschirmen, Vortrag beim PRAT 1982, VDI-Bericht Nr. 451, 1982

Dipl.-Ing. Manfred Harrer
Universität Stuttgart, Abteilung Stromerzeugung und Automatisierungstechnik (IVD)
Leitung: Prof. Dr.-Ing. E. Welfonder

Pfaffenwaldring 9
7000 Stuttgart 80

Eine allgemeine Dialogschnittstelle in PEARL für die Kommunikation mit PEARL-Prozessen über Bildschirm und Tastatur

Dipl.-Math. **Günter Stöhr**

Kurzfassung

Im Rahmen von Betriebsleitsystemen für den öffentlichen Personennahverkehr hat die Kommunikation des Fahrdienstleiters mit dem Prozeß eine übergeordnete Bedeutung. Deshalb wurde bei der Entwicklung des vom BMFT geförderten rechnergesteuerten Betriebsleitsystems BON auf die Realisierung eines allgemein verwendbaren Dialogkonzepts mit einem zentralen Dialogprogramm besonderer Wert gelegt.

Die Lösung der Aufgabe wurde in Form eines PEARL-Programms erreicht, das zu Dialog-Anwendungsprogrammen dazugebunden werden kann. Die Schnittstellen stehen als Unterprogrammaufrufe zur Verfügung. Die Ausgabeseite ist in Form einer sog. Einheitlichen Grafik-Schnittstelle (EGS) für Bildschirmgeräte verwirklicht, so daß bei Prozeßanwendungen übliche semigrafische Bildschirmgeräte aber auch alphanumerische Terminals verwendet werden können.

Abstract

Within the scope of a computerised management system for public transport the communication of the operator with the process has a superior importance. Therefore in the face of developing the computerised management system BON in the town of Hannover, supported by BMFT - the German Ministry for Research and Technology -, there was set a high priority on the creation of a universal dialogue-module with a central dialogue-program.

The solution of the problems was achieved by a PEARL-program, which can be added to the dialogue-user-programs. The interfaces are available as procedure-calls. The output-interface of the dialogue-module is a so-called "Einheitliche Grafik-Schnittstelle" (Standardized Graphic-Interface), giving the possibility of using semigraphic screens as well as alpha-displays as peripheral devices.

1. Idee und Entwicklung der Dialog-Schnittstelle im Rahmen BON

Eine der wichtigsten Aufgaben bei der Entwicklung rechnergestützter Leitsysteme ist die Verbindung des Prozesses mit dem Bedienungspersonal. Darunter fallen Informationen über Betriebszustände, Prozeßbeeinflussung durch Steuereingaben, Anzeige kritischer Ereignisse usw. Betrachtet man diese Mensch-Prozeß-Schnittstelle bei verschiedenen Leit- und Überwachungssystemen genauer, so fällt immer wieder die Ähnlichkeit der Struktur der Leitstellen bezüglich Hard- und Software auf (siehe auch Bild 1):

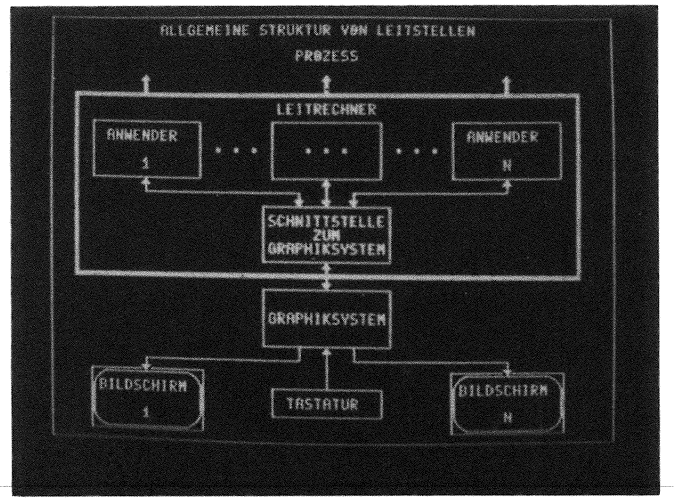


Bild 1: Allgemeine Struktur von Leitstellen

- Es gibt ein Ein- und Ausgabe-(Grafik-)-System mit Bildschirm, Tastatur und ggfs. weiteren peripheren Einheiten
- Es gibt einen oder mehrere Leitrechner mit Programm-Modulen, die eine Verbindung zum Bedienungspersonal am Leitstellenarbeitsplatz benötigen
- Die Verbindung zwischen Leitrechner und Grafiksystem wird über besondere Softwareschnittstellen hergestellt

Darüber hinaus zeigt sich deutlich, daß die Formalismen der Dialogführung im Konzept oft identisch sind und sich nur in Ausprägungen unterscheiden.

Trotz dieser Gegebenheiten ist die softwaretechnische Realisierung sehr unterschiedlich. Dialog- und Schnittstellenbausteine werden immer wieder "neu erfunden". Gleichzeitig werden sie mit der eigentlichen Prozeßsoftware so verwoben, daß eine Übertragung auf andere Leitsysteme selbst bei quasi identischer Aufgabenstellung nicht mehr möglich ist.

Bei der Entwicklung des vom BMFT geförderten rechnergesteuerten Betriebsleitsystem für den öffentlichen Personennahverkehr (BON) wurde gemäß der Forderung der Standardisierung und der Entwicklungsfähigkeit diesen Erkenntnissen Rechnung getragen (siehe auch Bild 2), indem:

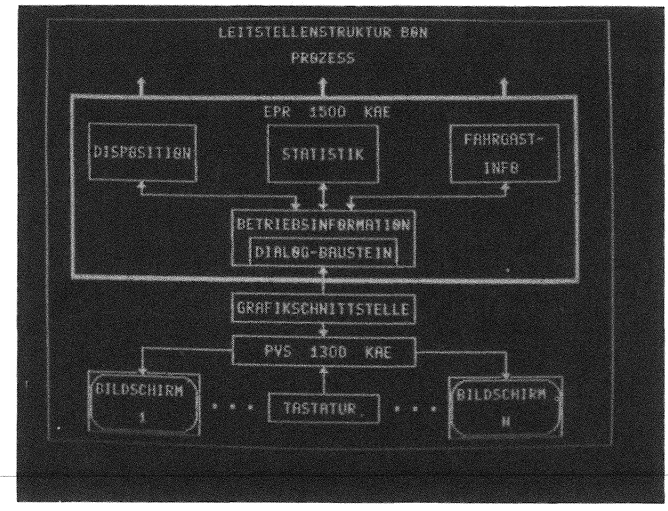


Bild 2: Leitstellenstruktur in BON

- ein zentrales Dialogprogramm mit einem allgemeinen Dialogkonzept entwickelt wurde, über das die Anwender-Module freizügige Dialoge mit dem Rechner führen können,
- zur Ansprache des Grafiksystems eine "Einheitliche Grafik-Schnittstelle für die Anbindung semigrafischer Farbbildschirmsysteme" (im folgenden EGS) formuliert und angewendet wurde,
- die Realisierung der Komponenten in PEARL erfolgte, um nicht auf einen bestimmten Prozeßrechnerhersteller von vornherein festgelegt zu sein.

Dadurch sollten die Anwenderpakete (hier Disposition, Soll-Vorgaben, Statistik, Fahrgastinformation) weitgehend von der Führung der Dialoge entlastet werden sowie die einheitliche Bedienerführung sichergestellt werden. Weiterhin wird durch die Verwendung der EGS die Unabhängigkeit von einer speziellen Grafik-Hardware erreicht.

Im Rahmen BON wurde das Dialogprogramm dabei im sog. Modulpaket Betriebsinformation angesiedelt. Es ist aber an sich unabhängig. Für die Pilotanwendung in Hannover findet als Grafiksystem ein Prozeß-Video-System PVS 1300 und als Leitrechner ein EPR 1500 der Firma Krupp-Atlas-Elektronik, Bremen (im folgenden KAE), Verwendung.

Im folgenden werden zunächst auf die Anforderungen an den Dialogbaustein und die sog. Einheitliche Grafikschnittstelle näher eingegangen. Anschließend wird die Realisierung in PEARL erläutert.

2. Anforderungen an den Dialogbaustein

Der Dialogbaustein soll von verschiedenen Bearbeitungsmoduln des BON-Systems - nämlich Soll-Vorgaben, Disposition, Statistik und Fahrgastinformation -, die eine Verbindung zum Disponenten, d.h. Bediener, benötigen, individuell benutzt werden können. Insgesamt wurden folgende Forderungen aufgestellt: (siehe auch Bild 3)

- freie Definition von Dialogformularen
- Eingabe beliebiger Werte in Eingabefelder
- Möglichkeit der Vorbelegung von Eingabefeldern mit Werten, die vom Disponenten akzeptiert oder korrigiert (überschrieben) werden können

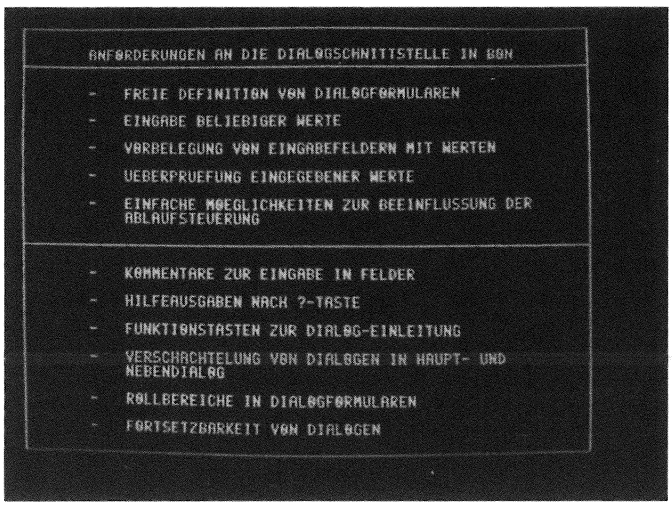


Bild 3: Anforderungen an die Dialog-Schnittstelle

- Überprüfung von eingegebenen Werten auf
  * Erfüllung von Eingabekonventionen (Format-Prüfung)
  * Sinnfälligkeit (Semantik-Prüfung)

  und, falls eine Fehleingabe erkannt wird, Ausgabe vorformulierter Fehlermeldungen mit anschließender Möglichkeit zur Korrektur der Eingabe
- einfache Möglichkeiten zur Beeinflussung der Ablaufsteuerung durch das anwendende Modulpaket

Ein Dialogformular besteht dabei im wesentlichen aus einem Grundbild (Raster) mit sogenannten Eingabefeldern, in die Werte sowohl ein- als auch ausgegeben werden können. Während einer Dialogführung gibt es immer genau ein aktuelles Eingabefeld, welches durch den Cursor (Schreibmarke) identifiziert wird. In dieses aktuelle Feld können Werte eingetragen oder, falls vorhanden, überschrieben oder akzeptiert werden. Jede abgeschlossene Eingabe in ein Feld wird überprüft und im Fehlerfall eine entsprechende Meldung ausgegeben. Anschließend wird dasselbe Eingabefeld zur Korrektur aktuell oder es wird, falls vorhanden, ein weiteres Feld behandelt. Ist kein weiteres Feld mehr vorhanden, so gilt der Dialog als abgeschlossen. Die Bedienführung in Form des Dialogablaufs kann dabei vom anwendenden Modulpaket leicht beeinflußt werden.

Zur Erhöhung des Komforts der Schnittstelle sowohl für den Bediener als auch für das anwendende Modulpaket wurden zusätzliche Anforderungen entwickelt:

- Ausgabe von Kommentaren vor der Eingabe zur Erläuterung des einzugebenden Wertes
- Ausgabe zusätzlicher, erläuternder Informationen, bezogen auf das aktuelle Eingabefeld, nach Betätigung einer Hilfstaste (?-Taste)
- Einsatz von Funktionstasten zur Dialog-Einleitung
- Verschachtelung von Dialogen in Haupt- und Nebendialoge
- Einführung von Rollbereichen in Dialogformularen
- Fortsetzbarkeit von Dialogen

Dies sind die wichtigsten Anforderungen, die im Rahmen der Entwicklung vor allem von der Anwenderseite aus zusammengetragen wurden. Desweiteren wurde von dieser Seite auch eine möglichst hardware-unabhängige Ansprache des Graphik-Systems gefordert. Darauf wird im nächsten Kapitel genauer eingegangen.

3. Die Grafikschnittstelle

Eine Anwenderforderung bei der Ansprache des Graphiksystems war die Hardware-Unabhängigkeit. Bei der Analyse verschiedener Graphiksysteme wurde nun festgestellt, daß diese einen im Prinzip ähnlichen logischen Aufbau besitzen. Dies legte die Schaffung einer Einheitlichen Grafik-Schnittstelle (EGS) nahe. Um eine eindeutige Trennung zwischen Leitrechner und Grafiksystem zu erzielen, wurde von der Anwenderseite gefordert, Hard- und Softwareschnittstellen an derselben Stelle anzuordnen. Deshalb wurde für die zu entwickelnde EGS die Form einer Alphic-Dation gewählt, die mit "PUT" und "GET" angesprochen werden kann. Als Graphiksystem werden dabei ein oder mehrere Graphikrechner mit Bildschirmen und Tastaturen, Lichtstift zugrunde gelegt. Als logische Grundeinheit wurden Bilder definiert, die geladen, auf Bildschirmen dargestellt und manipuliert werden können, indem etwa Texte eingetragen oder Farb- und Blinkinformationen hinzugefügt werden.

Zu einem Grafikbild gehören dann: (siehe auch Bild 4)

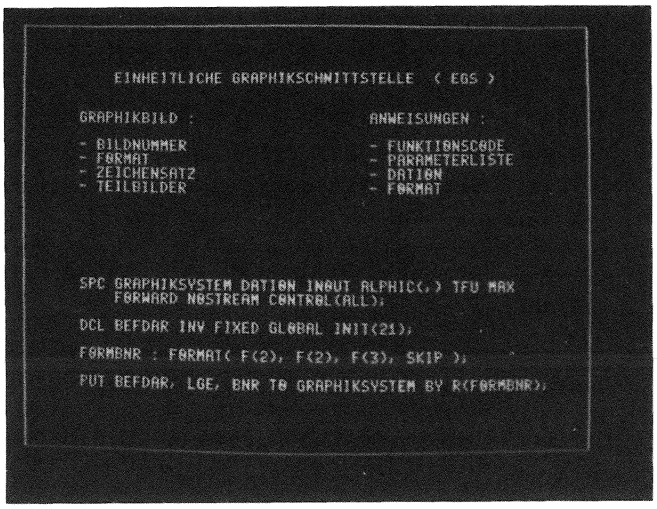


Bild 4: Die Einheitliche-Grafik-Schnittstelle

- eine Bildnummer
- ein Format (Zeilen * Spalten)
- ein Zeichensatz, mit dem das Bild erstellt wurde
- Teilbilder in verschiedensten Ausprägungen

Unter Teilbildern sind dabei etwa Bildausschnitte, vorgefertigte Großsymbole, Bildpositionen oder auch Eingabefelder zu verstehen. In einem Befehl an das Graphiksystem werden nun das anzusprechende Bild und die betroffenen Teilbilder usw. in einer Parameterliste zusammengefaßt. Der Befehl selbst enthält dann insgesamt:

- einen Funktionscode (BEF...)
- eine Parameterliste
- eine Dation
- ein Format

z.B. hat dann der Befehl an das Graphiksystem zum Darstellen eines Bildes mit der Nummer BNR auf dem Bildschirm, der im System unter dem logischen Namen LGE bekannt ist, das Aussehen:

```
PUT BEFDAR, LGE, BNR TO GRAPHIKSYSTEM
          BY R(FORMBNR);
```

Dabei ist BEFDAR der Befehlscode für Bild-Darstellen und FORMBNR das zugehörige Format.

```
FORMBNR: FORMAT(F(2), F(2), F(3), SKIP);
```

In der hier angesprochenen Graphikschnittstelle werden dabei alle Funktionscodes, Formate und sonstigen graphikbezogenen Werte wie etwa Farben vom Implementierer des Grafiksystems als benannte Konstanten zur Verfügung gestellt. Der Anwenderprogrammierer benötigt also keine speziellen Kenntnisse über Wertebereiche etc.

Im Falle der Entwicklung des Betriebsleitsystems BON ist der Dialogbaustein der Hauptanwender der EGS. Auf den Aufbau dieses Bausteins soll im folgenden eingegangen werden.

4. Aspekte der Realisierung des Dialogbausteins in PEARL

Die Dialogschnittstelle steht, von den sie benutzenden Modulpaketen aus gesehen, als Prozeduraufrufe zur Verfügung. Der Aufruf zur Anforderung einer Dialogführung sieht dabei wie folgt aus:

```
FUEHREDIALOG: PROCEDURE
  (MP FIXED,          /* Auftraggeber                  */
   KENNUNG FIXED,     /* Kennummer der
                         Aufforderung                  */
   MNR FIXED,         /* Monitor                       */
   BNR FIXED,         /* Bildnr. des Eingabe-
                         formulars                     */
   FORTSETZUNG BIT(1),    /* Dialog-
                             Fortsetzung?              */
   ANZEAFELDER FIXED,     /* Anzahl Ein-/
                             Ausgabefelder             */
   FELDEINTRAG() EINTRAG IDENT,  /* Liste der
                                    E/A-
                                    Felder
                                                       */
   ERR FIXED IDENT)       /* Return-Code               */
   RESIDENT GLOBAL;
```

Der Auftraggeber identifiziert sich dabei gegenüber der Dialogschnittstelle durch seinen Namen sowie eine Kennummer der Anforderung, die bei Dialogfortsetzung zur Feststellung der Zulässigkeit dient. Ferner wird die Bildnummer des Eingabeformulars, welches behandelt werden soll, angegeben. Der Dialogbaustein führt dabei unter dieser Nummer eine Beschreibung des Eingabeformulars, die alle wichtigen Informationen enthält. Dazu gehört unter anderem auch eine Beschreibung aller im Bild enthaltenen Ein-/Ausgabefelder mit: (siehe auch Bild 5)

- Anzahl möglicher Zeichen im Feld
- Liste der Nachbarteilbilder bei Betätigung einer der vier Cursor-Steuertasten
- zugehörige Konvertierungsroutine bei Ein-

```
/* DATENSTRUKTUREN DES DIALOG-BAUSTEINS                    */

TYPE EAFELDER STRUCT      /* E/A-FELD-BESCHREIBUNG          */
     (ANZ          FIXED, /* ANZAHL MOEGL. ZEICHEN IM FELD */
      TAB          FIXED, /* NACHBARFELD NACH TAB →        */
      TABM1        FIXED, /* NACHBARFELD NACH TAB ←        */
      PFEILOBEN    FIXED, /* NACHBARFELD NACH TAB ↑        */
      PFEILUNTEN   FIXED, /* NACHBARFELD NACH TAB ↓        */
      KONVROUT     FIXED, /* KONVERTIERUNGSROUTINE         */
      PRUEFROUT    FIXED, /* PRUEFROUTINE                  */
      ARBEITSPAR   FIXED, /* VERWEIS AUF ARBEITSPARAMETER  */
      HELPIDF      FIXED);/* NAME DER HELPLISTE            */

TYPE EINTRAG STRUCT       /* EINTRAG IN EIN E/A-FELD       */
     (TBNR         FIXED, /* TEILBILDNUMMER                */
      WERT     PARAMETER, /* EINZUTRAGENDER WERT           */
      STATUS       FIXED, /* FELDSTATUS                    */
      MELDUNG      FIXED);/* KOMMENTAR ZUM FELD            */

DER DATENTYP PARAMETER WIRD DABEI JE NACH ANWENDUNGSFALL
GESCHAFFEN.
```

Bild 5: Datenstrukturen der Dialog-Schnittstelle

oder Ausgabe in das Feld

- zugehörige Prüfroutine bei Eingabe in das Feld
- Arbeitsparamter für die Konvertierungs- oder Prüfroutine
- Informationen über die Helpliste, die bei Betätigung der ?-Taste aufgeschaltet werden soll

Der Auftraggeber wählt nun aus allen möglichen Ein-/Ausgabefeldern im Formular diejenigen aus, die er im nächsten Dialog behandeln will, und liefert:

- ihre Anzahl und
- ihre Feldeinträge

an.

Ein Feldeintrag enthält dabei:

- die Teilbildnummer des Ein-/Ausgabefelds,
- den einzutragenden Wert,
- den Feldstatus bei Dialoganfang,
- den Kommentar, der bei Freigabe dieses Feldes ausgegeben werden soll.

Welches Aussehen der einzutragende Wert dabei haben kann, ist dem Anwendungsfall überlassen. Für das Forschungsvorhaben BON wurden zum Beispiel

- Zahlen
- Linien/Kurs-Nummer
- Haltestellenkürzel
- Datums-Angabe
- Abfahrzeiten

als Parameterarten benötigt.

Der Feldstatus ist die wichtigste Einflußmöglichkeit des Anwenders; er kann folgende Werte annahmen:

- Vor Eingabe:
  Das Feld ist leer, und es soll eine Eingabe dort stattfinden.
- Vorbelegt:
  Das Feld ist mit einem Wert vorbelegt, dieser muß akzeptiert oder überschrieben werden.
- Falsch:
  Das Feld enthält eine falsche Eingabe.
- Veränderbar:
  Das Feld enthält einen Wert, der verändert werden kann, aber nicht muß.
- Inaktiv:
  Das Feld wird zur Zeit nicht benutzt.
- Ausgabefeld:
  Das Feld hat reine Ausgabefunktion. Dort kann zur Zeit nichts eingegeben werden.

Mit Hilfe dieses Feldstatus ist somit eine einfache Einflußnahme auf den Dialogablauf möglich.

Die angegebenen Datenstrukturen sind als PEARL-Structs implementiert. Dabei wurde auf das TYPE-Konzept zurückgegriffen. Insbesondere ist dadurch auch die einfache Anpassbarkeit an den speziellen Anwendungsfall sichergestellt. Diese leichte Anpassbarkeit wird erreicht durch:

- die Arten der zu behandelnden Parameter sind vom Anwender vorgebbar,
- die Konvertierungsroutinen zur Ein- und Ausgabe der Werte sind vom Anwender vorgebbar,
- die Prüfroutinen zur Überprüfung der eingegebenen Werte sind vom Anwender vorgebbar.
- bei der Software-Entwicklung wurden moderne Methoden aus der strukturierten Programmierung angewandt.

So sind zum Beispiel alle Datenbestände des Dialogbausteins in einem PEARL-Module zusammen mit ihren Zugriffsfunktionen zusammengefaßt. Dies bedeutet, daß die Daten nur über fest definierte Zugriffe erreichbar sind. Als Beispiel sei eine Procedure angeführt,

die die Cursorsteuerung in einem Eingabeformular übernimmt:

```
NXTFNRCUR: PROCEDURE
   (MNR FIXED,            /* (E): Monitor   */
   BNR FIXED,             /* (E): Bildnummer*/
   AKTCUR FIXED,          /* (E): Aktuelle
                             Cursorposition */
   RICHTUNG FIXED,        /* (E): Richtung  */
   NEWCUR FIXED IDENT)    /* (A): Neue
                             Cursorposition */
   RESIDENT GLOBAL;
```

Diese Procedure bestimmt in einem Eingabeformular, daß durch seine Bildnummer identifiziert wird, das nächste Eingabefeld. Dazu benötigt die Procedure außer der alten Cursorposition und der Suchrichtung auch die Angabe des Monitors, auf dem das Eingabeformular dargestellt ist, da die Cursorsteuerung durch aktuelle Dialogwerte wie Feldstatus beeinflußt wird. Diese aktuellen Dialogwerte sind nun monitorbezogen im Datenbereich hinterlegt.

Alle wichtigen Dimensionierungen wie Anzahl Monitore, Anzahl von Eingabeformularen, maximale Anzahl von Ein-/Ausgabefeldern sind generierbar. Dies bedeutet, daß, je nach Anwendungsfall, auf einfache Weise durch ein PEARL-Programm eine optimal an die Erfordernisse angepaßte Version des Dialogbausteins erstellt wird.

Implementiert wurde der Dialogbaustein auf einem Rechner vom Typ EPR 1400 der Firma KAE in 3 PEARL-Modules:

- Datenbereich mit Zugriffsfunktionen (40 Prozeduren)
- Dialogbearbeitung (1 Task, 8 Proceduren)
- Prüf- und Konvertierungsroutinen

Der Speicherplatzbedarf des im Anwendungsfall stark dialogorientierten Betriebsleitsystems BON beträgt mit Daten etwa 30-40 K-Worte. Für kleinere Anwendungen reduziert sich der Speicherbedarf entsprechend.

5. Schlußbemerkung

Die im Rahmen der Entwicklung des rechnergestützten Betriebsleitsystems geforderte Standardisierung und die hohen Anwenderanforderungen bezüglich Komfort und Geschwindigkeit, machten die Entwicklung des Dialogbausteins sowie einer allgemeinen Grafikschnittstelle notwendig. Zusammenfassend kann man nach eingehenden Tests des Dialogbausteins mit mehreren Leittischen sagen, daß dieser die Anforderungen erfüllt. Es besteht die begründete Aussicht, daß er sich auch in der betrieblichen Erprobung bewährt.

Anschrift des Autors:

Dipl.-Math. Günter Stöhr

Büro
Dr.-Ing. H. Heusch/
Dipl.-Ing. J. Boesefeldt GmbH
Liebigstraße 20

5100 Aachen

Tel.: 0241/16 50 01

PROKON – ein universelles, frei parametrierbares PEARL-System für Meßwerterfassung, -verarbeitung, -visualisierung und -archivierung

R. Brehm, R. Jaeckel, E. Rausch, Nürnberg

Zusammenfassung:

Viele unterschiedlichste technologische Prozesse machen die Ueberwachung einer grossen Zahl von analogen Messwerten, von Signalen und Zaehlwerten notwendig. Oft sind diese Ueberwachungsaufgaben haeufigem Wechsel unterworfen. Derartige Aufgabenstellungen erfordern deshalb ein System, das vom Technologen und nicht vom Rechnerfachmann durch einfachste Handhabung staendig und auf einfachste Weise auch an komplexe Aufgaben angepasst werden kann.

Nur durch den Einsatz einer modernen hoeheren Programmiersprache vie PEARL war es moeglich mit vertretbarem Aufwand dieser Aufgabenstellung gerecht zu werden.

Mit PROKON wird ein System vorgestellt, das fast ausschliesslich in PEARL programmiert ist und auf heute verfuegbare modernste Hard- und Software aufbaut.

Es dient der Signal- und Messverterfassung, der Verarbeitung, der Visualisierung und der Archivierung dieser Prozessgroessen und ist frei im Maskendialog parametrierbar.

Stichworte

Prozessueberwachung
Prozesssteuerung
Prozessdatenverarbeitung
Messverterfassung
Signalverarbeitung
Prozessvisualisierung
Datenarchivierung
Anlagendarstellung
Maskendialog
Stoerprotokoll
Meldeprotokoll
Wartungsprotokoll
Semigraphik
Vollgraphik

Summary

Many technological processes from a variety of application areas require the monitoring of a large number of analogue values, signal inputs and counter readings. This monitoring task is often subject to frequent change. Such a task requires a system which is simple to use for an engineer or operator, i. e. does not require a computer specialist, but is capable of the monitoring of complex processes.

This goal was achieved with commensurate effort by using the modern higher level programming language, PEARL.

The system PROKON which is programmed almost entirely in PEARL and uses the most modern available hardware and software is described in this paper.

It provides facilities for monitoring signals and measured values, displaying them in a suitable manner, and placing in a history file. The input to PROKON can be provided interactively using display masks.

Keywords

process monitoring
process control
data processing
measurement acquisition
signal acquisition
process displaying
data recording
configuration display
display mask
fault list
message list
maintenance list
semigrafic
fullgrafic

1. Aufgabenstellung - Zielsetzung

Die starke Preisreduktion der Hardware von Prozessrechnern, verbunden mit einer erheblichen Leistungssteigerung und die enorme Erhoehung der Personalkosten gaben den Anstoss fuer die Entwicklung des Systems PROKON.

Unter:

- Ausnutzung modernster Hardware
  - > schneller, leistungsfaehiger ZE´s
  - > grossvolumiger preiswerter Arbeitsspeicher
  - > schneller und sicherer Externspeicher
  - > intelligenter, mc-gesteuerter Datensichtgeraete
  - > vielseitiger Drucker
  - > unterschiedlichster Prozessperipherie
  - > leistungsfaehiger Semi- und Vollgraphik

und unter:

- Ausnutzung modernster Standard-Software
  - > leistungsfaehiger Betriebssysteme
  - > prozessfaehiger, hoeherer Programmiersprache
  - > anpassungsfaehiger Dialogsoftware
  - > schneller Datenverwaltungsverfahren
  - > vielseitiger Dienst- und Hilfsprogramme

sollte erreicht werden:

- groesstmoegliche Portabilitaet
  - > fuer unterschiedlichste Anwendungen
  - > fuer neue Hard- und Softwareprodukte
- hoechste Variabilitaet
  - > bei Erweiterungen
  - > bei Anpassungen
  - > bei Aenderungen
- sichere und einfache Bedienung
  - > fuer Anwender
  - > fuer Inbetriebnahmepersonal
- kuerzeste Einarbeitungszeit
  - > fuer Anwender
  - > fuer Programmpflege-Personal .

2. Hardware

Grundlage von PROKON sind Standard-Geraete aus dem SIEMENS System 300.

Die Mindestausstattung:

- Zentraleinheit R10V
  - > 192 kW (ohne Graphik)
  - > 256 kW (mit Graphik)
- Plattenspeicher 3949
  - > 2 x 13,2 Mbyte netto
  - > benoetigt 10 Mbyte
- Datensichtgeraet 3974R
  - > blockmaskenfaehig
- Matrix-Drucker 3918
- Prozesselement
  - > PE 3600 (1:1) oder
  - > PE 3600 (Matrix) oder
  - > Steuerung S5 (seriell) oder
  - > MC-System SMP oder
  - > Fernwirkanschluss (SEB)

optionell

- Semigraphik-Farbsichgeraet
  - > Darstellung von Anlagenbildern
  - > Darstellung von Kurven
- Vollgraphik
  - > Plotter
  - > Sichtgeraet

3. Standard-Software

Auch hier wurde auf Standard-Produkte fuer SIEMENS Systeme 300 zurueckgegriffen:

ORG 300PV	-	Externspeicherbetriebssystem mit virtueller Speicherverwaltung fuer Prozessrechnerserie 300R
PEARL 300	-	BASIC-PEARL mit Erweiterungen
ASS 300	-	Assembler
FORM 300	-	Erstellung und Interpretation von Blockmasken (Entwicklung der ZN-Nuernberg)
SIGSY	-	Sichtgeraetesystem fuer Semigraphik

GRIBS 300/P - Graphisches Interaktives Basis-System (PEARL)
(Entwicklung der S.E.P.P.-GmbH Roettenbach)

4. Dialog

Von der Eingabe der systemparametrierenden Daten ueber die Eingabe und Pflege aller im System benoetigen Stammdaten bis zur Festlegung des Protokollaufbaues wird PROKON ausschliesslich ueber gefuehrte Blockmasken bedient.

In einer bis zu 4 Stufen tiefen Hierarchie sind dabei alle Masken weitgehendst selbsterklaerend. Alle Eingaben werden, soweit von der Logik her moeglich, ueberprueft.

Bei kritischen Eingaben wird die gewuenschte Operation nur nach Quittung ausgefuehrt.

Archivkorrekturen sind ueber ein individuelles Codewort abgesichert; nur ueber eine zusaetzliche Sicherung koennen Systemgrundeinstellungen durchgefuehrt werden.

Fuer Maskenwechsel u.ae. werden die Kurztelegrammtasten der Datensichtstation benutzt.
Die grundsaetzliche Funktion der Dialoge soll nachstehend an einem Beispiel erlaeutert werden.

4.1 Einrichten oder Aendern eines Messwertes

Am Bedienterminal erscheint nach Neu- oder Wiederanlauf bzw. nach Betaetigung der Kurztelegrammtaste "G" die "Bediensystem Grundmaske Funktionswahl"

```
+-------------------------------------------------------+
I BEDIENSYSTEM Grundmaske Funktionsanwahl              I
+-------------------------------------------------------+

      *** Auswahl der Grundfunktionen ***

      System aktualisieren ..... AKT

      System darstellen ......... DAR

      Daten manuell erfassen ..... ERF

      Systemzeit einstellen ..... UHR

      gewuenschte Funktion : AKT
```

Nach Wahl "AKT" fuer die "Uebersicht Systemaktualisierung" erscheint die naechste Maske:

```
+-------------------------------------------------------+
I  Uebersicht  S y s t e m a k t u a l i s i e r u n g  I
+-------------------------------------------------------+

      *** moegliche Funktionen ***

      Signalverarbeitung ..... SIG
      Messwertverarbeitung ... MEW
      Zaehlwertverarbeitung .. ZAE
      Labordatenverarbeitung . LAB
      Sammelsignale ......... SAM
      Ereigniszaehler ........ EZA
      Wartungszeiterfassung .. WAR
      Berichtsaufbau ......... BER
      Bildvariablen ......... BIL

      gewuenschte Funktion ..... : MEW

      fuer Variable ........... : Druck1

      Stammdaten loeschen (J/N?) : N
```

Hier muss nun zusaetzlich zur Eingabe der gewuenschten Funktion "MEW" fuer die Messwertverarbeitung noch eine bis zu 6 Zeichen lange im System eindeutige Kurzbezeichnung fuer die Prozessendstelle vergeben werden.

