PEARL
ASSOCIATION
VEREIN E.V.

Fachtagung

Echtzeitsysteme

PEARL-Tagung `85

am 5. und 6. Dezember 1985
in Boppard

unter Mitwirkung von GI und GMR

Tagungsleitung: Dr. Dirk Krönig
Friedrichshafen

Herausgegeben vom PEARL-Verein e.V. Geschäftsstelle Stuttgart, Pfaffenwaldring 47, 7000 Stuttgart 80

Inhalt

(C) Copyright PEARL-Verein e.V., Postfach 1139, 4000 Düsseldorf 1

Tagungsleitung

Dr. D. Krönig
c/o Dornier System GmbH
Postfach 1360
7990 Friedrichshafen 1

Programmausschuß:

Dipl.-Ing. K.F. Bamberger	Siemens Karlsruhe
Prof. Dr. W. Gerth	Uni Hannover
Dr. P. Holleczek	Uni Erlangen
Dipl.-Ing. A. Küchle	Dornier System
Dr. D.Krönig	Dornier System
Dr. H. Windauer	Werum GmbH
Dr. K. Rebensburg	Uni Berlin

Technische Organisation

Geschäftstelle Stuttgart
Dipl.-Ing. V. Scheub
Pfaffenwaldring 47
7000 Stuttgart 80

Vorwort

PEARL-Tagungen waren schon immer Tagungen über Echtzeitsysteme. PEARL selbst ist ein Standard für die Programmierung von Echtzeitsystemen. Es ist nur folgerichtig, die Veranstaltung nach ihrem Gegenstand zu benennen. Daher die erstmals gewählte Bezeichnung `Fachtagung Echtzeitsysteme, PEARL-Tagung '85''.

Das weitergefaßte Thema hat Beiträge aus Gebieten angesprochen, die Echtzeitsysteme auch ohne PEARL bearbeiten. Hierin sollte eine fruchtbare Herausforderung für PEARL gesehen werden. Denn gerade im Vergleich, in der Gegenüberstellung beweist sich der Wert einer Sache.

Die 18 Vorträge behandeln Themen aus den Bereichen

* Software-Engineering-Environments
* Echtzeitdatenbanksysteme
* Kommunikation in verteilten Systemen
* Unkonventionelle Rechnerarchitekturen
* PEARL-Anwendungen

Die Veranstalter und der Programmausschuß hoffen, daß die Vorträge anregende Diskussionen induzieren, und die Tagung allen Gewinn bringt. Der ansprechende Rahmen sollte es den Teilnehmern erleichtern, zu regem Erfahrungsaustausch und persönlichen Kontakten zu finden.

Dem Programmausschuß gebührt Dank und Anerkennung für seine sorgfältige Arbeit und das ausgewogene Programm. Die Organisation der Tagung hat in bewährter Weise das Sekretariat des PEARL-Vereins übernommen. Dem scheidenden Geschäftsführer, Herrn Dipl.-Ing. V. Scheub, gebührt besonderer Dank.

Dr. Dirk Krönig

Ein Nachführsystem für Beschleunigerstrahlen auf Basis eines Mikrocomputers

R.Müller, R.Besold, H.W.Ortner
Universität Erlangen-Nürnberg

Zusammenfassung

Es wird ein Nachführsystem zur Kompensation von langsamen (~1 min) zeitlichen Schwankungen der Strahlposition an Teilchenbeschleunigern auf Basis eine Z80-ECB-Karten-Mikrocomputers und der multitasking Programmiersprache PEARL vorgestellt. Die Ablenkung des Teilchenstrahles erfolgt mit zusätzlichen rechnergesteuerten Dipolmagneten. Die Ortsdetektion wird am Beispiel von Szintillatorblenden erläutert.

A regulation system for slowly (~1 min) drifting partlicle beams is presented. The system is based on a Z80 ECB-Mikrocomputer and programmed in the multitasking program language PEARL. For deflection are computercontolled dipolmagnet used. The detection of the beamposition is shown at the example of szintillator strips near the beam.

1. Problemstellung

Bei physikalischen Experimenten an Teilchenbeschleunigern bereitet es häufig Schwierigkeiten die Strahlposition am Target zeitlich konstant zu halten. Gründe für Schwankungen liegen in veränderter Strahllage im Beschleuniger selbst, Drift der Magnetströme und Temperaturänderungen an den Ablenkmagneten, wodurch sich die Abbildungseigenschaften der Strahlführungselemente ändern. Derartig verursachte Störungen wirken sich besonders dann fatal aus, wenn die Wanderung sich in der Grössenordnung der Ausdehnung des Targets (=Wechselwirkungszone) bewegt.
Am Beispiel des Antiprotonenexperiments PS185 am LEAR-Beschleuniger am europäischen Kernforschungszentrum CERN werden die Anforderungen an die Strahlqualität beschrieben.
Der Targetdurchmesser beträgt dort lediglich 2.5mm (vgl. Bild 1 und /FRAN83/) um das Reaktionszentrum möglichst klein zu halten. Die das Target umgebenden Szintillationsdetektoren ermöglichen die Generierung eines Signals, dass kein geladenes Teilchen die Targetregion verlässt. Hinter dem Target befinden sich eine Proportionalkammer (1mm Auflösung), ein Hodoskop und zwei Driftkammern zur Detektion der Spuren der Reaktionsteilchen.
Als problematisch erweisen sich die Szintillationsdetektoren direkt neben dem Target. Bei teilweiser oder gar vollständiger Bestrahlung eines dieser Zähler vergrössert sich die Zählrate um bis zu 2 Grössenordnungen, wodurch neben der dann geringeren Ausnutzung der teuren Antiprotonen (ca. 50kSfr/h bei 1MHz Antiprotonen) zusätzlich Totzeit in der Elektronik produziert wird, da diese 'S2' genannten Detektoren für die Reaktionserkennung benutzt werden.
Weiter ist zu bedenken, dass durch eine Schwankung der Strahllage das Reaktionszentrum ungenauer bekannt ist. In der Auswertung werden wird mit Hilfe der Ortsinformation die Spuren der Reaktionsteilchen rekonstruiert. Bei genauer Kenntnis des Vertexpunktes (=Zerfallspunkt) können Untergrundereignisse erkannt und eliminiert werden. Der systematische Fehler des Apparates wird somit verkleinert.
Wie aus Bild 1 ersichtlich, besteht der S2-Detektor aus vier gegeneinander isolierten Teilen. Damit konnte in einer Strahlzeit im Mai 1984 in diesem Experiment parallel zur Datenaufnahme die Wanderung der Strahllage durch die Zählratenasymmetrie der einzelnen S2-Detektorsegmente aufgezeichnet werden. Es zeigte sich, dass während eines Spills (=Strahlextraktionszyklus von ca. 60

Minuten Dauer) die Position keiner erkennbaren Gesetzmässigkeit folgend, um mehrere Millimeter wandert, sowie die Startposition von Spill zu Spill unterschiedlich ist. Beispiele für derartige Wanderungen sind in Bild 2 dargestellt. Die Driftgeschwindigkeit liegt in der Grössenordnung 1mm/10min. Hier der Strahlschwerpunkt mit der statistischen Unsicherheit als Fehlerbalken eingezeichnet. Das Band deutet die Ausdehnung des Strahlflecks durch die Halbwertsbreite an (und wird als konstant angenommen). Die Lage des Targets ist durch die gestrichelte Linie markiert. Man erkennt, dass der Schwerpunkt meist noch am Target liegt, jedoch ein grosser Teil der Intensität den Szintillator trifft, bzw. das Target verfehlt.
Um nicht dauernd manuell Korrekturen an den Strahlführungselementen vornehmen zu müssen, erscheint es sinnvoll, diese Schwankungen automatisch zu kompensieren.

2. Regelkreis

Im folgenden werden nun die einzelnen Elemente (Sensor,Aktor,Regler) des Regelkreises vorgestellt und auf die physikalischen Anforderungen hin diskutiert.

2.1 Der Sensor

Es bieten sich verschiedene Methoden an, die Strahlposition zu ermitteln.
Bei niedriger Strahlrate kann eine direkte Zählung der einzelnen Strahlteilchen erfolgen. Mit zunehmender Intensität wird dies unmöglich und es kann lediglich eine indirekte Detektion über eine Wechselwirkung des Beschleunigerstrahls mit Materie durchgeführt werden. Bei sehr hohen Strahlströmen kann durch eine direkte Messung des makroskopischen elektrischen Stromes der Strahlpartikel auf den Strahlschwerpunkt geschlossen werden.
Der Sensor soll den Beschleunigerstrahl möglichst wenig beeinflussen (Verschmierung der Strahlenergie durch unterschiedlich starke Wechselwirkungen, Streuung, Untergrund). Andererseits ist eine Messung stark mit einer Störung verknüpft (Massenbelegung). Dies ist besonders bei indirekten Messmethoden der Fall. Die derart erzeugte Störung darf auf keinen Fall die Strahlqualität in einer Art vermindern, dass die Messgrössen nennenswert beeinflusst werden.
Viele Experimente haben schon Detektoren integriert, die es erlauben, Signalen abzugreifen, die auf den Strahlort schliessen lassen.
Im folgenden wird eine direkte Messmethode für Strahlintensitäten <1MHz vorgestellt.
Es wird dünnes Szintillatormaterial in direkte Nähe der Sollposition des Teilchenstrahles postiert und dessen Zählraten betrachtet (Bild 3).
Die Diskriminatorsignale von Teilchen aus der Randzone des Strahles, die das Szintillatormaterial direkt treffen, werden gezählt. Die Asymmetriefunktion

$$A_x = \frac{\text{Zählrate rechts} - \text{Zählrate links}}{\text{Zählrate rechts} + \text{Zählrate links}}$$

hängt nun stark vom Strahlprofil ab.
Ein Vorteil ist die geringe Materialdichte (dünne Plastik-Szintillatoren), die nur einen geringen Energieverlust verursacht. Der durch diese zusätzliche Materie bedingte Untergrund kann durch eine Antikoinzidenzschaltung mit den Blendensignalen eliminiert werden. Die Ortsbestimmung kann weitgehend ohne Beeinflussung des Teilchenstrahles erfolgen, soweit die Strahlposition nahe der Sollposition liegt, da dann nur die Randzone (Fuss) des Strahls die Szintillatoren trifft. Bei einer Fehlstellung steigt die Zählrate in einem Detektor stark an. Somit ist eine nicht zentrische Position schnell erkennbar.

2.2 Aktor

Um den Teilchenstrahl in der Position zu koorigieren, benötigt man eine geeignete Korrektureinrichtung, die präzise einstellbar sein muss.
Ein Beschleunigerstrahl besteht aus geladenen Teilchen. Bewegte elektrische Ladungen gehorchen der Lorentzkraft:

$$\underline{F} = q\ (\underline{E} + \underline{v} \times \underline{B})$$

q: Ladung des Teilchens
$\underline{E}$: äusseres elektrisches Feld
$\underline{v}$: Teilchengeschwindigkeit
$\underline{B}$: äusseres Magnetfeld
$\underline{F}$: resultierende Kraft

Mit den Bezeichnung aus Bild 4 und $\underline{v} = (0,0,v_z)$ ergibt sich durch ausmultiplizieren:

$$F_x = q\ (E_x - v_z\ B_y)$$
$$F_y = q\ (E_y + v_z\ B_x)$$
$$F_z = q\ E_z \qquad \text{(keine Ablenkung !)}$$

Im folgenden wird nur noch die Ablenkung in der x-z-Ebene betrachtet. Die Ablenkung in der y-z-Ebene ist analog.
Der Ablenkwinkel α ergibt sich zu:

$$\operatorname{tg} \alpha = \frac{v_x}{v}$$

mit

$$v_x = \frac{1}{m\, v_z} \int_{z_0}^{z_0+l} F_x(z)\, dz =$$

$$= \frac{q\, l}{m\, v_z} \left(E_x - v_z\, B_y \right)$$

und damit $(v = v_z;\ p = p_z)$

$$\operatorname{tg} \alpha = \frac{2\, q\, l\, E_x}{E_{kin}} - \frac{q\, l\, B_y}{p}$$

Eine Ablenkung kann einerseits mit Ablenkmagneten (= Steerermagnete) und andererseits mittels Plattenkondensatoren erfolgen. Um gleiche Brechkraft zu erreichen benötigt man beispielsweise die Äquivalente aus Tabelle 1. Die Werte aus Tabelle 1 gelten für eine Entfernung der Wechselwirkungszone zur Detektorebene von 3m, eine Wechselwirkungsstrecke von 10cm angenommen und eine Ablenkung um 1cm.

Teilchen	kin. Energie MeV	Impuls MeV/c	B oder Gauss	E kV/m
Protonen	10	137	151	329
Protonen	831	1500	1666	2757

Tabelle 1: Brechkraftäquivalente

Obwohl Steerermagnete für die Regelung ungünstige Effekte, wie

- Hysterese
- Remanenz
- Selbstinduktion

zeigen, findet man selten elektrische Felder zur Strahlführung. Dies liegt an der Problematik bei der Handhabung der grossen Spannung für das E-Feld (Vakuum,Überschlag).