Das System prueft, ob dieser "Variablenname" bereits vergeben ist. Bei Neudefinition der Variablen wird diese eingerichtet; sind bereits Daten fuer einen Messwert vorhanden, werden sie in der naechsten Maske der "Aktualisierung Analogstammdaten" automatisch aufgeblendet und koennen somit modifiziert werden.

```
+-------------------------------------------------------+
I   Aktualisierung   A n a l o g s t a m m d a t e n   I
+-------------------------------------------------------+

Symbol ..... : Druck1   Zyklus .. (S/M/L?) : S

Sammelsignal ..... : Hupe     Ereigniszaehler .. : Stat.1

Messstelle ADU/Mnr . : 0 / 112  Messbereich ..... :  7

Anpassung el/phys . :  0 /    0.0  100 /    20.0

Grenzwerte U2 .. O2 :    0.0      2.0      15.0      18.0

zulaessiger Gradient:   1.0   rel. Hysterese  :  20 /1000

aktiviert .. (J/N?) : J
melden ..... (J/N?) : J          Stoermeldung (J/N?) : N

Messwertbezeichnung : Druck 1 hinter Turbine Kessel 1

Einheit ..... : ( bar )
```

Neben allen relevanten Daten dieses Messwertes wird in dieser Maske festgelegt, ob der Messwert bei Grenzverletzung einer der 4 Grenzen, bzw. Ueberschreitung des zulaessigen Gradienten im Melde- oder Stoerprotokoll ausgedruckt wird.

In dieser Maske wird auch bereits eine bis zu 30 Zeichen lange Messwertbezeichnung vergeben (gilt auch fuer alle anderen Arten der Systemdaten).
In allen Protokollen auf Sichtgeraet oder Drucker wird dieser Langtext dann zusaetzlich zur symbolischen Variablenbezeichnung abgelistet und erhoeht somit den Informationsgehalt aller Ausgaben fuer das Bedienungspersonal.

4.2 Weitere Bedienungen

Alle weiteren Bedienungen im Gesamtsystem erfolgen nach dem oben beschriebenen Verfahren.

Ueber Maskendialog koennen z.B. noch folgende Funktionen ausgefuehrt werden:

- Anstoss aller moeglichen Protokolle und Anlagenuebersichten
- Festlegung von Inhalt und Aufbau der Tages-, Monats- und Jahresberichte
- Zuordnung von Prozessgroessen zu Bildvariablen der semigraphischen Anlagendarstellung
- Einstellen der Systemzeit
- Funktionen zur Softwareinstallation und Programmpflege.

5. Prozessdatenerfassung und -verarbeitung

PROKON erfasst und bearbeitet zyklisch bzw. ereignisorientiert bis zu 2048 Binaersignale, 512 analoge Messwerte und 32 BCD-Zaehler. Die erfassten Daten werden von den simultan ablaufenden Erfassungsprogrammen in drei Umlaufpuffern hinterlegt.

Diese Puffer stellen eindeutige Schnittstellen (Binaer-, Zaehl- und Analogwertepuffer, vgl. Bild 1) der Prozessdatenerfassung und der folgenden Prozessdatenverarbeitung dar. Auf diese Weise ist der Austausch von Prozessdatenerfassungskomponenten und die Anpassung an unterschiedlichste Prozessumgebungen problemlos moeglich.

Realisiert sind Erfassungsprogramme fuer eine Vielfalt von Prozessperipherie:

- Digitaleingaben (16 Bit/32Bit, zyklisch und interruptgesteuert)
- Binaersignalbus (Diodenmatrix)
- Analogeingaben (integrierende Erfassung)
- Zaehleingaben (BCD-Zaehler) .

Serielle Schnittstellen sind in Bearbeitung fuer

die Subsysteme:
- S5-Steuerungen
- intelligente Unterstationen wie z.B. SMP (SIEMENS Mikrocomputer Baugruppensystem).

Die Daten in den Eingangspuffern werden von den Verarbeitungsprogrammen in Echtzeit unterschiedlichen Plausibilitaets- und Grenzwertpruefungen unterzogen. Die Verarbeitungsprogramme fuehren die zentralen Prozessabbilder, bilden Betriebs- und Stoermeldungen zur weiteren Auswertung durch das Protokolliersystem (Melde- und Stoerprotokoll) und loesen gegebenenfalls Sammelsignale aus.

Sammelsignale dienen der Konzentration einer Gruppe von Betriebsereignissen (Schaltvorgaenge, Stoerungen usw.) auf ein Signal. Sammelsignale koennen sowohl fuer die Anlagendarstellung genutzt, als auch ueber Digitalausgabe ausgegeben werden (Hupe, Stellsignal, Warte usw.).

Die zentralen Prozessabbilder sind die Datenbasis fuer weitergehende Auswertungen:

- Archivierung
- Anlagendarstellung
- Protokollierung
- Steuerkomponente

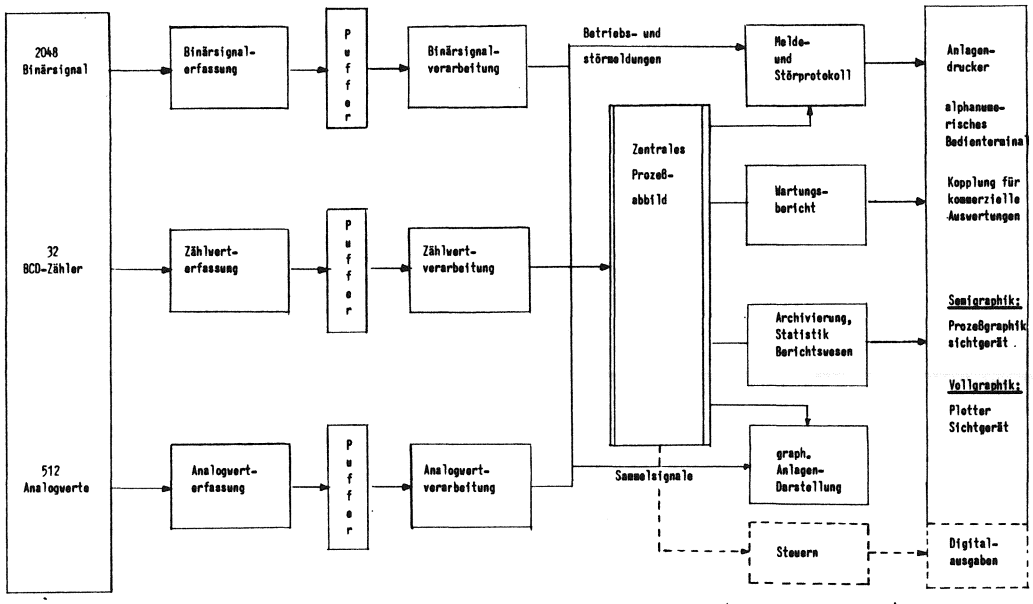


Bild 1 PROKON-Systemübersicht ohne Bediensystem (---- in Bearbeitung)

6. Prozesssteuerung

Um uebergeordnete Steuerfunktionen mit PROKON komfortabel realisieren zu koennen, wird zur Zeit eine Steuerkomponente in das System integriert. Der Sprachumfang, sowie die Notation der Steuersprache wurden stark an die bei den Technologen weit verbreiteten STEP-Sprachen angelehnt.

Der Befehlsvorrat umfasst 25 ausfuehrbare Anweisungen. Ein Steuerprogramm besteht aus bis zu 16 Steuermodulen unterschiedlicher Prioritaet.
Die einzelnen Module koennen miteinander kommunizieren und sich somit gegenseitig koordinieren.

7. Protokollierung

Im System erfolgt, je nach Festlegung bei der Systemparametrierung, schritthaltend der Ausdruck von Betriebs- und Stoermeldungen.
Beide Protokollarten koennen dabei gespoolt werden.

Das per Dialogbedienung parametrierbare Berichtswesen umfasst die Ausgabe von Tages-, Monats- und Jahresprotokollen.
Je nach Wahl werden Stundenwerte, Mittelwerte, Summen, Minima und Maxima ausgegeben.
Die Zusammenfassung verschiedener Zeilen zu Abschnitten, die Vorgabe von Abschnittsueberschriften, die Formatsteuerung u.ae. erfolgt ebenfalls ueber den Protokolldialog.

Der Ueberwachung einzelner Maschinen bzw. ganzer Anlagenteile dient das Wartungsprotokoll, ueber das eine Installations- und/oder Laufzeitueberwachung vorgenommen werden kann.

7.1 Beispiele fuer die Protokollarten in PROKON

Als Beispiel fuer die Darstellung der Protokolle am alphanumerischen Bildschirm soll ein Meldeprotokoll dienen.
Nach Wahl der Funktion "DAR"-System darstellen in der Grundmaske, erscheint die nachfolgende "Systemdarstellung: Grundmaske"

```
+-------------------------------------------------------+
I     Systemdarstellung :   G r u n d m a s k e       I
+-------------------------------------------------------+

          *** Auswahl der Grundfunktionen ***

   Stammdateien und Prozessabbilder ..... : STA
   Tages-, Monats-, Jahresprotokoll ..... : PR1
   Melde-, Stoer-, Wartungsprotokoll ..... : PR2
   Mess- und Zaehlwerte mit  Aktualisierung : AKT
   Einbringen der Werte fuer Aktualisierung : EIN
   Alphabetische Symbolliste ............. : SYM
   Meldeprotokoll-Einstellungen aendern ... : MEL
   Kurvendarstellung ..................... : KUR

   gewuenschte Funktion ..................... : PR2
```

Waehlt der Bediener hier mit "PR2" die entsprechende Protokollart vor, kann er mit der naechsten Maske das Meldeprotokoll und das Ausgabegeraet festlegen.

```
+-------------------------------------------------------+
I Systemdarstellung : Melde-, Stoer-, Wartungsprotokoll I
+-------------------------------------------------------+


          Meldeprotokoll ..... : MEL
          Stoerprotokoll ..... : STO
          Wartungsprotokoll .. : WAR


          gewuenschte Funktion : MEL

Ausgabe auf Drucker0 , Drucker1 , Sichtgeraet (0/1/S ?) : S

( KT + A  :  Abbrechen aller angestossenen Protokolle )
```

Das dann z.B. am Datensichtgeraet vorgelegte Meldeprotokoll zeigt die nachfolgende Hardcopy vom Schirm:

```
+-------------------------------------------------------+
I Systemdarstellung :   M E L D E P R O T O K O L L    I
+-------------------------------------------------------+
--Zeit-- --Bezeichnung-- Signal  Hinweis  -Soll-  -Ist- ( )

17:35:13 Sauerstoffgeh.1 O2GEH1  UNT UG1     1.5    1.4 mg/l
17:35:23 Sauerstoffgeh.2 O2GEH2  UNT UG1     1.5    1.4 mg/l
17:35:24 Inhalt Gasbeh.  NDGASI  UNT UG1   300.0   11.7 m3
17:45:35 Inhalt Gasbeh.  NDGASI  UEB UG1   300.0  354.0 m3
17:46:21 Wasserpumpe 1   BEWAP1  E I N
17:46:25 Wasserpumpe 2   BEWAP2  A U S
18:01:54 Rundraeumer     RRZKB   E I N
 *** ENDE MELDEPROTOKOLL ***
```

```
KT:   + vorwaerts   - rueckwaerts   A Anfang      E Ende
      G Grundmaske  R Vorher. Maske N Neuausgabe
```

(Die Langtextbezeichnung von 30 Zeichen im Originalprotokoll wurde aus drucktechnischen Gruenden verkuerzt.)

Ueber Kurztelegrammtasten kann in den Datenbestaenden beliebig geblaettert werden.

Um einfach und ohne Programmierung jederzeit Messwertkombinationen z.B. bei Stoerungen oder Einstellarbeiten genau ueberwachen zu koennen, wurde in PROKON die Moeglichkeit geschaffen bis zu 9 Bildschirmseiten mit bis zu 15 Messwerten im Dialog einzurichten. Die naechste Maske zeigt dieses "Formular".

```
+-------------------------------------------------------+
I  Mess- und/oder Zaehlwerte einbringen fuer Akt.       I
+-------------------------------------------------------+

          Aktualisierungsseite Nr.: 01

Geben Sie bitte die Mess- und/oder Zaehlwertsymbole fuer
diese Seite ein! (Unterstriche stehen fuer Leerstellen):

ZULAUF    TROPFK    BELSC1    TEMPF1    TEMPF2

______    ______    ______    ______    ______

______    ______    ______    ______    ______


     Naechste Eingabeseite ( 1 ... 9 ) oder
     Einbringen beenden    ( 0 ) ..... : 0
```

Die eingerichteten Messwerttafeln koennen nun jederzeit aufgerufen werden. Das System aktualisiert dann die gewaehlte Messwertkombination im 10 sec.-Zyklus am Bildschirm.

```
+-------------------------------------------------------+
I       Mess- und/oder Zaehlwerte mit Aktualisierung    I
+-------------------------------------------------------+
                    Seite Nr.: 01
--Zeit--  Symbol  --Bezeichnung--  Einheit  Akt.Wert  Hinweis

13:42:41 ZULAUF  Zulaufmenge ...     l/s      573.0
13:42:41 TROPFK  Tropfkoerper ..     l/s      410.0
13:42:41 BELSC1  Belebtschlamm 1     l/s      280.0  UNT UG1
13:42:41 TEMPF1  Temperatur 1 ..   gradC       35.5  UEB OG1
13:42:41 TEMPF2  Temperatur 2 ..   gradC       33.0

Kurztelegramme : KT +
n Neue Seite  /  N Neuausgabe  /  G Grundmaske  /  R Vorg.maske
```

(Die Langtextbezeichnung von 30 Zeichen im Originalprotokoll wurde aus drucktechnischen Gruenden verkuerzt.)

7.2 Beispiele fuer die Selbstdokumentation in PROKON

PROKON ist voll selbstdokumentierend, d.h. alle Systemdaten, von den Prozessgroessen bis zu den eingegebenen Protokollspezifikationen, werden in sinnvoller, den Benutzer unterstuetzender Art auf Wunsch ausgedruckt. Da bei diesen Ausdrucken der eingesetzte Drucker mit bis zu 192 Zeichen/Zeile voll ausgenutzt wird, wurde auf eine Darstellung in dieser Ausarbeitung verzichtet. Die Moeglichkeiten des Systems sollen deshalb die beiden dafuer vorgesehenen Dialogmasken demonstrieren:

```
+-------------------------------------------------------------+
I Systemdarstellung : Alphabetische Symbolliste(n) erstellen  I
+-------------------------------------------------------------+

     Ausgabe auf Drucker (0/1 ?) ..... : 1

     Alle unten aufgefuehrten Sy.listen  : N

              oder alternativ

     Getrennt nach Symbolarten ..... :
     (falls Liste gewuenscht, /J/ eingeben)

                  Analogwert - Symbole : J
                  Binaerwert - Symbole : J
                  Zaehlwert  - Symbole : N
                  Laborwert  - Symbole : N
                  Maschinen  - Symbole : J
                  Sammelsign.- Symbole : N
                  Ereignisz. - Symbole : N

  (Mit KT+A kann ein laufendes Protokoll abgebrochen werden)
```

```
+-------------------------------------------------------------+
I Systemdarstellung : Stammdateien und Prozessabbilder        I
+-------------------------------------------------------------+
( KT + A : Abbrechen aller angestossenen Protokolle )

Ausgabe auf Drucker (0/1 ?) : 1

Stammdaten :
------------
Analogwerte ..... : ANP          Wartungsdaten ... : WAP
Binaerwerte ..... : BIP          Protokoll-
Zaehlwerte ...... : ZWP          Buchhalter ..... : BUP
Laborwerte ...... : LAP          Protokoll-
Sammelsignale .... : SAP         Abschnitte ..... : ABP
Ereigniszaehler .. : EZP

Prozessabbilder :   Analogabbild ..... : AAB
--------------      Zaehlwertabbild .. : ZAB
                    Binaerabbild ..... : BAB

gewuenschte Funktion ..... : ANP
```

8. Archivierung

Eine besonders wichtige Bedeutung kommt in dem Online-System der staendigen Auskunftsbereitschaft zu. Eine begleitende Statistik aus allen erfassten Daten dient der langfristigen Bewertung der Betriebsablaeufe. Alle Daten bleiben - in stufenweise komprimierter Form - ueber drei Jahre gespeichert.
Das Archivierungsprogramm wird viertelstuendlich gestartet, liest das Prozessabbild, bildet Mittelwerte und fuehrt die Archive. Im Tagesarchiv werden sieben Tage lang die Daten der Analog- und Zaehlwerte viertelstundengenau gespeichert. Im Monatsarchiv werden entsprechend tagesgenaue Mittelwerte ueber sechs Monate gehalten. Ueber 3 Jahre hinweg werden Monatsmittelwerte archiviert.

Der Dialogteil "Handerfassung" bietet dem Bediener die Moeglichkeit, automatisch archivierte Werte direkt in den Archiven zu veraendern. Zusaetzlich koennen nicht automatisch messbare Groessen (Laborwerte) in die Archive eingebracht werdem, so dass sie fuer nachfolgende statistische Auswertungen (Berichte, Graphik) zur Verfuegung stehen.
Betriebs- und Stoermeldungen werden im Melde- und Stoerprotokoll auf den Anlagendruckern (abschaltbar) ausgegeben und gleichzeitig auf Externspeicher im Logbuch fuer Melde- und Stoerberichte abgelegt.

In den statischen Bereich faellt weiterhin die Ueberwachung der Betriebs- und/oder Standzeiten von Anlagenteilen. Ein automatisch oder auf Anforderung ausgegebener Wartungsbericht ermoeglicht somit eine ideale Wartungsstrategie.

9. Graphische Analagendarstellung

Zusaetzlich zu den umfangreichen Anlagenuebersichten auf den alphanumerischen Bedienterminals und den Anlagendruckern koennen in verschiedenen Ausbaustufen graphische Systeme angeschlossen werden.

S e m i g r a p h i k (Symbolgraphik):

Das eingebaute Prozessgraphiksystem erlaubt eine komfortable Bilderstellung. Der Anwender kann sich mittels eines interaktiven Bildkonstruktionsprogrammes entsprechende Anlagenbilder (als Hintergrundbilder) selbst definieren und ueber die Hardcopyfunktion auf Drucker ausgeben (s. Bild 2).

Die Bildkonstruktion erleichtern unter anderem die Funktionen:

- Virtuelle Tastatur: die Funktionen des Konstruktionsprogramms werden auf dem Monitor angezeigt und koennen ueber Steuerknueppel aufgerufen werden

- Symbolkonstruktion: 1024 frei konstruierbare Symbole (7x9 Bildpunkte)

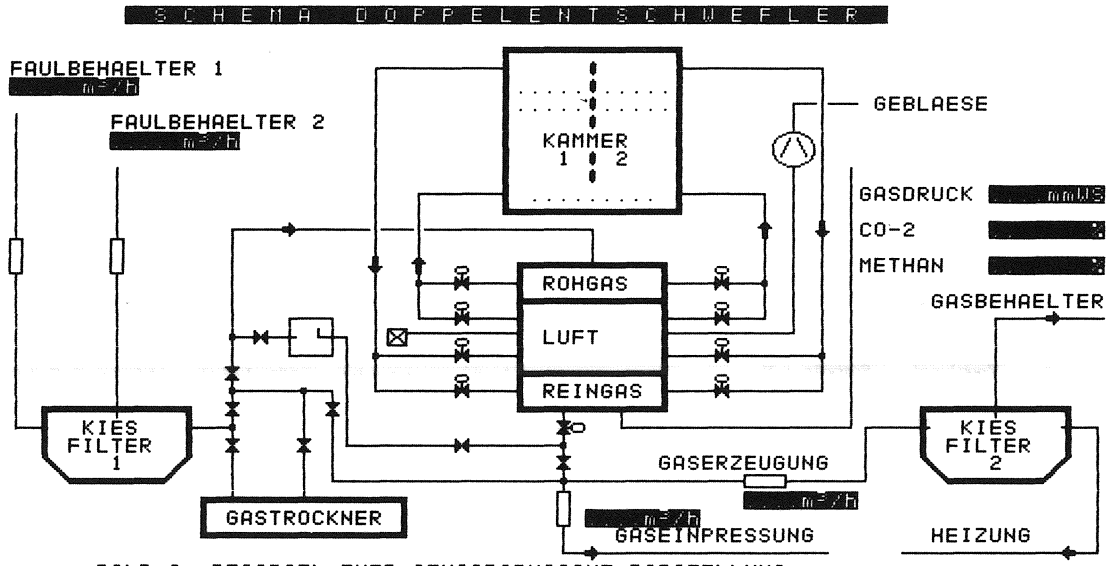


BILD 2. BEISPIEL FUER SEMIGRAPHISCHE DARSTELLUNG

- Farbkomposition: 16 Farben, die in 10 %-Anteilen aus den 3 Grundfarben mischbar sind.

- Konstruktionshilfsmittel:
  > Schieben und Doppeln von Bildausschnitten
  > Farbgebung Symbol/Hintergrund
  > Wiederholungsfunktionen

Der Hintergrundbildkonstruktion folgt die Definition der Bildvariablen, die in Abhaengigkeit vom Prozessgeschehen dynamisch veraendert werden sollen.

Die Verbindung von momentanem Prozessgeschehen zur graphischen Anlagendarstellung kann vom Bediener online erstellt oder geaendert werden. Mittels einer leicht erlernbaren Zuordnungssprache legt er fest, in Abhaengigkeit welcher Prozessvariablen Farb-, Figur- und andere Darstellungsaenderungen im Anlagenbild vorgenommen werden sollen. Die Zuordnungssprache bietet ausser Taschenrechnerfunktionen auch logische Operatoren (AND, OR, NOT, XOR) zur Verknuepfung von bis zu 16 Prozessvariablen bei der Beschreibung einer Bildvariablen.

Diese Anlagenuebersichten ensprechen der sonst ueblichen zentralen Warte, von der aus das Bedienpersonal Schieber, Pumpen usw. steuert, sind aber wesentlich uebersichtlicher z.B. durch individuelle Darstellung einzelner Prozessabschnitte und -ablaeufe sowie durch Darstellung der aktuellen Prozessverte direkt im Anlagenbild - und zwar in Ziffern-, Balken- oder Kurvenform.

V o l l g r a p h i k :

Fuer spezielle graphische Darstellungen (z.B. Kurvenverlaeufe), die eine hoehere Aufloesung der Geraete und somit Vollgraphik erforderlich machen, wird das in PEARL geschriebene, graphische interaktive Basis System GRIBS der Fa. S.E.P.P. - Gesellschaft fuer System-Entwicklung, Prozess-Programmierung und Computer Graphik mbH in Roettenbach eingesetzt. GRIBS ist eine Implementierung von Funktionen des Graphischen Kern-Systems GKS (DIN-7942) und erlaubt somit eine geraeteunabhaengige Programmierung von graphischen Darstellungen.
In PROKON koennen zur Zeit alle in den Archiven gespeicherten Daten in Form von kommentierten Kurvenverlaeufen auf zu Tektronix kompatiblen Sichtgeraeten und auf einen Mehrfarbenplotter ausgegeben werden.

10. Programmieraufwand

Die Verwendung von PEARL 300 fuehrte in allen Projektphasen (Entwurf, Codierung, Test) zu erheblichen Zeit- und damit Kostenersparnissen. Bereits in der Entwurfsphase wurden saemtliche Datenstrukturen, Zugriffsoperationen, Dateierklaerungen usw. sowie die Grobstrukturen der einzelnen Task's in PEARL oder PEARL-aehnlicher Notation festgeschrieben. Dies fuehrte zu klaren

Daten- und Programmschnittstellen und somit zu kleinen, getrennt codier- und testbaren Systembausteinen. Neber den als bekannt vorausgesetzten Faehigkeiten von PEARL bei der Programmkoordinierung fuehrten vor allem das Vorhandersein eines Testsystems und die Moeglichkeit zum Aufruf von Assemblerunterprogrammen zu niedrigen Gesamtkosten. Assembler wurde auschliesslich dann eingesetzt, wenn es galt bereits vorhandene Bausteine oder Standardsysteme zu benutzen und es teuerer geworden waere, diese Funktionen in PEARL neu zu programmieren.

An der Realisierung des Projektes waren drei Mitarbeiter der ZN-Nuernberg beteiligt, wobei es fuer zwei Mitarbeiter den ersten Einsatz von PEARL bedeutete. Das Projekt startete Mai 1981. Die erste Auslieferung des Systems PROKON fand noch im Dezember 1981 statt.

Die Auslieferung von PROKON erfolgt in der Form eines 'leeren' Rahmensystems, welches vom Anlagenpersonal parametriert wird. Auf Wunsch kann die Parametrierung, falls entsprechende Stammdaten vorliegen, auch bereits vor der Installation erfolgen. Waehrend der Installationsphase entsteht somit lediglich Parametrier- und keinerlei Programmieraufwand.

11. Programmtechnische Daten des Systems

Die wichtigsten Kenndaten von PROKON sind je nach Ausbaustufe bis zu

- 46 Task's
- 40 Masken
- 53 mit PEARL verwaltete Systemdateien.

Der Quellcode umfasst ca. 40.000 Quellzeilen.

Verfasser: Ruediger Brehm
Robert Jaeckel
Erich Rausch

SIEMENS AG
Zweigniederlassung Nuernberg
Vertrieb Energietechnik und
Automatisierungstechnik Systeme

Von-der-Tann-Str. 30

8500 Nuernberg

Tel. 0911/654-3785

Ein Vergleich von FORTRAN IV und PEARL an Hand eines umfangreichen Programmes

Burkhard Menzel, Bielefeld

1. Zusammenfassung

Es wurden gleiche Programme zur Textverarbeitung in PEARL und in FORTRAN IV erstellt. Diese Anwendung von PEARL in der "klassischen" Datenverarbeitung zeigte im Vergleich zu FORTRAN IV folgende Ergebnisse:

Die Laufzeiten der FORTRAN- und PEARL-Programme waren praktisch gleich.

Der Quellcode der FORTRAN-Programme enthielt ca. 3,5 mal soviele Zeilen wie die PEARL-Programme.

Die Untersuchung zeigt, daß PEARL wegen seiner besseren Eignung zur struktuierten Programmierung auch für Probleme der "klassischen" Datenverarbeitung bei weitem besser geeignet ist als FORTRAN IV.

2. Einleitung

In der Regel werden zwei verschiedene Programmiersprachen auf der Basis verglichen, daß die einzelnen Sprachelemente gegenübergestellt werden. Erheblich seltener wird man ein und dasselbe Programm in zwei Sprachen auf der gleichen Anlage benutzen. Es soll hier gezeigt werden, was beide Sprachen bei der Lösung eines Textverarbeitungsproblems zu leisten vermögen.

3. Die Aufgabe

Die Aufgabe bestand darin, einen sog. PREPROZESSOR und weitere Hilfsprogramme für die TOP-DOWN-Entwicklung von FORTRAN- und PEARL-Programmen in FORTRAN IV zu schreiben. Die FORTRAN IV Programme sollten in Anlehnung an bereits vorhandene PEARL-Programme erstellt werden. Um das Textverarbeitungsproblem, das der PREPROZESSOR löst, genauer verstehen zu können, soll zunächst das Verfahren zur TOP-DOWN-Entwicklung von Programmen näher beschrieben werden.

3.1 Beschreibung des Programmier-Verfahrens

Das Verfahren (1), (2) dient zur TOP-DOWN-Entwicklung von Programmen und verzichtet auf graphische Entwurfs -Hilfsmittel, wie Strukturdiagramme und ähnliches.

Die Programmentwürfe werden direkt in den Rechner eingegeben und bilden gleichzeitig die vollständige Programm-Dokumentation.

Bei der Entwicklung eines Programmes hat man zu Beginn meist nur eine vage Vorstellung von der Lösung des Problems. Man kann aber schon in dieser Phase das Programm in Abschnitte teilen, wie z. B. Deklaration der Datenbasis, Vereinbarung von Prozeduren, Beschreibung des Tasking (Abb. 1). Diese Abschnitte können dann durch schrittweise Verfeinerung in Unterabschnitte aufgeteilt und dadurch genauer erklärt werden.

```
1.      PROGRAM HAUPT
2.      /*
3.
4.
5.      #&S     SPEZIFIKATIONEN
6.      #D      DEKLARATIONEN
7.      #F      FORMAT-ANWEISUNGEN
8.      #P      PROGRAMM
9.      #&M     FEHLERMELDUNGEN
10.     #&T     TECHNISCHE-DATEN
11.     #&L     LOGBUCH
12.     */
13.     END
14.     /*
15.
```

```
221.#P      PROGRAMM
222.========================
223.
224.        #PI     INITIALISIEREN
225.        #P1     MELDE BEREIT, ERFRAGE UND BILDE DATEI-NAMEN
226.        #P2     HOLE BEARBEITUNGSBEFEHLE UND BILDE
227.                ZIELDATEINAMEN
228.        #P3     EROEFFNE DATEIEN
```

```
253.
254.#P1     */
255.        WRITE(TKAN,2011)
256.        CALL DIALG
257.        /*
258.
259.#PI     */
260.        CALL DATIN
261.        ZEINR=0
262.        GRBFEL=.FALSE.
263.        ADRBUL=.FALSE.
264.        /*
265.
266.#P2
267.        #P21    HOLE BEFEHLE
268.        #P22    UNTERSCHEIDE TESTVERSION UND ENT-
269.                GUELTIGES PROGRAMM DURCH VERSCHIEDENE
270.                VERSIONSNUMMERN.
271.
```

Abb. 1 Beispiel für die Aufteilung des Programms in Abschnitte

Die TOP-DOWN-Entwicklung besteht nun darin, daß zunächst die Abschnitt-Überschriften (aus gleich erkennbarem Grund als Pseudocodes bezeichnet) in der Reihenfolge notiert werden, in der die Abschnitte selbst später vom Compiler benötigt werden. Die

einzelnen Abschnitte selbst werden dann in beliebiger, jedoch von der Logik des Entwurfsprozesses diktierter Reihenfolge weiterbehandelt:

Sie werden in Unterabschnitte aufgeteilt, von denen zunächst wieder nur die Pseudocodes (Abschnitt-Überschriften) notiert werden. Die Unterabschnitte werden wiederum in Unterabschnitte aufgeteilt usw.

Bei diesem Prozeß der schrittweisen Verfeinerung können auf jeder Ebene schon Anweisungen der Programmiersprache eingestreut werden.

Ein Programm-Entwurf ist vollständig ausgearbeitet, wenn jeder Pseudocode seine Entsprechung in einem Code-Abschnitt der Programmiersprache gefunden hat. Wie gesagt, stehen die Verfeinerungen in scheinbar beliebiger Reihenfolge. Sie werden daher mit Hilfe von Schlüsseln (Kennziffern) mit ihren zugehörigen Pseudocodes assoziiert. Unter der oben erwähnten Voraussetzung, daß die Pseudocodes in der vom Compiler benötigten Reihenfolge stehen, kann dann durch Umsortieren der Zeilen ein kompilierbares Programm erzeugt werden. Dieses Umsortieren erledigt der erwähnte PREPROZESSOR.

Die Aufgabe des PREPROZESSORS ist es dabei, hinter jedem Pseudocode die zugehörige Verfeinerung einzuordnen.

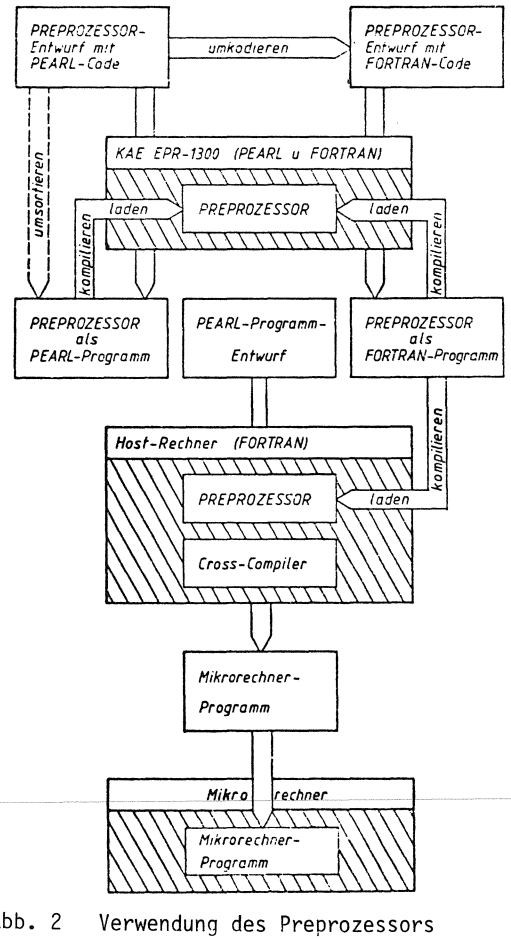


Abb. 2 Verwendung des Preprozessors

3.2 Entstehungsgeschichte der Programme

Ausgangspunkt für die Arbeit war die Aufgabe, daß PEARL-Programme für Mikroprozessoren auf einem Host-Rechner mit Hilfe des oben beschriebenen Entwurfsverfahrens erstellt werden sollten (Abb. 2). Der Cross-Compiler und alle weiteren Hilfsprogramme waren in FORTRAN IV geschrieben. Da zur Programmierung des Host-Rechners PEARL (wegen Fehlens eines Compilers) nicht verwendet werden konnte, mußte auch der PREPROZESSOR in FORTRAN IV vorliegen. Das bedeutete, daß ein bereits in PEARL vorhandener PREPROZESSOR nach FORTRAN IV umkodiert werden mußte.

Der in PEARL erstellte PREPROZESSOR existiert auf dem Rechner Krupp-Atlas Elektronik EPR 1300, auf dem auch der FORTRAN-PREPROZESSOR entworfen wurde. Der Prozessrechner EPR 1300 besitzt einen FORTRAN 77 Compiler, es wurden allderdings aus Kompatibilitätsgründen keine Sprachelemente benutzt, die über FORTRAN IV hinausgehen, wie z. B. die FORTRAN 77 CHARACTER-Verarbeitung.

Zwei PREPROZESSOR-Versionen waren zu erstellen:

1. Ein PREPROZESSOR mit Fehlerprüfung, z. B. richtige Verwendung der Kommentarzeichen, Schlüssel paarweise, ...
2. Ein PREPROZESSOR ohne diese Fehlerprüfung, der dadurch eine verkürzte Laufzeit hat.

3.3 Aufbau der FORTRAN-Programme

Der PEARL-PREPROZESSOR wurde etwas modifiziert, so daß er auch FORTRAN-Programme verarbeiten kann, damit die FORTRAN-PREPROZESSOREN mit Hilfe des beschriebenen Entwurfsverfahrens erstellt werden konnten.

Auch die Programmstrukturen wurden weitestgehend aus den PEARL-Programmen übernommen. Abweichungen gibt es nur an den Stellen, wo beide Programmiersprachen soweit verschieden sind, daß keine gleichwertigen FORTRAN-Statements vorhanden waren. Das betrifft in erster Linie die CHARACTER-Verarbeitung und die Ein-/Ausgabe, sowie Positionierungsprobleme innerhalb der Dateien.

Um auch in FORTRAN IV eine gute und übersichtliche Struktur beizubehalten, wurden ungefähr genauso viele Unterprogramme benutzt wie in PEARL. Das erforderte die Deklaration von COMMON-Blöcken, die in jedem Unterprogramm (nicht immer vollständig) wiederholt werden mußten. In der Regel wird man solche Deklarationen mit Hilfe eines Editors in die

Unterprogramme kopieren. Das Mehrfach-Deklarieren führt bei späteren Änderungen jedoch schnell zu Fehlern.

4 Vergleiche

4.1 Längenvergleiche

Der Entwurf des PEARL-PREPROZESSORS mit Fehlerprüfung ist 1504 Zeilen lang und enthält dabei 703 Zeilen mit reinem PEARL-Code. Der Rest besteht aus etwa 250 teilweise mehrzeiligen Pseudocodes (Quasi Kommentare) und Leerzeilen zur optischen Aufgliederung des Programmes (Tabelle 1). Der entsprechende FORTRAN-PREPROZESSOR weist dagegen eine Länge von 5571 Zeilen mit 2517 FORTRAN-Zeilen auf. Das bedeutet, daß der FORTRAN-PREPROZESSOR 3,7 mal so lang ist wie das gleichwertige PEARL-Programm. Setzt man die Anzahl der Zeilen mit reinem FORTRAN-Code ins Verhältnis zu den Zeilen, die PEARL-Code enthalten, so erhält man den gleichen Faktor.

	PREPROZESSOR mit Fehlerprüfung		*PREPROZESSOR ohne Fehlerprüfung*	
Texte + Statements	*PEARL*	*FORTRAN*	*PEARL*	*FORTRAN*
	1504	*5571*	*842*	*3065*
Statements	*703*	*2517*	*326*	*1151*

Tabelle 1 Tabelle zu den Längenvergleichen

Daß das keine Zufall ist, bestätigt der Vergleich der beiden PREPROZESSOR-Versionen ohne Fehlerprüfung. Der FORTRAN-Programmentwurf ist 3,6 mal so lang wie der entsprechende PEARL-Entwurf. Bei der Anzahl der Zeilen, die Statements enthalten,ergibt sich der Faktor 3,5 (FORTRAN:PEARL).

Die Länge der FORTRAN-Programme hat folgende Ursachen:

1. Mehrfach-Deklarationen der COMMON-Blöcke.

   In PEARL wurde das ganze Programm in ein einziges MODUL geschrieben. Variable und Konstanten, die über die Grenzen eines Unterprogramms bzw. einer TASK hinaus bekannt sein sollten, wurden im PROBLEM-Teil deklariert. Diese Deklarationen brauchten hier nur ein einziges Mal niedergeschrieben werden. Damit waren alle dort aufgeführten Größen im gesamten MODUL bekannt. Bekanntlich wird in FORTRAN IV jedes Unterprogramm für sich übersetzt. Daten, die von mehreren Unterprogrammen gemeinsam benutzt werden, müssen in COMMON-Blöcke geschrieben werden, wenn man auf die Übergabe mittels Parameterlisten verzichten will. Die Deklarationen dieser COMMON-Blöcke müssen grundsätzlich in jedem Unterprogramm, in dem die globalen Größen verarbeitet werden sollen, erneut niedergeschrieben oder mit einem Editor kopiert werden.
   Diese Vorschrift erhöht die Länge einiger Dateien beträchtlich. Es sind hier oft um die 100 Programmzeilen. Bei kurzen Unterprogrammen mit z. B. 20 ausführbaren Anweisungen ist die Benutzung solcher COMMON-Block-Deklarationen dafür verantwortlich, daß diese Dateien erheblich länger sind als ihre PEARL-Äquivalente.

   Würde man in PEARL ein Programm aus mehreren MODULEN aufbauen, so müßten in den Spezifikationen solche globalen Daten ebenfalls aufgeführt werden. PEARL hat jedoch den Vorteil, daß man in den Spezifikationen nur solche Größen beschreiben muß, die auch wirklich benutzt werden. In FORTRAN würde man das nur dadurch erreichen können, daß für jedes Datum ein eigener COMMON-Block geschaffen werden müßte.

   Ein Beispiel soll das veranschaulichen:

   Der PREPROZESSOR mit Fehlerprüfung arbeitet mit 19 Unterprogrammen. Nehmen wir an, daß in jedem Unterprogramm 70 Zeilen benötigt werden, um die nötigen COMMON-Blöcke zu deklarieren, (dieser Wert ist für das gegebene Beispiel sehr realistisch.) Wie sich leicht errechnen läßt, entfallen auf die Deklarationen innerhalb der Unterprogramme 1330 Zeilen.

2. Aufwendigere Zeichenverarbeitung

   In FORTRAN IV wurde eine Zeile in ein INTEGER-Feld eingelesen. Dadurch war es nicht möglich, Zeichenketten so einfach wie in PEARL zu bearbeiten. Um Zeichenketten herauszugreifen, zu vergleichen oder zu manipulieren mußten DO-Schleifen benutzt werden.

3. Fehlen moderner Sprachelemente

   Konstruktinen wie 'REPEAT-WHILE'-Schleifen oder Case-Anweisungen sind in FORTRAN IV nur durch erhöhten Programmaufwand zu realisieren.

4.2 Laufzeitvergleiche und Speicherplatzbedarf

Die Messungen wurden auf dem KAE-Prozeßrechner EPR 1300 durchgeführt. Dieser Rechner arbeitet normalerweise im Time-Sharing-Betrieb, jedoch nahm beim Aufzeichnen der Meßwerte kein anderer Benutzer am Rechnerbetrieb teil. Die PEARL- und FORTRAN-Programme waren praktisch gleichschnell (FORTRAN ca. 4 % schneller).

Aus diesem Sachverhalt kann jedoch keine pauschale Schlußfolgerung gezogen werden. Die Meßwerte be-

treffen nur die EPR 1300 und die hier beschriebene Programmieraufgabe und werden von Anlage zu Anlage anders sein. Dazu noch eine Anmerkung:

Die FORTRAN-PREPROZESSOREN benutzten zum Positionieren innerhalb der Input-Datei eine Anweisung an das Betriebssystem, die dem Find-Statement ähneln dürfte, das auf einigen Anlagen implementiert ist. (In PEARL wurde mittels der SKIP-Anweisung positioniert.)

In den ursprünglichen Versionen der FORTRAN-Programme, die diesen maschinenspezifischen Trick nicht benutzten, wurde durch entsprechende DO-Schleifen-Konstruktionen positioniert. Das hatte zur Folge, daß die Programme um bis zu 30 % langsamer waren als die jetzigen Versionen und deren PEARL-Äquivalente.

Das Beispiel zeigt, daß die Laufzeiten eines Programmes extrem stark von der Verfügbarkeit eines einzigen Statements abhängen können.

Zum Speicherplatzbedarf ist noch folgendes zu bemerken:

Die FORTRAN- und PEARL-Programme liefen hier auf einem Mehrbenutzer-Betriebssystem mit PEARL-Tasking, das sicher umfangreicher ist als ein Mehrbenutzer-Betriebssystem nur für FORTRAN. Andererseits lädt dieses für PEARL entwickelte Betriebssystem für mehrere PEARL-Benutzer das PEARL-Laufzeitsystem nur einmal, während die FORTRAN-Laufzeitroutinen normalerweise zu jedem Benutzerprogramm dazugebunden werden. Deshalb ist auf diesem speziell für PEARL entwickeltem System der Speicherverbrauch für mehrere PEARL-Benutzer kleiner als für mehrere FORTRAN-Benutzer, die ähnliche Aufgaben lösen.

Literatur:

Weiterführende Literatur zum Thema PREPROZESSOR:

{1}

Top-Down-Spezifikation, -Entwicklung und Dokumentation von PEARL-Modulen.

L. Frevert
PEARL-Rundschau Band 2, Nr. 6, S. 53 (Dez. '81)

{2}

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

L. Frevert

PEARL-Rundschau Band 2, Nr. 3, S 55 (Sept. '81)

Alle übrigen Daten stammen aus einer Diplom-Arbeit an der Fachhochschule Bielefeld.

Anschrift des Autors

Burkhard Menzel
Haspelstr. 25
4800 Bielefeld 1

Untersuchungen zur Elimination des Rechenzeiteinflusses auf die Regelgüte in Prozeßregelungssystemen unter Verwendung der prozeßorientierten Programmierungssprache PEARL

Dipl.-Ing. **U. Mohr**, Erlangen; Prof. Dr.-Ing. **D. Popović**, Bremen; Dr.-Ing. **G. Thiele**, Bremen

Zusammenfassung

Die notwendige Zeit zur Berechnung der Stellgröße wirkt sich in der Verschlechterung der Regelgüte aus, insbesondere wenn eine Software-Gleichkommaarithmetik angewendet wird. Dies kann nachträglich berücksichtigt werden, wenn die Stellgröße mit Hilfe eines Prädiktor- oder Filter-Algorithmus berechnet wird. Dadurch kann - bis auf die Verzögerung durch das Betriebssystem - die Stellgröße unverfälscht ausgegeben werden. Die Untersuchungen hierzu wurden auf dem Prozeßrechner PDP-11/45 mit der BBC-PEARL-Version PAS 2 durchgeführt.

Schlüsselworte: PEARL, Abtastregelung, Kalman-Filter, Prädiktor

Summary

The time necessary for computation of the actuating value of the controller can deteriorate its performance-index, especially when a floating point software package is used for computation. This can be compensated by computing the control-signal by the use of a predictor or of a filter algorithm. In this way the actuating value can be outputted with no distortion save the one caused by the time-delay due to the operating system. The relevant studies were done by the use of a PDP-11/45 process-computer, programmed in the BBC-version of PEARL.

Key words: PEARL, DDC, Kalman-Filter, Predictor

Einführung

Mit der Entwicklung moderner Systemtheorie entstand der Bedarf an der unmittelbaren Beobachtung sog. Zustandsgrößen des zu regelnden bzw. steuernden dynamischen Systems. Die optimale Steuerung und Regelung von Mehrgrößensystemen stützt sich auf die Zustandsraum-Methode, die die Rückführung des gesamten Zustandsvektors verlangt, der bei technischen Systemen nicht direkt meßbar ist. Deswegen werden die sog. Beobachter benötigt, die an Hand des gemessenen Ausgangssignals des Systems seinen Zustandsvektor errechnen. Sind Störgrößen bzw. Meßrauschen vorhanden, so wird die rein deterministische Beobachtungsaufgabe eine statistische Komponente enthalten, die zum Einsatz eines Zustandsschätzers führt. Die Aufgabenstellung dieser Arbeit besteht in der Beobachtung der "realen technischen Welt", d.h. eines technischen Prozesses, mit Hilfe eines Prozeßrechners (PDP-11/45), um den technischen Prozeß zu regeln. Zur Implementierung dieser Aufgabe wird die Realzeit-prozeßorientierte Programmierungssprache PEARL (BBC-Version PAS 2) eingesetzt. Der technische Prozeß selbst (die Lageregelung mit einem Gleichstrommotor) wird auf einem Analogrechner (RA 770) nachgebildet.

Um der meßtechnischen Realität näher zu kommen, werden dem Prozeßrechner verrauschte,d.h. stochastisch gestörte Meßsignale seitens des Analogrechners geliefert, so daß hier eine statistische Lösung der Zustandsbeobachtung notwendig ist. Als Zustandsbeobachter wird der Kalman-Filter gewählt, und zwar zwei seiner Versionen: der Prädiktor und der Filter selbst.

Die Programmierungssprache PEARL wurde gewählt, weil sie blockorientiert und somit relativ einfach zu programmieren ist. Andererseits bietet die Sprache die Möglichkeit der Programmierung von parallelen Realzeit-Prozessen und deren zeitliche Einplanung und, was nicht vernachlässigt werden darf, der Selbstdokumentation. Andererseits sollten dadurch einige als nachteilhaft zu betrachtende PEARL-Eigenschaften näher untersucht werden, wie z.B.

die mögliche Verringerung der Optimalität der Programme nach ihrer Übersetzung und der Verlängerung der Rechenzeit der programmierten Algorithmen durch das benötigte Betriebssystem.

Mathematische Problemdefinition

Die im vorhergehenden Abschnitt bereits gestellte Aufgabe wird, wie im Bild 1 gezeigt, mathematisch definiert.

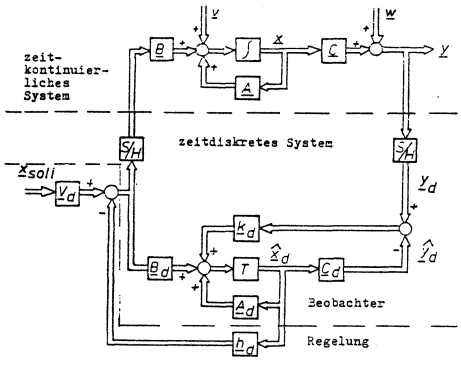


Bild 1: Darstellung der Zusammenschaltung eines Beobachters und des zu beobachtenden Systems

Der technische Prozeß wird als ein dynamisches Mehrgrößensystem im Zustandsraum durch

$$\dot{\underline{x}}(t)=\underline{A}(t)\cdot\underline{x}(t)+\underline{B}(t)\cdot\underline{u}(t)+\underline{v}(t) \qquad (1)$$

und

$$\underline{y}(t)=\underline{C}(t)\cdot\underline{x}(t)+\underline{w}(t) \qquad (2)$$

dargestellt, wobei $\underline{x}(t),\underline{u}(t),\underline{y}(t),\underline{w}(t)$ und $\underline{v}(t)$ der Zustands-, Eingangs-, Ausgangs-, Meßstörungs- und Systemstörungsvektor, und $\underline{A}(t)$, $\underline{B}(t)$ und $\underline{C}(t)$ die System-, Steuer- und Beobachtungsmatrix sind.

Da die Ein- und Ausgangsgrößen des Systems vom Prozeßrechner fortlaufend abgetastet werden, so benötigt man seine zeitdiskrete Beschreibung im Zustandsraum:

$$\underline{x}(k+1)=\underline{A}(k)\cdot\underline{x}(k)+\underline{B}(k)\cdot\underline{u}(k)+\underline{v}(k) \qquad (3)$$

und

$$\underline{y}(k)=\underline{C}(k)\cdot\underline{x}(k)+\underline{w}(k)\ , \qquad (4)$$

die im Prozeßrechner als ein Programm implementiert wird.

Die Aufgabe des Rechners ist:

1. den Schätzwert des Zustandsvektors $\underline{x}(k)$ fortlaufend zu errechnen, d.h. für $k=1,2,3,\ldots$, und
2. durch Rückführung des errechneten Zustandsvektors eine Regelung des Systems durchzuführen.

Die Zustandsgrößen des gewählten technischen Systems, d.h. des Gleichstromantriebes, (Bild 2) sind

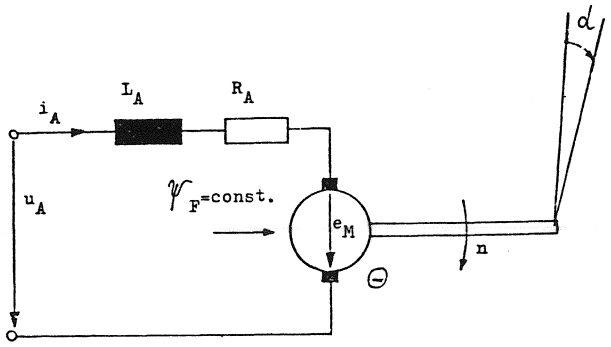


Bild 2: Schematische Darstellung eines Gleichstromantriebes

der Ankerstrom i_A, die Drehzahl n, die Lage und die Stellgröße u_A. Die mathematischen Beziehungen zwischen diesen sind

$$\dot{i}_A=-\frac{c\,\Psi_F}{L_A}\cdot n-\frac{R_A}{L_A}\cdot i_A+\frac{1}{L_A}\cdot u_A \qquad (5)$$

$$\dot{n}=\frac{k\cdot\Psi_F}{\Theta}\cdot i_A \qquad (6)$$

$$\dot{\alpha}=k_\alpha\cdot n, \qquad (7)$$

wobei R_A u. L_A den Widerstand und Induktivität des Ankerkreises, Ψ_F den als konstant angenommenen Fluß der Erregungswicklung und c,k und k_α gewisse Konstante darstellen. Im Zustandsraum kann das System durch

$$\dot{\underline{x}}(t)=\begin{bmatrix}-\frac{R_A}{L_A} & -\frac{c\,\Psi_F}{L_A} & 0\\ \frac{k\,\Psi_F}{\Theta} & 0 & 0\\ 0 & k_\alpha & 0\end{bmatrix}\underline{x}(t)+\begin{bmatrix}\frac{1}{L_A}\\ 0\\ 0\end{bmatrix}u_A(t) \qquad (8)$$

und

$$y(t)=\begin{bmatrix}0 & 0 & 1\end{bmatrix}\cdot\underline{x}(t) \qquad (9)$$

beschrieben werden, wobei der Zustandsvektor $\underline{x}(t)$ wie folgt definiert ist

$$\underline{x}(t)=(i_A \quad n \quad \alpha)^T\ . \qquad (10)$$

Als Beobachter (Bild 1) wurde hier, bedingt durch die verrauschten Meßdaten, der Kalman-Filter gewählt.

Problemlösung

Bei dem Einsatz des Kalman-Filters zur Errechnung der Stellgröße eines Regelsystems mit dem geschätzten Wert des Zustandsvek-

tors in der Rückführung kann man folgende zwei Algorithmen unterscheiden:

1. den Prädiktor-Algorithmus, bei dem die Stellgröße u(k) nur von dem vorhergesagten Schätzwert $\hat{\underline{x}}^*(k)$ abhängig ist, d.h. es gilt

$$u(k)=V\alpha_{soll} - \underline{h}^T\cdot\hat{\underline{x}}^*(k) \qquad (11)$$

mit $\underline{h}$ als Rückführungsvektor und

2. den Filter-Algorithmus, bei dem der Meßwert y(k) zusätzlich berücksichtigt wird, so daß die Stellgröße u(k) unter Ansatz des korrigierten Schätzwertes $\hat{\underline{x}}(k)$ berechnet wird als

$$u(k)=V\alpha_{soll} - \underline{h}^T\cdot\hat{\underline{x}}(k) \qquad (12)$$

Vorteil des erstgenannten Algorithmus ist die kurze Zeitspanne Δt, die zwischen der Messung und der Ausgabe der Stellgröße liegt (Bild 3). Von Nachteil ist die große Varianz des Schätzfehlers des vorhergesagten Schätzwertes $\hat{\underline{x}}^*(k)$, da die Einwirkung einer Störung erst beim übernächsten Schritt berücksichtigt wird (Bild3). Dagegen vergeht zur Korrektur des vorhergesagten Schätzwertes $\hat{\underline{x}}^*(k)$ und zur Berechnung der Stellgröße u(k) beim reinen Filter-Algorithmus eine recht große Zeitspanne, in der sich natürlich das zu regelnde System weiterbewegt (Bild 3). Hier wird jedoch der Schätzwert mit weit kleinerer Fehlervarianz errechnet, da die Einwirkung einer Störung bereits im nächsten Abtastschritt berücksichtigt wird (Bild 3).

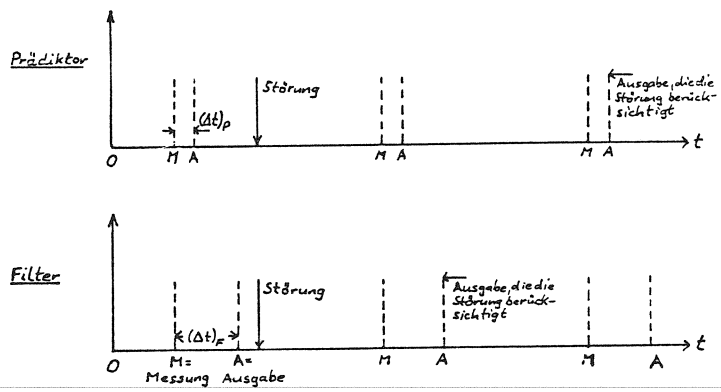


Bild 3: Zeitdiagramm des Prädiktors und des Filters

Besondere Vorteile des durch Prädiktion korrigierenden Kalman-Filters werden aus folgenden Überlegungen deutlich. Bei dieser Filter-Version wird die Stellgröße u(k) nicht zum Zeitpunkt t_k, sondern um Δt verzögert ausgegeben, bis zu welchem Zeitpunkt der Wert u(k-1) gilt, d.h. es gilt, generell,

$$\hat{\underline{x}}^*(k+1)=\underline{A}(t_{k+1},t_k)\hat{\underline{x}}(k)+\underline{B}(t_k+\Delta t,t_k)u(k-1)+ \\ +\underline{B}(t_{k+1},t_k+\Delta t)\cdot u(k). \qquad (13)$$

Sind die Stellgrößenamplituden relativ klein, so kann der Einfluß von Δt vernachlässigt werden; trotzdem ist aber die Korrektur nach (13) problematisch. Um die auszugebende Stellgröße dem Uhrtakt des Betriebssystems von 5 ms anzupassen, wird die Bewegung des Systems im Intervall $[t_k, t_{k+1}]$ in die Abschnitte $[t_k,t_k+\Delta t]$ und $[t_k+\Delta t,t_{k+1}]$ aufgeteilt und geschrieben

$$\hat{\underline{x}}^*(t_k+\Delta t)=\underline{A}(t_k+\Delta t,t_k)\hat{\underline{x}}(t_k)+\underline{B}(t_k+\Delta t,t_k)\cdot \\ \cdot u(k-1) \qquad (14)$$

und

$$\hat{\underline{x}}^*(t_{k+1})=\underline{A}(t_{k+1},t_k+\Delta t)\hat{\underline{x}}(t_k+\Delta t)+ \\ +\underline{B}(t_{k+1},t_k+\Delta t)\cdot u(k). \qquad (15)$$

Somit wird die aktuelle Stellgröße u(k) mit Hilfe von (11) und (14) errechnet und ausgegeben, und anschließend die Bewegung der Regelstrecke im Zeitintervall $[t_k+\Delta t,t_{k+1}]$ geschätzt.

Bei der Feststellung des Ausgabezeitpunktes von u(k) muß berücksichtigt werden, daß

1. der geschätzte Zustandsvektor korrigiert,
2. bis zur Ausgabe eine Prädiktion durchgeführt und
3. die Stellgröße berechnet werden muß.

Diese zur Korrektur notwendige Rechenzeit kann mit Hilfe der TIME-Prozedur bestimmt werden. In unserem Beispiel der Lageregelung beträgt sie 45 ms. Unsererseits wurde eine Verzögerung von 50 ms gewählt, um eventuell noch die Abspeicherung von Zwischenergebnissen zu ermöglichen. Unter Berücksichtigung der Verzögerung der D/A-Wandler-Ausgabe durch das Betriebssystem ergibt sich schließlich eine Aufteilung des gewählten Zeitintervalles $t_k-t_{k-1}=1s$ in t=50,35 ms und $t_k-t_{k-1}-\Delta t$=949,65 ms. Zusammenfassend ausgedrückt wird zum Zeitpunkt t_k zur Korrektur von $\hat{\underline{x}}^*(t_k)$ durch

Messung

$$\hat{\underline{x}}(k)=\hat{\underline{x}}^*(k)+\underline{K}(k)\left[y(k)-\underline{c}^T\,\hat{\underline{x}}^*(k)\right] \quad , \qquad (16)$$

für die Prädiktion bis zum Ausgabezeitpunkt (15) und zur Berechnung der Stellgröße

$$u(t_k+\Delta t)= V\cdot\alpha_{soll}- \underline{h}^T\hat{\underline{x}}^*(t_k+\Delta t) \qquad (17)$$

benutzt. Anschließend wird gemäß (16) die Vorhersage des Zustandsvektors für den nächsten Abtastzeitpunkt t_{k+1} berechnet. Bei den Berechnungen wurden zur Ermittlung des Fehlerverstärkungsfaktors K(k) die von Kalman definierten Gesichtspunkte berücksichtigt.
Es ist zu bemerken, daß die Berechnungen nach (14) ,(16) und (17) innerhalb von Δt abgeschlossen werden müssen, und daß anschließend nach (15) eine Prädiktion bis zum nächsten Abtastpunkt vorgenommen werden muß. Dabei kann die Berechnung von $\underline{K}(k)$ auch off-line erfolgen.

Experimentelle Ergebnisse

Wie bereits erwähnt, stellt bei der Regelung mit Prozeßrechnern, wenn der aktuelle Wert $\underline{y}(k)$ zur Berechnung der Stellgröße $\underline{u}(k)$ eingesetzt wird, die Zeitdifferenz zwischen der Messung und der Ausgabe der Stellgröße ein prinzipielles Problem dar. Es hat sich gezeigt, daß mit einem gemäß (15) modifizierten Kalman-Filter, insbesondere bei stochastischen Meßstörungen,ein verallgemeinerter Beobachtungsfehler auf etwa 50% des Wertes ohne diese Korrektur reduziert werden kann. Um den Einfluß dieser Rechenzeit zu verdeutlichen, wurde die Ausgabe der Stellgröße um Δt = 50, 70, 120, 220, 320 und 520 ms verzögert. Die entsprechenden Kurvenscharen für die unkorrigierte Filter-Version sind im Bild 4 dargestellt.

Bei dem Einsatz von Beobachtungsalgorithmen wurde folgende Definition eines verallgemeinerten Beobachtungsfehlers eingeführt:

$$K_o = \sqrt{\frac{1}{n+1}\sum_{k=0}^{n}\left\|\tilde{\underline{x}}_{norm}(k)\right\|^2} \quad , \qquad (18)$$

mit

$$\tilde{\underline{x}}_{norm}(k)=\underline{x}_{norm}(k)-\hat{\underline{x}}_{norm}(k) \quad , \qquad (19)$$

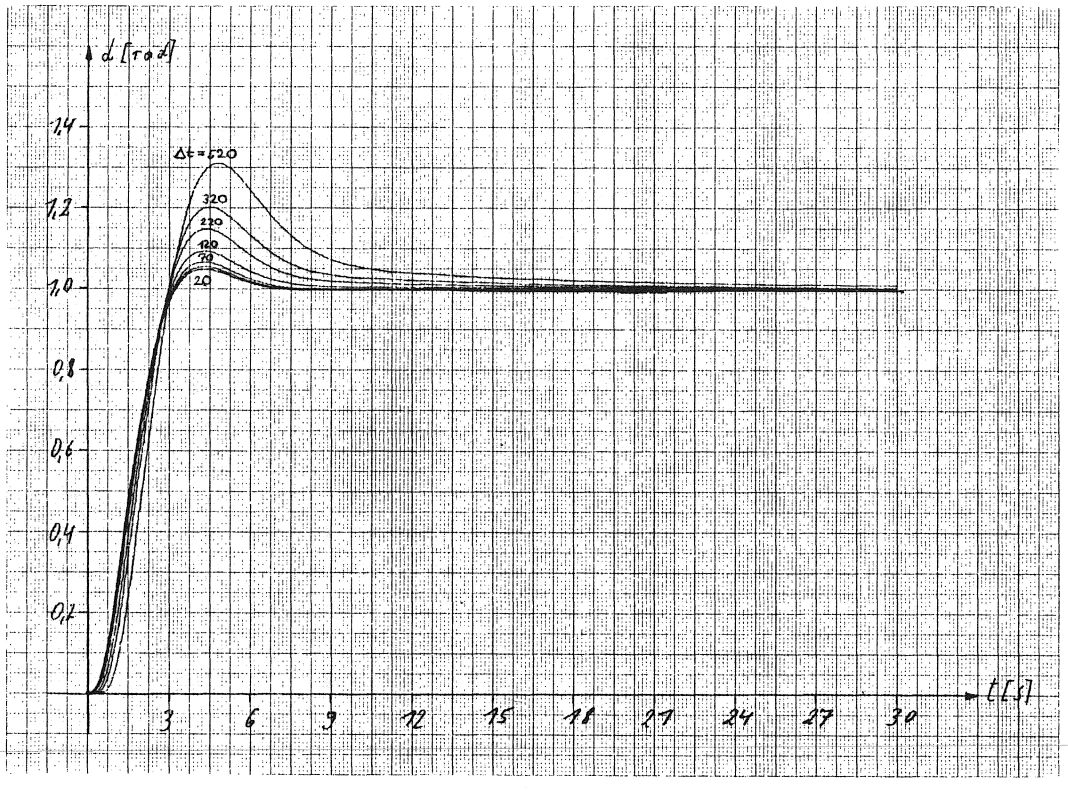


Bild 4: Einfluß der Ausgabe-Verzögerungszeit Δt auf das Regelverhalten eines dynamischen Systems

wobei die durch "norm" gekennzeichneten Größen die auf den jeweiligen Maximalwert bezogenen Systemgrößen sind.
Für den Fall,daß bei Berechnung der Stellgröße gemäß dem (1-Schritt-) Prädiktor-Algorithmus (11)
direkt der bereits bekannte Prädiktionswert $\hat{\underline{x}}^*(t_k)$ verwendet, d.h. der Meßwert $\underline{y}(t_k)$ noch nicht berücksichtigt wird, wurde eine Rechenzeit von 0,2 ms (meßtechnisch mittels Oszillograph) festgestellt, die sich auf mehr als 15 ms vergrößert, wenn gemäß dem Filter-Algorithmus (16) die Zustandschätzung zunächst korrigiert und anschließend die Stellgröße gemäß (12) errechnet wird, d.h. der aus regelungstechnischen Gründen wegen der um ein Intervall früheren Berücksichtigung von Störungen vorteilhafte Schätzalgorithmus angewandt wird.
In Bild 5 wird die Regelung gemäß dieser beiden Schätzalgorithmen mit der gemäß dem modifizierten Kalman-Filteralgorithmus nach (14) und (17) verglichen.

$y(0)/_{rad}$	-0,5	0	0,5	Mittelwert
K_0(Präd.)	0,241	0,345	0,242	0,2759=100%
K_0(Filt.)	0,547	0,443	0,644	0,5449=197,5%
K_0(Pr./Filt.)	0,369	0,349	0,138	0,2853=103,4%

Bild 5: Vergleich des Prädikator-, Filter- und modifizierten Filteralgorithmus an Hand des verallgemeinerten Beobachtungsfehlers K_0 für Sprungantworten des Regelungssystems

Im Vergleich zu dem Prädiktor ist der Beobachtungsfehler, bedingt durch die Berücksichtigung des aktuellen Meßwertes, bei einem Filter etwa doppelt so groß. Dagegen kann dieser Fehler durch zusätzliche Korrektur um etwa 50% reduziert werden. Dabei ist die Erfassung der Daten, die Durchführung der Berechnungen, sowie die Einplanung der Ausführung der Ausgabe der Stellgröße u(k) zu einem festen, bekannten Zeitpunkt mit einfachen PEARL-Anweisungen auch nachträglich möglich.

PEARL-Programme

In Bild 6 ist die programmtechnische Realisierung der Regelung mittels modifiziertem Kalman-Filter-Algorithmus in PEARL veranschaulicht.

```
 1   MØDULE KALFIL ØPTIØNS (/40,MM3,EFP);
 2     SYSTEM;
         .
         .
14     SYSEND;
15     PFØBLEM;
         .
         .
       /* TASK-DEKLARATIØNEN */
26     DCL HAUPT      TASK PRIØ(8) ØPTIØNS(MAIN);
27     DCL BEØBACHTER TASK PRIØ(9);
28     DCL AUSGABE    TASK PRIØ(10);
         .
         .
80     HAUPT:TASK;
         .
         .
       /* EINPLANUNG DER TASK BEØBACHTER */
90     EVERY TA DURING SIMT ACTIVATE (BEØBACHTER);
         .
         .
107    END; /* ENDE DER TASK BEØBACHTER */
108    BEØBACHTER:TASK;
         .
         .
       /* MESSUNG*/
113    DØ I=0 TØ 3;
114       TAKE (ADW(I)) INTØ (AX(I));
115    END;
       /* EINPLANUNG DER TASK AUSGABE */
116    AFTER 50 MILSEC ACTIVATE (AUSGABE);
       /*  BERECHNUNG VØN U(TK+DELTA) */
         .
         .
132    U= . . .
       /*ENDE DER BERECHNUNG VØN U(K+DELTA) */
133    . . . .
       /* TASK AUSGABE WIRD NICHT VØR */
       /* STATEMENT 133 AKTIV */
         .
         .
177    END; /* ENDE DER TASK BEØBACHTER */
178    AUSGABE:TASK;
179      SEND (DAW(0)) FRØM (U);
181    END; /* ENDE DER TASK AUSGABE */
         .
         .
         .
412    PFØBEND;
413  MØDEND;
```

Bild 6: PEARL-Programm zur Regelung mittels modifiziertem Kalman-Filter

Die Zeit zur Berechnung der Stellgröße $u(t_k+\Delta t)$ ist kürzer als die feste Zeit von 50 ms, nach der die mit Statement 116 jeweils in der Task BEOBACHTER eingeplante Task AUSGABE aktiviert wird. Die Task BEOBACHTER wird in der Task HAUPT zyklisch zu den Zeitpunkten $k \cdot T_a$, k=0,1,2,..., eingeplant, die weiterhin die Steuerung des Analogrechners und die Protokollausgabe übernimmt.