2.3 Der Regler

Dieser Teil des Regelsystems beschäftigt sich mit dem zentralen Modul, der die vom Sensor gelieferten Daten mit den Solldaten vergleicht und bei Unstimmigkeit eine Korrekturgrösse (Stellwert) produziert. Diese Funktion kann auf unterschiedliche Weise realisiert werden. Einerseits ist eine Hardwarelösung durch eine (Analog-) Schaltung möglich und andererseits kann ein Programm diese Aufgabe erfüllen.

2.3.1 Hardwarelösung

Im einfachsten Fall des Regelproblems (der proportionalen Korrektur) besteht der Regler aus einem Differenzverstärker mit einstellbarer Verstärkung. Eine Berücksichtigung von nichtlinearen Effekten bereitet enorme Schwierigkeiten, da dann Funktionsgeneratoren für die jeweilige Kennlinie zu erstellen sind. Der grosse Vorteil derartiger Lösungen liegt in der extrem kurzen Reaktionszeit, die es erlaubt, auch extrem schnelle Schwankungen (~10µs) zu kompensieren.
Allerdings kann bei einer anderen Kennlinie (Sensor, Aktor) der Regler eventuell vollkommen neu gebaut werden müssen, da die Kennlinie fest verdrahtet ist.

2.3.2 Softwarelösung

Die Erstellung der Stellgrösse mit Hilfe eines Mikroprozessors erfordert zusätzlichen Aufwand bei der Adaption der analog arbeitenden Teile des Regelkreises. Analoge Eingangssignale müssen digitalisiert werden, sowie die nach einem Zyklus erzeugte digitale Stellgrösse oft in ein analoges Referenzsignal für die nachfolgende Stelleinrichtung gewandelt werden.
Die Reaktionszeit auf eine Störung ist durch die Taktfrequenz des Computers, sowie dem notwendigen Operationsaufwand festgelegt. Bei wenigen Operationen kann die Stellgrösse schon nach einigen 100µs vorliegen. Wenn komplizierte Berechnungen durchgeführt werden, ergeben sich Zykluszeiten von ~100ms.
Der Vorteil einer Softwarelösung liegt

- in den vielfältigen Möglichkeiten auch komplizierte Sensorsysteme (MWPC, Hodoskop) benutzen zu können.
- darin, exterme Nichtlinearitäten in der Abbildungsfunktion leicht berücksichtigen zu können.
- darin, ein anderes Sensorsystem ohne grössere Probleme an den Regler anschliessen zu können.
- in der freien Wahl für die Parameter der

Abbildungsfunktion.

- darin, dass Programme leichter auf eine andere Funktion anpassbar sind, als eine Hardwarelösung.

3. Die Implementierung

In Anbetracht der grossen Driftzeiten im vorliegenden Problem und der langen Einstellzeit für den Magnetstrom (grosse Induktivität der Wicklung) sind kürzere Reaktionszeiten als 100ms nicht sinnvoll.

Daher wurde das Regelprogramm auf einem Mikrocomputer erstellt. Als Programmiersprache empfahl sich PEARL wegen seiner Unterstützung bei Ansteuerung von Prozessperipherie. Darüber hinaus konnte durch Ausnutzung der Multitaskingfähigkeit eine Diagnostikfunktion für den Regelautomaten leicht installiert werden.

Die Spezifikation des gesamten Problems erfolgte in PASS (=Parallel Activities Spezification Scheme /FLEI84/). Als Zielrechner wurde ein Z80-ECB-System gewählt, da für diese (preiswerte) Maschine ein komplettes PEARL-System verfügbar war.

Vom eigentlichen Regelprogramm wurde als Vorlauffunktion die Aufnahme der Eichfunktion abgetrennt. Dies erwies sich als sinnvoll, da die Charakteristik der Strahlführung über einen grösseren Zeitraum unverändert bleibt.

Aus der Kommunikationsstruktur (Bild 5) ist das Zusammenspiel der 20 Prozesse leicht zu erkennen. Von diesen 20 parallelen Aktivitäten sind lediglich die Prozesse 'BEDIE','DISPL','PROTO','HAND', 'REGLE' und 'PROTO' als PEARL-Tasks implementiert. Die weiteren Prozesse sind durch Hardware realisiert.

Die Task 'BEDIE' wickelt den gesamten Benutzerdialog ab (Start/Stop, Parameter einstellen, ...). Die Task 'DISPL' sorgt für die Anzeige des aktuellen Systemzustandes auf einem Statusschirm. Für eine Dokumentation der Regelvorgänge können alle wichtigen internen Daten auf einen Protokolldrucker oder File von der Task 'PROTO' abgeschrieben werden. Die Task 'HAND' (=manuelle Steuerung) liest ein Tastenfeld ein und verstellt dann den Magnetstrom entsprechend der Tasteninformation zur Vorgabe der Sollposition. Die Task 'REGLE' startet bei Regelbetrieb die 4 Zähler 'Z1'-'Z4' und liest diese zyklisch aus, berechnet die Regelgrösse aus der Zählratenasymmetrie und der Eichfunktion, wenn eine signifikante Fehlstellung (statistische Fehler!) erkannt wurde. Diese neue Regelgrösse wird dann sofort an die Stelleinrichtung (DAC liefert Referenzspannung für Magnetstromversorgung) ausgegeben. Anschliessend startet ein neuer Regelzyklus. Die Task 'POSDI' liest zyklisch die Zähler 'Z5'-'Z8' und gibt die sich daraus ergebende Strahllage als Asymmetriewert an die Task 'DISPL' weiter. Wenn 'Z5'-'Z8' mit den gleichen Signalen versorgt werden wie 'Z1'-'Z4', dann ist so eine unabhängige Messung der Strahlposition gegeben. Der Experimentator kann sich so von der ordnungsgemässen Funktion überzeugen.

4. Erfahrungen

Das System war im August 1985 am Antiprotonenstrahl des CERN cirka eine Woche lang ohne Unterbrechung im Einsatz. Dabei zeigte sich, dass die Rechenleistung des Z80-Mikroprozessors für dieses Regelproblem voll ausreichte und der Algorithmus in der Lage war selbst mit unvorhergesehenen Ansprechverhalten des Sensors zufriedenstellend arbeitete. Auf diese Art gelang es den sehr teuren Antiprotonenstrahl auf ca. 0.05mm genau auf der Sollposition zu halten, und damit die optimalen Bedingungen für das Experiment herzustellen. Lediglich eine Kopplung zum Datenaufnahmerechner (hier VAX 750) ist für die endgültige Version der Regelapparatur noch zu installieren.

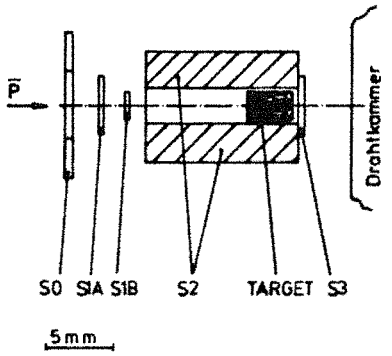


Bild 1: Targetregion des LEAR-Experimentes
S1 - S3 sind Szintillationsdetektoren

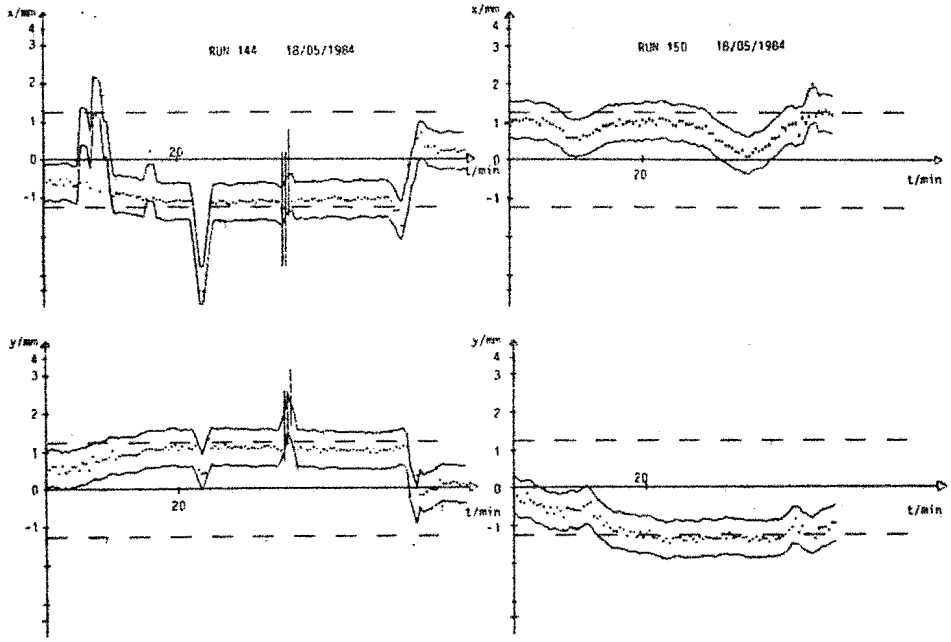


Bild 2: Strahlposition am Target

+ Strahlposition mit statistischem Fehler

—— Halbwertsbreite der Strahlintensität

– – Targetbegrenzung

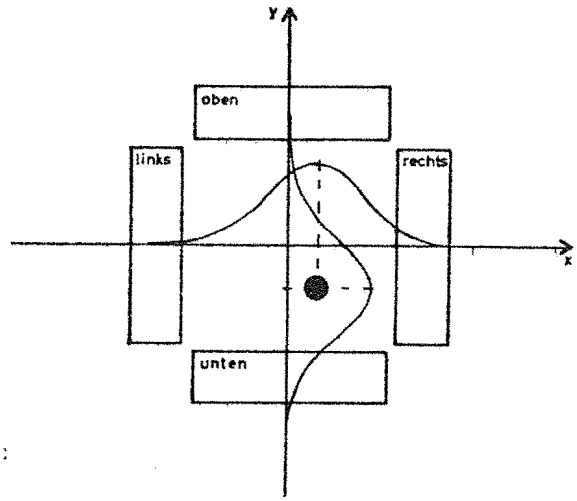


Bild 3: Geometrie der Szintillatorblenden mit eingezeichnetem Strahlprofil

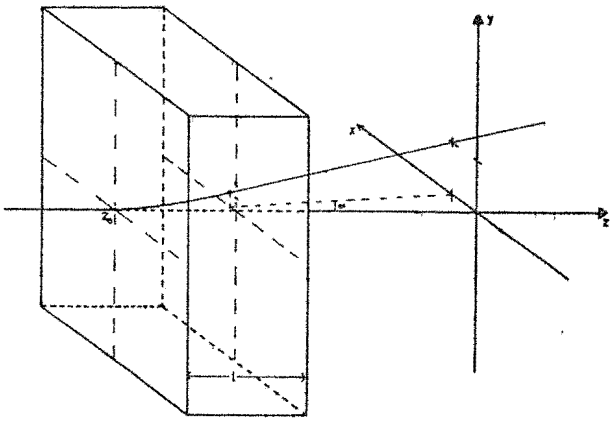


Bild 4: Koordinatensystem im Aktor

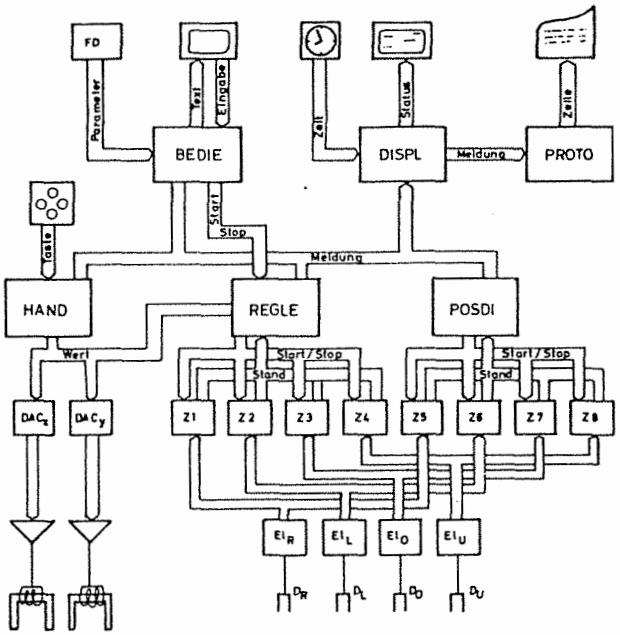


Bild 5: Kommunikationsstruktur des Regelprogramms in PASS

Schrifttum:

/FLEI84/: A.Fleischmann
Ein Konzept zur Darstellung und Realisierung von verteilten Prozessautomatisierungssystemen
Dissertation Erlangen 1984

/FRAN83/: R.v.Frankenberg
Ein triggeraktives Targetsystem
Diplomarbeit, Physikal. Inst.,
Erlangen 1983

Adresse:

Universität Erlangen-Nürnberg
Physikalisches Institut Abt. III
z. Hdn. Rainer Müller
Erwin Rommelstr. 1
8520 Erlangen

Telefon: 09131/85-7080

PEARL - Anwendungen bei der ESG

M. Fenzl, F. K. Nonhoff, W. Hirschel, München

Zusammenfassung

Es werden drei Projekte vorgestellt, die mit ATM-PEARL realisiert wurden. Die dabei in bezug auf PEARL gesammelten Erfahrungen werden diskutiert.

Summary

Three applications implemented in ATM-PEARL are presented. The common PEARL specific aspects are discussed.

1. Einleitung

Die PEARL-Aktivitäten der Firma ESG reichen ca. 10 Jahre zurück. Die Erfahrungen im Projekteinsatz von PEARL (seit 79) beruhen vorwiegend auf realzeitorientierten, nationalen militärischen Systemen.

Es werden drei Projekte mit unterschiedlichem Systemcharakter, Umfang und Entwicklungsrandbedingungen vorgestellt. Bei allen Anwendungen kommen ATM Rechner und ATM-PEARL zum Einsatz. Die beim Entwickeln der PEARL-Programmsysteme gewonnenen Erfahrungen werden diskutiert.

2. Übersicht über die Projekte JADE, HFLA, ADLER

2.1 Projekt JADE

Im Bereich der Deutschen Bucht und der angrenzenden Wasserstraßen entsteht derzeit das "Verkehrssicherungssystem Deutsche Bucht", das mit seinen technischen Komponenten wie Synthetikradar, Präzisionspeiler und Datenverarbeitung die Sicherheit des Schiffsverkehrs und die Verkehrssteuerung verbessern wird.
Innerhalb dieses Vorhabens wurde das Programmsystem für die Komponente "Schiffsdatenverarbeitung mit Datenverbund" realisiert, das folgende Hauptaufgaben umfaßt:

- Speicherung, Organisation und Darstellung von Daten für die Verkehrsüberwachung
- Dokumentation des Verkehrssystems über einen längeren Zeitraum
- Automatischer Austausch von Schiffsdaten zwischen den Schiffsdatenrechnern der Revierzentralen Wilhelmshaven, Bremerhaven, Cuxhaven und Brunsbüttel über DATEX-L.

Das System ist als Doppelrechnersystem für hot-stand-by-Betrieb mit teilweise umschaltbarer Peripherie konzipiert und hat folgende Hardware-Konfiguration (B i l d 1) :

- 2 ATM 80-30 mit 384 KByte, gekoppelt über Selektorkanal
- 2 Plattenspeicher (10 MByte)
- 2 Protokollfernschreiber
- 4 Datensichtgeräte
- 2 graphische Sichtgeräte
- 2 Magnetbandsysteme mit Doppelzugriff.

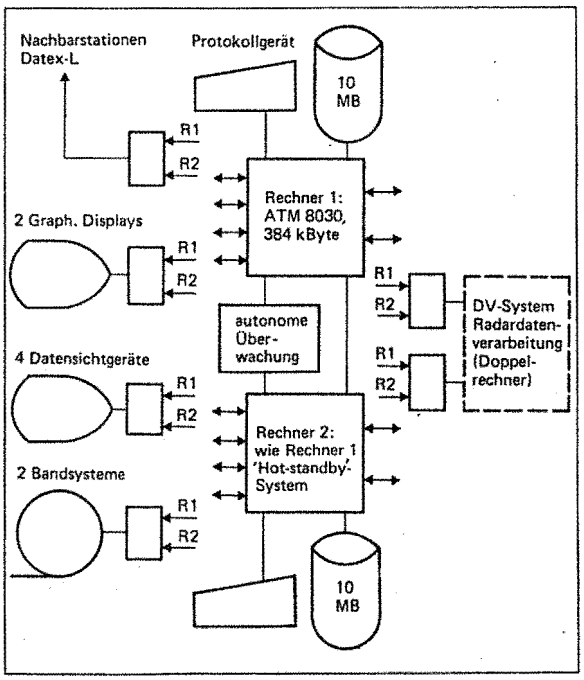


Bild 1. Hardware-Konfiguration JADE

Softwarekonzept
Die Konsistenz der Datenbasis im Doppelrechnersystem wird dadurch gewährleistet, daß alle, die Datenbasis und systemglobale Zustände verändernden Aufträge in Ringpuffern in beiden Rechnern abgelegt werden. Die Bearbeitung dieser Aufträge erfolgt sequentiell.

Das Anwendersystem ist in vier Subsysteme aufgeteilt:
- Ablaufsteuerung und Verwaltung der Schiffsdatenbestände
- Dialogsubsystem
- Datenverbundsteuerung
- Verkehrsdokumentation

Die Kommunikation der Subsysteme erfolgt über Botschaften und Ringpuffer. Jedes Subsystem enthält einen Monitor, der ihm zugeordnete Verarbeitungstasks koordiniert.

2.2 Projekt HFLA
Im Rahmen der Entwicklung HFlaAFüSys (Heeres-Flugabwehr-Aufklärungs- und Gefechtsführungssystem, kurz HFla) erstellt Firma ESG die Software für den Gerätesatz "Luftlagedatenverarbeitung" (LLDV) mit folgenden Hauptfunktionen:
- Erstellung einer Sensorluftlage aus den Informationen der lokalen Radaranlage
- Einarbeiten der eigenen Luftlage und der über Datenfunk empfangenen Luftlageinformation benachbarter Information in eine Gesamtluftlage (Multi-Radar-Tracking)
- Eliminierung von Festzielen
- Ortung von Störern

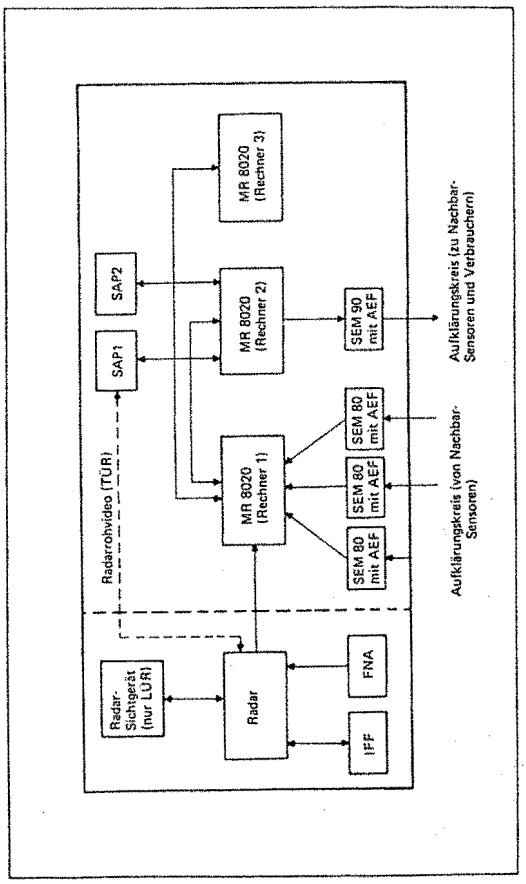
Die Track-Algorithmen und die Bedienung der Peripherie, nämlich:
- zwei autonome Systemarbeitsplätze (SAP)
- vier Datenfunkgeräte SEM 80/90 mit Anpaßeinheit Funk (AEF)
- Radarschnittstelle

induzieren eine hohe Rechnerlast. Drei MR8020, die über den Speicher-Bus gekoppelt sind, erbringen die geforderte Rechenleistung. Virtuell verfügen die Rechner über einen Hauptspeicher von 256 KByte. Der gemeinsame Speicherblock von 32 KByte liegt physikalisch im Rechner 1 (B i l d 2).

Softwarekonzept
Die logische Kopplung der drei Rechner erfolgt über einen Satz von Ringpuffern, die im gemeinsamen Speicherbereich angelegt sind. Da pro Ring nur ein Leser und ein Schreiber existiert, können die Zugriffe mit minimalem Aufwand implizit über Schreib- und Lesezeiger synchronisiert werden. Die Ablaufsteuerung erfolgt in den drei Rechnern einheitlich über Monitortasks, die die relevanten Ringpuffer nach vorgegebenen Prioritäten zyklisch abarbeiten und die zugehörigen Verarbeitungsprogramme anstoßen. Die Synchronisierung zwischen Monitor und Verarbeitungsprogrammen erfolgt über Semaphore.

2.3 Projekt ADLER
ADLER (Artillerie-Daten-Lage-Einsatz-Rechnerverbund) ist das Führungssystem für die Artillerie. Die Hauptaufgaben sind dabei:
- Führung des Feuerkampfes
- Planung und Leitung des Feuerkampfes
- Befehlsgebung
- Artilleristische Aufklärung
- Verbindung innerhalb ADLER und zu anderen Verbänden

Das Ziel, den Einsatz der Artillerie effizienter zu gestalten, wird erreicht durch Einführung von Rechnerunterstützung bei der Lösung von Teilaufgaben und Beschleunigung des Informationsaustausches über Datenfunk in einem gemäß den militärischen Führungsebenen gegliederten, hierarchischen Netz von Funkkreisen.



Hardware-Konfiguration Luftlagedatenverarbeitung

In ADLER gibt es Dienstposten mit unterschiedlichen funktionellen Anforderungen. Die für die verschiedenen Varianten benötigte Geräteausstattung wird aus logistischen Gründen aus einer einheitlichen ADLER-Zelle (B i l d 3) konfiguriert. Die Varianten bestehen aus einer, zwei oder vier Zellen. Die Geräteausstattung einer ADLER-Zelle ist wie folgt:

- Rechner MR8020 1 MByte
- 1 oder 2 Funkgeräte
- 1 oder 2 Bildschirme mit Tastatur
- Drucker
- Programmladegerät (Bubble-Speicher 1 MByte)
  Plattenspeicher (40 - 70 MByte; nur in Operationszentrale)
- Schnittstellen für Gefechtsstandskommunikation

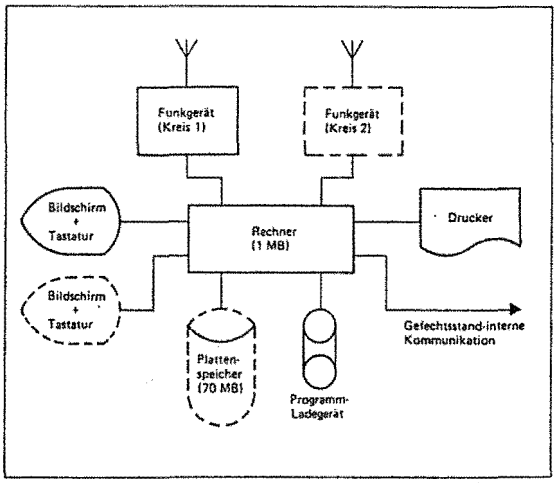


Bild 3. Konfiguration einer ADLER-Zelle

Software-Konzept

Neben Funktionsumfang und Randbedingungen bezüglich Hardware und Systemsoftware beeinflußten folgende Punkte die Wahl des Software-Konzepts wesentlich:

- N-Teilnehmersystem
  Jeder ADLER-Teilnehmer sollte (prinzipiell) Zugriff auf das gesamte Anwendungsspektrum haben, mit funktionsabhängigen Anwendungsprofilen.
- Leichte Konfigurierbarkeit der Software im Hinblick auf den Ablauf auf unterschiedlichen Hardware-Konfigurationen und verteilte Verarbeitung.
- Erweiterbarkeit
  Die Realisierung in mehreren Ausbaustufen erfordert ein die Modularisierung förderndes, von den Anwenderfunktionen unabhängiges Softwarekonzept.
- Die lange Lebensdauer militärischer Systeme stellt hohe Anforderungen in bezug auf Pflegbarkeit, Wartbarkeit und Zuverlässigkeit des Softwareprodukts.
- Die Anforderungen der Nutzer an das Realzeitverhalten erfordern eine sorgfältige Abbildung, insbesondere der systemnahen Software-Anteile, auf die Gegebenheiten der Hardware und Vorgaben für die Anwendung des PEARL-Sprachumfangs in der Implementierung.

Die Ablaufsteuerung des Mehrbenutzersystems erfolgt aus Effizienzgründen über Tabellen, die von verteilten Abwicklern abgearbeitet werden. Den Benutzern werden segmentierte Arbeitsspeicherbereiche zugeordnet (realisiert als PEARL-Datenmodul), in denen sie sich voneinander unabhängig entwickeln können. Außerdem wurde den Entwicklern ein Stack-Mechanismus, der Parameterübergabe und Aufrufschachtelung erleichtert, sowie eine Pufferverwaltung und Dateiverwaltung zur Verfügung gestellt.
Die Intertaskkommunikation wird über Botschaften abgewickelt. Das Konzept unterstützt asynchronen und synchronen Nachrichtenaustausch (Auftrag ohne, bzw. mit warten auf Quittung). Es gibt kurze Botschaften mit maximal 5 Byte Nutzinformation für den Empfänger sowie variabel lange Extensionen, die im Stack-Bereich der Arbeitsspeicher oder in Puffern abgelegt werden. Als Hilfsmittel für Integration und Test ist ein Task-Trace integriert.
Im Hinblick auf Modularität, Erweiterbarkeit und Flexibilität werden im Programmsystem Tasks, Benutzer und Geräte auf logische Bezeichner abgebildet. Für die Implementierung wurden streng formalisierte PEARL-Rahmenprogramme vom Design-Team festgelegt, die alle konzeptionell notwendigen Datenstrukturen und Zugriffsfunktionen auf Betriebsmittel enthielten.
Nach anfänglichen Widerständen bei den Implementierern (sie fühlten sich in ihrem schöpferischen Freiraum eingeschränkt) zeigten sich große Vorteile in bezug auf Implementierungsgeschwindigkeit und bei der Softwareintegration.