In Bild 7 ist das Programm angegeben, das zur Bestimmung der Ausgabeverzögerung durch das Betriebssystem verwendet wurde. Die Messung wurde dabei so durchgeführt, daß die Zeitverzögerung auf einem Oszillographen abgelesen werden konnte. Dieser wurde durch das über das Digitalausgabe-Interface DR 11-C ausgegebene Singal getriggert und die Zeit zwischen 2 folgenden Analogausgaben gemessen.Bis auf eine Verzögerung durch die Ausgabe-Hardware und die bei dieser Messung nicht eingehende Zeit für die

Zeitverwaltung ist der Abstand zwischen den 2 Digitalausgaben die gesuchte Zeitverzögerung der Ausgabe. Durch zyklische Wiederholung konnte die Zeitdifferenz an einem stehenden Oszillographenbild abgelesen werden.

```
1   MØDULE TEST2 ØPTIØNS(/40,MMØ,EFP);
2      SYSTEM;
3         DEVADR AD01009=0;
            .
5         DAW(0:1)MØDE(12): <- AD01009*(0:1);
6         DEVADR DR11C=167770L;
7         DIGAUS EIT(2): <-> DR11C*2*14;
            .
            .
10     SYSEND;
11     PRØBLEM;
12        DCL A BIT(2)INIT('11'B);
13        DCL B BIN FLØAT(24)INIT(-0.5);
14        DCL C BIN FIXED(15);
            .
            .
21        DCL HAUPT TASK ØPTIØNS(MAIN);
22        DCL T1 TASK;
23        DCL T2 TASK;
24        DCL T3 TASK;
25        HAUPT:TASK;
26            EVERY 100 MILSEC ACTIVATE (T1);
27        END; /* HAUPTTASK */
28        T1:TASK;
29            A:=\A;
30            B:=-B;
              /* FLØATING TØ INTEGER */
31            CALL FDAW(B,C);
32            SEND(DIGAUS)FRØM(A);
33            ACTIVATE(T2);
34            ACTIVATE(T3);
35        END; /* TASK T1 */
36        T2:TASK;
37            SEND (DAW(0)) FRØM (C);
38        END; /* TASK T2 */
39        T3:TASK;
40            SEND (DAW(1)) FRØM (C);
41        END; /* TASK T3 */
42     PRØBEND;
43  MØDEND;
```

Bild 7: Testprogramm zur Ermittlung der durch das PEARL-Betriebssystem verursachten Ausgabeverzögerung

Adressen der Autoren:

Dipl.-Ing. U. Mohr, Siemens AG, Ingenieurkreis Erlangen

Prof.Dr.-Ing. D. Popović, Universität Bremen, 28 Bremen 33, Postfach 330 440

Dr.-Ing. G. Thiele, Universität Bremen, 28 Bremen 33, Postfach 330 440

Literatur

[1] BBC-PEARL-Subset, Sprachbeschreibung, 1977

[2] Ludyk, G., Theorie dynamischer Systeme Elitera-Verlag, Berlin, 1977

[3] Föllinger, O., Regelungstechnik, 3. Auflage, AEG-Telefunken, Berlin, 1980

[4] Brammer, K., Siffling, G., Deterministische Beobachtung und stochastische Filterung, R. Oldenbourg Verlag, München, 1975

[5] Mohr, U., Kalman-Filter: Probleme der Implementierung der Genauigkeit und der Stabilität Diplom-Arbeit, Universität Bremen, 1982

Instanzen zur Datenakquisation bei verfahrenstechnischen Prozessen

Dr.-Ing. **Karl Maurer**, Speyer a. Rh.

Zusammenfassung

Aufbau und Funktion eines zur Meßwertverarbeitung an einem verfahrenstechnischen Prozeß eingesetzten Prozeßrechensystems werden beschrieben. Detailliert wird die Organisation des Prozesses der zyklischen Einlesung der Meßwerte mittels verschiedenartiger Meßgeräte und Multiplexer durch ein Instanzennetz dargestellt. Neben dem Konzept der Ablaufstruktur der den Erfassungsprozeß realisierenden, in PAS2 geschriebenen Software in der Realzeitumgebung werden die Lösung von Synchronisations- und Koordinationsproblemen behandelt. Eine Beschreibung der Hardwarekonfiguration des Gesamtsystems und ein Abriß BASF-eigener Hard- und Grundsoftwareentwicklungen zur Meßwerterfassung werden gegeben.

Stichworte: Prozeßautomatisierungssprache PAS2, Meßwertverarbeitung, Multiplexersteuerung, Design von Programmablaufstrukturen, Synchronisation und Koordination

Summary: Design and performance of a process computer system, which is used for measurement processing in a chemical plant, are described. The organizational details which realises the cyclic acquisition of measurements by different instruments and multiplexers, is depicted by a staged network. Besides of the concept of the running structures of the software, which is written in PAS2 for the acquisition process in a realtime environment, problems of synchronization and coordination are treated. A description of the hardware configuration of the entire system and a brief survey of the development of hardware and fundamental software for measurement acquisition by BASF are given.

Key-words: processautomation language PAS2, measurements processing, control of multiplexers, design of structures for program running, synchronization and coordination

1. Zur Problemstellung

1.1 Aufgaben einer Meßwertverarbeitung

Zur Führung verfahrenstechnischer Produktionsprozesse, speziell der sogenannten Fließprozesse, die in möglichst stationärem Fließgleichgewicht aus Einsatzstoffen Produkte herstellen, müssen Zustandsgrößen analoger und binärer Natur in regelmäßigen Abständen oder auch sporadisch, beim Auftreten bestimmter Ereignisse im Prozeßablauf, in großer Anzahl beobachtet und ausgewertet werden. Zur Automatisierung der Beobachtungs- und Auswertevorgänge werden seit den sechziger Jahren Digitalrechner eingesetzt. Diese Prozeßrechnereinsätze werden mit dem Begriff Meßwertverarbeitung umschrieben.

Kennzeichnend für diese Aufgabe ist, daß die Meßwerte mittels unterschiedlicher Meß- und Übertragungstechniken (analoge Meßtechnik, Zählermeßtechnik, Chromatographie / elektrische oder pneumatische Übertragung) an den Rechner gebracht werden. Dieser muß sie dann in geeigneten Zeitzyklen oder auch sporadisch einziehen, in Zahlenwerte konvertieren und als Prozeßabbild in Datenbereichen zur weiteren Verarbeitung in Meldungen und Protokolle vorhalten.

Der ursprüngliche Entwurf der Programme des Systemteils "Meßwerterfassung" für die diesem Aufsatz zugrunde liegende Anwendung stammt von Dr. W. Pfeffer.

Der Verfasser dankt für die freundliche Genehmigung zu seiner Benutzung bei dieser Publikation.

1.2 Technische Lösungsmöglichkeiten

Nach den ersten Realisierungen solcher Projekte erkannte man schon frühzeitig, daß derartige Prozeßrechnereinsätze weitgehend standardisierbare Aufgabenstellungen enthalten, und so entstanden die ersten "Programmpakete für Meßwertverarbeitung und DDC" [1], die zum Lauf auf größeren Prozeßrechnern bestimmt, zentralistische Lösungen darstellten. Seit Ende der siebziger Jahre werden den Fortschritten der Mikroelektronik entsprechend dezentrale Lösungen in Form sogenannter "Digitaler Prozeßautomatisierungssysteme"[1] in großer Vielfalt auf dem Markt angeboten [2]. Beiden Lösungen haftet jedoch der Nachteil an, daß man sich relativ starr an die von den Herstellern des Systems vorgesehenen Möglichkeiten halten muß, sowohl bezüglich der Datenerfassung als auch bei deren Verarbeitung und der Darstellung des Ergebnisses. Ersteres mag bei Neuanlagen noch ohne weiteres möglich sein, Letzteres stößt fast immer auf Widerstände bei den Anlagenbetreibern.

1.3 Eine Lösung mittels PEARL

Als bei der BASF Aktiengesellschaft 1973 die Realisierung einer Meßwerterfassung an einer bestehenden Anlage mit umfangreicher und vielseitiger Meßperipherie und vielen Sonderwünschen des Betreibers hinsichtlich der Datendarstellung zur Diskussion stand, entschloß man sich damals zum Einsatz eines BBC-Prozeßrechensystems DP1000, programmiert in der 1.PEARL-Implementation PAS2, wegen der Problem- und Benutzernähe dieses neuen Sprachkonzepts. Das so realisierte System läuft seit 1975 in softwaremäßiger Hinsicht problemlos und wurde vor zwei Jahren an den neuesten Stand von PAS2 angepaßt. Es hat sich bei häufigen Änderungswünschen des Benutzers als flexibel erwiesen.

2. Formulierung der Aufgabe

Die Funktion des aus Operateur, Anlage und Rechner bestehenden Mensch-Maschinesystems ergibt sich aus dessen Kommunikationsstruktur, die für den realisierten Fall durch Abb. 1 dargestellt wird.

(B i l d 1)

Unter anderem ist zu ersehen:

- Der Rechner wirkt nicht direkt auf den Prozeß ein, er läuft zwar on-line, aber open-loop.

- Der Operateur beeinflußt den Prozeß aufgrund von Meldungen und Protokollen über den Prozeßablauf, er korrespondiert mit dem Rechner über ein Bedienpult (Eingabe von nicht gemessenen Werte, Initiierung von Ausgaben). Funktion und Format der Protokolle sind in [3] beschrieben.

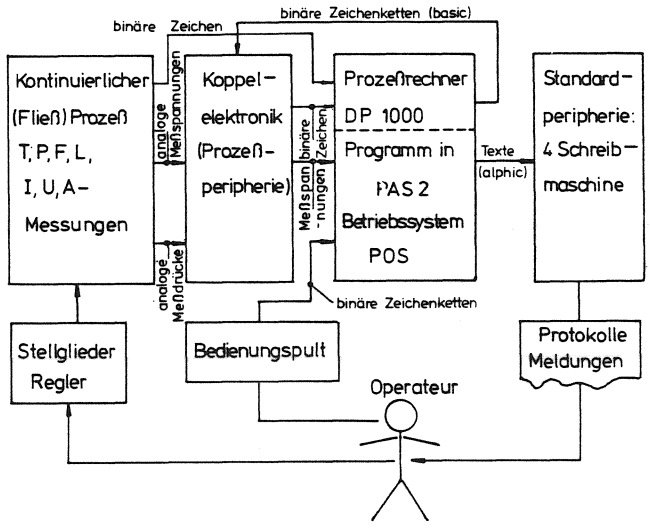


Bild 1. Kommunikationsstruktur des Meßwertverarbeitungssystem

- Prozeß und Rechner sind miteinander über eine Koppelelektronik verbunden, die die aus dem Prozeß kommenden Meßsignale in weiter unten besprochener Weise in eine vom Rechner benötigte elektrische Signalspannung einheitlichen Niveaus umsetzt. Die Koppeleinheit wird vom Rechner über binäre Zeichenketten gesteuert, setzt die analogen pneumatischen Meßdrücke oder elektrischen Meßspannungen in elektrische Spannungen im Signalniveau des Rechners um und gibt außerdem Informationen über ihren eigenen Zustand ebenfalls in Form binärer Zeichenketten an den Rechner ab.

Die von dem Prozeßrechensystem auszuführenden, d.h. von der Software abzudeckenden Funktionen sind in dem Funktionsnetz von Abb. 2 dargestellt.

(B i l d 2)

Die Akquisition der analogen und binären Daten und ihrer Umwandlung in die interne Rechnerdarstellung des "physikalischen Wertes" in Form von Zahlen des Typs "Gleitkomma" erfolgt dabei in den Instanzen (2) und (3), deren hard- und softwaremäßige Realisierung im folgenden behandelt wird.

3. Die Schnittstelle zum Prozeß (Prozeßperipherie)

Die Klassifizierung der zu erfassenden Messungen erfolgt aus der Sicht des Anlagenbetreibers nach der physikalischen Art der zu messenden Größe: Temperaturen (T), Drücke (P), Durchflüsse (F), Füllstände (L), Stoff-

[1] Die hierfür gebrauchte Abkürzung "PAS" ist nicht zu verwechseln mit PAS2 (= Prozeßautomatisierungssprache)

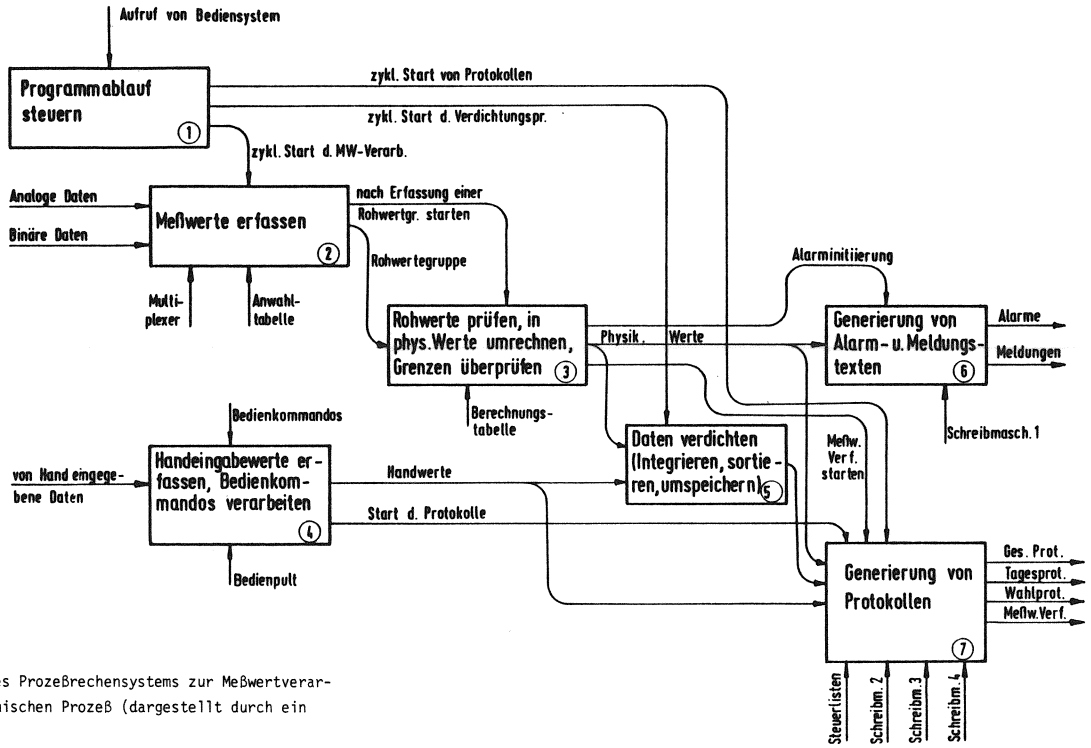


Bild 2. Funktionen des Prozeßrechensystems zur Meßwertverarbeitung an einem chemischen Prozeß (dargestellt durch ein SADT-Aktigramm)

zusammensetzungen (A) u.a. Für die Konstruktion eines Meßwertverarbeitungssystems kommen dagegen andere Klassifizierungsmerkmale in Betracht:

- Nach dem Meßprinzip (Blendenmessung, Zählermessung, Gaschromatograph)
- Nach dem verwendeten Meßstellenumschalter. Die übertragenen Meßwerte müssen - wie schon gesagt - in eine analoge Meßspannung am Eingang des Rechners umgesetzt werden und dann im Rechner auf ein Bitmuster, interpretiert als ganze Zahl, abgebildet werden Diese beiden Funktionen übernehmen Meßumformer und AD-Umsetzer (ADU). Beide Geräte sind relativ teuer und müssen in einem Zeitmultiplexverfahren von einer Vielzahl gleichartiger Messungen zeitlich nacheinander genutzt werden. Der Typ des verwendeten Meßstellenumschalters wird im wesentlichen von der Art der Meßwertübertragung (pneumatisch oder elektrisch) bestimmt.
- Nach der zeitlichen Größe des erforderlichen Abtastintervalls. Diese richtet sich nach der Geschwindigkeit, mit der sich die zu beobachtende Größe ändern kann. Im beschriebenen System waren zwei Abtastzyklen zu realisieren:
ein "1-min.-Zyklus" und ein "20 sec-Zyklus",
d.h. alle Messungen, die laufend mit den vorgegebenen zeitlichen Abständen abgetastet werden sollen, werden in den betreffenden Zyklus eingereiht. Alle Messungen des Zyklus werden der Reihe nach abgetastet mit derart gewählten zeitlichen Abständen, daß spätestens nach der vorgegebenen Zykluszeit alle Messungen einmal erfaßt wurden. Die Abtastung beginnt dann genau nach Verstreichen der Zykluszeit mit der ersten Messung wieder von vorne.

Folgende Multiplexertypen werden eingesetzt:

- Relaismultiplexer (MSU)[1].
Eine matrixförmige Anordnung(16*16), die über zwei $(\frac{1}{16})$-codierte binäre Worte vom Rechner angesteuert wird.
- Temperaturanwahlgerät (ZMA)[2].
Eine Einrichtung, durch die zum einen Mal Temperaturmessungen von Hand angewählt und durch Zahlenröhren angezeigt werden können, zum anderen Mal vom Rechner mittels einer in $(\frac{1}{10})$-Codierung dargestellten dreistelligen Zahl angewählt und zum Rechner übertragen werden.

Der Zugriff ist in beiden Fällen wahlfrei durch die momentan an den Klemmen angelegte Adresse bestimmt (random acess).

- Pneumatische Multiplexer mit der Handelsbezeichnung Scanivalve (SCV). Hier werden die pneumatischen Signale sequentiell abgetastet und durch einen Transmitter in das elektrische Normsignal umgeformt. Die Weiterschaltung von Abgriff zu Ab-

1) MSU = Abkürzung für die bei BASF gebräuchliche Bezeichnung "Meßstellenumschalter".

2) ZMA = Abkürzung für "zentrale Meßstellenanwahl".

griff erfolgt mittels eines vom Rechner auszugebenden Schrittpulses. Der Zustand ist hier also nicht allein durch die Eingangsgrößen, sondern auch durch den Vorzustand bestimmt. Die Rückkehr in die Grundstellung erfolgt mittels eines "Homepulses".

Um eine zahlenmäßige Vorstellung von der Größe des behandelten Problems zu geben, wurde die Aufteilung der verschiedenartigen Messungen auf die Multiplexer und Abfragezyklen in Tab. 1 dargestellt.

Geräteart	Abfragezyklus	Messungen A	D	F	L	P	T	Σ
Turbinenzähler	2min			14				14
Temperaturanwahlgerät	1min						103	103
Temperaturanwahlgerät	1min						51	51
Relaismultiplexer	1min	8	1		64		1	74
Scannivalve	1min		1	43	15	2		61
Scannivalve	1min			40	7	16		63
Scannivalve	20sec			41	4	19		64
Gaschromatograph	Interrupt	16						16
Σ		24	2	138	90	37	155	446

Tabelle 1. Anzahl der verschiedenartigen im System verarbeiteten Messungen und ihre Aufteilung auf die Multiplexer

Nicht typisch oder erforderlich ist dabei, daß die Messungen, die mittels der Multiplexer mit wahlfreiem Zugriff eingezogen werden, jeweils nur zu einem Zyklus gehören.

4. Die Instanzen der Meßwerterfassung

4.1 Zur Darstellungsmethode durch Instanzennetze

Nachdem nun die hardwaremäßigen Einrichtungen des Systems beschrieben sind, soll nun das Verfahren der Erfassung der Messungen in seinen einzelnen Stationen näher beschrieben werden. Für eine solche Station, die in einem System "eine abgegrenzte Einheit mit definierten Schnittstellen" bildet und dabei "Eingangsobjekte gemäß einer vorgegebenen Funktion in Ausgangsobjekte umformt" [4], hat sich neuerdings in der Fachsprache die Bezeichnung "Instanz" eingebürgert. Instanzen können prinzipiell hard- oder softwaremäßig sein.

Die Darstellung der Ausführung des Verfahrens durch die beteiligten Instanzen erfolgt in Form eines Instanzennetzes [5]. Dargestellt werden in einem solchen Netzsystem Funktionsbausteine ▭ (Rechtecke) und Verbindungsknoten ▽ (Kreise oder Rechtecke mit abgerundeten Ecken). In ▭ sind die Wertzuweisungsfunktionen formal oder verbal beschrieben, die angeben, wie sich die Werte der unten angeordneten Anschlußvariablen (Ausgangsvariablen) aus den oben angeordneten Eingangsvariablen und gegebenenfalls den internen Zustandsvariablen des Bausteins ergeben. Die wesentliche funktionale Eigenschaft der ▽, die die Beobachtungsvariablen darstellen, besteht darin, daß an ihren Anschlüssen (oben und unten) immer Wertegleichheit besteht. Da es sich bei den ▽ letzlich auch um die Darstellung der im System fließenden Daten handelt, werden ihre Anschlüsse entsprechenden Konventionen aus Datenflußplänen folgend mit Doppellinien gezeichnet.

4.2 Erfassung der zyklischen Messungen

Ein weiterer wesentlicher Gesichtspunkt ist der der Zuständigkeit einer Instanz für ihre Ausgangsvariablen die eigentlich durch eine gesonderte Zuständigkeitsfunktion G beschrieben werden müßte. Hier wird vereinfachend, wie bei Ablaufplänen angenommen, daß die Zuständigkeit (Aktivität) von oben nach unten durch die Instanzen wandert, d.h. die Zuständigkeit ist mit der einmaligen Erzeugung der Ausgangsdaten aus den Eingangsdaten abgeschlossen, bis das Instanzennetz wieder neu durchlaufen wird. Wird die Zuständigkeit abweichend von dieser Reihenfolge übertragen, wird dies durch einen einfachen Pfeil mit Angabe einer eventuell eingebauten Verzögerung $\triangle t$ angegeben.

Der Prozeß der Meßwerterfassung, der durch das Zusammenspiel von Software sowie Hardwarekomponenten des Systemherstellers (Ein/Ausgabewerke) und des Anwenders (Multiplexer) gekennzeichnet ist, ist in Abb. 3 auf die beschriebene Weise dargestellt. Durch Hardware realisierte Instanzen sind dabei an der oberen rechten Ecke gekennzeichnet.

Es handelt sich hierbei um die Beschreibung eines Schritts des Verfahrens, bei dem 6 in dem betreffenden Meßzyklus zu bearbeitende Messungen, die je einem der 6 verschiedenen Multiplexer zugeordnet sind, ausgewählt werden und mit Hilfe der in dem ihnen zugeordneten Anwahlcodewort verzeichneten Adresse angewählt werden. Dieser Schritt wird innerhalb der Zykluszeit so oft wiederholt, bis alle zum Zyklus gehörigen Messungen eingezogen sind. Multiplexernummer und Anwahlcode sind in den Anwahlcodewörtern zusammengefaßt, gemäß dem in Abb. 4 beschriebenen Format.

Die Anwahlcodewörter sind in einer aus einer Systembeschreibung generierten Anwahlliste MEANW niedergelegt [1]. Der Index ist die Verarbeitungs-Nr.

---

[1] Steuerlisten sind in Abb. 3 durch doppelt umrandete Verbindungsknoten dargestellt.

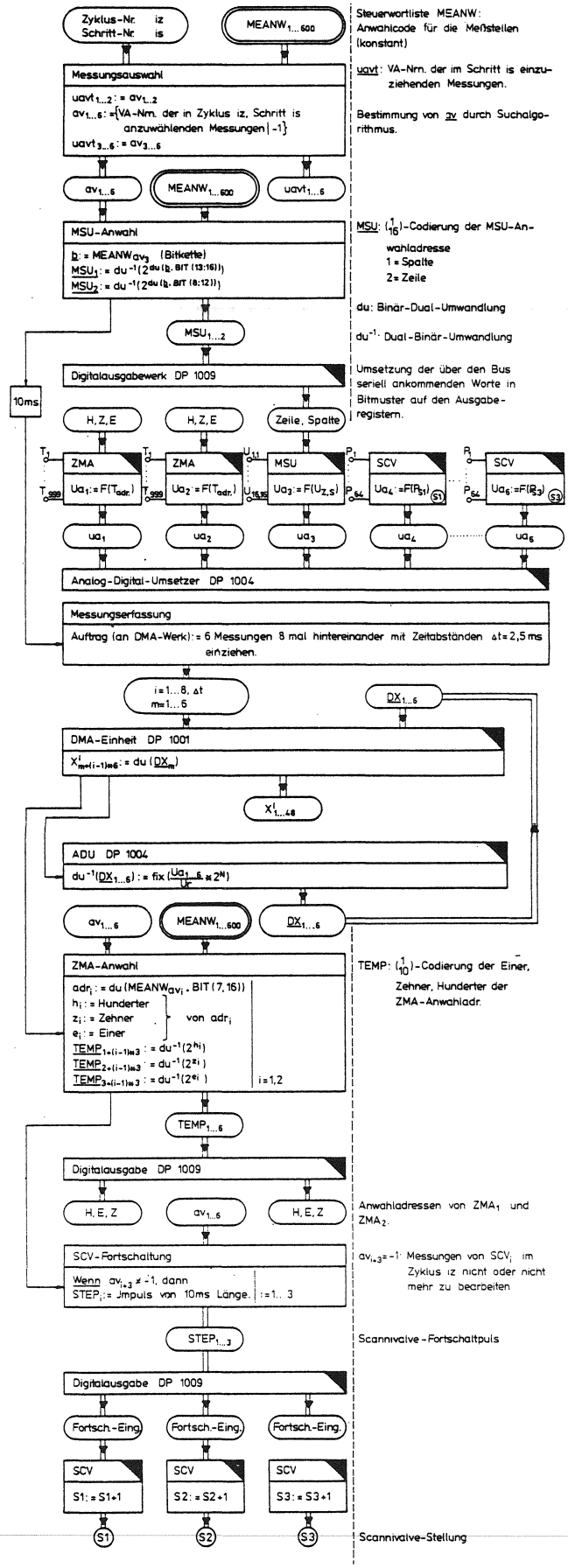
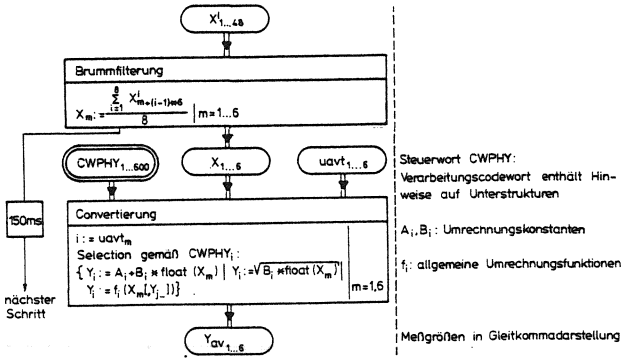


Bild 3. Instanzen zur Durchführung eines "Schritts" eines Meßwerterfassungszyklus



(VA-Nr.), die einer jeden Messung fest zugeordnet ist. Die in einem Schritt zu bearbeitenden VA-Nrn. werden aus MEANW durch einen Suchalgorithmus bestimmt und in einem Feld av nach dem Multiplexerindex abgelegt. Es folgt dann die Ausgabe der Anwahladresse an den MSU, dessen Ausgangsspannung

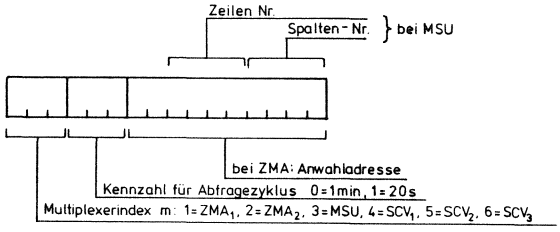


Bild 4. Codewort für die Meßstellenanwahl

wegen seiner kurzen Einschwingzeit bereits nach 10 ms über den ADU eingezogen werden kann. Die ZMAs und SCVs sind wegen der langen Einschwingzeit dagegen schon am Ende des voraufgegangenen Schrittes in die gegenwärtige Position gebracht worden. Die Messungen werden von allen 6 Multiplexern quasiparallel und unter Hardwaresteuerung durch die DMA-Einheit zu 8 äquidistanten Zeitpunkten innerhalb einer Netzperiodendauer (20 ms) eingezogen (Abb. 5).

Die Umsetzung der Spannung an einem ADU-Eingang in ein N Stellen langes Bitmuster Dx auf dem Ausgangsregister wird formal vermittels einer Funktion du beschrieben, die die Interpretation eines Bitmusters D (D_1.....D_N) als Dualzahl liefert:

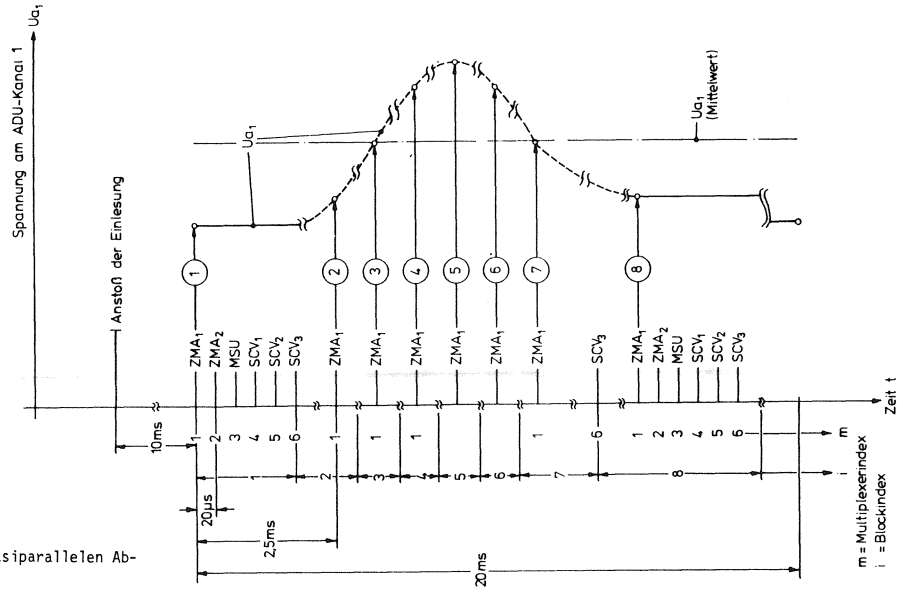


Bild 5. Zeitlicher Verlauf der 8-maligen quasiparallelen Abtastung von 6 Meßwerten

$$x = du(\underline{D}) = \sum_{j=1}^{N} D_j * 2^{j-1} \qquad (1)$$

du^{-1} bedeutet die zugehörige Umkehrfunktion.
U_r ist die Referenzspannung des ADU.

Nach dem Ablesen der 6 Kanäle des ADU erfolgt dann die Fortschaltung der ZMAs und SCVs für den nächsten Schritt.
Die 8 Meßwerte, die von jeder der 6 Messungen vorhanden sind, werden gemittelt, um den in die Übertragungsleitungen eingekoppelten 50 Hz-Brumm herauszufiltern.
Nach dieser Aktivität wird 150 ms gewartet, bis mit der Abwicklung des nächsten Schritts begonnen wird.
Zeitlich parallel dazu werden die 6 X-Werte der Messung konvertiert, d.h. in einen als Gleitkommavariable dargestellten Zahlenwert umgerechnet, der dem Momentanwert der Messung entspricht, und unter der VA-Nr. in der Analogwertliste Y abgelegt. Diese wird einem Feld uavt entnommen, daß die VA-Nrn. der in diesem Schritt eingezogenen Messungen enthält, im Gegensatz zu av, das die VA-Nrn. der in diesem Schritt angewählten Messungen enthält. A_i und B_i sind den Messungen zugeordnete Konstanten, die sich aus den Parametern der Meßwertübertragung ergeben. Umfangreichere Formeln treten z.B. bei der Kennwertermittlung oder bei der Bestimmung von Behälterinhalten aus dem Flüssigkeitsstand auf.
Der zeitliche Ablauf der Aktivitäten eines Schritts zeigt Abb. 6.

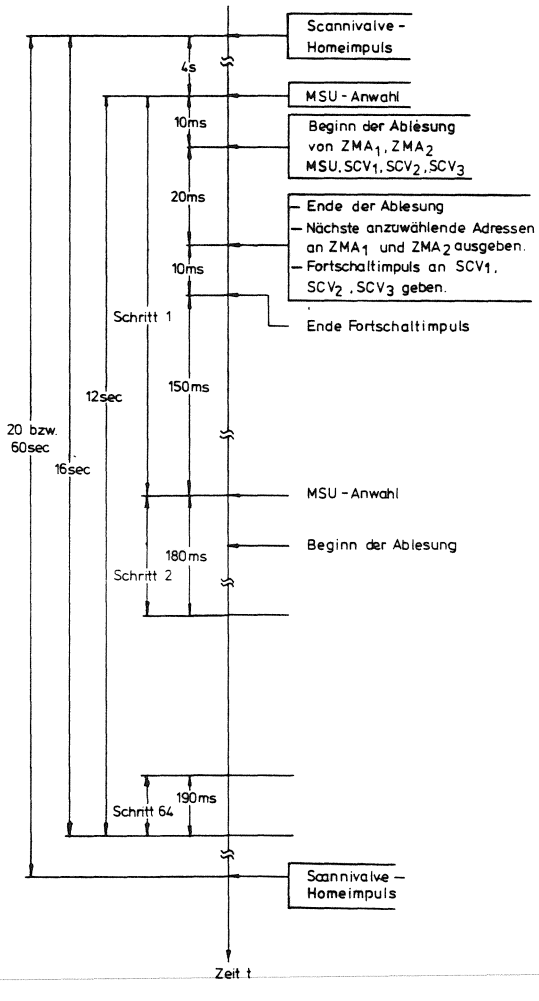


Bild 6. Zeitliche Abfolge der Aktivitäten der Schritte während eines Zyklus

4.3 Die Erfassung der Gaschromatographen- und der Zählermessungen

Die nach diesen beiden Prinzipien erfaßten Messungen nehmen gemäß Tab. 1 eine Sonderstellung ein. Die Funktionen der zugehörigen Softwareinstanzen werden pauschal beschrieben.

Der Gaschromatograph arbeitet nach folgendem Verfahren: Er wird mit einer Probe beschickt. Nach Bearbeitung der Probe sendet er zuerst eine Kennung des Probestroms codiert als Höhe der analogen Spannung an seinem Ausgang, sodann sendet er seriell die Peakhöhen, die den Anteilen der enthaltenen Komponenten entsprechen, in einer für jeden Probestrom fest vorgegebenen Reihenfolge, da dessen qualitative Zusammensetzung bekannt ist. Das Aussenden eines analogen Wertes wird durch ein binäres Signal INTGAS angekündigt. Ein weiteres binäres Signal GCMOD gibt an, ob sich der GC im Adressübertragungsmodus oder Meßwertübertragungsmodus befindet. Die Identifikation einer aus dem Prozeß ankommenden GC-Messung erfolgt also gemäß der hierarchischen Struktur "GC-Nummer.Probestrom-Nummer.Komponenten-Nummer".
Die zugehörige Softwareinstanz tastet den GC-Ausgang nun nicht zyklisch ab, sondern wird durch INTGAS über einen Interruptmechanismus aktiviert. Die GC-Nummer wird anhand des Interrupts identifiziert und der GC-Ausgang am MSU angewählt. Ist der GC im Adreßmodus, wird die Probestrom-Nummer aus seiner Ausgangsspannung decodiert. Ist der GC anschließend im Meßwertmodus, werden die Peakhöhen der einzelnen Komponenten der Reihe nach aufgenommen und aus einer Beschreibung des identifizierten Probestroms die VA-Nummern der zugehörigen Komponenten bestimmt. Die Konvertierung und Ablage in dem Y-Bereich erfolgt in bekannter Weise.
Bei den von Turbinenzählern erfaßten Durchflußmessungen muß die Software die Zählimpulse, die von einem jeden solchen Gerät zugeordneten Digitaleingang aufgenommen werden, zählen und nach folgender Formel die Durchflußgeschwindigkeit bestimmen:

$$Y_i = \frac{(\text{Anzahl der Impulse in } \Delta t)_i}{\Delta t} * (\text{Menge/Impuls})_i \qquad (2)$$

Die Impulse müssen also durch eine im Verhältnis zur Impulsfrequenz genügend häufige Abfragen der zugeordneten Digitaleingänge identifiziert und aufsummiert werden. Nach Ablauf des Abfrageintervalls Δt wird die Anzahl der Impulse gemäß (2) umgerechnet und als Y-Wert abgelegt.

5. Die Realisierung der Softwarekomponenten der Meßwerterfassung in PAS2

5.1 Spezifikation der Hardware im Systemteil

Eine solche Realisierung beginnt u.a. mit der Festlegung der logischen Namen in einem Systemteil des Programms, mit denen die zu und von den Werken der Rechnerprozeßpheripherie transportierten PAS2-Objekte in den Transportanweisungen des Problemteils bezeichnet werden. Hierbei spiegelt sich die hardwaremäßige Belegung der Klemmen dieser Werke direkt in den Spezifikationen des Systemteils, eine hoch einzuschätzende selbstdokumentarische Eigenschaft von PEARL. Als Beispiel ist in Abb. 7 die Belegung des Digitalausgabewerkes DP1009, das die Benutzerprozeßperipherie steuert, und des Digitaleingabewerkes DP1016 dargestellt.

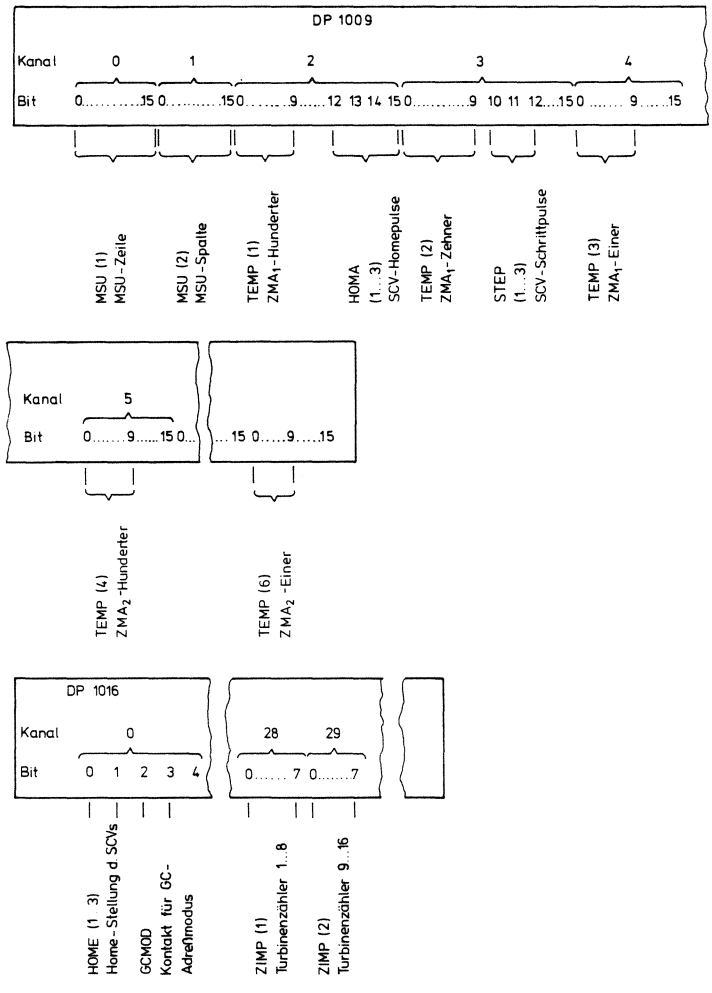


Bild 7. Belegung des Digitalaus- und des Digitaleingabewerkes

Die logischen Namen und die Steuereingänge der Multiplexer sind nebeneinander angegeben. Den entsprechenden Ausschnitt aus dem Systemteil des PAS2-Programms zeigt Abb. 8, wo außerdem die Namensvereinbarung für die Ansprache des Analogeingabewerkes DP1004 zu sehen sind.

```
SYSTEM;


/* Werknamen */

   /* Direct-Memory-Acess-Werk DPi001 */
   DEVNAM DMA = BLOCK;

/* Werkanschluesse */
   /* Analog-Digital-Ausgabe DPi009 */
   DEVADR ADOi009 = 0;
   /* Digitaleingabe DPi016 */
   DEVADR DIi016  = 512;
   /* Analogeingabe DPi004 */
   DEVADR AIi004  = i024;
   /* Vorrangunterbrechungswerk DPi010 */
   DEVADR INTi010 = 4088;

/* Systeminterrupts */
   ADOi009 ERRORINT :-) PDC#0;

/* Digitalausgaenge */

    /* Zeile und Spalte des MSU */

    MSU(2) BIT(16) :(- ADOi009#(0:1)#0 STEP(16);

   /* Hunderter, Zehner, Einer von ZMA  und  ZMA  */
                                        1       2
   TEMP(6) BIT(10) :(- ADOi009#(2:4)#0 STEP(16);

   /* Homepulse SCV , SCV  , SCV  */
                  1     2     3
   HOMA(3) BIT(1) :(- ADOi009#2#12;

   /* Steppulse SCV , SCV , SCV  */
                 1     2     3
   STEP(3) BIT(1) :(- ADOi009#3#10;



/* Digitaleingaenge */

   /* Homestellung SCV , SCV , SCV  */
                   1     2     3
    HOME(3) BIT(1) :-) DIi016#0#0;

  /*Uebertragungsmodus des GC */
   GCMOD BIT(1) :-) DIi016#0#3;



  /*Zaehlimpulse der Turbinendurchflussmesser */
  ZIMP(1:2) BIT(8) :-) DIi016#(28:29)#0 STEP(8);


/* Analogeingaenge */

   /* Eingaenge von ZMA , ZMA , MSU, SCV  , SCV  ,SCV  */
                    1     2          1     2     3
    ANEIN(1:6) :-) AIi004#(0:5);


SYSEND;
```

Bild 8. Ausschnitt aus dem Systemteil des Programms

Die in eine /*.....*/ -Beklammerung eingeschlossenen Texte sind Kommentare. Die Syntax der Zuordnung der logischen Namen zu den Kanal- und Anschlußnummern und die Bezeichnung der Übertragungsrichtung erfolgt gemäß den Festlegungen in [6].

Die sinnfällige Ansprache der Werke der Prozeßperipherie mittels logischer Namen in den SEND- und TAKE-Anweisungen des Problemteils ist sicher ein für den gegebenenfalls mit den Programmen arbeitenden Verfahrenstechniker schwerwiegender Vorteil gegenüber den verklausulierten Directive-Calls, die für diese Aktivitäten z.B. in Prozeßfortran abgesetzt werden müssen.

5.2 Konzept der Ablaufstruktur der Erfassungsprozesse

Ein für den System-Designer wertvolles Hilfsmittel sind die Realzeiteigenschaften des Systems, die es erlauben, Nebenläufigkeiten im zu realisierenden Meßwerterfassungsprozeß nach dem Multitasking-Konzept zu formulieren.
Die Prozesse zum Einziehen der Meßwerte des 20 s-Zyklus, des 1 min-Zyklus, der Gaschromatographen, der Turbinenmengenmesser sind nebenläufig, d.h. sie können im Prinzip quasiparallel ablaufen und müssen lediglich dann synchronisiert werden, wenn sie gemeinsame Ressourcen benützen wollen, das sind in diesem Fall die Multiplexer ZMA, MSU und der ADU. Die SCVs mit dem sequentiellen Zugriff können eo ipso nur einem Zyklus zugeordnet werden. Eigene Tasks werden also kreiert für die Prozesse der beiden Zyklen (ZYK1 und ZYK0) und der GC-Erfassung (GCIN)und die Zählerabtastung(IMPZ). Die beiden Zyklen haben dabei an sich vollkommen gleiche Funktionen lediglich mit einem vielleicht unterschiedlich zusammengesetzten Satz von Multiplexern auszuführen, wie z.B. keines der SCVs in beiden Zyklen gleichzeitig vertreten sein kann. Das legt nun den Gedanken nahe, die eigentlichen Aktivitäten der beiden Tasks ZYK1 und ZYK0 in eine gemeinsame Prozedur ERFAS zu legen, die mit jeweils unterschiedlichen Parametern aufgerufen wird. Die Turbinenmengenmessungen werden wie auch die Gaschromatographenmessungen von eigenen Tasks erfaßt.

Eine Möglichkeit zur Vergrößerung des Datendurchsatzes besteht darin, die Konvertierung der Daten, die wegen der umfangreichen dabei auszuführenden Rechenoperationen lange dauert, nicht synchron zu den Erfassungsprogrammen, sondern asynchron auszuführen. Damit kann der Prozessor in Wartezeiten, die während der Anwahl wegen der Zusammenarbeit mit der Hardware unumgänglich sind, noch Werte des letzten Schritts konvertieren. Die Konvertierung wird also in einer eigenen Task ausgeführt, die nacheinander für alle Erfassungsprogramme arbeitet. Die sich daraus ergebende Ablaufstruktur des Systems zeigt Abb. 9.

Die Programm-Module sind in drei Schichten dargestellt:

- Programmobjekte zur übergeordneten Steuerung des Ablaufs
- Objekte zur Durchführung der Erfassung
- Datenmodule zur Aufnahme von Steuerdaten (CWPHY und MEANW) und der zwischen den asynchron ablaufenden Rechenprozessen auszutauschenden Informationen (NUM,X,Y, FLUSS).

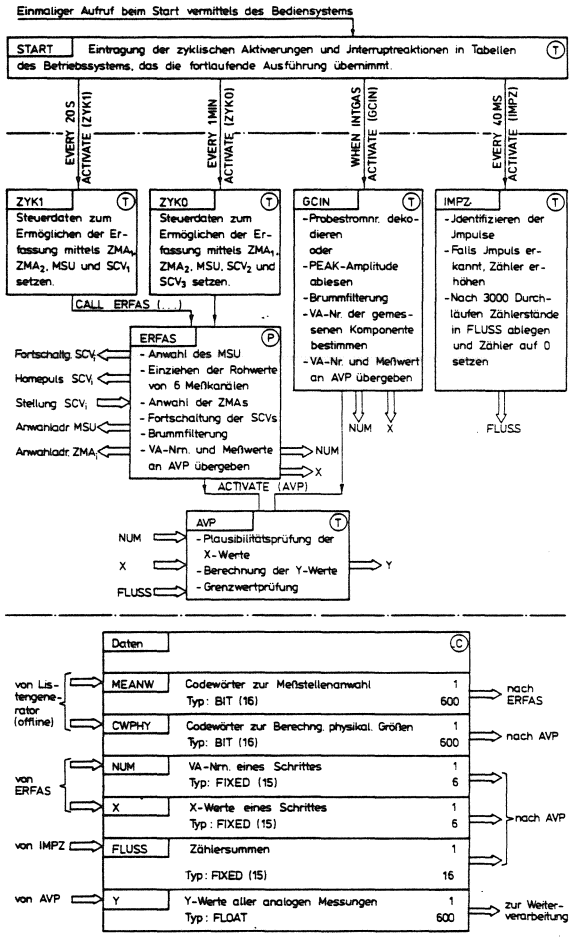


Bild 9. Ablaufstruktur der Prozesse der Meßwerterfassung

Die Arten der Module sind angegeben durch T = Task, P = externe Prozedur, C = globale Daten (COMMON). AVP (Analog Value Processing) ist hier die Task, die die von ZYK0, ZYK1, GCIN eingezogene Gruppe von bis zu 6 Analogwerten, deren VA-Nrn. in NUM und deren gemittelte ADU-Ablesung in X übergeben werden, mit Hilfe der in CWPHY abgelegten Informationen über Umrechnungskonstante oder Sonderformeln in physikalische Werte umrechnet, die in Y den nachfolgenden Programmen als Prozeßabbild zur Verfügung stehen. Die die Zählereingänge abtastende Task IMPZ legt dagegen die nach jeweils $\triangle t$ = 2 min aufgelaufenen Zählerstände in einem eigenen Datenbereich FLUSS ab. Die VA-Nrn. der Zählermessungen werden von ZYK0 oder ZYK1 in NUM an AVP als während dieser Zyklen zu bearbeitende Messungen übergeben. AVP greift dann vermittels einer Sonderformel auf die Werte in FLUSS zu und rechnet sie gemäß (2) in physikalische Werte um.

6. Synchronisations- und Koordinationsprobleme

Synchronisiert werden muß erstens der Zugriff auf die Multiplexer MSU, ZMA_1 und ZMA_2 und den ADU. Dies ist in PAS2 relativ einfach möglich, indem jedem Multiplexer eine binäre Semaphorvariable zugeordnet wird die vor Benutzung mit einer P-Operation belegt und nach Einlesung mit einer V-Operation wieder freigegeben wird.

Komplizierter ist das Problem der Synchronisation der Verarbeitungstask AVP mit den Erfassungstasks ZYK0, ZYK1 und GCIN. Die Kommunikation erfolgt hier über die Pufferbereiche NUM und X.

Das Synchronisationsproblem wird am sinnfälligsten durch ein Kausalnetz (Petri-Netz) modelliert (Abb. 10). Hierbei wird zur besseren Hervorhebung der Aktionen das Petri-Netz in Form eines "Start-Stop-Netzes" [7] gezeichnet. Man erkennt die Vielfalt der Kommunikationspfade, die die Bedingungen für das Einleiten der Aktion "Füllen des Puffers" in den drei zu synchronisierenden Prozessen versinnbildlichen.

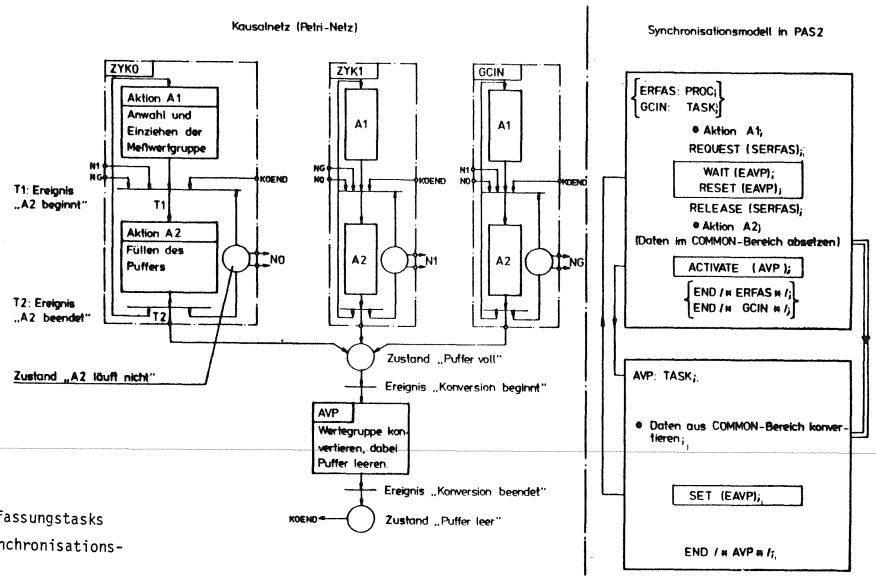


Bild 10. Petri-Netz-Modell zu Synchronisation der Erfassungstasks mit der Verarbeitungstask und Lösung des Synchronisationsproblems in PAS2

Die Synchronisation der Erfassungstasks untereinander und mit AVP sollte folgende Kriterien erfüllen:

1. Beschreibt ein Prozeß P1 den Puffer, so müssen die anderen Prozesse mit dem Beschreiben des Puffers warten, bis AVP die Daten von P1 verarbeitet hat.

2. Warten mehrere Prozesse (P2 und P3) auf die Freigabe des Puffers, aus dem noch Daten von P1 abgearbeitet werden, so sollte der Prozeß mit der höchsten Priorität als erster zum Laufen kommen.

Das 1. Kriterium ist durch die im Petri-Netz angegebene Verknüpfungen erfüllt, das 2. liegt vor, wenn mehrere Transitionen gleichzeitig schaltbereit sind, und durch das Schalten einer Transition die Schaltbereitschaft der anderen zerstört wird.

Diese Konfliktsituation kann nur durch den Eingriff eines übergeordneten Koordinators, in diesem Fall der Scheduler des Betriebssystems, gelöst werden.

Das Problem kann in PAS2 durch ein in dieser Sprache im Gegensatz zu PEARL definiertes Objekt vom Typ "EVENT", ein Ereignisflag gelöst werden, für das eine Warteoperation (WAIT(eventname)) definiert ist. Ein solches Eventflag EAVP wird zur Signalisierung des Endes der Bearbeitung der Pufferdaten in AVP gesetzt. Alle Erfassungsprozesse bleiben auf dem Statment WAIT(EAVP) stehen, bis EAVP gesetzt ist. Der Prozeß höchster Priorität wird daraufhin als erster weitergeführt, löscht als erste Aktion EAVP, womit konkurrierende Prozesse dann wieder warten müssen, setzt seine Daten im COMMON-Bereich ab und aktiviert anschließend AVP.

Bei dieser evidenten Lösung übersieht man allerdings die Möglichkeit des Eintretens folgender Situation: Ein Prozeß P1 könnte nach Passieren des WAIT-Statements vor Ausführung des RESET von einem Prozeß P2 mit höherer Priorität überholt werden. Dieser würde dann seine Daten im Puffer absetzen, die anschließend von P1 überschrieben würden.

Durch zusätzliche Anordnung einer Semaphorvariablen SERFAS in der in Abb. 10 dargestellten Art kann diese Situation verhindert werden. Da nun aber der Prozeß P2 höherer Priorität generell nicht bis zum Statement WAIT(EAVP) durchdringen kann, wenn ein Prozeß P1 dort steht, kann Kriterium 2 nicht mehr eingehalten werden, die Prozesse kommen in der Reihenfolge ihres zeitlichen Eintreffens an der kritischen Stelle zur Durchführung. Dieser Zielkonflikt läßt sich nach Ansicht des Verfassers mit den in PAS2 zur Verfügung stehenden Mittel nicht lösen.

7. Einiges zum Aufbau des Systems

Die Gesamthardwarekonfiguration ist in Abb.11 dargestellt.
Eingesetzt ist ein Prozessor ohne Floatingpoint-Unit, was bei der Vielzahl der Messungen zu einer fast totalen zeitlichen Auslastung führt. Das Soft-

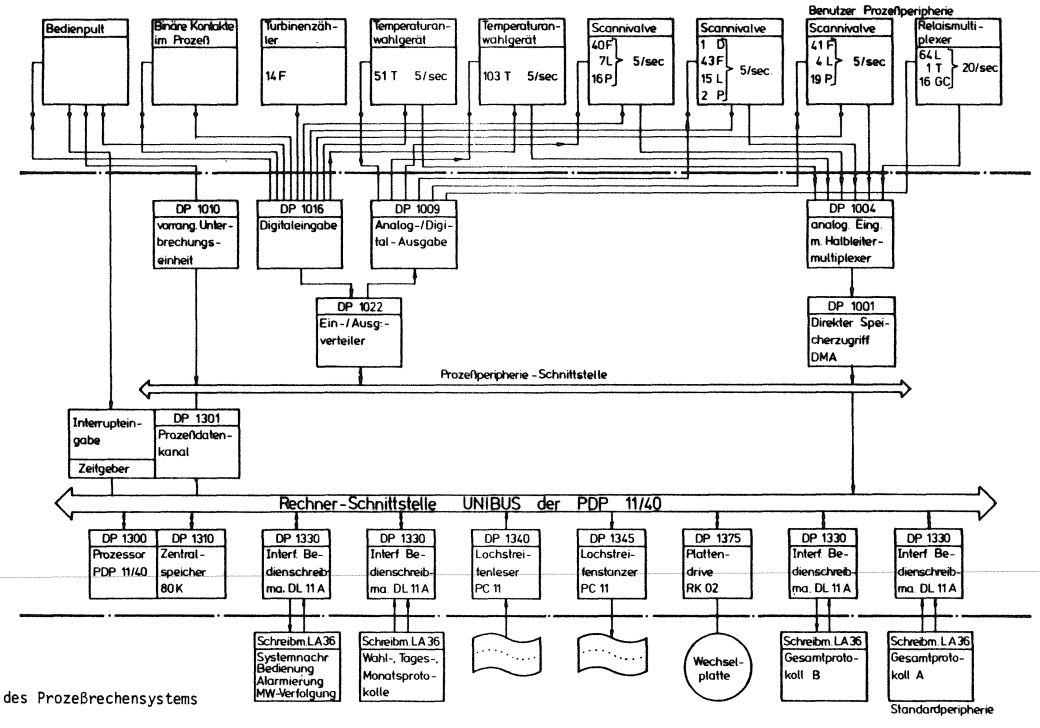


Bild 11. Hardwarekonfiguration des Prozeßrechensystems

waresystem ist rein arbeitsspeicherorientiert. Die Wechselplatte wird nur zum Laden des auf einem Programmentwicklungsrechner gleichen Typs compilierten und gelinkten Softwaresystem benötigt. Die die Meßstellen und ihre Verarbeitung beschreibenden Steuerdaten werden aus einer auf Lochkarten abgelegten Beschreibung auf einem Großrechner generiert, auf Karten ausgestanzt und bei der Kompilierung des Anwenderprogramms als initialisierte Datenbereichsdeklarationen mit eingegeben.

8. Weitere Entwicklungen

Bei der BASF Aktiengesellschaft werden zum Zwecke der Produktionsüberwachung und Steuerung häufig Rechner der PDP11-Familie eingesetzt. Während bei kleineren oder Standardsystemen ein eigenentwickeltes Realzeitbetriebssystem zum Einsatz kommt und in Assembler programmiert wird, greift man bei größeren Anwendungen auf das Betriebssystem RSX11-M von Firma DEC und Fortran als Programmsprache zurück.

Als Rechnerprozeßperipherie kommen eigenentwickelte Interfaces zur Ankopplung der Multiplexer und der analogen und binären Datenein- und -ausgaben an den UNIBUS oder den Q-BUS der DEC-Rechnerfamilie zum Einsatz. In diesen Interfaces wurden Funktionen wie Adressdekodierung, Erzeugung der Scanivalve-Pulse und Brummfilterung gleich hardwaremäßig integriert. Die eigenentwickelten Treiberroutinen für diese Geräte können ins Betriebssystem RSX11-M eingebracht und über die QIO-Directive aufgerufen werden. Die begonnene Entwicklung der DEC-PEARL-Implementation [9] sieht eine Möglichkeit vor, solche Treiber über BASIC-Dations anzusprechen. Der Verfasser empfindet die Aufgabe der PEARL-Entwicklung bei Fa. DEC in diesem Zusammenhang als sehr bedauerlich, begibt man sich dadurch doch der Möglichkeit, dieses ausgefeilte und erprobte Sprachkonzept bei diesen neueren Entwicklungen einzusetzen.

Anhang

Die Sourcecode-Listen der Erfassungstasks ZYKO (Abb. 12), ZYK1 (Abb. 13) und der von diesen gemeinsam benutzten Prozedur ERFAS (Abb. 14) sind dargestellt. Um die Niederschrift allzu PAS2-spezifischer Statements oder bekannter Algorithmen, wie z.B. zur Umkodierung zu vermeiden, wurden solche Passagen durch Pseudocode-Statements umschrieben. Diese sind an der Schreibweise: kleine Buchstaben mit vorangestelltem Punkt und der Formulierung in normaler Sprache wie ein Kommentar zu erkennen. Genaueres hierzu in [8].

Schrifttum

[1] V D I / V D E-Richtlinie 3555: Programmpakete für Meßwertverarbeitung und DDC. Düsseldorf 1977

[2] W ü c h n e r, W.: Erfahrungen mit dezentralen Systemen. VDI-Ber. Nr. 451, S. 59/62

[3] M a u r e r, K.: Effiziente Programme zur Generierung von Meßstellenprotokollen in Prozeßdatenverarbeitungssystemen nach dem Interpreterprinzip aufgrund des Ansatzes eines Automatenmodells. Angew. Inform. 24(1982) Nr. 5, S. 282/289

[4] K u s s l, V.: Logik des Programmierens. Düsseldorf 1979, S. 1/2

[5] W e n d t, S.: Einführung in die Begriffswelt allgemeiner Netzsysteme. Regelungstechnik 30(1982) Nr. 1, S. 5/12

[6] B r o w n, B o v e r i & C i e A G: PAS2(BBC-PEARL-Subset), Prozeßautomatisierungssprache, Sprachbeschreibung. Best.-Nr. D BBC 70049 D, Juli 1977

[7] W e n d t, S.: Petri-Netze und asynchrone Schaltwerke, Elektron. Rechenanl. (1974) Nr. 6, S. 206/216

[8] N e t t e s h e i m, H. (Bearb.): Programmentwurf und Programmdokumentation, Methoden und Techniken bei der Prozeßdatenverarbeitung. Düsseldorf 1982, S. 100/109

[9] D i g i t a l E q u i p m e n t C o r p.: PEARL Language User's Guide. Maynard 1979, S. 5-6/5-7

Anschrift des Autors

Maurer, Karl
c/o BASF Aktiengesellschaft
Abt. D-DEI/P (Technische Informatik)
D-6700 Ludwigshafen
Tel.-Nr. 0621/60-54160

```
ZYK0: TASK;

     /* Anwahlprogramm fuer 1 min.-Zyklus
        angesteuerte Multiplexer ZMA ,ZMA ,MUS ,SCV ,SVC
                                  1   2        2   3   */
/*****************************************************************/

/* Deklarationen

   Im letzten Schritt festgestellte VA-Nr. fuer ZMA ,..SCV */
                                                  1      3

   DCL AV(6) FIXED(15);

   /* Multiplexerindex */

   DCL M FIXED(15);
/*****************************************************************/

/* Aktivitaeten */

/* Es koennen Messwerte von ZMA ,ZMA ,MSU eingezogen werden */
                            1   2
   DO M = 1 TO 3; AV(M) = 0; END;

/* es koennen keine Messwerte von SCV  eingezogen werden
                                    1                    */
   AV(4) = -1;

/* ES koennen Werte von SCV  und SCV  eingezogen werden
                        2       3                    */
   DO M = 5 TO 6; AV(M) = 0; END;

/* Erfassung fuer Zyklus 0 (1 min) aufrufen */
   CALL ERFAS(AV,0);

   END /*ZYK0*/;
```

Bild 12. Code der Erfassungstask für 1 min.-Zyklus

```
ZYK1: TASK;

     /* Anwahlprogramm fuer 20 sec.-Zyklus
        angesteuerte Multiplexer ZMA , ZMA , MSU, SCV
                                  1     2         1 */
/*****************************************************************/

/* Deklarationen */

   DCL (AV(6),M) FIXED(15);

/*****************************************************************/

   DO M = 1 TO 4; AV(M) = 0; END;
   DO M = 5 TO 6; AV(M) = -1; END;

   CALL ERFAS(AV,1);

   END /*ZYK1*/;
```

Bild 13. Code der Erfassungstask für 20 s-Zyklus

```
ERFAS: PROC(AV,IZ);

     /*Fuer alle Anwahlprogramme gemeinsame Prozedur mit
       folgenden Funktionen:
      - Homelauf der anzusteuernden Scannivalves,
      - Aufsuchen der VA-Nrn. fuer alle Multiplexer aus
        der Liste MEANW,
      - solange VA-Nrn. zu bearbeiten:
        Ansteuern der Multiplexer,
        Einziehen der Messwerte 8 mal hintereinander,
        Mittelwertbildung aus je 8 Messungen (X-Werte),
        Uebergabe der VA-Nrn. und X-Werte an AVP,
        Steuerung der fuer das Einschwingen der Multi-
        plexer erforderlichen Wartezeiten              */

/*****************************************************************/

/* Deklarationen */
/* Referenz des Bereichs zur Aufnahme der VA-Nrn. aus der
   aufrufenden Task */

   DCL AV(*) FIXED(15);

 /* Kennung des Zyklus der aufrufenden Task */
   DCL IZ FIXED(3) VAL;
 /* Abfrageschrittzaehler */
   DCL IS FIXED(15)
 /* Multiplexerindex */
   DCL M FIXED(3);
 /* Laufindicer */
   DCL (I,IAV,J) FIXED(15);
 /* Zwischenspeicher */
   DCL CODE BIT(6), T1 CLOCK, BITS BIT(16);
 /* VA-Nrn. der in diesem Schritt eingezogenen Messungen */
 DCL UAVT(6) FIXED(15);
 /* Bereich zur Aufnahme der ADU-Auslseung (je 8 Werte fuer
    6 Messungen) */
 DCL FELD(48) FIXED(15);
/*****************************************************************/

/* Aktivitaeten */

/* Homelauf der anzusteuernden Scannivalves */
 DO /*1*/ M = 4 TO 6;
 IF AV(M) \= -1 THEN
 IF \(.SCV   in Homestellung.) THEN .Sende Homepuls an SCV   ;
          M-3                                              M-3
 END /*1*/;

 /* Warten, bis evetuell erfolgter SCV-Homelauf ausgefuehrt */
 DELAY DURING 4 SEC;

  /* Beginn der Schrittdurchfuehrung
  aus formalen Gruenden wird der Endwert des Laufindex an-
  gegeben  */
 DO /*1*/ IS = 1 TO 100;
 /* im vorhergehenden Schritt angewaehlte VA-Nrn. werden
    jetzt eingezogen */
 UAVT = AV;

/* Suchalgorithmus */
/* Fuer jeden in diesem Zyklus zu bearbeitenden Multiplexer
   wird die VA-Nr. der als naechste einzuziehenden Messung
    gesucht */
   DO /*2*/ M = 1 TO 6;
   IAV = AV(M);
/* Wird der Multiplexer in diesem Zyklus bearbeitet? */
   IF IAV \= -1 THEN DO /*3*/;

/* CODE fuer die Multiplexer/Zyklus-Kombination bilden, nach
   der in MEANW gesucht wird */
   CODE = BIT(M)!! BIT(IZ)
/* Durchsuchen von MEANW beginnend bei der auf die im
   letzten Schritt gefundene folgenden VA-Nr. */
   DO /*4*/ IAV = IAV+1 TO 600;
   IF (MEANW(IAV)&'111111'B) = CODE THEN
   DO /*5*/; AV(M) = IAV; GOTO M1; END /*5*/;
   END /*4*/;
/* Falls keine passende VA-Nr. mehr gefunden wurde, wird
   dies durch -1 kenntlich gemacht */
   AV(M) = -1; END /*3*/;
M1: END /*2*/;
```

Bild 14. Code der Erfassungsprozedur

```
/* die fuer MSU bis SCV3 gefundenen VA-Nrn. werden nach UAVT
   umgespeichert, da sie noch in diesem Schritt eingezogen
   werden */
   DO /*2*/ M = 1 TO 6; UAVT(M) = AV(M); END /*2*/
 /* Ist noch eine Messung von irgendeinem der 6 Multiplexer
    einzuziehen ? */
   DO /*2*/ M = 3 TO 6;
   IF UAVT(I) \= -1 THEN GOTO FORTS;
   END /*2*/;
 /* Wenn nein, ist der Abfragezyklus beendet */
   RETURN;



FORTS: /* Anwahl des MSU, falls erforderlich */
       IAV = UAVT(3); IF IAV \= -1 THEN DO /*2*/;
 /*Semaphore fuer MSU und ADU-Belegung anfordern */
  REQUEST(SMSU);
 /* Absenden der Zeilen- und Spalten-Nr. */
  DO /*3*/ I = 1 TO 2;
  J = .Ganze Zahl, gebildet aus MEANW(IAV).BIT(5+4*I:4+5*I)
  BITS = .(1 aus 16)- Codierung von J;
  SEND (MSU(I) FORM  BITS; END /*3*/; END /*2*/;
 /* Einziehen der.6 Analogeingaenge mit 10 ms Verzoegerung
    8 mal innerhalb 20 msec */
  CALL STATUS(BLOCK,19); TAKE(ANEIN) INTO (FELD);
 /* Freigabe der belegten Semaphoren */
  IF IAV \= -1 THEN RELEASE(SMSU);
  DO /*2*/ M = 1 TO 2;
  IF UAVT(M) \= -1 THEN GOTO L(M); ELSE GOTO E;
L(1): RELEASE(SZMA1); GOTO E;
L(2): RELEASE(SZMA2);
E: END /*2*/;

/*Anwahl von ZMA1 und ZMA2, falls erforderlich */
  DO /*2*/ M = 1 TO 2;
  IAV = AV(M); IF IAV \= -1 THEN DO /*3*/;
  /* Belegung der zugeordneten Semaphoren */
  GOTO MS(M);
MS(1): REQUEST(SZMA1); GOTO W;
MS(2): REQUEST(SZMA2);
W: J = .MEANW(IAV).BIT(7 : 16);
   DO /*4*/ I = 0 TO 2;
   K = . Stelle I von J gemaess : (I=0) (=) Hunderter,
         (I=1) (=) Zehner, (I=2) (=) Einer
   BITS = .(1 aus 10)-Codierung von K;
   SEND (TEMP(2*I+M)) FROM (BITS);
   END /*4*/; END /*3*/; END /*2*/;

  /* Die SCVs werden solange fortgeschaltet, bis wieder
     Position 1 erreicht ist */
  IF IS(6> THEN DO /*2*/ M = 4 TO 6;
  IAV = AV(M);
  IF IAV \= -1 THEN .Sende Steppuls an SCV   ;
                                        M-3
  END /*2*/;
 /* Aufnahme des Zeitpunkts der Fortschaltung */
  T1 = TIME;

 /* Uebergabe der Messwerte und VA-Nrn. an AVP */
  /*warten bis AVP fuer die Verarbeitung der Messungen
    frei ist */
    REQUEST(SERFAS); WAIT(EAVP);
  /* Zustandskennung "Puffer frei" zurucksetzen */
   RESET(EAVP); RELEASE(SERFAS);
 /* VA-NRN. in Puffer uebertragen */
  NUM = UAVT;
 /* Mittelung der X-Werte aus 8 Messungen fuer die 6
    Multiplexer udnd Uebertragung in den Puffer */
  DO /*2*/ M = 1 TO 6;
  If UAVT(M) ) 0 THEN X(M) =.Mittelwert aus den 8 Mes-
                            sungen fuer M aus FELD;
  END /*2*/;

 /* Verarbeitungsprozess anstossen */
  ACTIVATE(EAVP);
 /* Einschwingzeit fuer Multiplexer abwarten */
  DELAY DURING 150 MSEC - (TIME - T1);

 /* Ende der Schleife ueber Abfrageschritte */
  END /*1*/;

  END /*ERFAS*/
```

PEARL – spezifische Rechnerkopplung

G. Landsberg, H. Meyerhoff, Bremen

1. Zusammenfassung

Ausgehend von dem Ein-/Ausgabe-Konzept der Realzeitsprache PEARL in Form von DATIONs werden Probleme bei der Implementierung bezüglich Universalität und Effizienz beschrieben.
In einem konkreten Beispiel einer Rechnerkopplung wird eine Lösung beschrieben, bei der die Forderungen nach höchster Effizienz ohne Umgehung des PEARL-Sprachumfangs realisiert wurde:
Die PEARL-Dation wurde weitgehend in Hardware abgebildet.