3. Programmkenngrößen

In T a b e l l e 1 sind einige Kenngrößen der Programmsysteme zusammengestellt. Bemerkenswert ist der hohe PEARL-Anteil in allen drei Anwendungen.

Kenngrößen \ Projekt	JADE	HFLA	ADLER
PEARL-Tasks	69	36	100
Anwendercode [KB]	500	350	1500
Betriebssystem [KB]	80	3 x 60	80
Tabellen	50	-	1600
Interrupts/sec.	50 - 100	200	50 - 100
PEARL-Anteil	93 %	95 %	98 %

Tabelle 1. Kenngrößen der Programmsysteme

4. Schalenmodell

Die Software wurde in hierarchischen Schalen strukturiert; diese Vorgehensweise bietet, ähnlich wie beim OSI-Modell der Kommunikation, Vorteile in bezug auf die Zuordnung von Subsystemen, Funktionskomplexen und Dienstleistungen innerhalb eines Softwaresystems und die Kommunikation der einzelnen Komponenten untereinander.
Bei der Realisierung von einheitlichen Daten- und Aufrufschnittstellen zwischen den Schichten und Programmen bietet PEARL sehr gute sprachliche Voraussetzungen. Blockstruktur und Modulkonzept unterstützen einen bei der Modularisierung und bei der Festlegung von Sichtbarkeitsgrenzen.
Die folgende T a b e l l e 2 gibt einen Überblick über die Zuordnung von Funktionen zu Softwareschichten.

Projekt / Schicht	JADE	HFLA	ADLER
ANWENDER-SW (Hauptaufgaben)	Schiffsdatenverwaltung Meldungsverarbeitung Aufzeichnung/Replay Verbundsteuerung	Sensorluftlage Gesamtluftlage Störerortung Falschziele/Clutter	Feuerkampf Befehlsgebung Meldungsverarbeitung Aufklärung Lagebeurteilung Verbundsteuerung
PROZESS-SOFTWARE			
Ablaufsteuerung	x	x	x
Systeminitialis.	x	x	x
Systemüberwachung	x	x	x
Kommunikation	x	x	x
Datenhaltung	x		x
Dialog (MMI)	x		x
SYSTEMNAHE ANWENDER-SW			
Rechnerkopplung	x		x
Botschaftsdienst	x		x
Pufferverwaltung	x	x	x
Zugriffsroutinen	x	x	x

Tabelle 2. Zuordnung von Funktionen im Schalenmodell

5. Entwicklungsumgebung

Für die Softwareentwicklung steht eine umfangreiche DV-Konfiguration zur Verfügung, die über ein lokales Netz auf Ethernet-Basis (Bandbreite 10 MBaud) miteinander gekoppelt ist. Als Host-Rechner dienen Siemens-Rechner der Serie 75xx, die unter BS2000 laufen sowie VAX 750 mit den Betriebssystemen UNIX und VMS. Neben zahlreichen Netzterminals und Arbeitsplatzrechnern sind eine Reihe von projektspezifischen Rechnern angeschlossen.
B i l d 4 gibt am Beispiel ADLER einen Überblick über die Zuordnung einzelner Entwicklungstätigkeiten auf die Rechner.
Die Entwicklung erfolgt unter AMDES, in dem UNIX-Host-Systeme mit ATM-Entwicklungsrechnern gekoppelt sind, die unter AMBOS laufen. Dies erlaubt eine komfortable Entwicklung unter UNIX. Die AMBOS-Maschinen dienen als Compile-Prozessoren sowie zum Generieren und Testen der PEARL-Programmsysteme.

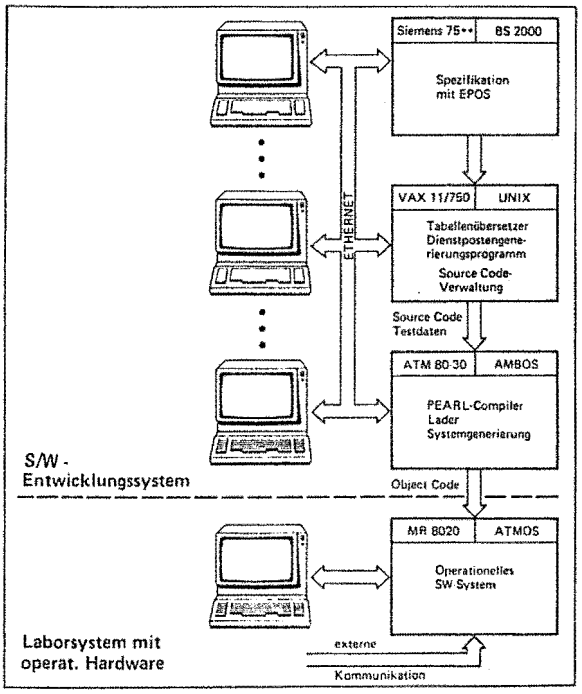


Bild 4. ADLER-Software-Entwicklungssystem

6. Projektübergreifende PEARL-Erfahrungen

Zunächst einige Bemerkungen zu Schwachpunkten:
Die meisten Mängel sind implementierungsabhängig, bzw. auf Hardware-Eigenschaften zurückzuführen.

- Der PEARL-Compiler ist nicht reentrant. Dies führt zu Wartezeiten in der Implementierungsphase.
- Größere PEARL-Module können häufig erst nach iterativer Anpassung von Compiler-Startparametern übersetzt werden, weil gewisse interne Listen linear abhängig sind.
- Das Laufzeitpaket ist zwar gut modularisiert, verlängert jedoch die Programmlaufzeiten.
- Manche Sprachkonstrukte wie GET, PUT, CAT, OPEN erzeugen viel Code.
- Es fehlt ein PEARL-Binder. Die globalen Querbezüge werden in Modul-Tasks abgelegt, die eine Prozeßnummer belegen und in Hauptspeichersystemen statisch Speicherplatz beanspruchen.
- Sprachliche Mängel sind in der Ausprägung der CASE-Anweisung und dem Fehlen von variablen RECORDS zu sehen.
- Für viele Prozessanwendungen wäre es wünschenswert, bei Task-Anweisungen Parameter mitgeben zu können.

Positive Erfahrungen mit PEARL:

- Sprachumfang

ATM-PEARL weist gegenüber Full PEARL gemäß DIN 66253-F nur geringe Einschränkungen auf, wie z. B.:

- o Keine expliziten Operatordeklarationen (Standardoperatoren)
- o Keine expliziten Interface-Deklarationen (Standardinterfaces)
- o Keine BOLT-Variable (nur Semaphore)
- o Keine Semaphor-Listen in Semaphoranweisungen
- o Keine dynamische Änderung von Taskprioritäten

Die Einschränkungen wirkten sich bei unseren Anwendungen nicht erschwerend aus und wurden ggf. auf Anwenderebene umgangen.

Bei größeren Programmsystemen ist es neben anderen Maßnahmen sinnvoll, den Entwicklern in der Anwendung von Sprachkonstrukten Vorgaben zu machen, z. B. um einen einheitlichen Programmierstil zu erzwingen, Integrationsproblemen vorzubeugen und Implementierungsschwächen (Code-aufwendige Sprachkonstrukte etc.) zu umgehen.
Folgende, über Basic PEARL hinausgehende Elemente erwiesen sich dabei als besonders nützlich:

- Typspezifikation (hauptsächlich mit Strukturattribut) trägt in Verbindung mit INCLUDE-Pragma sehr zur Programmiersicherheit bei
- Beliebige Strukturschachtelung erlaubt optimale Darstellung von Daten nach funktionellen Gesichtspunkten
- Identitätsspezifikation (SPC...IDENT(...)) zur Zuordnung von weiteren Zugriffsfunktionen bzw. Bezeichnern auf existierende Objekte
- Alle Deklarationsarten (außer TASK's und SEMA's) auf beliebiger Blocktiefe
- Keine spezielle Deklarationsreihenfolge
- Referenzobjekte
- Arrays in Strukturen
- Beliebige Arrayindexgrenzen (dynamisch)
- Zuweisung von Arrays und Strukturen als ganzes
- Beliebige Ausschnitte (SLICES) aus ARRAY's, Zeichen- und Bitketten
- Array und Strukturdisplays

Weitere positive Aspekte sind:

- Gute Unterstützung durch ATM beim Lösen von Problemfällen
- Interesse des Herstellers an Weiterentwicklungen, z. B. Code-Datentrennung für Anwenderprogramme und Einführung eines Optimierungslaufes im PEARL-Compiler
- Mehrstufiges INCLUDE-Pragma
- Große Anzahl von Fehlermeldungen durch Compiler und LZO
- Quellbezogenes PEARL-Testsystem (QPTS) bietet gute Hilfestellung beim Test von Anwendersystemen
- Erlernbarkeit/Akzeptanz
  PEARL ist leicht erlernbar (Aufwand 1 bis 4 Wochen). Die Erstellung optimierter Programme erfordert ca. 6 Monate Erfahrung. Die Akzeptanz der Sprache im Entwickler-Team ist sehr gut.
- Die Anwendung von PEARL fördert die Produktion von gut strukturierten, transparenten Programmsystemen. Dies beeinflußt die Kosten für Wartung und Pflege günstig.

Schlußbemerkung:
Zusammenfassend kann man sagen, daß sich PEARL als Implementierungssprache für Prozeßanwendungen in unserem Hause gut bewährt hat.
In das zur erfolgreichen Projektabwicklung nötige Software-Engineering läßt sich PEARL gut einbetten.

Anschrift des Autors:

Dr. M. Fenzl
Elektronik-System-Gesellschaft mbH
Vogelweideplatz 9
8000 München 80

Anwendung von unterschiedlichen PEARL-Umgebungen in einem Projekt

Dr. M. Ammann, Tettnang

Zusammenfassung

Dornier entwickelte und implementierte für das rechnergestützte Waffeneinsatzsystem ARES die Anwender- und Testsoftware in PEARL nach dem Phasenmodell mit Unterstützung durch EPOS.

Zum Einsatz kamen zwei verschiedene PEARL-Entwicklungsumgebungen, nämlich für ATM 80-PEARL und das DEA (GPP)-PEARL. Das Arbeiten mit diesen verschiedenen Umgebungen wird anhand von Aspekten aus ARES diskutiert. Aus den Erfahrungen werden Schlußfolgerungen gezogen.

Stichworte

PEARL, Entwicklungssystem, Sprachumfang, Systemleistung

Summary

Dornier developped and implements application software and test software for the compute-aided weapon operation system ARES with respect to the phase model and with support by EPOS.

Two different PEARL development systems were used, one for ATM 80-PEARL and one for DEA (GPP)-PEARL. Working with these different systems is discussed with respect to aspects from ARES. Out of the experiences there will be done some conclusions.

1. Einleitung

Das Artillerie-Raketen-Einsatz-System ARES hat die Aufgabe, als rechnergestütztes Waffeneinsatzsystem in der Führungsebene einer Batterie die taktische und technische Feuerleitung zu unterstützen. Dornier entwickelt und implementiert für dieses System die Anwender- und Testsoftware in PEARL.
Die Softwareentwicklung erfolgte nach dem Phasenmodell mit Unterstützung durch EPOS.

Der geforderte Funktions- und Leistungsumfang wird durch ein 3-Rechner-System (mit zwei verschiedenen Rechnertypen) realisiert, dessen Komponenten verschiedene Teilaufgaben bearbeiten unter Berücksichtigung von Redundanz und Ausfallverhalten. Für weitere Einzelheiten wird auf [1] verwiesen.

Für Software-Entwicklung und Test wurden zwei unterschiedliche PEARL-Umgebungen eingesetzt, die mit ihren Eigenschaften und Leistungen sowie den daraus zu ziehenden Schlußfolgerungen vorgestellt und diskutiert werden.

2. Verwendete Entwicklungsumgebungen

Für den einen Rechnertyp, einen MR 8020, wird ein ATM 80-Entwicklungssystem eingesetzt; Bild 1 zeigt die verwendete Konfiguration, die sowohl als Gast- wie auch als Zielrechner eingesetzt wird.

Folgende Software wird dabei eingesetzt:

ATMOS-Betriebssystem
AMBOS-Produktionssystem
PEARL-Compiler/-Testsystem
BAPAS-DB (Fa. Werum)

Für den anderen Rechnertyp, ein DEA 2020 (Basis Intel 8086), wird das PEARL-System der Firma GPP eingesetzt auf zwei verschiedenen Gastrechnern. Das zuerst verwendete System bestand aus einer PDP 11-Konfiguration für den PEARL-Compiler sowie einem Intel Serie III-Entwicklungssystem, angekoppelt über eine V24-Verbindung. Zielsystem ist ein DEA 2020 mit Drucker, Kommunikationsschnittstellen und Display, ohne Externspeicher.