2. Einleitung

Die Programmiersprache PEARL (Process and Experiment Real-time Language) bietet im Gegensatz zu anderen Sprachen [1; 2; 3;] im Sprachumfang festgelegte Möglichkeiten für Prozeß-Ein-/Ausgabe.
Der wichtigste Begriff ist hierfür die Datenstation (DATION), ein virtuelles Gerät oder Kanal, das mit Attributen beschrieben wird (z. B. IN, OUT, DIM, INTERRUPT usw.) und auf das Anweisungen (z. B. TAKE, SEND usw.) ausgeführt werden können.

Bei vorhandener Hardware und einer dafür implementierten PEARL-E/A bietet das für den Prozeß-Programmierer einen hohen Komfort und eine leichte und schnelle Programmierung bei relativer Fehlerfreiheit.
Die Probleme werden - wie es auch sinnvoll ist - auf den PEARL-Implementator verlagert, der das Kunststück fertigbringen muß, einen festgelegten Einheitshut über den ausgedehnten Bereich der Prozeß-Ein-/Ausgabe der verschiedensten Prozesse zu stülpen.

Der wesentliche Gegensatz besteht dabei zwischen den Forderungen

- universeller Einsatz des festgelegten E/A-Konzeptes für alle Prozeßanwendungen und
- höchste Effizienz bezüglich der zeitlichen Ausführung und der Rechnerbelastung in besonderen Fällen.

Die Lösung des Konfliktes kann nur ein Kompromiß sein, der dann tragfähig ist und für das festgelegte Sprachkonzept in PEARL spricht, wenn z. B. 80% der Anforderungen mit der realisierten Implementierung abgedeckt werden kann.

Die übliche Methode, die restlichen 20% abzudecken, ist die Einführung von Assembler-Unterprogrammen in einem PEARL-Modul.

In einem konkreten Anwendungsfall wird geschildert, wie ohne Verlassen des PEARL-Sprachraumes durch weitgehende Implementation einer DATION in Hardware die Forderungen nach zeitlicher Effizienz bei geringer Rechnerbelastung realisiert wurden.

Zum besseren Verständnis wird vorher das DATION-Konzept in PEARL geschildert und auf die bei der Implementation auftretenden Probleme eingegangen.

3. Datenstationen

Datenstationen haben die Aufgabe, Information zwischen dem Arbeitsspeicher des Systems und der Umgebung zu übertragen. Ihr grundsätzlicher Aufbau ist in Bild 1 dargestellt.

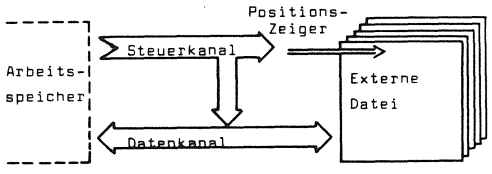


Bild 1 Bestandteile einer Datenstation

Vier Teile jeder Datenstation sind für den Programmierer von Interesse:

- Der Datenkanal
  Der Datenkanal bewirkt die eigentliche Informationsübertragung zwischen dem Arbeitsspeicher und der Umgebung (der externen Datei). Gewöhnlich werden die zu übertragenden Daten zunächst im Datenkanal zwischengespeichert und dann blockweise übertragen. Die Zwischenspeicherung kann sich gelegentlich auf die Funktionsweise auswirken, etwa bei einem Wechsel der Übertragungsrichtung.

- Der Steuerkanal
  Der Steuerkanal bewirkt die Steuerung des Betriebs der Datenstation entsprechend den Anforderungen des Programms. Er beeinflußt die Umwandlung der Daten zwischen ihrer externen Form und ihrer internen Form (der Form, in der sie im System gespeichert werden) und er steuert den Positionszeiger, der auf ein Element der externen Datei zeigt.

- Die externe Datei
  Eine Datenstation hat die Aufgabe, Daten zwischen dem Arbeitsspeicher des Systems und der Umwelt zu übertragen. Zum Verständnis des Konzepts der Datenstation ist es hilfreich, sich vorzustellen, daß die Daten außerhalb des Systems eine externe Datei bilden, so daß die Übertragung zwischen dem Arbeitsspeicher und der externen Datei stattfindet.

  Bei gewissen Datenstationen ist die Vorstellung, daß zur Datenstation eine externe Datei gehört, unmittelbar einleuchtend; beim Schnelldrucker z. B. besteht diese externe Datei aus dem bedruckten Papier, das den Drucker verläßt. Bei anderen Datenstationen existiert die externe Datei ebenfalls, allerdings nicht immer in einem so konkreten Sinne.

- Der Positionszeiger
  Dieser Zeiger gibt an, welche Position innerhalb der externen Datei von der Übertragung des nächsten Elements betroffen sein wird. Wenn der nächste Übertragungsvorgang eine Eingabe ist, zeigt der Zeiger auf dasjenige Element der externen Datei, das als nächstes gelesen wird, und wenn der nächste Übertragungsvorgang eine Ausgabe ist, zeigt der Zeiger auf diejenige Position in der externen Datei, auf die das nächste auszugebende Element geschrieben wird.
  Bei jeder Übertragung eines Elements schaltet der Zeiger automatisch um ein Element weiter (implizite Positionierung), so daß die Elemente der externen Datei ohne weiteres Zutun in fortlaufender Reihenfolge gelesen oder geschrieben werden können. Außerdem ist es möglich, mit Übertragungsanweisungen die Position, auf die der Zeiger weist, willkürlich zu verändern (explizite Positionierung) und so beim Schreiben oder Lesen von der natürlichen Reihenfolge der Dateielemente abzuweichen.

Grundlage jeder Ein- und Ausgabe sind die vorhandenen Geräte des Systems. Sie werden dem PEARL-Programm dadurch zugänglich gemacht, daß gewisse Datenstationen, die diesen Geräten unmittelbar entsprechen, dem System a priori bekannt sind. Die Namen und Eigenschaften dieser Datenstationen sind vollständig implementationsabhängig. Im Systemteil des PEARL-Programms werden die systemdefinierten Datenstationen in das PEARL-Programm eingeführt, indem der a priori bekannte Systemname der betreffenden Datenstation mit einem oder mehreren frei gewählten Benutzernamen assoziiert wird. In den Problemteilen können dann die Datenstationen unter ihren Benutzernamen angesprochen werden, nachdem sie jeweils spezifiziert worden sind.

Die Art der Spezifikation muß sich nach den Gelegenheiten der Hardware richten; dies ist

im einzelnen im Implementationshandbuch angegeben [7] .

Basis-PEARL erlaubt es dem Programmierer darüber hinaus, neue Datenstationen zu definieren; diese müssen über Datenwege mit den Datenstationen des Systems verbunden werden, da in Wirklichkeit natürlich Ein- und Ausgaben nur unter Verwendung der Hardware- und damit der Datenstationen des Systems - möglich sind. Die benutzerdefinierten Datenstationen können mit dem Attribut GLOBAL versehen werden und sind dann in allen Problemteilen des Programms nach entsprechender Spezifikation ansprechbar. Einzelheiten sh.[7].
Der Umgang mit Datenstationen ist für den Programmierer anfangs ungewohnt und für den Vorgang der Deklaration und der Eröffnung nicht ganz einfach. Das liegt u. a. darin, daß hiermit der Bezug zur speziellen Hardware des Systems hergestellt wird.
Die Anwendung im Problemteil ist dann jedoch einfach.
Eine Anweisung wie
SEND BYTE TO LAMPENFELD
ist schon Umgangssprache und leicht verständlich.

Warum die Ausführung obiger Anweisung dann aber ca. 1 ms dauert, ist vielen "Assemblerprogrammierern", die eine Ausgabe in 10 Assemblerbefehlen hinschreiben, restlos unklar.

Dazu muß man bedenken, daß die universelle DATION viele Sonderfälle abfangen muß. Im Laufzeitpaket für die DATION werden z. B. folgende Funktionen abgewickelt:

- Aufbereitung von Parametern und Adressen für die interne Schnittstelle.
- Überprüfung der Gültigkeit der Operation (z. B. ist die DATION vorhanden und eröffnet).
- Zugriffserlaubnis: Darf die aufrufende Task eines Laufbereiches (Users) auf die DATION zugreifen.
- Einstellung der DATION auf die Transferrichtung.
- Transfer des Objektes in den DATION-Buffer und evtl. Blockung der Daten.
- Status-und Fehleraufbereitung und Rücksprung.

Die genannte Zeit von 1 ms ist für die meisten Anwendungen ausreichend. Es muß jedoch berücksichtigt werden, daß für 1 ms auch Rechenzeit benötigt wird. Es kann in Ausnahmefällen Anwendungen geben, bei denen die Dauer der Ausführung und/oder die Rechnerbelastung mit den Anforderungen nicht in Einklang zu bringen sind. Von einem derartigen Fall wird im folgenden berichtet.

4. Das Projekt " BON "

Bei dem "BON - System" handelt es sich um eine zentrale Leitwarte für ein Nahverkehrssystem (sh. Bild 2).

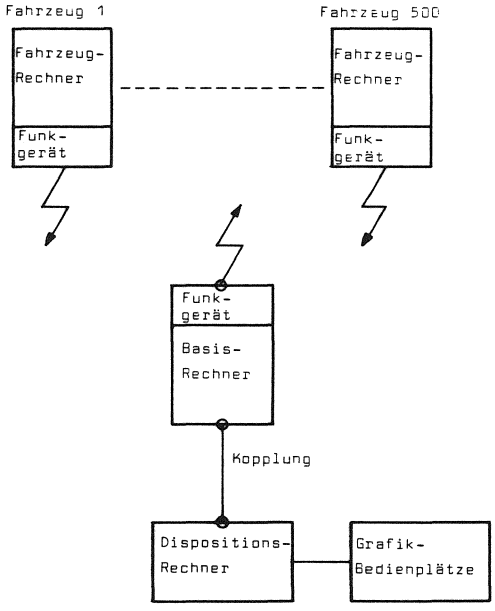


Bild 2: Leitsystem Nahverkehr (BON)

Über ein Funkinterface sind bis zu 500 Rechner in Fahrzeugen (Busse, Bahnen) mit der Leitwarte gekoppelt.
Die Aufgaben der Leitwarte werden auf dem Basisrechner (Aufgaben z. B. Fahrzeugaufrufe, Standortverfolgung) und auf dem Dispositionsrechner (Aufgaben z. B. Fahrplan Soll-Ist-Vergleich, Statistik) verteilt. Die Bedienung und die Ergebnisdarstellung (z. B. Standortdarstellung der Fahrzeuge) erfolgt auf Farb-Bildschirmgeräten.

Die bezüglich der Effizienz und der Rechnerbelastung kritischen E/A-Schnittstellen sind:

1. Schnittstelle des Basisrechners zum Funkinterface:
   Sämtliche Informationen von und zu den Fahrzeugrechnern werden über dicht gepackte ASCII-Telegramme der verschiedensten Formen abgewickelt (Funkbetrieb und 2400 Baud über 2 Schnittstellen). 50 Telegramme pro Sekunde müssen bei 30% Rechnerauslastung verarbeitet werden können.

2. Schnittstelle des Basisrechners zum Dispositionsrechner (Rechnerkopplung). Der Dispositionsrechner benötigt für seine Aufgaben einen schnellen Zugriff zu dem Datenmodell des Basisrechners (und umgekehrt). Die Datenkopplung muß schnell und ohne wesentliche Belastung des Basisrechners erfolgen.

Die Software des Projekts sollte in PEARL erstellt werden. In einer Vorstudie wurde festgestellt, daß für die beiden kritischen Schnittstellen die Forderungen unter Verwendung der Standard-DATION-Ein-/Ausgabe nicht erfüllt werden konnten.

Vom Lieferanten der Hardware und des PEARL-Systems - die Firma Krupp Atlas-Elektronik - wurde daraufhin für diese beiden Schnittstellen ein Verfahren vorgeschlagen und realisiert, das im folgenden am Beispiel der Rechnerkopplung ausführlicher geschildert wird.

5. Rechnerkopplung

Bild 3 zeigt die Hardwarekonfiguration.

Die eigentliche Parallel-Schnittstelle wird von einem programmierbaren E/A-Prozessor betrieben, der nach Initialisierung durch die CPU zum Speicher des Rechners Zugriff hat (per DMA, Direct-Memory-Access).

Ohne Mitwirkung der CPU werden Daten vom E/A-Prozessor aus dem Speicher geholt (oder zum Speicher gesendet), evtl. interpretiert und zur Parallelschnittstelle übertragen. Der Prozessor kann einen Hardware-Interrupt auslösen.

Das Verfahren besteht darin, die durch ein PEARL-Programm festgelegten strukturierten

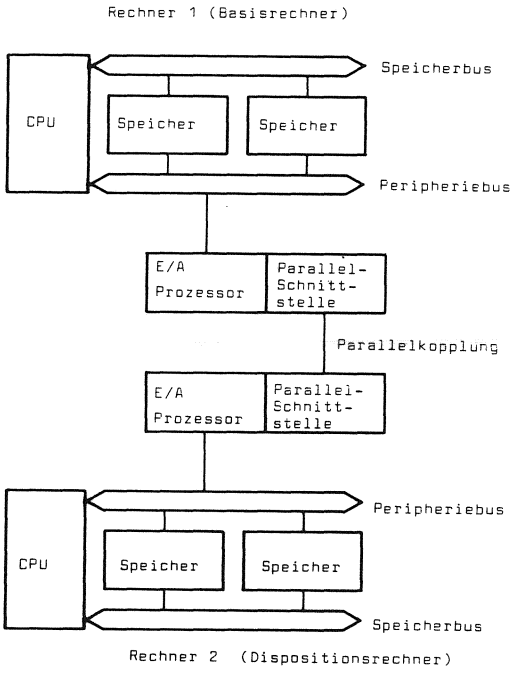


Bild 3: Hardware-Blockschaltbild, Rechnerkopplung

Daten nicht durch SEND- und TAKE-Anweisungen einer "DATION Parallelkopplung" zu transportieren, sondern diese Funktionen in den E/A-Prozessor zu verlegen.

Für den Fall der Datenkopplung von Rechner 2 (Dispositionsrechner) zum Rechner 1 (Basisrechner) werden die Hardwarefunktionen und die dazugehörigen PEARL-Anweisungen näher erläutert.

In einem globalen Datenmodul KOPLNG (Kopplung) werden die zu übertragenden Daten zusammen mit Steuerungsparametern durch Deklarationen aufbereitet (sh. Bild 5).

Der Block der Steuerungsparameter besteht jeweils aus drei Worten:

LNG, die Länge des Datenblocks

AUFT, eine gesetzte 1 bedeutet Start der Übertragung

SEMA, Die Nummern des globalen Semaphor des Partnerrechners, auf die nach Übertragungsende ein RELEASE erfolgen soll, womit ein Auswerteprogramm fortgesetzt wird.

Dem E/A-Prozessor wurde in der Initialisierungs-

phase des Gesamtsystems die Anfangsadresse des globalen Datenmoduls mitgeteilt. Die am Anfang des jeweiligen Datenblocks stehende Länge des Datenblocks gibt ihm die Informationen, um zyklisch das Auftragsbit AUFTR abzufragen.

Bei Vorliegen einer 1 werden die Daten einschließlich der Parameter zum anderen Rechner über die Parallelschnittstelle übertragen und die Daten dort in einem völlig symmetrisch aufgebauten Datensegment abgespeichert. Es kann nach Abspeicherung ein Interrupt ausgelöst werden. Daraufhin liest ein Treiber die mit übertragene Nummer des Semaphor aus. Durch ein RELEASE des Semaphor wird das aufgerufene Programm auf dem Rechner 2 fortgesetzt. Wenn der E/A-Prozessor eine Längenangabe 0 vorfindet, beginnt er seinen Poll-Zyklus wieder bei der Anfangsadresse (sh. Bild 5). Die Software-Struktur zeigt Bild 4.

Ein Modul FUB (Funkbedienung) will das Modul FZA (Fahrzeugaufruf) ansprechen. Da es sich in einem anderen Rechner befindet, wird ein Modul FZAKOP (Dummy Fahrzeugaufruf) angesprochen. Dieses Dummy-Modul startet die Kopplung. Die Parameter und die Daten werden mittels der Parallelkopplung übertragen. Auf der anderen Seite wird ein Auftragsmonitor angesprochen, der den eigentlichen Fahrzeugaufruf aktiviert.

Diese Struktur mit dem Dummy-Modul wurde gewählt, um eine Software-Struktur zu erhalten, bei der ohne

```
MODUL     KOPLNG     GLOBAL (1);
PROZEDUR;
.
.
.
.
.
.
DCL       LNG1       FIXED     GLOBAL    INIT(20);
DCL       AUFTR1     BIT(1)    GLOBAL    INIT ('0'B);
DCL       SEMA1      FIXED     GLOBAL;
DCL       DATEN1 (20) FIXED    GLOBAL;
.
.
.
DCL       POLL       FIXED     INIT (0);
MODEND;
```

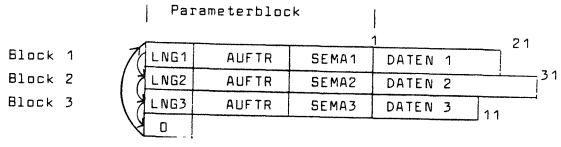


Bild 5 Speicherabbild der im PEARL-Modul KOPLNG (Kopplung) aufbereiteten Daten

große Änderungen die Module FUB (Funkbedienung) und FZA (Fahrzeugaufruf) auch auf einem Rechner ablaufen könnten.
Für das Modul FUB ist es "unsichtbar", ob das Modul FZA auf dem eigenen oder auf dem gekoppelten Rechner abläuft.

Ein vereinfachtes PEARL-Programm ist in Bild 6 und Bild 7 dargestellt.

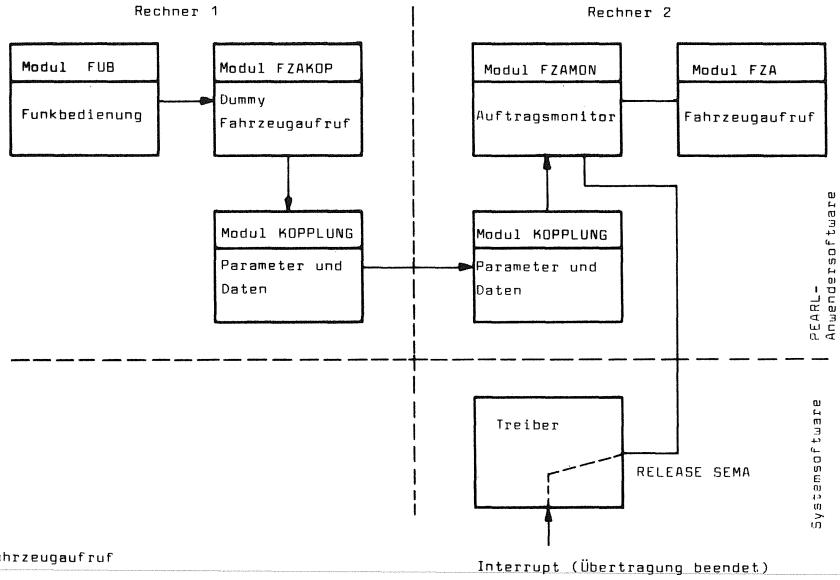


Bild 4: Struktur des PEARL-Systems für Rechnerkopplung: Funkbedienung bewirkt Fahrzeugaufruf

```
MODULE FUB;
PROBLEM;
SPC SCHNITTSTELLE ENTRY (FIXED) RESIDENT GLOBAL;
.
.
.
CALL SCHNITTSTELLE (E);    /* Schnittstellenaufruf */
.
.
.
MODEND;


MODULE FZAKOP;
PROBLEM;
.
.
.
SPC      AUFTR1      BIT(1);
SPC      DATEN1(20)  FIXED;
DCL      START       INV        BIT(1)  INIT ('1'B);

SCHNITTSTELLE: PROC (E FIXED) RESIDENT GLOBAL;
DATEN1 (1):= E;      /* Übergabe des Eingabeparameters */
AUFTR1:= START;      /* Start der Parallelkopplung */
END;
MODEND;
```

Bild 6 Vereinfachtes PEARL-Programm für die Rechnerkopplung, MODULE FUB (Funkbedienung) und FZAKOP (Dummy Fahrzeugaufruf) auf dem Rechner 2 (Dispositionsrechner)

```
MODULE FZAMON;
PROBLEM;
SPC SEMA1     SEMA GLOBAL;
SPC SCHNITTSTELLE ENTRY (FIXED) RESIDENT GLOBAL;
SPC DATEN1 (20) FIXED;

AUFTRAGSMONITOR: TASK PRIO1 RESIDENT GLOBAL;
REPEAT;
REQUEST SEMA1;  /* Warten auf Fertigwerden Parallelkopplung */
CALL SCHNITTSTELLE (DATEN1 (1)); /* Schnittstellenaufruf */
END;
MODEND;

MODULE FZA;
PROBLEM;
SPC DATEN1 (20) FIXED;
.
.
.
SCHNITTSTELLE: PROC (E FIXED) RESIDENT GLOBAL;
.
.
.
MODEND;
```

Bild 7 Vereinfachtes PEARL-Programm für die Rechnerkopplung, MODUL FZAMON (Fahrzeug-Monitor) und FZA (Fahrzeugaufruf) auf den Rechner 1/ Basisrechner

6. Ergebnis

Das Ergebnis des Verfahrens kann durch folgende Angaben charakterisiert werden:

- Ohne Verlassen des PEARL-Sprachraumes wurde ein effizientes Verfahren der Daten-Ein-/Ausgabe realisiert. Eine in Software implementierte DATION wurde umgangen.
- Pro Auftrag wird ein Grundaufwand von 100 µsec benötigt. Dazu ist u. a. der hier nicht beschriebene Aufwand für die Rückmeldung des erfolgreichen Transfers und die Datensicherung enthalten. Pro Datenwort sind 3,5 µsec für die Übertragung erforderlich. 170 µsec nach dem PEARL-Startbefehl (AUFTR1:=START, sh. Bild 6) steht der Datenblock von 20 Worten im Speicher des anderen Rechners.
  Die Zentraleinheit wird dabei so gut wie nicht belastet.
- Die Flexibilität einer Softwarelösung bleibt erhalten: Ohne Änderungen der Hardware können die Anzahl der Software-Koppelverbindungen und die Struktur der Daten verändert werden.
- Mit diesem Verfahren wurde gleichzeitig ein Beispiel gegeben, wie PEARL-Programme auf mehrere Rechner verteilt werden können.
- Das Ergebnis kann damit beschrieben werden, daß eine PEARL-DATION weitgehend in Hardware realisiert wurde.
  Eine komplette - für den Programmierer unsichtbare - Lösung einer Hardware-DATION kann durch folgende zusätzliche Maßnahmen erreicht werden:
  - o Die Aufbereitung der Parameterblöcke - in dem geschilderten Beispiel ein Teil des Anwenderprogramms - könnte der Kompilierer übernehmen.
  - o Ebenso könnte der Kompilierer die TAKE- und SEND-Anweisungen in entsprechende Start-Anweisungen für die Hardware umwandeln (z. B. SEND entspricht dem PEARL-Befehl nach Bild 6 in Modul FZAKOP (AUFTR1:=START).

7. Schlußbemerkung

Es wurde gezeigt, wie bei hohen Anforderungen an die Effizienz der Ein-/Ausgabe eines PEARL-Automatisierungssystems die Forderungen erfüllt werden können, ohne den PEARL-Sprachraum zu verlassen.

Es wurde ansatzweise die PEARL-Dation in Hardware realisiert.

Die gezeigte Struktur ist geeignet, PEARL-Programme auf mehrere Rechner zu verteilen.

Gleichzeitig ist die Realisierung ein Beispiel dafür, wie eine Hardware an die Struktur einer Software angepaßt werden kann.

Derartige Lösungen können nur bei enger Zusammenarbeit zwischen Systemhersteller und Anwender erreicht werden.
Der Lösungsansatz wird zur Nachahmung empfohlen.

8. Literatur

[1] P. Elzer:
Kurzüberblick über Entwicklung und Eigenschaften von PEARL im Vergleich mit anderen Programmiersprachen.
PEARL Rundschau Band 2 Nr. 2 1981

[2] A. Schwald, R. Baumann:
PEARL im Vergleich mit anderen Echtzeitsprachen.
PEARL Rundschau Band 2 Nr. 2 1981

[3] H. Sandmayr:
Sprachen für Echtzeitprogrammierung.
Fachtagung Prozeßrechner 1981, München.

[4] DIN 66253:
Teil 1 - Basic PEARL (Vornorm)
Teil 2 - Full PEARL (Normentwurf)

[5] B. Boysen:
PEARL auf einer 16 Bit-Rechenanlage mit Mehrbenutzersystem.
PEARL Rundschau Band 2 Nr. 6 1981

[6] M. Amann:
PEARL für verteilte Systeme.
Informatik-Fachberichte 39 1981,
Springer Verlag

[7] Krupp Atlas-Elektronik:
PEARL-Sprachbeschreibung BL 2052 A 179 4.82
PEARL-Implementationsbeschreibung BL 2052 A 166 8.81
PEARL Testmonitor BL 2052 A 158 6.81.

Wirtschaftliche Implementierung von PEARL auf Mikroprozessoren: INTEL RMX86 -PEARL

P. Heine, F. Kaiser, IITB Karlsruhe; **R. Schneider,** INTEL München

Zusammenfassung

Es wird eine Implementierung von PEARL auf einer INTEL 8086/87-basierenden Hardware-Konfiguration vorgestellt. Unter dem Gesichtspunkt der Wirtschaftlichkeit wurden dabei, soweit als möglich, bereits existierende Programm-Pakete benutzt.

Summary

We present an implementation of PEARL on an INTEL 8086/87-based hardware configuration. For economic reasons as far as possible we explicitely selected already existing program packages to be parts of the system.

1. Einleitung

Für viele der derzeitig verfügbaren Mikro-Rechner existieren leistungsfähige Betriebssysteme, die alle einen vergleichbaren Satz an multi-tasking and realzeit-Eigenschaften als Minimum enthalten. Auch kristallisiert sich für einige Hochsprachen heraus, daß sie künftig zur Standard-Ausrüstung solcher Rechner gehören werden. Es liegt damit nahe, daß das Ausnutzen dieser beiden Tatsachen eine äußerst wirtschaftliche Implementierung von PEARL auf Mikro-Rechnern ermöglichen kann.
Deshalb wurden bei der hier vorgestellten Implementation von PEARL auf einem Entwicklungssystem und Single-board-computer der Fa. INTEL nach Möglichkeit bereits existierende Standard-Bausteine, -Schnittstellen und -Hochsprachen benutzt.

Die Schnittstellen des Compilers, sowie der Laufzeit-unterstützenden Prozeduren zu den darunter liegenden Schichten, können somit weit oberhalb der Maschinen-Ebene angelagert werden. Die darunter liegenden Schichten sind dann durch die lokalen Implementierungen verwirklicht. Im vorliegenden Fall wurden der existierende INTEL PASCAL-Compiler: PASC86 [1] und das RMX86-Betriebssystem [2] bei der PEARL-Implementierung miteinbezogen.

Bei diesem Vorgehen wurde ein Verlust an Leistungsfähigkeit dadurch vermieden, daß wenige, aber für die Laufzeit wesentliche Anteile klar abgetrennt und in Assembler-Sprache codiert wurden. Die Abb. 1 gibt einen Überblick über das gesamte Programmsystem, das im folgenden näher beschrieben wird.

2. PEARL-Compiler

Der PEARL-Compiler ist von der Fa. Werum erstellt worden.

Sein Sprachumfang ist in [3] festgelegt. Er geht weit über BASIC-PEARL hinaus. Der Code-Generator des PEARL-Compilers erzeugt PASCAL-Quellcode als Zwischensprache. Diese wird zum Erzeugen des Maschinencodes, als letzter Schritt des Übersetzens, vom PASC86-Compiler weiterverarbeitet. Der Sprachumfang dieses Compilers ist eine Obermenge des ISO-Normungsvorschlags für STANDARD PASCAL. Der PEARL-Compiler selbst liegt in PASC86-Code vor. Er läuft auf dem INTEL MDS SERIES III Entwicklungssystem [4] als Overlay-Programm unter der Kontrolle des standard ISIS-Betriebssystems [7]. Zugriff zum ISIS-Betriebssystem erfolgt über die standardisierte INTEL-interne UDI-Schnittstelle [5]. Dadurch ist der PEARL-Compiler auch unter künftigen INTEL-Entwicklungssystemen ablauffähig, die der UDI-Schnittstelle genügen.

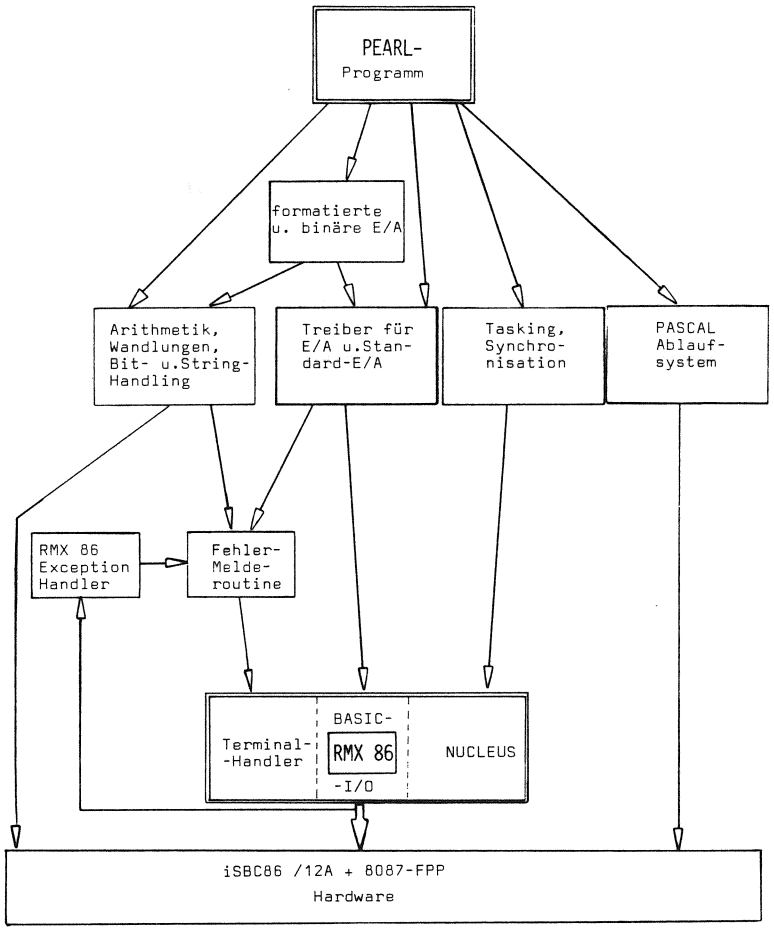


Abb. 1: Ablaufschema eines PEARL-Jobs unter dem RMX86-Betriebssystem

Wesentliche Abbildungen der Strukturen PEARL- nach PASC86-Moduln sind:

- Ein PEARL-Modul wird in einen PASC86-non-main-Modul abgebildet.
- Eine PEARL-Task wird in ein PASC86-Prozedur abgebildet. Diese wird dann später, zur Laufzeit, dem Betriebssystem als Task bekannt gemacht.
- Jeder PASCAL-Modul enthält eine Initialisierungs-prozedur. Diese erledigt alle Modul-spezifischen Initialisierungen. So macht sie z.B. die Tasks des Moduls dem Betriebssystem bekannt oder erzeugt Objekte, die zur Verwaltung von Warteschlangen dienen.
- Der PEARL-Compiler generiert eine INCLUDE-Anweisung für ein File, in dem die Typ-Definitionen für alle eigen erzeugten Objekte stehen wie Task-, Interrupt-, Semaphor- und Bolt-Kontrollblöcke.

Die Initialisierung dieser Struktur der generierten PASC86-Moduln ist in Abb. 3 skizziert. Sie wird in Abschnitt 6 näher behandelt.

3. Betriebssystem

Ein gebundenes PEARL-Programm läuft unter der Kontrolle des INTEL RMX86-Betriebssystem ab. Dieses Betriebssystem bietet die wesentlichen Funktionen, die bis auf wenige Ausnahmen (BOLT-Variablen, SIGNALs, zeitliche- und Unterbrechungs-Einplanungen von Tasks) eine einfache Abbildung der PEARL-auf den RMX86-Funktionen gestatten.

So sind z.B. die PEARL-Anweisungen ACTIVATE, SUSPEND, CONTINUE, RESUME, REQUEST, RELEASE nahezu unmittelbar auf RMX86-Aufrufe abbildbar; für Zeit- und Interrupt-Einplanungen, sowie für die Behandlung von BOLT-Variablen müssen eigene Warteschlangen aufgebaut werden. Die vorliegende Implementierung des RMX86 ist eine Konfiguration bestehend aus NUCLEUS, BASIC I/O und TERMINAL-HANDLER.

4. Laufzeit-Unterstützung

Die PEARL-Laufzeitroutinen dienen der Abarbeitung in PASC86 nicht vorhandener Operationen wie der Behandlung von Bit- und Zeichen-Ketten,

dem Aufruf von RMX86-Funktionen zur Manipulation von RMX86-Objekten oder der Manipulation eigener Objekte.

Im einzelnen lassen sich die PEARL-Laufzeitroutinen wie folgt klassifizieren:

- PEARL-Arithmetik
- Formatierte und binäre PEARL-Ein-/Ausgabe
- PEARL-Ein-/Ausgabe-Treiber
- PEARL-Tasking und -Synchronisation
- Fehlerbehandlung
- Standard INTEL PASC86- und 8087-floating-point-prozessor-Unterstützung

4.1 PEARL-Arithmetik

Diese Routinen enthalten die in PASCAL nicht vorhandenen arithmetischen und logischen Operationen mit PEARL-Variablen vom Typ BIT, CHARACTER, Doppelwort Integer FIXED (31) u.ä., sowie Typwandlungen.

Diese Routinen sind in Assembler-Sprache geschrieben.

4.2 Formatierte und binäre PEARL-Ein-/Ausgabe

Diese Gruppe von Laufzeitroutinen bearbeitet die vielfältigen Möglichkeiten der Ein-/Ausgabe, die in PEARL geboten werden und bildet diese auf eine Geräteunabhängige Schnittstelle zum Ein-/Ausgabetreiber ab.

Diese Routinen sind in PEARL geschrieben und stellen den umfangreichsten Teil des Laufzeitsystems dar.

4.3 PEARL-Ein-/Ausgabe-Treiber

Der Ein-/Ausgabetreiber stellt die Verbindung von formatierter und binärer Ein-/Ausgabe zum RMX86 Basic I/O Betriebssystemanteil dar. Unterstützt werden alle Geräte, die auch vom Basic I/O System unterstützt werden und die vom File-Typ 'named' oder 'physical' sind.

Die Aufgabe des Ein-/Ausgabetreibers ist die Datenübertragung aus einem bzw. in einen von der formatierten und binären Ein-/Ausgabe bereitgestellten Datenpuffer, das Einstellen der aktuellen Schreib-/Leseposition sowie das Öffnen und Schließen von Files.

Die Abbildung der PEARL-Ein-/Ausgabe auf die Basic I/O des RMX86 ist auf diesem Niveau der Treiber stark dadurch vereinfacht, daß beide Seiten der Abbildung nach dem Konzept der Geräteunabhängigkeit gestaltet sind, d.h. die Struktur der Funktionsaufrufe ist dieselbe für alle Geräte.

Der Ein-/Ausgabetreiber ist vorwiegend in PLM86 programmiert.

4.4 PEARL-Tasking und -Synchronisation

Der Zugriff zu PEARL-Tasks, um unbedingte-, Zeit- oder Ereignis-gesteuerte Zustandsänderungen zu bewirken, erfolgt über diese Laufzeitroutinen.

Unbedingte Aufrufe, wie z.B. ACTIVATE, SUSPEND, CONTINUE bewirken unmittelbar die entsprechenden RMX86-NUCLEUS-Aufrufe.

Von den PEARL-Synchronisationsobjekten Semaphor- und Bolt-Variablen sind nur die Semaphoren direkt auf das RMX86 abbildbar. Für die Bolt-Variablen müssen eigene Warteschlangen aufgebaut und verwaltet werden. Ebenso werden Task-Einplanungen für externe Interrupts und zeitgesteuerte Ereignisse von eigenen Warteschlangen verwaltet. Das Laufzeitsystem enthält dazu pro externem Interrupt eine Interrupt-Task, welche die Laufzeitroutinen für die zugehörigen Einplanungen aufruft. Und insbesondere existiert eine Timer-Task, die externe Interrupts bedient und die Warteschlangen für zeitliche Task-Einplanungen bearbeitet. Ganz allgemein werden für die Interrupt-Behandlung die Möglichkeiten genutzt, die das RMX86 zur Verfügung stellt, nämlich die 2-stufige Behandlung eines Interrupts über einen Interrupt-Handler und eine Interrupt-Task (Abb. 2). Die Routinen für Tasking und Synchronisation sind überwiegend in PASCAL programmiert.

4.5 Fehlerbehandlung

Werden innerhalb der Laufzeitroutinen oder der RMX86-Operationen Fehler entdeckt, so folgt über eine zentrale Fehlerroutine eine Fehlermeldung auf dem Bildschirm. Ein Abbruch der fehlerhaften Task ist durch den Aufruf einer entsprechenden Routine möglich.

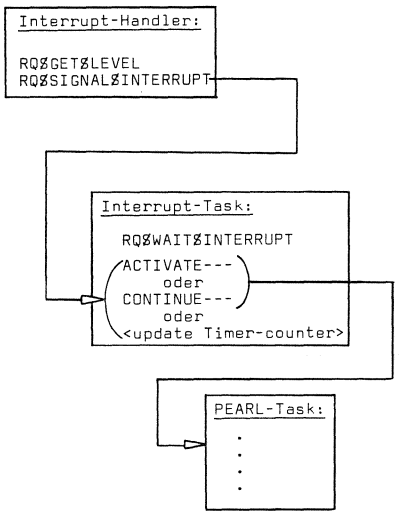


Abb. 2: Ablaufschema eines externen Interrupts

4.6 Standard INTEL PASC86- und 8087-Floating-point-prozessor-Unterstützung

Die von PASC86 benötigten Laufzeitroutinen werden aus den dazugehörigen Bibliotheken nach Bedarf dazugebunden. Es werden allerdings lediglich Routinen zur dynamischen Speicherplatzverwaltung (PASCAL-Anweisung NEW) und solche für einige Einbaufunktionen (z.B. SIN, COS) aus dem PASC86-Laufzeitsystem benötigt.

Der PASC86-Compiler nutzt voll die Möglichkeiten des 8087-Floating-point-Prozessors [6] aus. Und dies gilt damit auch für PEARL-Programme. Hierzu sei bemerkt, daß die Benutzung des 8087-Floating-point-Prozessors nicht nur einen erheblichen Laufzeitgewinn bringt, sondern auch das Einhalten des IEEE Floating point Standards garantiert. Der PASC86-Compiler erzeugt direkten Code für den 8087-Prozessor. Lediglich eine Laufzeitroutine muß zur Initialisierung der 8087-Register während der Initialisierungsphase einer PEARL-Task aufgerufen werden.

5. Binder

Die Möglichkeit, Moduln verschiedener Sprachen zusammenbinden zu können, war eine wesentliche Voraussetzung dafür, wirtschaftlich u n d Laufzeit-effizient sein zu können. So konnten kleine, laufzeitintensive Teile in Assembler geschrieben werden, maschinenabhängige größere Teile in PLM86 und der Hauptteil als portable PASC86-Moduln; und wie unter 2. beschrieben, werden aus den PEARL-Anwenderprogrammen ebenfalls PASC86-Programme generiert.

Für das Binden, Verabsolutieren und Einbringen in eine Bibliothek aller benötigter Laufzeit-Routinen werden die Standard-Dienstprogramme LINK86, LOC86, LIB86 [7] benutzt.

Gebundene und verabsolutierte PEARL-Moduln bilden einen RMX86-Job. Zusammen mit den vorkonfigurierten RMX86-Jobs 'Terminal Handler' und 'Basic I/O' ist der PEARL-Job ein Abkömmling des RMX86-root-Jobs. Nach Binden und Verabsolutieren des PEARL-Jobs muß deshalb dieser root-Job noch erzeugt werden. (Dabei wird die Endadresse des PEARL-Jobs in das Konfigurationsfile des root-jobs eingetragen.) Danach müssen der RMX86 Nucleus, die RMX86-Jobs, der PEARL-Job und der root-Job in einer Bibliothek bereitgestellt werden, um ein ablauffähiges Programmsystem zu erhalten.

All dies erfolgt automatisch, gesteuert von einem Dialog-Programm, das die Schnittstelle zum Benutzer darstellt. Der Benutzer ist somit davon befreit, über die Techniken der Implementierung informiert sein zu müssen. Als Ergebnis des Dialogs werden Kommandoprozeduren erzeugt, die dann die oben beschriebenen Funktionen ausführen. Im einzelnen werden dabei die folgenden Schritte durchlaufen:

-- Erzeugen eines verabsolutierten PEARL-Jobs:
  - Binden der PEARL-Job Initialisierungs-Task
  - Binden der PEARL-Moduln
  - Binden der PEARL-Laufzeitroutinen
  - Verabsolutieren des Gebundenen

-- Erzeugen eines verabsolutierten root-jobs:
  - Eintragen der Endadresse des PEARL-Jobs in das Konfigurationsfile des root-jobs
  - Binden und Verabsolutieren des root-jobs

-- Erzeugen einer ladefähigen Bibliothek aus:
  - nuclus
  - basic I/O
  - terminal-handler
  - PEARL-job
  - root-job

6. Initialisierung

Die Initialisierung des gesamten PEARL-Programmsystems erfolgt durch die Initialisierungs-Task des PEARL-jobs. Diese Task wird ebenfalls als Ergebnis des unter 5. beschriebenen Dialogs generiert.

Im einzelnen werden dabei die folgenden Initialisierungen durchgeführt:

- pro PEARL-Task
  (z.B. Initialisieren des 8087-floating-point-prozessors)
- pro PEARL-Modul
  (z.B. Erzeugen von RMX86- und PASC86-Objekten entsprechend den PEARL-Objekten wie Tasks, Semaphoren etc.)
- global für alle PEARL-Moduln
  (z.B. Initialsieren des PEARL-Timers)
- global für das gesamte PASC86-Laufzeitsystem
  (z.B. Aufrufen der PASC86-Laufzeitroutine TQ001 zum Initialisieren der dynamischen Speicherverwaltung in PASC86)
- global für das Gesamtsystem:
  Aktivieren der als PEARL-Start-Task bestimmten Task(s)

Diese Struktur der Initialisierung ist in Abb. 3 skizziert.

7. Implementierungserfahrung

Wie bereits in der Einleitung betont, wurden bei der vorliegenden Implementierung von PEARL auf einem INTEL Mikrocomputer System soweit wie möglich die Funktionen des bereits existierenden Betriebssystems ausgenutzt. Weiterhin wurde eine höhere Programmiersprache bei der Übersetzung als Zwischensprache wie auch als System-Implementationssprache verwendet. Die Erfahrungen, die bei der Implementierung gewonnen wurden, betreffen im wesentlichen die Verwendung dieser Programmpakete.

Das RMX86 ist ein mächtiges, modulares und bis in feine Strukturen konfigurierbares Realzeit- und Multitasking-Betriebssystem. Um bei der Implementierung der Laufzeitfunktionen von PEARL möglichst wirtschaftlich sein zu können, mußten für die Verwendbarkeit des RMX86 wesentliche Betriebssystem-Funktionen zur Task- und Speicherplatz-Verwaltung sowie Standard-Ein-/Ausgabe einfach abbildbar bereits darin vorhanden sein. Die feinkörnige Konfigurierbarkeit des RMX86 half dabei, ein Betriebssystem zu erzeugen, das genau den gewünschten Satz an Funktionen enthält. Daß diese minimale Konfiguration dennoch einen

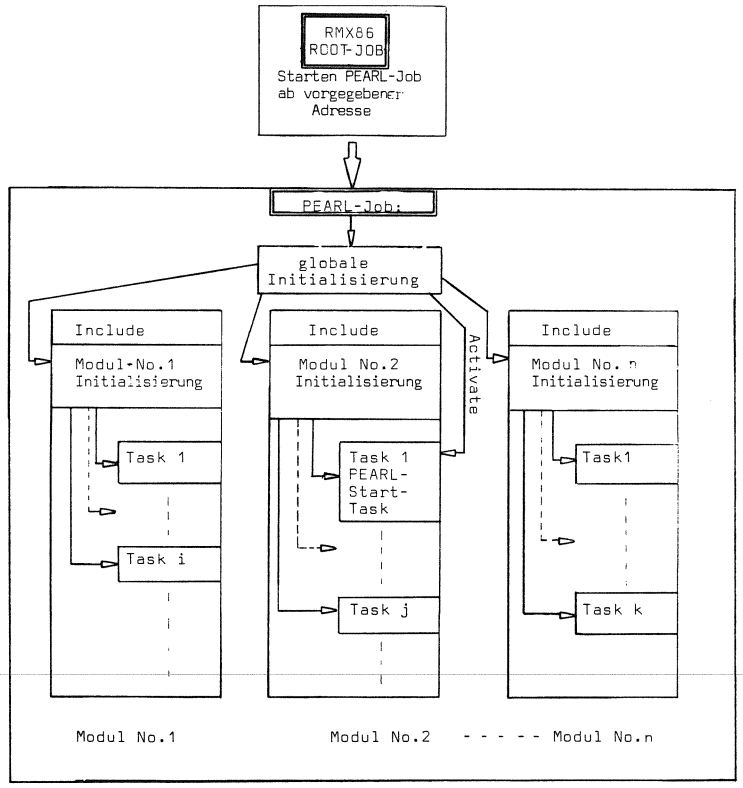


Abb. 3: Initialisierung des PEARL-Jobs

nach bisherigen Maßstäben hohen Bedarf an Speicherplatz erfordert (75 Kbyte), ist eine Eigenschaft des RMX86. Allerdings ist dieser hohe Bedarf an Speicherplatz bei einer Adressierbarkeit bis zu 1 Mbyte von geringerer Bedeutung. Dahingegen bleiben dem Benutzer sämtliche Funktionen erhalten, die durch die Struktur des RMX86 ermöglicht werden. So kann er z.B. für den Anschluß neuer Gerätetreiber voll die vom RMX86 dafür zur Verfügung gestellten Funktionen ausnutzen.

Bei der Implementierung der Laufzeitroutinen erwies sich als äußerst nützlich, daß standardisierte Aufrufkonventionen es ermöglichen Moduln verschiedener Sprachen problemlos zusammenzubinden. Dadurch konnten Funktionen entsprechend ihren Leistungsanforderungen in der geeignetsten Sprache implementiert werden z.B. wurden PEARL-Funktionen, die nicht unmittelbar auf RMX86-Funktionen abbildbar sind, in PASC86 portabel verwirklicht. An dieser Stelle sei zur Verwendung von PASCAL als System-Implementationssprache noch folgendes bemerkt:

PASCAL kristallisiert sich gegenwärtig nicht nur als eine der Standard-Sprachen für Mikrorechner heraus, sondern es erwies sich auch aufgrund seiner vorzüglichen Eigenschaften bzgl. Rekursivität als sehr geeignet für die Verwaltung eigener Warteschlangen, wie sie z.B. für die Implementierung von Zeiteinplanungen und BOLT-Variablen erforderlich waren. In der kombinierten Verwendung mit anderen Programmiersprachen, die für Hardwarenahe Programmierung wesentlich geeigneter sind, hat sich damit PASCAL auch für die Systemimplementierung als recht brauchbar erwiesen.

System-nahe Programmteile, wie z.B. die Abbildung der PEARL Ein-/Ausgabe auf RMX86 BASIC I/O Aufrufe wurden in der höheren INTEL-Systemimplementierungssprache PLM86 programmiert. Sie erlaubt für die betriebssystemabhängigen Anteile dennoch die Vorteile einer höheren Sprache ausnutzen zu können. Ansprüche auf Portabilität wurden bei diesen Betriebssystem-nahen Programm-Anteilen nicht gestellt, da bisher keine umfassenden standardisierten Schnittstellen für Tasking., realzeit- und Ein-/Ausgabe-System-Aufrufe existieren. Allerdings sind die UDI-Schnittstelle und das Konzept der Geräteunabhängigkeit ein erster Schritt in diese Richtung.

Für wenige zeitkritische Anteile wurde die ASM86-Assembler-Sprache benutzt.

Zusammengefaßt ist nach unserer Erfahrung die Forderung, problemlos Moduln verschiedener Sprachen zusammenbinden zu können, notwendig für eine leistungsfähige System-Implementierung mit vorgefertigten Programmteilen.

Als eine weitere Möglichkeit, portabel zu sein, wurde neben der Verwendung einer höheren Programmiersprache die Beschränkung auf die standardisierte INTEL-interne UDI-Schnittstelle angesehen. Sie wurde bei der Implementierung des PEARL-Compilers berücksichtigt. Dadurch ist dieser nicht nur unter der Kontrolle des ISIS-Betriebssystems ablauffähig, sondern auch unter allen künftigen Systemen, die der UDI-Schnittstelle genügen.

Die Berücksichtigung der UDI-Schnittstelle bei der Implementierung des PEARL-Laufzeit-Systems wurde jedoch nicht als vorteilhaft angesehen, da diese Schnittstelle erstens nur eine Untermenge der vom PEARL-Laufzeit-System benötigten und vom RMX86-Betriebssystem zur Verfügung gestellten Funktionen beinhaltet und zweitens eine weit umfangreichere Konfiguration des RMX86 erfordert.

Die Verwendung von PASCAL als Zwischensprache bei der PEARL-Übersetzung erwies sich als guter Kompromiß zwischen einer verlängerten Übersetzungszeit auf der einen und der Portabilität sowie der Sicherheit gegen Generierungs-Fehler auf der anderen Seite. So wurden die meisten Fehler, die zwangsläufig bei einer Installation eines neuen Compilers entstehen, bereits zur Übersetzungszeit durch den PASC86-Compiler abgefangen. Dieses Argument ist nach unserer Erfahrung für die Wirtschaftlichkeit der Implementierung von Bedeutung.

Zum PASC86-Compiler bleibt noch zu bemerken, daß bei der vorgestellten Implementierung nicht die offizielle Version verwendet wurde, sondern eine von der Fa. INTEL zur Ver-

fügung gestellte nicht-offizielle Version. Diese erlaubt auch, PASCAL-Programme zu bearbeiten, die sehr große Namenslisten erfordern; Beispiele dafür sind die einzelnen Pässe des PEARL-Compilers, die in PASC86-Code vorliegen.

8. Fazit

Zusammenfassend ist zu sagen, daß beim derzeitigen Stand der auf Mikrorechner existierenden Software, eine wirtschaftliche und dennoch leistungsfähige Implementierung von PEARL in wachsendem Maße möglich wird, durch Ausnutzung der Funktionen leistungsfähiger Realzeit-Betriebssysteme, sowie standardisierter Schnittstellen und Hochsprachen.

Literaturverzeichnis

[1] INTEL, PASCAL-86, USER's Guide Manual

[2] INTEL, Introduction to the RMX86 Operating System Manual

[3] W. Werum, H. Windauer: PEARL, Vieweg-Verlag 1978

[4] Intel, INTELLEC Series III, Microcomputer development console Operating Instructions Manual

[5] INTEL, Runtime Support Manual for iAPX86, 88 Applications

[6] INTEL, The 8086 Family Users Manual Numerics Supplement, July 1980

[7] INTEL, ISIS-II Users Guide Manual

Ein Mechanismus zur Kommunikation für verteilte Systeme in PEARL*)

A. Fleischmann, P. Holleczek, G. Klebes, R. Kummer

0. Einleitung

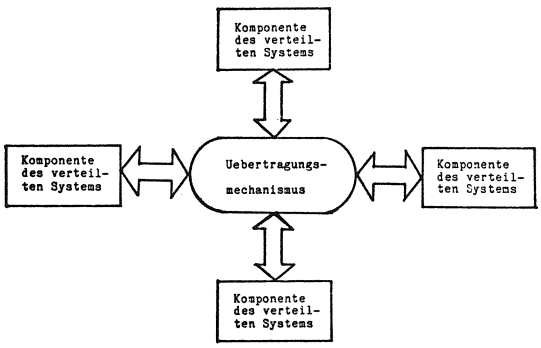
Die Programmierung von verteilten Systemen stellt besondere Anforderungen an die Programmiersprache. Hier soll ein Konzept vorgestellt werden, das sich mit der Kommunikation lose gekoppelter verteilter Systeme beschäftigt. Gemeinsame Objekte zur Kommunikation zwischen Prozessen, die sich auf verschiedenen Prozessoren befinden, werden nicht betrachtet, ebensowenig auch Aspekte, wie Sicherheit, Zuverlässigkeit, Wiederaufsetzverfahren etc.. Transportmechanismen, wie sie z.B. durch Netzwerke geliefert werden, werden vorausgesetzt.
Das hier vorgestellte Konzept basiert auf Botschaftsoperationen und nichtdeterministischen Kontrollstrukturen. Die genannten Operationen sind funktionale und formale Erweiterungen der Basissprache PEARL. Sie wurden in den Compiler und das Betriebssystem integriert.
Durch die Festlegung auf reine Botschaftskommunikation zwischen Prozessen auf verschiedenen Prozessoren und durch die vollständige Integration in Compiler und Betriebssystem unterscheiden sich die vorgeschlagenen Konstrukte von anderen Ansätzen /BONN81/,/AMMA81/.
Im folgenden wird zunächst der derzeitige Stand der Diskussion um Kommunikationsmechanismen skizziert, darauf die vorgeschlagenen Konstrukte erklärt und schließlich die Implementation vorgestellt.

*) gefördert von der DFG im Rahmen des Vorhabens Ho824

1. Kommunikation und Synchronisation in verteilten Anwenderprogrammen

1.1 Begriffsbestimmungen

Ein häufig benutztes Strukturierungskonzept für Programme zur Steuerung technischer Prozesse ist die Aufteilung in einzelne Rechenprozesse. Prozesse sind Berechnungen, die parallel zueinander ausgeführt werden können. Die einzelnen Prozesse können auf den Komponenten eines verteilten Systems laufen.
Ein verteiltes System besteht aus den Systemkomponenten und dem Übertragungsmechanismus /BOCH79/ (siehe B i l d 1).



B i l d 1 : Grundstruktur eines verteilten Systems

Systemkomponenten können Recheneinheiten, intelligente E/A-Geräte, usw. sein. Der Übertragungsmechanismus kann z.B. von einem lokalen Netz oder von einem öffentlichen Netz (z.B. Datex P) angeboten werden. Laufen die Rechenprozesse eines Programmes auf verschiedenen Komponenten eines verteilten Systems, spricht man von einem verteilten Programm.

1.2 Kommunikations- und Synchronisationskonzepte

Obwohl Rechenprozesse unabhängig voneinander ablaufen, müssen sie u.U. untereinander in Verbindung treten bzw. ihre "Arbeit" koordinieren. Außerdem müssen in der Prozeßautomatisierung die Rechenprozesse mit dem technischen Prozeß Steuerinformationen austauschen bzw. sich mit diesem Synchronisieren.
Um die Zusammenarbeit von Prozessen (Synchronisation und Kommunikation) zu organisieren, wurden verschiedene Konzepte vorgeschlagen. Dabei werden zwei Konzeptklassen unterschieden /STAU82/:

- Synchronisations- und Kommunikationskonzepte, die auf gemeinsamen Objekten aufbauen,
- Synchronisations- und Kommunikationskonzepte, die auf Botschaftsmechanismen basieren.

Diese beiden Konzeptklassen schlagen sich auf Spezifikation und Implementation nieder. In der Spezifikation wird ein Programm auf abstrakterer Ebene beschrieben, als dies in der programmiersprachlichen Formulierung der Implementation der Fall ist /KKST79/.
Ein Vergleich von Programmiersprachen mit verschiedenen Synchronisations- und Kommunikationskonzepten findet sich in /STOT82/.

1.2.1 Gemeinsame Objekte

Objekte können z.B. Variable, Dateien, Prozeduren usw. sein. Gemeinsame Objekte sind Objekte, die von mehreren Prozessen benutzt werden können. Eine Datei kann von mehreren Prozessen beschrieben oder gelesen werden. Eine Prozedur, die nicht wiedereintrittsfähig ist, wird von mehreren Prozessen u.U. gleichzeitig aufgerufen.
Um zu gewährleisten, daß der Zustand eines gemeinsamen Objektes konsistent ist, müßten die einzelnen Prozesse sich über dessen Verwendung "absprechen". Die Kommunikation von Prozessen wird als Spezialfall der Synchronisation betrachtet. Prozesse tauschen Daten über eine entsprechendes gemeinsames Objekt aus (i.A. ein Puffer) /KERA82/.
Spezifikationskonzepte für die Beschreibung von parallelen Programmen, die gemeinsame Objekte benutzen, sind z.B. die in /GOEH81/, /KERA79/ und /LAV78/ vorgestellten Modelle. Synchronisationskonzepte, die in Programmiersprachen verwendet werden und auf gemeinsamen Objekten beruhen, sind z.B. Semaphore, Monitore, kritische Abschnitte, Programmiersprachen, die solche Konzepte enthalten, sind z.B. PEARL, CONCURRENT PASCAL, MODULA.
Synchronisationskonzepte, die auf gemeinsamen Objekten basieren, gehen davon aus, daß die Rechenprozesse einen gemeinsamen Speicher benutzen. Sollen deshalb gemeinsame Objekte zur Programmierung verteilter Systeme verwendet werden, so gibt es vor allem folgende Probleme:

- die Implementierung von Synchronisationskonzepten basierend auf gemeinsamen Objekten wird ineffizient /LEVI81/, /KEME79/, und
- es gibt Probleme durch die Fehleranfälligkeit des Übertragungsmechanismus /BONN81/.

Außerdem ist es schwierig, die Kommunikation eines Programmes mit dem technischen Prozeß durch gemeinsame Objekte zu beschreiben. So werden z.B in PEARL zur Synchronisation der Rechenprozesse Semaphore benutzt, für die Kommunikation und Synchronisation mit dem technischen Prozeß indessen Ein/Ausgabeanweisungen und Anweisungen zur Einplanung von Interrupts. Diese Anweisungstypen haben aber große Ähnlichkeit mit Botschaftsoperationen.

1.2.2 Botschaftsmechanismen

Botschaftskonzepte wurden zur Programmierung von verteilten Systemen vorgeschlagen. Da verteilte Systeme in den letzten Jahren erheblich an Bedeutung gewonnen haben, wurden die meisten Botschaftskonzepte auch erst in jüngster Zeit veröffentlicht (/HOAR78/, /HANS78/,

/ADA79/, /BOOS81/, /FELD79/, /BOCH79/, /MAYE80/,/COOK80/).
Diese Konzepte unterscheiden sich vor allem in folgenden Punkten /GENT81/:
- Blockiert und Nichtblockiertbedingungen
- Adreß- und Absenderangaben
- Botschaftsformate
- Verhalten bei Übertragungsfehlern
- Abstraktionsgrad.

Programmiersprachen, in die Botschaftskonzepte eingebettet sind, sind z.B. ADA, *MOD.
Spezifikationssprachen für verteilte Programme werden in /BOCH79/ und /BRR081/ beschrieben.
Botschaftskonzepte sind zur Programmierung von Rechensystemen mit gemeinsamen Speichern nicht sehr effizient. Wird z.B. eine Datei von mehreren Prozessen beschrieben oder gelesen, so wird bei Programmiersprachen, die zur Synchronisation und Kommunikation nur Botschaftsmechanismen besitzen, ein Verwaltungsprozeß vorgeschaltet, der die Lese- und Schreibaufträge für die Benutzerprozesse ausführt und dann die Ergebnisse an diese zurückgibt.
Allerdings läßt sich die Zusammenarbeit technischer Prozeß/Rechenprozesse natürlicher beschreiben. Signale vom technischen Prozeß zu den Rechenprozessen bzw. umgekehrt lassen sich leicht als Botschaften interpretieren.

1.2.3 Nichtdeterministische Kontrollstrukturen

In Zusammenhang mit Botschaftskonzepten werden häufig die von Dijkstra /DIJK75/ eingeführten nichtdeterministischen Kontrollstrukturen verwendet /HOAR78/, /HANS78/, /ADA79/.
Mit dieser Anweisungsart wird es möglich, auf verschiedene Nachrichten von verschiedenen Prozessen gleichzeitig zu warten. In den Sprachen mit Botschaftskonzepten werden unterschiedliche nichtdeterministische Kontrollstrukturen verwendet. Dabei lassen sich vor allem folgende vier Typen unterscheiden:

- Guarded Regions
- Guarded Commands
- Guarded Loops
- Guarded Cycles

Nichtdeterministische Kontrollanweisungen bestehen aus mehreren sogenannten "Guards". Ein Guard besteht aus einer Bedingung und einer beliebigen Anweisungsfolge. Je nach Programmiersprache kann die Bedingung anders aufgebaut sein. In /ADA79/ und /HOAR78/ kann in der Bedingung auch eine erwartete Nachricht auftreten. Eine solche Nachrichtenbedingung ist 'wahr', wenn die erwartete Nachricht von dem entsprechenden Prozeß auch gesendet wird.
Je nachdem, welcher Typ einer nichtdeterministischen Kontrollanweisung nun vorliegt, wird bei der Abarbeitung einer solchen Anweisung unterschiedlich verfahren.
Ein Prozeß, der eine Guarded Region abarbeitet, prüft, ob eine Bedingung eines Guards wahr ist und führt dann die zugehörige Anweisungsfolge aus. Danach gilt die nichtdeterministische Kontrollanweisung als abgeschlossen. Ist keine der Bedingungen wahr, wird der Prozeß solange blockiert, bis eine Bedingung wahr wird. Sind mehrere Bedingungen wahr, wird ein beliebiges Guard ausgeführt.
Bei Guarded Commands wird genauso verfahren. Nur wird hier der Prozeß nicht blockiert, wenn keine der Bedingungen wahr ist. Das Guarded Command wirkt dann wie eine Leeranweisung.
Führt ein Prozeß eine Guarded Loop aus, wird jede Bedingung geprüft und falls sie wahr ist, die dazugehörige Befehlsfolge ausgeführt. Dies wird solange wiederholt, bis keine Bedingung mehr wahr ist. Erst dann gilt die Guarded Loop als beendet. Eine Guarded Loop, bei der keine Bedingung wahr ist, wirkt wie eine leere Anweisung.
Ein Guarded Cycle hat eine ähnliche Wirkung wie eine Guarded Loop. Der Unterschied besteht darin, daß der Guarded Cycle nicht beendet wird, wenn keine Bedingung mehr wahr ist, sondern der abarbeitende Prozeß blockiert wird, daß

bis wieder eine oder mehrere Bedingungen wahr werden. Ein Guarded Cycle wird nie beendet.

2. Erweiterung von (AVIONIC-) PEARL

Die folgenden Sprachkonstrukte sind echte Erweiterungen von AVIONIC-PEARL, d.h. die bereits vorhandenen Sprachelemente blieben unverändert.

2.1 Botschaften

2.1.1 Vereinbarung

In PEARL werden Prozesse durch Tasks dargestellt. Da es sich bei Botschaften um Mechanismen zur Prozeß- (Task-) Kommunikation handelt, scheint es sinnvoll, den Taskkopf um die Kommunikationsstruktur zu erweitern, die angibt, welche Tasks welche Botschaften an welche anderen Tasks schicken bzw. von diesen empfangen.
Die erste Zeile in Bild 2 stellt den ursprünglichen Taskkopf dar.

```
taskname": " "TASK" ["PRIO" prioritaet] ["GLOBAL"] ["RESIDENT"]
        ["TRANSMITS" sendebotschaftsname(n)
                     [parameterbeschreibung]
                   "TO" empfaengername(n)
                     {,sendebotschaftsname(n)
                      [parameterbeschreibung]
                   "TO" empfaengername(n)
                     }... ]
        ["RECEIVES" empfangsbotschaftsname(n)
                     [parameterbeschreibung]
                   "FROM" sendername(n)
                     {, empfangsbotschaftsname(n)
                      [parameterbeschreibung]
                   "FROM" sendername(n)
                     }... ]
                    ["NOBUFFER" /
                     "BUFFER" wartebereichsgroesse ] ";"
```

Bild 2
Halbformale Darstellung des erweiterten Taskkopfs

Mit der TRANSMITS-Klausel werden die Sendebotschaften vereinbart. Sie besteht aus dem Botschaftsnamen bzw. einer geklammerten Namenliste, gefolgt von einer optionalen Parameterliste. Diese setzt sich zusammen aus dem Parameternamen (bzw. einer geklammerten Namenliste), dem der Parametertyp (ein elementarer PEARL-Datentyp) folgt. Hierauf folgt das Schlüsselwort "TO" und der bzw. die Empfängernamen.
Die RECEIVES-Klausel ist analog der TRANSMITS-Klausel aufgebaut. In ihr werden die Empfangsnachrichten vereinbart. An Stelle des Schlüsselworts "TO" steht das Schlüsselwort "FROM". Anschließend folgt die Wartebereichsvereinbarung (für die Empfangsnachrichten). Bei Angabe von NOBUFFER existiert kein Wartebereich Bei Angabe von BUFFER existiert ein Wartebereich endlicher Größe. Die Größenangabe legt die Zahl der Botschaften fest, die im Wartebereich Platz finden.
Die Bedeutung von Wartebereichen ist unter 2.1.2 beschrieben.
Das zugehörige Anschlußschema wird im Systemteil beschrieben. Es gibt an welche Tasks über welche Verbindungen kommunizieren.

2.1.2 Verwendung

Die Darstellung der Botschaftsoperationen ähnelt den PEARL-E/A-Anweisungen:

Sendeanweisung:

```
"TRANSMIT" "FROM" botschaftsname
"TO" empfängername ";"
```

Mit der TRANSMIT-Anweisung wird diejenige Botschaft, die durch den angegebenen Namen identifiziert wird, mit ihren Parametern gesendet. Letztere haben vorher ihren Wert per Zuweisung erhalten.
Ist der Empfänger nicht empfangsbereit, wird der Sender blockiert, falls der Empfänger nicht über einen Wartebereich verfügt. Ist der Wartebereich des Empfängers voll, wird der Sender ebenfalls blockiert.