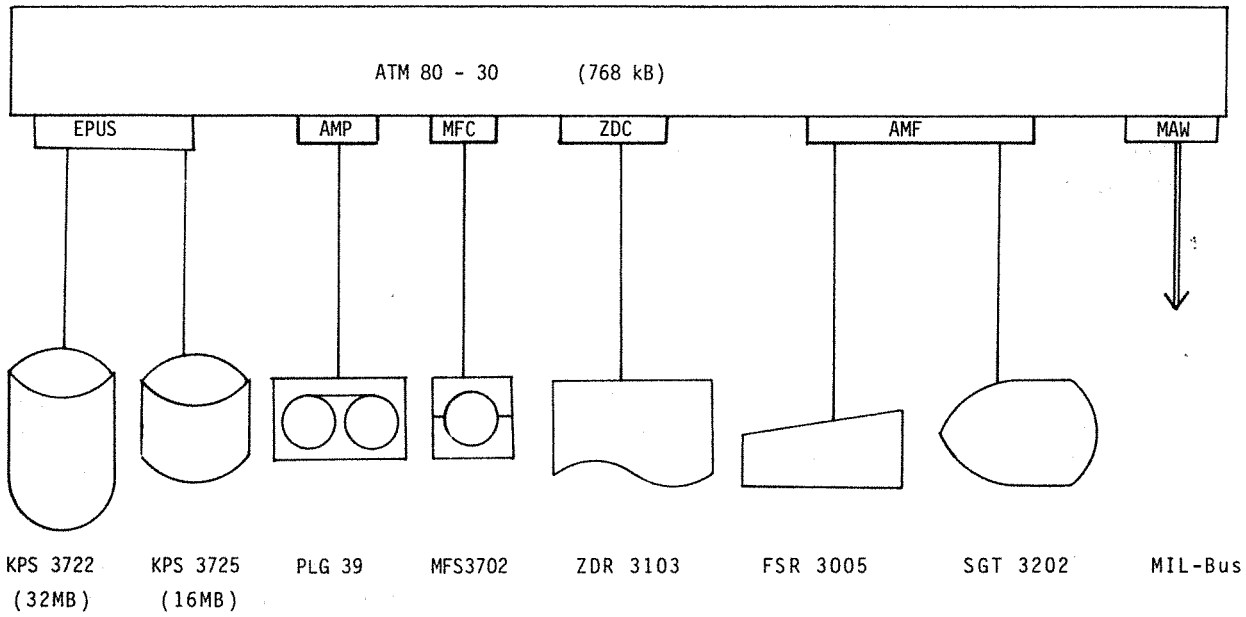


Bild 1 Entwicklungssystem ATM 80-30

Folgende Software wird dabei eingesetzt:

RSX 11-M V 4.1
PEARL-Compiler, -Systemdatengenerator > PDP 11
ISIS II
PEARL-System > Serie III

Zur Steigerung des Durchsatzes werden weiterhin Entwicklungssysteme vom Typ Intel SBC 86/3xx eingesetzt; Bild 2 zeigt die verwendete Konfiguration.

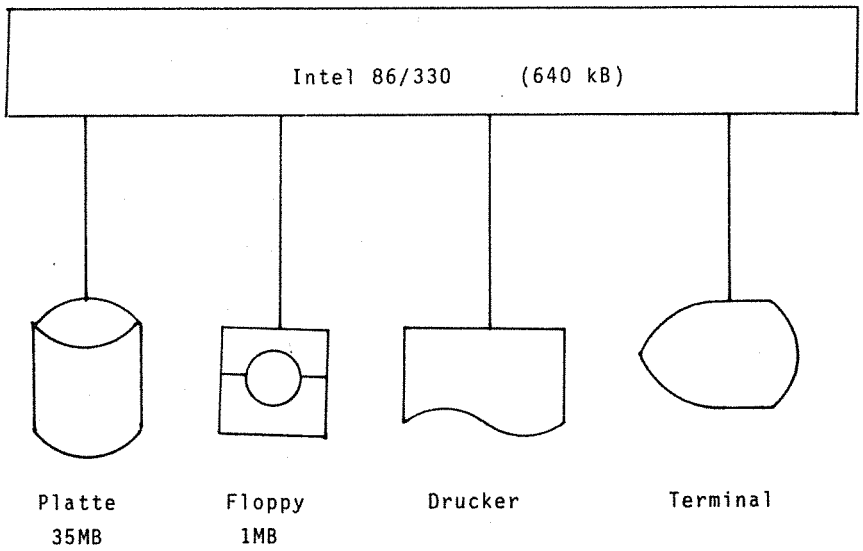


Bild 2 Entwicklungssystem Intel 86/330

Diese Konfiguration kann auch als Zielsystem verwendet werden in Testphasen, die noch keine Peripherie benötigen. Ansonsten ist wieder ein DEA 2020 das Zielsystem. Folgende Software wird hier eingesetzt:

iRMX 86 V6.0
PEARL-Compiler, -Systemdatengenerator
PEARL-System

Für das DEA 2020 ist das Projekt ARES Erstanwender.

Die beiden kurz vorgestellten Entwicklungsumgebungen werden im folgenden nun näher besprochen im Hinblick auf die konkrete Anwendung und das Handling. Dabei werden die Punkte

- Sprachumfang
- Systemleistung
- Test
- Releases
- Stördienst

angesprochen.

3. PEARL-System ATM 80

Anhand der o.a. Stichworte wird nun die PEARL-Entwicklungsumgebung auf ATM 80 aus der Sicht der erfolgten Anwendung diskutiert.

Sprachumfang: Der ATM-PEARL Sprachumfang geht weit über Basic-PEARL hinaus und umfaßt große Teile von Full-PEARL. Für ARES wurden insbesonders folgende Sprachelemente benutzt:

- Strukturen geschachtelt sowie mit Feldern als Element
- TYPE-Definition
- Identitätsspezifikation SPC ... IDENT
- Character-Selektion und -Handling
- Slices
- Referenzen

Damit war es möglich, sauber strukturierte Programme mit klaren Datenstrukturen zu entwickeln und zu implementieren. Als positiver Nebeneffekt war festzustellen, daß die Programme wesentlich effektiver und kleiner wurden als wenn der zu verwendende Sprachumfang auf Basic-PEARL beschränkt würde. Eine effektive Anwendung des ATM-PEARL ist jedoch im Hinblick auf die unterliegende Implementation nur möglich, wenn der Entwickler vertiefte Kenntnisse über Compiler, Laufzeitsystem und interne Struktur von Programmen und Daten hat.

Systemleistung: Für eine zügig abzuwickelnde Test- und Integrationsphase ist die Übersetzungsgeschwindigkeit des PEARL-Systems (bis zum Vorliegen eines ausführbaren Programms) ein wichtiges Leistungsmerkmal. Gemessene Zeiten ergaben Werte von ca. 150-200 PEARL-Statements pro Minute je nach Umfang der dazuzubindenden Bibliotheken und Objekte. Mit dieser Leistung sind vernünftige Turnaround-Zeiten in der Testphase realisierbar.

Ein weiteres Leistungsmerkmal sind die Compilergrenzen bei der Übersetzung größerer Einheiten in Bezug auf Anzahl Symbole, Anzahl Initialisierungswerte, Operandenkeller. Zu Beginn der Entwicklung wurden relativ kleine Werte festgestellt, die für den geplanten Umfang und Komplexitätsgrad der Software nicht ausreichend waren, ohne größere Auswirkungen auf die Implementation. In Zusammenarbeit mit Fa. ATM wurde hier Abhilfe geschaffen, so daß nunmehr in einem Übersetzungsvorgang maximal 800 Zugriffsfunktionen bei 500 Typattributen bearbeitet werden können. Diese Werte sind jedoch noch immer für mehrere geplante Übersetzungseinheiten zu klein und bedingen eine Aufspaltung mit entsprechendem Overhead.

Das letzte hier angesprochene Leistungsmerkmal sind die zulässigen Größen von Adressräumen für Code (d.h. hier Tasks) und Daten.
Ohne die Entwicklung der letzten Jahre darzulegen sei festgestellt, daß hier (derzeit noch) zwei wesentliche Grenzen existieren: die Größe eines Laufbereichs für eine Task (mit oder ohne Daten) und die Größe des Commonbereichs; beide Werte sind nicht unabhängig. Die erreichbaren Laufbereichsgrößen von 32 kB reichen oftmals nicht aus und erzwingen eine Aufteilung von Tasks. Die Problematik des Common-Bereichs wurde inzwischen dadurch entschärft, daß Anwender-Prozeduren und -Daten in das LZO-Segment bzw. in Zonen verlagert werden können, jedoch mit harten Einschränkungen.

Test: Für das PEARL-System existiert ein Testsystem (QETS), das umfangreiche Funktionen für sprachbezogenes Testen anbietet. Dieses System wurde jedoch nur beschränkt eingesetzt, da seine Anwendung die ohnehin knappen Adressräume zu sehr belastet. Der Hauptteil der Tests wurde mit einer eigenen allgemeinen Testumgebung durchgeführt, die ein effektives Testen ermöglicht.
Diese enthält dazu einen Satz von Grundfunktionen zur Ablaufsteuerung von Tests sowie für Dumps und Protokolle. Sie werden testspezifisch ergänzt um Funktionen, wie spezielle Dumps, Dialog für Parameteränderung und Anstoß spezieller Testhilfsroutinen.

Ein großer Vorteil des ATM-PEARL-Systems ist die Tatsache, daß während Entwicklung, Test und SW-Integration der Test auf der gleichen Anlage stattfindet wie die Übersetzung, d.h., Gast- und Zielmaschine sind identisch. Nachteilig ist, daß während des Tests die Anlage zu einem Einplatz-System wird, was in intensiven Testphasen zu Engpässen führt.

Releases: Die ATM-Systemsoftware inkl. PEARL-System wird ständig gepflegt und weiterentwickelt. In Abständen von ca. 1/2 bis 1 Jahr werden neue Versionen herausgegeben. Der positive Teil dieser Tatsache ist ein nach und nach immer leistungsfähigeres System, was der Entwicklung und Implementation von Software sehr zugute kommt. Die unangenehmere Seite ist zum einen der daraus resultierende beträchtliche Aufwand für Systemgenerierungen und Umstellungen, zum anderen die Feststellung, daß neue Releases zu oft mit Fehlern behaftet sind, die die SW-Entwicklung wesentlich beeinträchtigen und deren Behebung oder Umgehung einen beträchtlichen Aufwand verursacht.
Möglicherweise ist ein Grund darin zu sehen, daß vor Auslieferung eines neuen Release kein Feldversuch stattfindet, der in konkreten Anwendungen Fehler beseitigen hilft.

Stördienst: Für einen guten Projektablauf ist ein wirksamer Stördienst für die verwendete Systemsoftware von großer Wichtigkeit. Hier kann von sehr guten Erfahrungen mit dem ATM-Software-Stördienst gesprochen werden, der in der Lage ist, auf schnelle und unbürokratische Art und Weise Störungen zu beheben oder Umgehungen anzubieten.

4. PEARL-System DEA

Im folgenden sollen nun analog zum vorhergehenden Kapitel Aspekte der Arbeit mit dem PEARL-System DEA näher diskutiert werden.

Sprachumfang: Der Sprachumfang des verwendeten GPP-PEARL entspricht weitgehend Basic PEARL. Es sind jedoch folgende Konstrukte nicht vorhanden

- READ/WRITE (in der neuesten Version vorhanden; für ARES ohne Bedeutung)
- kein Character-Selektor
- Einschränkungen bei globalen Daten
- SIGNAL-Reaktionen nur auf Taskebene

Der fehlende Character-Selektor hatte zur Folge, daß für Stringhandling (dies ist der überwiegende Teil der DEA-Aktivitäten) ein Satz von 90 Assemblerprozeduren erstellt werden mußte. Die große Anzahl resultiert aus der strengen Typprüfung in PEARL, die hier eine effektive Lösung verhindert. Zudem fehlt in Basic-PEARL die Identitätsspezifikation, die elegante Problemlösungen durch Umgehen der Typprüfung erlaubt.

Einschränkungen existieren bei Verwendung des INV-Attributs in Spezifikationen globaler Daten; es darf nur angewendet werden, wenn auch die Deklaration mit INV erfolgte. Damit fehlt die Möglichkeit eines modulspezifischen Schreibschutzes für Variable.

Systemleistung: Gemessene Übersetzungszeiten führten zu folgenden Werten:

PDP 11 + Serie III	: 9-12	Statements/Minute
Intel 86/3xx mit DEA	: 12-16	" "
Intel 86/3xx	: 16-20	" "

Diese sehr schlechten Werte bedeuten für die Testphase eine wesentliche Beeinträchtigung; eine vollständige Übersetzung der DEA-Software dauert damit mehrere Tage.

Die Symboltabelle des PEARL-Compilers auf PDP 11 erlaubt maximal 300 Einträge. Dieser sehr kleine Wert erforderte eine übermäßige Aufspaltung aller geplanten Übersetzungseinheiten und verhinderte meist das Arbeiten mit Signalen in der Testphase. Auf Intel 86/3xx sind nach einer Änderung nunmehr 800 Einträge möglich, so daß einigermaßen erträgliche Testbedingungen herrschen.

Bezüglich Adressraum nutzt die Implementation die Möglichkeiten des Intel 8086 hinsichtlich Segmente und Register, so daß hier keine Schwierigkeiten aufgetreten sind.