Empfangsanweisung:

```
"RECEIVE" "FROM" sendername "TO"
botschaftsname ";"
```

Mit der RECEIVE-Anweisung werden Botschaften aus dem Wartebereich bzw., falls kein Wartebereich vorhanden ist, von anderen Tasks anstehende Botschaften übernommen, d.h. ihre Parameterwerte werden in die Parameter der Empfangsbotschaft kopiert.

Befindet sich die Botschaft nicht im Wartebereich bzw. ist der Sender nicht sendebereit, so wird der Empfänger blokkiert.

2.2 Guarded Regions/Commands

2.2.1 Aufbau und Bedeutung

Der Aufbau eines Guarded Statement ist B i l d 3 zu entnehmen.

```
guarded-statement: "GUARDED"  region / command
                   "GUARDEND" ";"

region: "REGION"
        guards
        ["TIMEOUT" "AFTER" duration-ausdruck
                   "REACT"  anweisungsfolge ]

command: "COMMAND"
         guards
         ["OUTREACT" anweisungsfolge]

guards: "GUARD" guard-element; {guard-element; } ...
              "REACT" aneisungsfolge
        {"GUARD" guar-delement; {guard-element} ...
                 "REACT"  anweisungsfolge } ...

guard-element:  "WHEN" interrupt /
                 request-anweisung /
                 get-anweisung /
                 receive-anweisung /
                 release-anweisung /
                 put-anweisung /
                 transmit-anweisung
```

Bild 3
Halbformale Darstellung der nichtdeterministischen Kontrollanweisungen

Die Guarded Statements werden von oben nach unten, die Guards von links nach rechts abgearbeitet. Falls innerhalb eines Guards alle Anweisungen ausführbar sind, wird der zugehörige Reaktionszweig abgearbeitet. Ist keines der Guards ausführbar, wird bei einem Guarded Command die Anweisungsfolge nach OUTREACT abgearbeitet; fehlt dieser Zweig, wirkt das Guarded Command wie eine Leeranweisung. Ist in einer Guarded Region keines der Guards ausführbar, wird gewartet, bis entweder eines von ihnen ausführbar wird oder die nach TIMEOUT angegebene Zeitdauer verstrichen ist. Danach wird der zugehörige Reaktionszweig ausgeführt. Fehlt der TIMEOUT-Zweig, wird gewartet, bis eines der Guards ausführbar wird.
Guarded Loops und Guarded Cycles wurden zunächst nicht implementiert.

2.2.2 Verwendung

Einige Beispiele:

- alternatives Warten auf verschiedene Interrupts (B i l d 4):

```
MODULE;
   :
PROBLEM;
SPC (POSI, ENDE) INTERRUPT;
   :
SCHRITTMOTORSTEUERUNG: TASK GLOBAL;
   :
        GUARDED REGION
          GUARD WHEN POSI; REACT /* REGULAERE BEARBEITUNG
                                    FORTSETZEN            */
          GUARD WHEN ENDE; REACT /* FEHLERBEHANDLUNG      */
          TIMEOUT AFTER 1 MIN;
                           REACT /* SCHRITTMOTOR KAPUTT   */
        GUARDEND;
   :
END;
MODEND;
```

Bild 4
Alternatives Warten auf mehrere Interrupts

Ein Prozeß zur Steuerung eines Schrittmotors soll alternativ auf zwei Interrupts warten:

a) die gewünschte Position wird erreicht; dies wird durch einen Interrupt (hier: POSI) angezeigt.
b) einer der beiden Endkontakte wird erreicht; die Rückmeldung hierfür erfolgt durch einen anderen Interrupt (hier: ENDE).

- Zeitüberwachung von Semaphoroperationen (B i l d 5)
  Ein Prozeß soll an einem Semaphor nur eine maximale Zeitdauer blockiert werden können.

```
MODULE;
   :
PROBLEM;
SPC MUTEX SEMA;
   :
ZUGRIFF: TASK GLOBAL;
   :
        GUARDED REGION
          GUARD REQUEST MUTEX; REACT /* ZUGRIFFSAKTION */
          TIMEOUT AFTER 5 SEC; REACT /* AUSWEICHAKTION */
        GUARDEND;
   :
END;
MODEND;
```

Bild 5
Zeitüberwachen von Semaphoreoperationen

- Zeitüberwachung von Eingaben
  Analog der Zeitüberwachung von Semaphoren kann das Warten bei Eingabeoperationen ebenfalls zeitlich begrenzt werden.

- gleichzeitiges Empfangen mehrerer Botschaften (B i l d 6)

```
MODULE;
SYSTEM;
AUSWAHL:<-ASNR*1; /* BOTSCHAFTEN AN AUSWAHL
   :                  GEHEN UEBER LEITUNG 1  */

PROBLEM;
SPC AUSWAHL TASK;
      :
MESS1: TASK GLOBAL
       TRANSMITS MESSW1 (DRUCK FLOAT) TO AUSWAHL;
          :
       TRANSMIT FROM MESSW1 TO AUSWAHL;
          :
END;

/* MESS2 UND MESS3 SIND ANALOG AUFGEBAUT UND BEFINDEN SICH IN
   ANDEREN MODULN AUF ANDEREN PROZESSOREN                    */

MODEND;


MODULE;
SYSTEM;
MESS1:->ASNR*1;
MESS2:->ASNR*2;
MESS3:->ASNR*3;
   :
PROBLEM;
SPC (MESS1,MESS2,MESS3) TASK;
      :
AUSWAHL: TASK GLOBAL
         RECEIVES MESSW1 (DRUCK FLOAT) FROM MESS1,
                  MESSW2 (DRUCK FLOAT) FROM MESS2,
                  MESSW3 (DRUCK FLOAT) FROM MESS3
         NOBUFFER;
            :
         GUARDED REGION
           GUARD RECEIVE FROM MESS1 TO MESSW1;
                 RECEIVE FROM MESS2 TO MESSW2;
                 RECEIVE FROM MESS3 TO MESSW3;
             REACT /* VERGLEICHEN UND AUSWERTEN */

           TIMEOUT AFTER 0.5 SEC;
             REACT /* FEHLERBEHANDLUNG */

         GUARDEND;
            :
END;
MODEND;
```

Bild 6
Gleichzeitiges Warten auf Nachrichten

Drei Prozesse messen unabhängig voneinander die gleiche Prozeßgröße (z.B. Druck, Temperatur, usw.). Ein vierter Prozeß vergleicht die Meßwerte und nimmt denjenigen als richtig an, der (mit gewissen Toleranzen) von mindestens zwei Meßtasks geliefert wurde. Bei Ausfall eines Meßprozesses sollen entsprechende Fehlerbehandlungen eingeleitet werden.
Die Meßtasks sollen erst fortgesetzt werden, wenn alle Meßwerte bei der Auswahltask eingetroffen sind.

3. Implementierung

3.1 Compiler und Codegenerator

Der maschinenunabhängige obere Compiler wurde um die oben genannten Sprachkonstrukte erweitert. Hierfür mußten neue Anweisungen für die Zwischensprache (CIMIC/AV) geschaffen werden. Ausserdem wurde der Codegenerator den neuen Anforderungen angepaßt.

3.2 Betriebssystem

Ein bereits vorhandenes, eigens für das gesamte PEARL-System entwickeltes, portables Betriebssystem mußte den neuen Anforderungen angepaßt werden. Die Änderungen erforderten u.a. die Einführung von sog. "Blockiertbereichen" für jede Empfängertask. Diese dienen zur Speicherung von Sendertask- und Botschaftsnamen, die entweder im Wartebereich keinen Platz fanden oder, falls kein Wartebereich vorhanden ist, die Empfängertask nicht empfangsbereit vorfanden.

4.3 Implementationseinschränkungen

Da die Überlegungen über die Verklemmungsfreiheit der entworfenen Guarded Statements noch nicht abgeschlossen sind (eine kurze Schilderung der Problematik ist zu finden in /MAYE79/), scheint es ratsam, die Mischung von Eingabe- (GET-, RECEIVE-, REQUEST-, WHEN-) und Ausgabe- (PUT-, TRANSMIT-, RELEASE-) Anweisungen (vorerst) nicht zuzulassen.

5. Abschlußbemerkungen

Absicht des voliegenden Artikels ist, einige Sprachkonstrukte zur Programmierung von verteilten Systemen vorzustellen und ihre Implementierbarkeit zu demonstrieren. Nichtsdestoweniger liegen bisher nur wenig praktische Erfahrungen vor. Es sollte deswegen daraufhingewiesen werden, daß einerseits die vorgeschlagene Syntax nur ein erster Ansatz ist, und daß andererseits eine gründliche Diskussion der Semantik der Konstrukte erfolgen sollte.

Literatur

/ADA79/ Ichbiah et. al.
Rationale for design of ADA programming language.
SIGPLAN Not. (ACM) Juni 1979

/AMMA81/ M. Ammann
PEARL für verteilte Systeme
Prozeßrechnertagung 1981
Informatik Fachberichte (39)
Springer Verlag

/BOCH79/ G. Bochmann
Distributed systems
Lecture Notes of Com. Science
Springer Verlag 1979

/BONN81/ G. Bonn, L. Lorenz
Eignung von Mehrrechner-PEARL zur Programmierung paralleler Prozesse, Erfahrung und Folgerungen
Prozeßrechnertagung 1981
Informatik Fachberichte (39)
Springer Verlag 1981

/BOOS81/ J. Boos, R. Plasmeijer, J. Stoet
Process communication based on input specifications
ACM TOPLAS juli 1981

/BRRO81/ P. Brück, J. Robra
Verfahren und Hilfsmittel für die Softwareentwicklung von Nebenstellenanlagen
TE-KA-DE Technische Mitteilungen 1981

/COOK80/ R. B. Cook
*MOD A language for distributed programming
IEEE Trans. Softw. Eng.
November 1980

/DIJK75/ E. W. Dijkstra
Guarded commands, nondeterminacy and derivation of programs
Comm. ACM August 1975

/FELD79/ J. Feldman
High level programming for distributed computing
Comm. ACM Juni 1979

/GENT81/ W. M. Gentlemen
Message passing between sequential processes: the reply primitive and the administrator concept

/GOEH81/ P. Göhner
Spezifikation der Synchronisierung paralleler Rechenprozesse in EPOS
Prozessorrechnertagung 1981
Informatik Fachberichte
Springer 1981

/HANS78/ P. B. Hansen
Distributed processes: A concurrent programming concept
Comm. ACM November 1978

/HOAR78/ C. A. R. Hoare
Communicating sequentiell Processes
Comm. ACM August 1978

/KEME79/ H. Kemen
Zur Programmierung verteilter Systeme
Workshop der GI
Informatik Fachberichte (22)
Springer 1979

/KERA82/ S. Keramidis
Eine Methode zur Spezifikation und korrekten Implementierung von asynchronen Systemen
Habilitationsschrift, IMMD Bericht der Univ. Erlangen 1982

/KKST79/ Kimm et. al.
Einführung in Software Engeneering
de Gruyter Berlin 1979

/LAVE79/ M. S. Laventhal
A constructive approach to reliable synchronisation code
Proc. 4th intern. conf. of software engeneering 1979

/LEVI81/ P. Levi
Betriebssysteme für Realzeitanwendungen
Data Context Verlag Köln 1981

/MAYE80/ T. W. Mao, R. T. Yeh
Communication port: A language concept for concurrent programming
IEEE Tran. Softw. Eng. März 80

/STOT82/ P. D. Stotts
A comparative survey of concurrent programming languages
SIGPLAN (ACM) Oktober 1982

Ein PEARL-Testsystem zum Einsatz in verteilten Systemen*)

C. Andres, A. Fleischmann, P. Holleczek, W. Mühlbauer

0. Einleitung

Die Akzeptanz von höheren Realzeitsprachen in der Prozeßsteuerung hängt wesentlich von den Testmöglichkeiten ab. Dieser Tatsache wurde auch bei PEARL bereits vielfach Rechnung getragen /GMH079/. Der Einsatz von PEARL in verteilten Systemen und der Programmtest auf Mikroprozessoren ohne eigene Bedienperipherie bedarf neuer Ansätze. Im vorliegenden Artikel soll ein PEARL-Testsystem vorgestellt werden, das den Test von "physikalisch" verteilten Programmen erlaubt. Es berücksichtigt darüber hinaus Botschaftsmechanismen, wie sie zur Programmierung verteilter Systeme vorgeschlagen wurden /FLH082/.

1. Eigenschaften des Testsystems

Bei Crosscompilern, wie sie oft für Mikroprozessoren verwendet werden, scheidet eine Fehlerlokalisierung durch Programmänderungen aufgrund der langen Programmerstellungszeiten praktisch aus. Deshalb wurde ein sprachorientiertes Testsystem entwickelt, das keine Neuübersetzung von Programmen verlangt,so daß wertvolle Zeit eingespart wird.
Man bediente sich dabei der dynamischen Instrumentierung /GMH079/, bei der ggf. (d.h. bei Traces und Haltepunkten) im bereits geladenen Programm der ursprüngliche Code durch Test-Code ersetzt wird. Das bedingt, daß zur Laufzeit noch verschiedene vom Übersetzer erzeugte Listen vorhanden sein müssen.

Das Testsystem besitzt einen modularen Aufbau und ist sowohl für Ein- wie Mehrprozessor-Konfigurationen einsetzbar.

2. Leistungsumfang des Testsystems

Neben den herkömmlichen Auskünften über Prozesse und Variable liefert das Testsystem Informationen über Betriebssystem-Warteschlangen sowie über den Inhalt der Botschaftspuffer.
An Manipulationen sind die üblichen Operationen auf PEARL-Datentypen sowie das Erzeugen von Botschaften erlaubt.

Das Testsystem liefert folgende Informationen:

(1) Auskunft über den Status aller Prozesse.
Ausgabe: -Taskpriorität
-Taskzustand

(2) Auskunft über Variable und Arrays.
Ausgabe: -Speicherinhalt

(3) Auskunft über Semaphore.
Ausgabe: -aktueller Wert
-Semaphore-Warteschlange
(ausgegeben werden die an dem Semaphore blockierten Tasks)

(4) Auskunft über den Inhalt der Botschaftspuffer.
Ausgabe: -Inhalt des Warte- und des Blockiertbereichs der Task
-Anzahl der Botschaften im Wartebereich der Task

(5) Auskunft über Geräte

*) gefördert von der DFG im Rahmen des Vorhabens Ho824

Ausgabe: -belegende Task
-Geräte-Warteschlange (ausgegeben werden die auf die Zuteilung wartenden Tasks)

(6) Auskunft über Interrupts
Ausgabe: -Interrupt-Warteschlange (ausgegeben werden diejenigen Tasks, die auf das Eintreffen des Interrupts warten)

(7) Auskunft über die Prozessor-Warteschlangen
Ausgabe: -Prozessor-Warteschlange (ausgegeben werden die Tasks, die auf die Zuteilung des Prozessors warten)

Dabei ist zu berücksichtigen, daß die Auskünfte überholt sein können, wenn die Prozesse nicht angehalten werden.

Das Testsystem läßt folgende Manipulationen zu:

(1) Zustandsänderungen von einzelnen Prozessen
- ACTIVATE
- CONTINUE
- PREVENT
- SUSPEND
- TERMINATE

(2) Anhalten und Fortsetzen einzelner Prozesse
- Haltepunkt setzen
- Haltepunkt löschen
- Fortsetzen des angehaltenen Prozesses

(3) Anhalten und Fortsetzen des Gesamtsystems

(4) Erzeugen von Botschaften

(5) Ändern von Semaphore-Werten

(6) Ändern der Werte von Variablen und Arrays

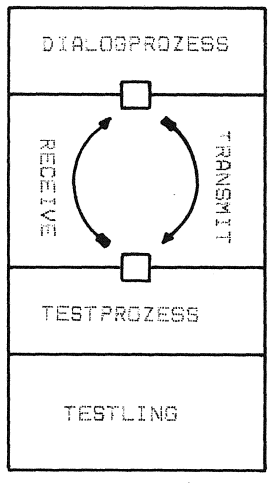
(7) Ein- und Ausschalten von Trace
- Setzen und Löschen von Traceberei-
chen

(8) Trigger-Interrupt

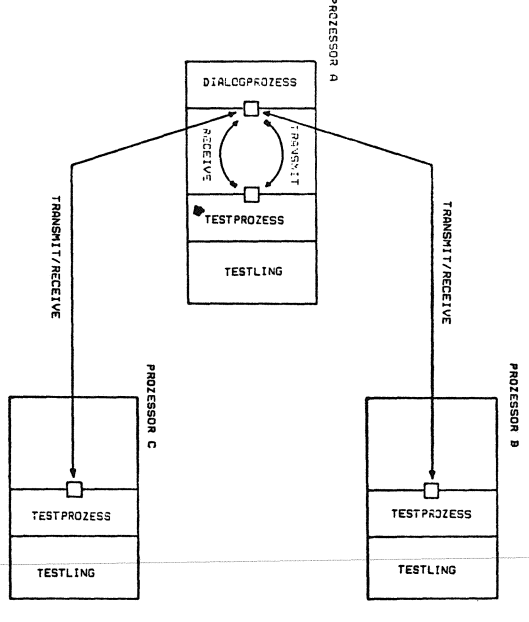
Dabei ist bei manchen Manipulationen zu beachten, daß sie zwar gefährliche Seiteneffekte haben können, aber durch Überwinden von Stillständen wertvolle Testzeit sparen helfen.

3. Aufbau des Testsystems

Die Funktionsweise des Testsystems wurde so spezifiziert, daß es sowohl auf Ein- wie Mehr-Rechnerkonfigurationen läuft (B i l d 1, B i l d 2).



B i l d 1: Testsystem in Monoprozessorkonfiguration



B i l d 2: Testsystem in Mehrprozessorkonfiguration

Da es ein Testsystem für Anwender-Programme darstellt, kann es die Funktionsfähigkeit des Betriebssystems voraussetzen und Funktionen des Betriebssystems mitbenutzen (soweit Seiteneffekte ausgeschlossen sind).
Das Testsystem wird immer in Form von zwei miteinander kommunizierenden Prozessen, einem Dialog-Prozeß und einem Test-Prozeß realisiert.
Der Dialog-Prozeß läuft immer auf dem zentralen Prozessor (Mittelpunkt eines logischen Sterns) des verteilten Systems, der mit einem Bedienterminal ausgerüstet ist.
Der Test-Prozeß läuft stets auf dem Prozessor, der das zu testende Programm beherbergt.
Diese Konfiguration unterstützt Mikroprozessorsysteme, deren Frontend-Prozessoren üblicherweise über keine Bedienperipherie verfügen.
Der Dialog-Prozeß, bestehend aus Kommandointerpreter und Protokollaufbereitung, interpretiert die vom Anwender abgegebenen Testaufträge (Kommandointerpreter) und komprimiert sie in Form von Botschaften, die dem Test-Prozeß übergeben werden. Der Test-Prozeß führt die Testaufträge durch und gibt die Ergebnisse oder Bestätigungen in Form von Botschaften an den Dialog-Prozeß zurück. Der Dialog-Prozeß entkomprimiert die Ergebnisse (Protokollaufbereitung) und protokolliert sie auf dem Bedien-Terminal.

4. Implementation

Wie schon in Kapitel 2 erwähnt, wird der Programmcode des Testlings nicht expandiert (statische Instrumentierung), sondern zur Laufzeit dynamisch ersetzt (dynamische Instrumentierung).

Vorteile der dynamischen Instrumentierung:

(1) Zum Testen ist kein eigener Übersetzungslauf notwendig, was vor allem bei Cross-Übersetzungssystemen viel Zeit spart und sehr zur Benutzerfreundlichkeit des Testsystems beiträgt.

(2) Das Echtzeitverhalten des Testlings zwischen Haltepunkten wird nicht beeinflußt, weil in diesen Bereichen der ursprüngliche Benutzercode - durchlaufen wird.

(3) Die vom Testsystem benötigten Informationen (in Form von Listen) können auf Externspeicher ausgelagert, und bei Bedarf als Teillisten wieder eingelagert werden.
Dies ist besonders wertvoll bei Mikroprozessorsystemen mit eingeschränktem Arbeitsspeicherausbau. Statische Expansion würde an dieser Stelle ein Overlay-Binden erforderlich machen.

Einen Nachteil hat diese Methode:

Derjenige Teil des Testsystems, der den Codeaustausch an den Testpunkten durchführt, ist aufgrund der unterschiedlichen Befehlsformate stark rechnerabhängig. Deshalb muß dieser Teil bei einer Übertragung auf andere Rechner neu codiert werden (allerdings nicht neu spezifiziert).

Im folgenden sollen die vom Testsystem zur Laufzeit benötigten Listen beschrieben werden.

(1) Modulliste
Zur Realisierung von Haltepunkten und Traces muß es möglich sein
- bei gegebenem Modulnamen und gegebener Zeilennummer die zugehörige Adresse und
- bei gegebener Adresse die zugehörige Zeilennummer und den Modulnamen zu ermitteln.

(2) Taskliste
Mit Hilfe der Taskliste ist es möglich
- bei gegebenem Tasknamen die zugehörige Kontrollblockadresse und den Prozessor, auf dem sich diese Task befindet
- bei gegebener Kontrollblockadresse den zugehörigen Tasknamen zu ermitteln.

(3) Semaphoreliste

(4) Geräteliste
Die Semaphore- und Geräteliste sind analog aufgebaut, so daß man bei gegebenem Semaphore- bzw. Gerätenamen die zugehörige Kontrollblockadresse ermitteln kann.

(5) Variablenliste
Die Variablenliste enthält neben den Variablennamen und den zugehörigen Adressen noch Angaben über den Datentyp und die Datentyplänge, um eine typgerechte Ein- bzw. Ausgabe zu ermöglichen.

(6) Arrayliste
Die Arrayliste besitzt denselben Aufbau wie die Variablenliste, erlaubt aber zusätzlich noch eine Überprüfung von Indexgrenzen.

Der Dialogteil des Testsystems wurde in PEARL geschrieben, um einen gewissen Grad an Portabilität zu erreichen.
Die Implementation der Testfunktionen erfolgte weitestgehend im gleichen abstrakten Assembler ("SPASS") wie die des PEARL-Betriebssystems, so daß mit Installation des Betriebssystems auch die Testfunktionen auf einem anderen Zielrechner vorhanden sind.

Einige wenige Testfunktionen und die Listenverwaltung (für die oben beschriebenen Listen) wurden aus Effizienzgründen in Assembler programmiert.

5. Zusammenfassung

Entsprechend den geringen Erfahrungen im Umgang mit verteilten Systemen bilden die entwickelten Testfunktionen lediglich eine Basis für eine Weiterentwicklung des Testsystems.
Der modulare Aufbau des Testsystems ermöglicht es, nachträglich Funktionen hinzuzufügen, die sich im praktischen Einsatz als notwendig erweisen, bzw. überflüssige oder "gefährliche" Funktionen wegzulassen.

Literaturhinweise

/GMH079/ L. Gmeiner, G. Hommel (Hrsg.): Testen und Verifizieren von Prozeßrechner-Software, KfK-PDV 179, Dez.1979, KfK, Karlsruhe

/FLH082/ A. Fleischmann, P. Holleczek, G. Klebes, R. Kummer: Ein Mechanismus zur Kommunikation für verteilte Systeme in PEARL, in diesem Heft

Bemerkungen zur Implementation von PEARL auf Mikrorechner

Prof. Dr. K. W. Pleßmann, Aachen

Zusammenfassung:
Durch die Typenvielfalt der heute angebotenen Mikrorechner und den daraus folgenden Konsequenzen für Übersetzungssysteme sollte die Portierbarkeit des PEARL-Compilers mit besonderem Interesse verfolgt werden. Hierbei geht es also darum, das Übersetzungssystem für unterschiedliche Prozessoren oder Mikrorechner mit gleichem Prozessorkern bereitzustellen.

Schlüsselworte: PEARL-Compiler, Mikrorechner

Abstract
The portability of the PEARL compiler is of special interest because of the great number of different microcomputer on the market. This problem has to do with the implementation for microcomputer with the same or different processors as processing unit.

Keywords: PEARL compiler, microcomputer

Die Diskussion um die Vorzüge der Anwendung von Mikrorechnern in Automatisierungssystemen sind in der Vergangenheit zunächst ausschließlich unter dem Gesichtspunkt der Rechnerleistung geführt worden. Hierbei trat zu Beginn das Problem der Herstellung der Anwenderprogramme in den Hintergrund, da zur Hauptsache auf der maschinennahen Ebene programmiert worden ist. Erst durch die Bereitstellung höherer Verarbeitungsleistungen insbesondere durch den Übergang zu 16 bit-Prozessoren - und erst recht beim Übergang zu 32 bit-Systemen - hat die Diskussion der Verwendung von Hochsprachen belebt.

Durch systemspezifische oder auch allgemeine Betriebssysteme, insbesondere für Echtzeitanwendungen hat sich damit eine Situation ergeben, die Gedanken zur Implementation von PEARL auf Mikrorechner notwendig macht. Hierbei ergeben sich neben allgemeinen Fragestellungen, die aus dem Sprachzusammenhang hervorgehen, noch folgende Gesichtspunkte:

1. Portabilität des Compilers auf unterschiedliche Systeme
2. Anpassung der Schnittstelle, Objektmoduln-Betriebssystem.

Während aus der gegenwärtigen Sicht die Typenvielfalt im Bereich der Prozessoren kleiner wird, werden immer mehr konfektionierbare Mikrorechnersysteme auf den Markt kommen. Dies wird, um eine langfristige Anwendung von PEARL sicherzustellen, auch eine größere Anzahl von Compilern möglicherweise mit nur einer geringen Anzahl von Prozessorkernen zur Folge haben müssen. Diesen Fragestellungen soll im weiteren nachgegangen werden.

1. Portierbarkeit des Compilers

Bei der Diskussion über die Portierbarkeit von Software ist in der Vergangenheit zumeist der Anwenderbereich und nicht zu sehr das Übersetzungssystem betrachtet worden. Es wurde davon ausgegangen, daß durch die Hersteller entsprechende Programmiersysteme bereitgestellt werden, so daß sich zwangsläufig die Frage auf den Anwenderbereich konzentrieren mußte. Durch die Typenvielfalt im Bereich der Mikroprozessoren und insbesondere der hieraus ableitbaren Rechnersysteme ergeben sich jedoch weitere Probleme. Gerade weil eine größere Anzahl unterschiedlicher aber konfektionierbarer Mikrorechner, möglicherweise mit dem gleichen Prozessorkern, auf den Markt kommen, muß die Bereitstellung des Programmiersystems mit in den Bereich der Übertragbarkeit von Software aufgenommen werden.

Wegen der hohen Kosten bei der Herstellung eines Compilers und aller hiermit zusammenhängenden Bereiche bis hin zum Laufzeitpaket kann also nicht erwartet werden, daß der Mikrorechnerhersteller, der, wie die Vergangenheit zeigt, zumeist ausschließlich die Hardware unterstützt, das Programmiersystem bereitstellt. Damit sind die Anwender auf jene "Hilfsmittel" angewiesen, die ihnen, von wem auch immer, zur Verfügung stehen. Ein Teil des Erfolgs des CP/M-Systems beruht nämlich darin, daß hierfür eine größere Anzahl von Übersetzungshilfen zur Verfügung stehen.

Wegen der besonderen Schwierigkeiten von PEARL empfiehlt es sich also eine portierbare Fassung eines Compilers herzustellen. Dieser sollte nach Möglichkeit in einer Sprache geschrieben werden, die auf anderen Systemen zur Verfügung steht. Beim gegenwärtigen Stand bietet sich eigentlich nur PASCAL [GRO-82] an.

Welche sind nun die Vorteile eines solchen Systems:

1. Auf der Basis dieser Grundidee läßt sich ein PEARL-Compiler herstellen, der auf den meisten Entwicklungssystemen eingerichtet werden kann.
2. Die Herstellung von Cross-Systemen ist ebenfalls möglich.
3. Durch Bereitstellung einer Compiler-Version würde sich eine weitgehende Normierung und damit Portierbarkeit von Anwendersystemen ergeben bzw. gewährleisten lassen.

Aus diesen Punkten ergeben sich gleich zwei Folgerungen. Diese sind:

1. Der Übergang auf einen neuen Prozessor hat die Modifikation des Generatorteils zur Folge.
2. Die Einrichtung des Compilers auf einem Mikrorechner mit einem Prozessorkern, für den es bereits einen Generator gibt, hat ausschließlich Änderungen an der Schnittstelle zum Betriebssystem zur Folge.

Unter diesen Voraussetzungen kann erwartet werden, daß eine größere Anzahl von Anwendern sich des Übersetzungssystems PEARL bedienen werden. Für sie würde der Übergang von einem zu einem anderen Rechner nicht mehr die Konsequenzen haben, die heute im allgemeinen zu erwarten sind. Hierzu gehören die Kompatibilität auf der sprachlichen Ebene, die Portierbarkeit von Anwenderprogrammen und die Benutzung ein und desselben Entwicklungssystems - u.U. als Gastmaschine - sogar für unterschiedliche Prozessortypen.

2. Betriebssystem-Interface

Von Bedeutung für die Nutzbarkeit im Zusammenhang mit unterschiedlichen Betriebssystemen ist die Schnittstelle zwischen dem lauffähigen Objekt und dem für eine spezielle Implementation vorgesehenen Betriebssystem. Diese Problematik ergibt sich zwangsläufig dadurch, daß selbst bei gleichem Prozessorkern einige Mikrorechner-Hersteller dazu übergehen, eigene Betriebssysteme auf den Markt zu bringen oder ausgehend von generellen Systemen, wie z.B. RMX, Varianten abzuleiten. Es kann nicht erwartet werden, daß in Zukunft eine andere Strategie verfolgt wird. Es ist daher bei Einrichtung des Compilers über ein geeignetes Interface die Schnittstelle zu dem zu verwendenen Betriebssystem einzurichten.

Eine vereinfachte Darstellung dieser Gedanken zeigt Bild 1. Auf der ersten Ebene befindet

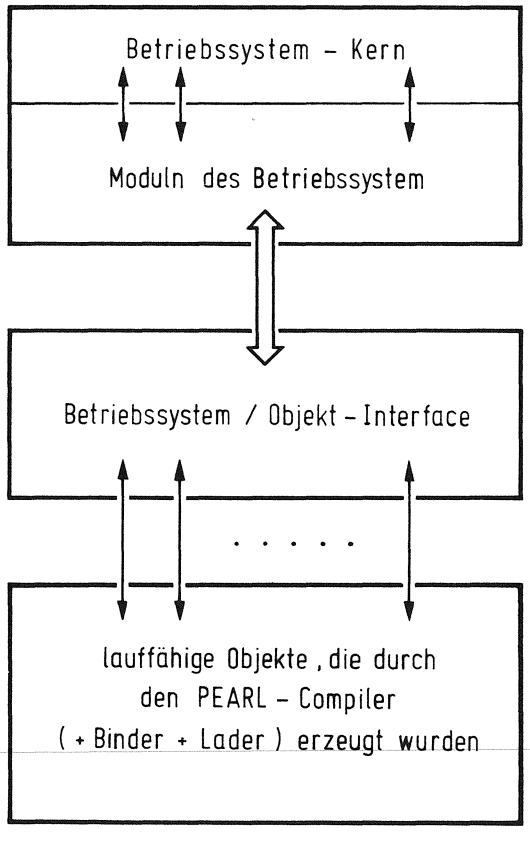


Bild 1. Interface-Modell

sich das benutzte Betriebssystem mit seinen nach außen zur Verfügung gestellten Schnittstellen für den Benutzer. U.U. sind einige Moduln zu ergänzen, um die Effektivität beim Zugriff auf die Dienste innerhalb des Betriebssystems zu vergrößern. Bei den heute auf dem Markt angebotenen Systeme ist jedoch auch ohne Änderung der Anschluß über die Außenbezüge möglich.

Auf der zweiten Ebene, hier Betriebssystem/ Objekt-Interface genannt, werden die "Transportmittel" für sämtliche Aktionen zwischen dem Anwendersystem und dem Betriebssystem bereitgestellt.

Da nach bestimmten Regeln die Erweiterbarkeit vor allem bei modular aufgebauten Systemen sichergestellt werden kann, ergibt sich auch gleichzeitig die Möglichkeit, jene Hardware-Elemente mit in den Gesamtkontext einzubeziehen, die durch den Anwender spezifiziert worden sind und die nicht ursprünglich zum Mikrorechnersystem gehören.

In diesem Zusammenhang bietet sich die Normierung der Systemkonnektoren nahezu zwangsläufig an. Dies würde insbesondere auf der Systemebene in vieler Hinsicht Erleichterungen für die Anwender mit sich bringen.

Zusammenfassung:

In den beiden vorhergehenden Punkten ist versucht worden, einen Motivationskatalog für ein PEARL-System zu entwerfen, welches übertragbare Compiler für unterschiedliche Mikrorechnersysteme zur Folge haben kann. Der Grundgedanke basiert auf der Überlegung, ein möglichst durchgängiges und durchgehendes Programmiersystem PEARL aufzubauen und für unterschiedliche Rechnerkonfigurationen und darüber hinaus ebenso Prozessoren bereitstellen zu können. Daß in diesem Zusammenhang eine Reihe von weiteren Festlegungen möglich sind, ist offensichtlich.

Dieser Zusammenhang muß in der Konkurrenzsituation anderer Programmiersprachen aber vor allem zu ADA gesehen werden. Die Komponentenhersteller, allen voran ZILOG und MOTOROLA, unternehmen die größten Anstrengungen, um für ihre Systeme, wenn auch auf größeren Time-sharing-Systemen, Compiler zu entwickeln. Dies hat zur Folge, daß nicht so sehr die Sprache PEARL gegen die Sprache ADA in Wettbewerb steht, sondern vielmehr das Angebot geeigneter Übersetzungssysteme als Gesichtspunkt der Anwendbarkeit herangezogen werden muß. Dies bedeutet, daß ein möglichst über alle Mikrorechnersysteme hinweg durchgängiges Konzept notwendig ist, um PEARL auch langfristig im Markt etablieren zu können.

Literaturverzeichnis:

[GRO-82] W.J. Grodde: Zwischensprachen in Übersetzern für Mikrorechner
Dissertation an der RWTH Aachen, 1982