Test: Für das PEARL-System DEA wurde von der Firma GPP ein sprachbezogenes Testsystem entwickelt, das in ARES zum Einsatz kam. Anders als das ATM-Testsystem, das mit Externspeicher arbeitet, müssen hier sämtliche Listen und Daten zusammen mit dem Testling gebunden werden, was für den Umfang Beschränkungen ergibt. Diese sowie Probleme der Generierung auf Intel 86/3xx ließen deshalb nur einen beschränkten Einsatz zu. Wie bei ATM 80-Rechner wurde wesentlich mit der eigenen Testumgebung gearbeitet.

Das PEARL-System mit den beiden Gastrechnern PDP 11 und Intel Serie III erlaubt zwar paralleles Compilieren auf PDP 11, die weiteren Bearbeitungsgänge sind aber im Single-User-Betrieb vorzunehmen, einschließlich der jeweiligen Transfers. Die Systeme

86/3xx sind reine Einplatzsysteme, bieten aber den Vorteil, auch als Zielmaschine dienen zu können.

Releases: Das PEARL-System wird gepflegt und weiterentwickelt. In größeren Abständen (ca. 1 Jahr) werden neue Versionen herausgegeben. Es sind hier ähnliche Aussagen zu machen wie bei ATM, jedoch waren die negativen Auswirkungen nicht immer so ausgeprägt. Insbesondere ist der Aufwand für die Generierung eines PEARL-Systems wesentlich kleiner als auf Seiten ATM 80.

Stördienst: Für das PEARL-System ist ein Stördienst eingerichtet, der gut funktioniert. Verbesserungen sind jedoch noch möglich in Bezug auf Reaktionszeit.

5. Schlußfolgerungen

In den beiden vorhergehenden Kapiteln wurde die Arbeit mit den beiden im Projekt verwendeten unterschiedlichen PEARL-Systemen anhand von für eine Entwicklung wichtigen Aspekte erläutert. Die hierbei gemachten Erfahrungen sollen nun zusammengefaßt werden; dabei können unter Einbeziehung von Anforderungen aus dem Projekt Schlußfolgerungen gezogen werden bezüglich Eignung, Einsatz und sinnvoller Erweiterungen.

Sprachumfang:

Der Vergleich der Arbeiten mit den beiden verschiedenen Sprachumfängen hat klar gezeigt, daß für Projekte der vorliegenden Größe und Komplexität ein komfortabler Sprachumfang wie der des ATM-PEARL sinnvoll und notwendig ist für eine effektive Programmierung. Insbesonders zu erwähnen ist die Identitätsspezifikation, die ein Umgehen der Typprüfung erlaubt und wesentlich für leistungsfähige Software ist. Die Beschränkung auf Basic-PEARL, das u.a. praktisch keine Textbearbeitung kennt, führt in Verbindung mit der starren Typprüfung zu grotesken Umgehungen und Assemblereinschüben. Generell fehlt in beiden Systemen die Möglichkeit von (auch rechnerübergreifender) Prozeßkommunikation, die in praktisch allen größeren Vorhaben notwendig ist. Dieser Mangel wurde durch eigene Entwicklung geeigneter Software-Moduln behoben.

Ein anderer Mangel in PEARL bezüglich E/A zeigte sich während der Entwicklung, nämglich das Fehlen von Anstoßanweisungen sowie keine Möglichkeit für kombinierte E/A, wie sie praktisch überall zu finden ist. Als Folge davon ergaben sich teilweise nicht sachgerechte Lösungen.

Zusammenfassend kann gesagt werden, daß Basic-PEARL für größere Projekte nicht die geeignete Programmiersprache ist. Die Verwendung eines Sprachumfangs, der große Teile von Full-PEARL umfaßt, ist dringend zu empfehlen. Darüberhinaus sollte in jedem Einzelfall geprüft werden, ob PEARL die geeignete Sprache ist.

Systemleistung

Für eine zügige Abwicklung größerer Projekte ist ein entsprechender Durchsatz an den Entwicklungssystemen zu fordern; ein Wert von 200 Statements pro Minute (über alles) ist anzustreben. Das ATM-System erreicht diese Leistung, während das DEA-System um eine Grössenordnung darunter liegt und damit unerträglich lange Durchlaufzeiten verursacht. Eine Leistungssteigerung um mindestens eine Größenordnung ist hier dringend notwendig.

Beiden Systemen gemeinsam sind Compilergrenzen bezüglich Umfang Symboltabelle und Anzahl Initialwerte, was für die Entwicklung u.U. gravierende Folgen haben kann. Mit ein wesentlicher Grund sind speicherresidente Tabellen und Listen, die nicht beliebig erweiterbar sind. Hier wäre eine Auslagerung auf einen Externspeicher angezeigt und erforderlich.

Bezüglich Adressraum bleibt nur zu erwähnen, daß mit der Einführung der neuen Maschinen ATM 80-16 mit Gate-Array-Technologie und Segmentadressierung eine weitere entscheidende Verbesserung der Situation zu erwarten ist.

Test

Beide PEARL-Systeme enthalten ein sprachbezogenes Testsystem mit einem umfangreichen Funktionssatz. Die Anwendung ist jedoch infolge von Rückwirkungen und Einschränkungen für den Testling mit Schwierigkeiten verbunden. Zu fordern ist hier eine praktisch rückwirkungsfreie Implementation, die den Zielrechner/Testling nicht mit Listen belastet. Die Identität von Gast- und Zielrechner muß gewahrt bleiben.

Nachteilig bei beiden Testsystemen ist die Tatsache, daß sie Einplatzsysteme darstellen, was in intensiven Testphasen zu Engpässen führt, die sich nur durch vermehrten Hardwareeinsatz vermeiden lassen. Mehrplatz-Testumgebungen bringen hier Vorteile, obwohl sie auch nicht ohne Probleme sind (Aufwand, Echtzeitverhalten, ...).

Unabhängig von obigen Überlegungen ersetzt ein Testsystem auf keinen Fall eine anwendungsspezifische

Testumgebung, deren Erstellungsaufwand sich durch systematisches und effektives Testen auszahlt.

Releases, Stördienst

Aufgrund der bisherigen Erfahrungen mit Releases ist zu empfehlen, vor Auslieferung einen Feldtest mit geeigneten Anwendungen vorzunehmen. Dies erscheint das angemessene Vorgehen zu sein, um Aufwand und Verzögerungen durch intensive Fehlersuche und Umgehung bei den Anwendern zu vermeiden.

In diesem Zusammenhang ist die Notwendigkeit einer exakten, funktionierenden Konfigurationskontrolle seitens der Hersteller besonders hervorzuheben. Die Vernachlässigung dieses Punktes führt zu sehr unangenehmen Störungen im Projektablauf und ist deshalb unbedingt zu vermeiden.

Schrifttum:

[1] Ammann, Martin: Entwicklung der Software für das Waffeneinsatzsystem ARES
PEARL-Tagung '83, Düsseldorf

Ammann Dr., Martin
Dornier System GmbH, Abt. ZITW
Postfach 1360
7990 Friedrichshafen
Tel. 07545/85472

Erfahrungen mit PEARL im wissenschaftlichen und industriellen Bereich

W. Freimann-v. Werder, Dr. A. Kantner, D. Hatting, K. Löhr, D. Müller, L. Pedron, Herne.

Zusammenfassung:

Die Vorteile von PEARL kommen dann zum Tragen, wenn Groß- oder Minirechner eingesetzt werden. Bisher fehlt jedoch die Möglichkeit, PEARL auch in jenen Bereichen einzusetzen, wo digitale Regler Anlageneigenschaften verbessern können, jedoch aus Preisgründen nur einfache Mikrocomputerregler möglich sind. Für die weitere Verbreitung von PEARL ist entscheidend, die Portabilität dieser Sprache von Minirechnern über PCs bis hin zu kleinen Mikrocomputern zu erweitern.

1. ONLINE-Simulation zur Optimierung der Regelung von Blockheizkraftwerken.

Blockheizkraftwerke (BHKW) sind kleinere dezentrale Anlagen der Kraft/Wärme-Kopplung mit höchstem Nutzungsgrad (80-90%). Sie bestehen aus mehreren Verbrennungsmotoren mit Wärmetauschern und Generator, einem Spitzenkessel und einem Wärmespeicher. Die Regelung hat dann Optimierungsfunktionen zu erfüllen, wenn der Bedarf an Wärme und Strom nicht der gleichzeitigen Produktion entspricht. Einer Entkopplung durch Speicher sind in finanzieller (Strom) oder baulicher Hinsicht (Gewicht von Wärmespeichern) in aller Regel Grenzen gesetzt.
Im Rahmen der Dissertation /1/ wurde versucht, durch eine Simulation des thermischen Teiles eines BHKWs und des damit beheizten Hauses auf einem Analogrechner die Regelung zu optimieren. Durch diesen Ansatz wird es möglich, auf dem Prozeßrechner Programme in einer Simulationsumgebung zu entwickeln und diese möglichst unverändert auf die reale Anlage anzuwenden. Die Schnittstellen zwischen der simulierten bzw. realen analogen Welt und dem Prozeßrechner sollten möglichst identisch sein. Neben einem fernsteuerbaren Analogrechner (EAI 180), der über integrierte Digitale Elemente verfügt (Gatter, Sample and Hold Verstärker, Analogschalter) wurde als Prozeßrechner ein HP 1000 F eingesetzt. Bild 1 zeigt den Hardwareaufbau der Simulation.

Nach einer Systemanalyse mit SADT /2/ wurden die Regelalgorithmen in PEARL implementiert. SADT war hilfreich, um die Problemstellung richtig zu verstehen, sowie zur Abgrenzung von Modulen und bei der Festlegung von Datenübergabeprozeduren. Die Darstellung von Parallelitäten und Synchronisationsmechanismen ist mit SADT nicht möglich. Der Programmumfang zur Durchführung der Simulation inklusive einer rechnerischen und graphischen Auswertung betrug ca. 1300 Anweisungen, wovon auf den eigentlichen Regler etwa 10 % entfielen. Für die grafische Darstellung wurden Teile von GRAPHIC-1000-II eingebunden, einem von HP vertriebenen Softwarepaket in FORTRAN. Das PEARL-Laufzeitsystem setzte während meiner Arbeit auf dem RTE-IV B von HP auf.

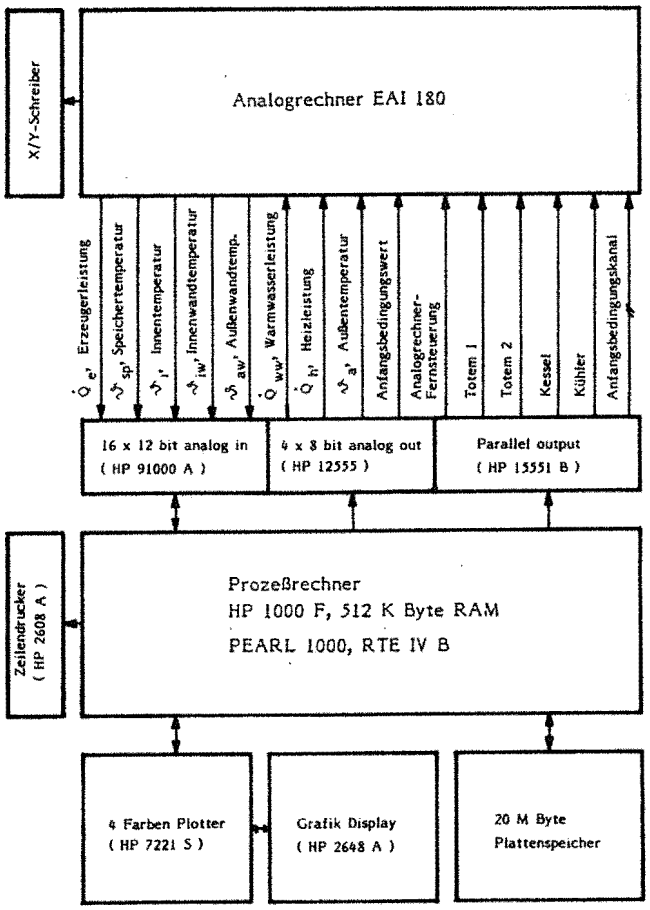


Bild 1 Hardwareaufbau der Simulation eines BHKWs

2. Erfahrungen mit PEARL-1000.

PEARL ist eine wunderbare Sprache, weil leicht zu erlernen, leicht zu handhaben und nahezu selbstdokumentierende Programme formulierbar sind. Die mächtigen Sprachkonstrukte brauchen hier nicht im Einzelnen gewürdigt zu werden. Für die Einarbeitung in das Programmieren mit PEARL-1000 waren 2 Monate erforderlich, wobei die Lektüre von /3/ nützlich war.

Auf der HP 1000 unter dem RTE-IV B war kein screenorientierter Editor verfügbar; auch der modernste HP-Editor für die A900 bietet nicht den Komfort, den man etwa von UNIX-Maschinen gewohnt ist. Die Übersetzungszeiten des PEARL-Compilers waren trotz der F-Maschine zu lang, die deutschen Fehlermeldungen nach kurzer Gewöhnung hilfreich. Im Bereich der Operatoren-Behandlung wurden Fehler im Compiler festgestellt.

Für die automatische Überprüfung von Schnittstellen wird der PUBS eingesetzt. Auch hier sind Ausführungszeiten zu bemängeln. Die Schnittstellenüberprüfung sollte abschaltbar sein, um Änderungen, die innerhalb eines Moduls erforderlich sind, schneller auf die Maschine zu bringen. Der Link-Vorgang ist hinsichtlich der Ausführungsdauer ärgerlich lange: Pro Task 1 bis 2 Minuten. Weiter ist zu bemängeln, daß unter RTE-IV B keine reentranten Bibliotheken angelegt werden, die z.B. das Benutzen eines Formatters von mehreren Prozessen erlauben. Dadurch werden der Umfang der Tasks und die Linkzeiten unnötig erhöht. In diesem Zusammenhang ist besonders ärgerlich, daß unter RTE-IV B ein Prozeß maximal 32 K-Worte groß sein darf. Im Falle des Simulationsprogrammes mußten deshalb an sich sequenzielle Programmteile in mehrere Tasks aufgeteilt werden.

Bei der Kopplung von Analog- und Prozeßrechner wird man geneigt sein, die Simulationszeit möglichst schnell ablaufen zu lassen, damit der Analogrechner geringe Fehler integriert. Dies war mit PERAL-1000 nicht in dem wünschenswerten Maße möglich. Bei ohne Kompromisse an die Rechenzeit mit sauberer Synchronisation und Zugriffsschutz über BOLT-Variable formulierten Programmen zeigte sich in dieser Anwendung, daß die kleinste Reglerabtastzeit in der Größenordnung 10^{-1} sec liegt; für die reale Prozeßsteuerung eines BHKWs mit Sicherheit keine relevante Einschränkung. Allerdings konnte die Zeitraffung nur 120fach sein, wodurch die Simulation der Abläufe eines Tages sich immerhin über 12 Minuten erstreckte.

3. Beispiel einer industriellen Anlagensteuerung.

Die Firma Happel stellt Anlagen für die Heizungs- Lüftungs- und Klimatechnik her. Als Beispiel für eine solche Anlage soll hier eine Heizungszentrale für Ein- und Zweifamilienhäuser herausgegriffen werden, die aus einem öl- oder gasbefeuerten Kessel und einer zweistufigen Wärmepumpe besteht (Bild 2). Aufgabe der Regelung ist es, unter den gegebenen Randbedingungen die zur Brauchwasserbereitung und zum Heizen günstigsten Erzeu-

gereinheiten auszuwählen und unter Beachtung bestimmter Bedingungen zusammen mit Umwälzpumpen, Lüftern und Ventilen zu steuern und regeln. Dies ist eine Aufgabenstellung, die sowohl das Erfassen von Meßgrößen, die adaptierende, optimierende Regelung mehrerer

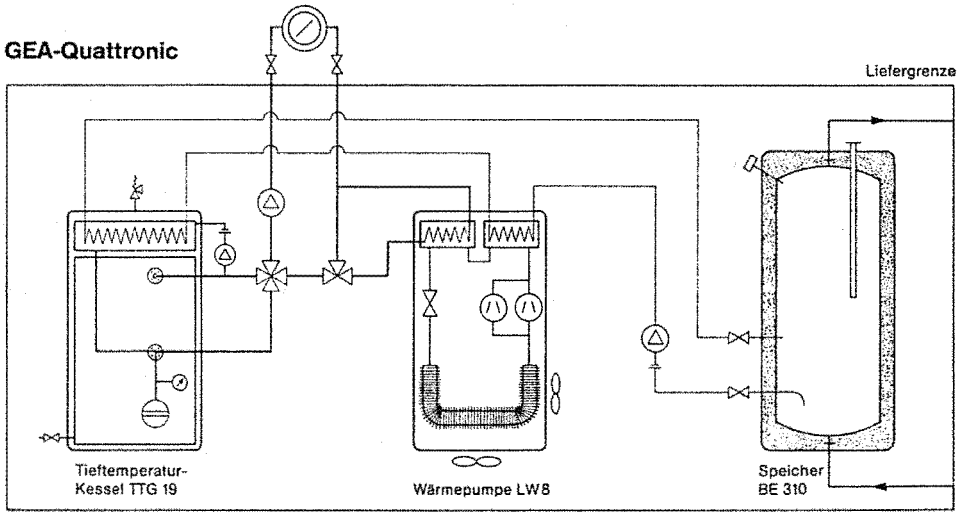


Bild 2 Prinzipschaltbild Heizzentrale GEA-Quattronic

Kreise, die Ablaufsteuerung mit bestimmten Sicherungsmechanismen, die Ansteuerung von Aggregaten und nicht zuletzt eine leichte Bedienbarkeit der Anlage umfaßt. Zusätzlich bestehen hohe Anforderungen an die Zuverlässigkeit der Regelung, die ihre Funktionen auch nach mehrstündiger Versorgungsspannungsunterbrechung sicher automatisch wieder aufnehmen muß. Für die Regelung dieser oder vergleichbarer Anlagen wird vom Markt ein Preis von etwa 1000 DM akzeptiert. Die Herstellungskosten müssen entsprechend niedriger sein.

Seit 1983 wird bei Happel intensiv, zum Teil gefördert mit Bundesmitteln, an der Entwicklung digitaler Regler gearbeitet. Erst durch Ablösen der tradierten analogen Regler wurde es möglich, die oben aufgelisteten Aufgabenstellungen befriedigend zu lösen. Im ersten Ansatz haben wir eine auf dem Z80 basierende Regelung realisiert, die ausschließlich in Assembler programmiert ist, wobei ein selbst geschriebenes einfaches Multitasking-Betriebssystem unterlegt wurde. Der Umfang des Problems soll dadurch annähernd charakterisiert werden, daß 24 K Bytes Programmcode zusammengeschrieben wurden.

Auf den damit gewonnenen Erfahrungen aufbauend wurde folgende Spezifikation zukünftiger Regler erarbeitet:

- Verwendung einer standardisierten Hardware für verschiedene Anlagen,
- Reduktion der Software-Entwicklungszeit,
- weitestgehender Einsatz von Hochsprachen,
- Anwendung eines Multitasking-Betriebssystemes,
- Klarschrift-Bedienoberfläche, für verschiedene Anlagen und Sprachen per Software anpaßbar,
- Menü-Bedienung über wenige Tasten,
- Gliederung der Regelung in 3 Funktionsgruppen: Bedienung/Regelung, Prozeßinterfaces und Netzteil,
- Kopplung der Funktionsgruppen über einen seriellen Bus,
- Verlagerung der Assembler-Programmteile in ein mit einem Microcontroller bestücktes Interface zur Meßdatenerfassung und Ansteuerung von Aggregaten.

4. Eine Sprache für drei Rechner unterschiedlicher Leistungsfähigkeit.

Durch Einführen der Interfaces und eines standardisierten Datenübertragungsbusses wird

es möglich, herkömmliche Rechner, insbesondere PCs ohne Prozeßperipherie für die Anlagenregelung einzusetzen. Dies ist für die Entwicklung der Programme im Labor nützlich, besteht doch die Möglichkeit, durch Simulation der Interfacekarten mit anderen PCs eine Überprüfung und Validierung der Reglerprogramme ohne reale Anlage mit realistischen oder auch extremen Eingangsdaten vorzunehmen. Diese sonst für allgemeine Datenverarbeitung ausgerichteten Computer, die z.B. unter UNIX oder MS-DOS arbeiten, müssen Multitasking-Eigenschaften bekommen; hinsichtlich der Reaktionsschnelligkeit sind unsere Anforderungen gering, da die Zeitkonstanten der thermischen Prozesse im Minuten oder Stundenbereich liegen.
An die Programmentwicklung schließt sich in aller Regel eine Eprobungsphase an, bei der an den eingesetzten Rechner höhere Anforderungen gestellt werden, als an die Serienhardware. Für diese in der Industrie notwendige Feldtestphase ist zweierlei erforderlich:

- Die für die Regelung wichtigen Zustandsgrößen müssen zusammen mit denjenigen, die für die Beurteilung der Anlage wichtig sind, auf einem Datenträger protokolliert werden. Es ist mit einer sehr großen Datenflut zu rechnen, die nur maschinell auswertbar ist, wobei die Daten nach bestimmten Ereignissen absuchbar sein müssen (Datenbankproblematik).

- Als Folge des Feldtests ist mit Änderungen am Regelprogramm zu rechnen, die leicht und vor Ort möglich sein sollten.

Der letzte Schritt besteht dann darin, die auf dem PC verbesserten Programme automatisch (deswegen fehlerfrei) auf die wegen der finanziellen Beschränkungen sehr einfachen Zielsysteme (Bedienung/Regler) herunterzubringen. Eine nochmalige Felderprobung sollte überflüssig sein.
Es sollten somit 3 Rechnersysteme verfügbar sein, die alle fähig sein müssen, über den seriellen Datenbus mit den Interfacekarten zu kommunizieren:

1. Komfortable, Multi-User- und Multi-Tasking-Maschine zur Systemanalyse, Dokumentation, Programmentwurf und -implementierung; Feldtestauswertung. Graphische Ausgabeeinheiten.

2. Feldtestrechner mit Massenspeicher und begrenzter Programmvariationsmöglichkeit (schlichter Editor, Compiler). Graphikfähiges Terminal.

3. Mikrocomputersystem als Serienhardware, nur mit dem Notwendigsten ausgestattet.

In der ersten Hälfte 1985 haben wir auf dem Markt nach einem Softwarewerkzeug gesucht, das unseren Bedürfnissen entspricht. Obwohl PEARL vom Ansatz her geeignet wäre, fehlt die Möglichkeit, bis hinunter auf die einfachen Mirkocomputer zu kommen. Selbst für die üblichen PCs ist kein PEARL-Sstem verfügbar, obwohl sie mittlerweile die Leistungsfähigkeit einer HP 1000 F erreicht haben. Ich möchte durch diesen Beitrag entsprechende Entwicklungen anregen.

Literatur:

/1/ W. Freimann-v. Werder:
Beitrag zur Einsatzoptimierung von Blockheizkraftwerken,
Deutsche Dissertation, Berlin 1985.

/2/ Scientific Control Systems GmbH:
Analyse mit SADT, Hamburg 1982.

/3/ Brinkkötter, Nagel, Nebel, Rebensburg:
Systematisches Programmieren mit PEARL,
Wiesbaden 1982.

Anschrift:

Abteilung LTRE
Happel GmbH & Co.
Südstraße
4690 Herne 2
Tel 02325 468 321

Der Einsatz von PEARL für eine Echtzeitsimulation auf einem Prozessrechner in Verbindung mit einem Hybridrechner

Dipl.Ing. Claudia Schmidt

1 Aufgabenstellung

1.1 Allgemeine Beschreibung des Problems

Das Projekt beschäftigt sich mit der Entwicklung neuer Antriebsregelkonzepte für Personenaufzüge. Dabei steht neben der Verbesserung von Komfort und Einfahrgenauigkeit die Realisierung einer digitalen Antriebsregelung zur Aufgabe. Der Vorteil einer digitalen Lösung gegenüber einer herkömmlichen liegt in der Zusammenfassung von Antriebsregelung und Ablaufsteuerung, die heute schon weit verbreitet mit Hilfe der Mikroprozessortechnik erfolgt.

Im Zuge einer digitalen Reglerimplementierung sollen gleichzeitig Parameterschwankungen der Aufzugsstrecke (Eigenfrequenzen,Dämpfung) und andere Einflüsse berücksichtigt werden, was bei einer analogen Ausführung nur schwer möglich ist.

Einen Teil des Projektes bildet die Simulation unterschiedlicher Antriebsregelungen. Dabei sollen im Wesentlichen folgende Punkte untersucht werden:

- Einflüsse der in der Regelung nicht berücksichtigten Modellabschnitte,
- Einflüsse der sich während des Betriebes verändernden Parameter (Last,Wegänderung),
- Störungen (Bremse,Schienenstöße),
- Anforderungen an den elektrischen Antriebskreis,
- Anforderungen an den Prozessor,die Schnittstelle und die Messerfassungssysteme.

Bild 1 zeigt eine Übersicht über das physikalische Modell einer geregelten Aufzugsanlage. Hierbei ist die Sicherheitskontrolle nicht enthalten.

1.2 Simulationsmethoden

Bei der Wahl des Simulationskonzeptes wurde darauf geachtet, nahe der Realisierung zu liegen, um eine hohe Sicherheit in der Voraussage des Verhaltens der Anlage zu erzielen. Es wurde eine Echtzeitsimulation gewählt, die den Vorteil

- der Eingriffsmöglichkeit während des Prozesses,
- der Aussage über Laufzeiten der Prozeßglieder,
- der Nachbildung paralleler Prozesse bei der Aufzugsstrecke,
- der Einbeziehung der Schnittstellen, die bei der Realisierung erforderlich sind (A/D - D/A - Wandlung)

beinhaltet.

Die Nachbildung des Streckenmodells wurde als kontinuierliches, dynamisches System auf einem Hybridrechner ausgeführt. Dieser Rechner vereinigt analoge und digitale Rechenelemente, so daß eine Berücksichtigung nicht linearer Elemente (Reibung,Bremsverhalten) möglich ist. Der Motion Control - Teil (Sollwertbildung,Regelung) wurde in einen Prozessrechner verlegt. Zum Test der Abtastzeiten hilft die Verwendung der Zeitraffung und -dehnung, die beim Hybridrechner möglich ist. Die Grenze dieser Zeitmanipulation wird durch die Offsetdrift der Integrierer bei langen Prozeßlaufzeiten gesetzt.

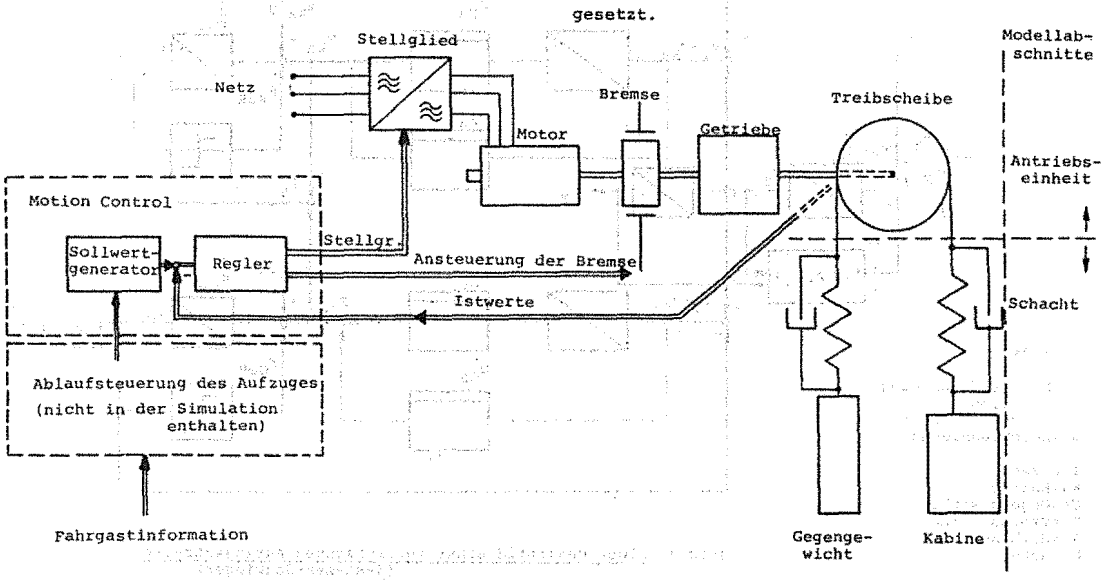


Bild 1 Übersicht des physikalischen Modells

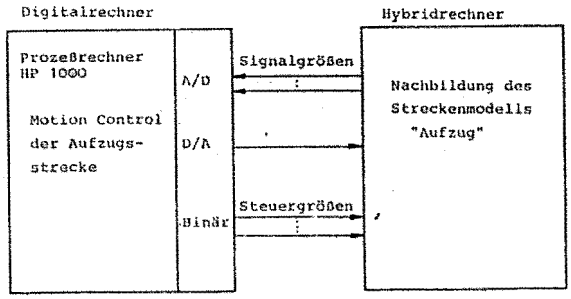


Bild 2 Simulationsaufbau

2 Realisierung der Simulation

2.1 Das Streckenmodell auf dem Hybridrechner

Das Modell des Aufzuges wird unterteilt in einen elektrischen und einen mechanischen Teil. Der mechanische Teil ist ein gekoppeltes Mehrmassensystem,der je nach Ausführung und Vereinfachung von einem Differentialgleichungssystem 6.Ordnung beschrieben werden kann (Antriebseinheit - Kabine - Gegengewicht). Der elektrische Teil wird als drehmomentgeregelter "schwarzer Kasten" angesehen und als VZ1-Glied 1. Ordnung (plus Totzeit) realisiert.
Nichtlinearitäten und Störgrößen (Reibungs- und Bremsverhalten) können mit Hilfe von Funktionsgebern nachgebildet werden. Es besteht auch die Möglichkeit über den Prozeßrechner Kennlinien von Störgrößen zu realisieren. Bei dem jetzigen Stand der Untersuchung wurde jedoch nur von der Sign-Funktion (Haftreibung) Gebrauch gemacht. Bild 3 zeigt das auf dem Hybridrechner realisierte Blockschaltbild wie es aus dem Bereich der Regelungstechnik üblich ist.

2.2 Reglerimplementierung auf dem Prozeßrechner

2.2.1 Reglerrealisierung

Auf dem Prozeßrechner erfolgt die Sollwertberechnung, Regelung und Protokollierung (mit anschließender Auswertung). Dabei wurden alle 3 Funktionen als eigenständige Programmteile gesehen, die miteinander kommunizieren. Das hat den Vorteil, daß bei Änderung des Regelkonzeptes nur der Reglerteil berührt ist, Sollwertberechnung und Messauswertung aber erhalten bleiben.

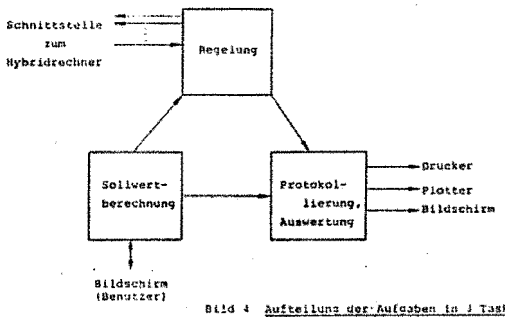


Bild 4 Aufteilung der Aufgaben in 3 Tasks

Bei der Auslegung des Reglers geht man von einer diskreten Strecke aus, die in Abhängigkeit der Abtastzeit und physikalischen Parameter ihre Pole ändert.
Folgende Anforderungen an die Regelung sind zu berücksichtigen:

- ein zeitoptimales Führungsverhalten,
- daß die für den menschlichen Körper spürbaren Grenzwerte der Beschleunigung und der Beschleunigungsänderung auch bei betriebsnormalen Störungen (Bremse,Haftreibung) nicht überschritten werden,
- Vermeidung eines Überschwingen des Wegistwertes,
- daß die Stellgröße beschränkt ist,
- eine hohe Positionsgenauigkeit,
- geringer Aufwand bei der Messerfassung.

Der Einfachheit halber wurde bei der Auslegung von einer Strecke 3. Ordnung ausgegangen, d.h. die Treibscheibe ist mit dem Gegengewicht und der Kabine starr gekoppelt. Das hat nur dann seine Gültigkeit, wenn die Anregung des

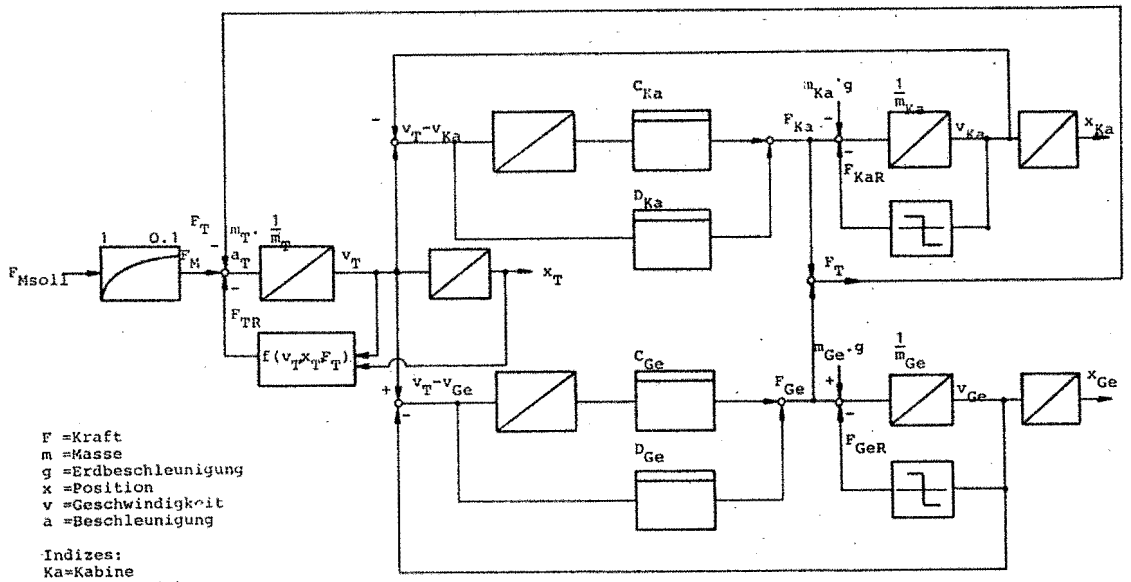


Bild 3 Blockschaltbild einer vereinfachten Aufzugsstrecke (3-Massen-Schwinger)

Systems unter der Eigenfrequenz des Schachtes bleibt. Hierbei wird gerade der letzte Punkt der Anforderungen an die Regelung erfüllt.

Die Auslegung der Regelung geschieht auf Grundlage verschiedener Regelverfahren:

Kaskadenregelung (Nachteil: alle 3 Zustände - x,v,a - müssen erfaßt werden)

Kompensationsregler für den Wegregelkreis (Nachteil: kein Einfluß auf andere Zustände,z.B. v,a)

Zustandsregler (ausbaufähig, jedoch großer Rechenaufwand)

Auf die Regelverfahren wird hier nicht weiter eingegangen. Es wird nur ein Beispiel genannt, um die anschließende Problematik mit der Programmiersprache PEARL besser zu verdeutlichen.

Zu Testzwecken wurde zunächst ein Kaskadenregler aufgebaut, wie er auch aus der kontinuierlichen Regelungstechnik bekannt ist.

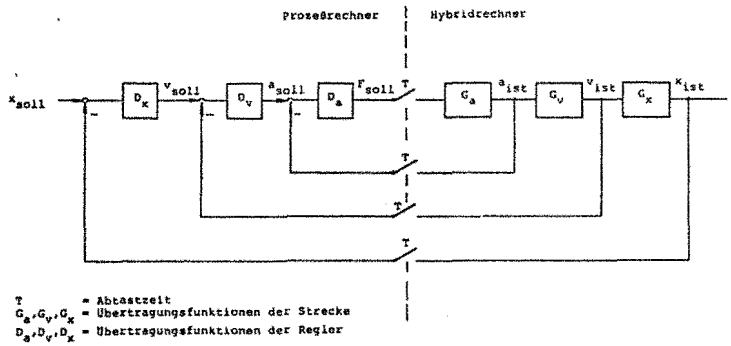


Bild 5 Vereinfachtes Blockschaltbild der Kaskadenregelung

Der gesamte Regelalgorithmus für die einzelnen Regelkreise wurde sequentiell im Rhythmus der Taktzeit T abgearbeitet (Bild 6).

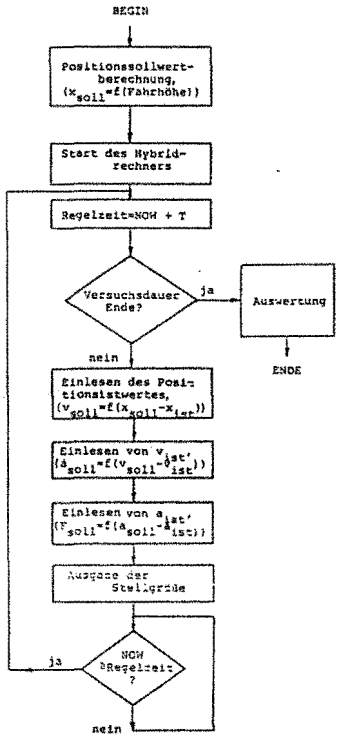


Bild 6 Übersicht der Datenfluß-Software

Alternativen zur Realisierung des Kaskadenreglers

Gedacht war auch an eine Kaskadenregelung mit unterschiedlichen Abtastzeiten für die einzelnen Regelkreise. Da der Beschleunigungsregelkreis sehr schnell auf Laständerungen an der Treibscheibe reagieren muß,ist es sinnvoll diesen Regelkreis erheblich schneller zu gestalten als den Drehzahlregelkreis.
Wählt man die Abtastzeit des äußeren Regelkreises um ein Vielfaches des inneren, so darf die Rechenzeit des gesamten Reglers die Abtastzeit des inneren Kreises nicht überschreiten. Das Problem ist einfach mit einer sequentiellen Programmgestaltung zu lösen. Soll der Beschleunigungsregler immer im Eingriff sein (Rechenzeit des Beschleunigungsregelkreises= Taktzeit des inneren Reglers) und nur von der Überlagerten Regelung unterbrochen werden, so könnte theoretisch das Reglerprogramm mit Hilfe zweier Tasks und Semaphoren gestaltet werden.

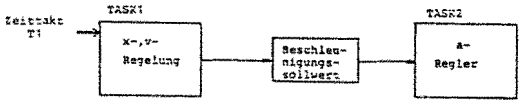


Bild 7 Aufteilung der Regelung in 2 Tasks

Die Starttask(Task 1 = Geschwindigkeits- und Positionsregler) aktiviert Task2 (Beschleunigungsregler). Beide Tasks greifen auf den Beschleunigungssollwert zurück. Die Task2 wartet bis die Semaphore "Sollwert_a" frei ist. In der Starttask wird der Sollwert der Beschleunigung berechnet und die Semaphore frei gegeben. Task2 kann nun die Stellgröße berechnen.
Da es sich bei der Task2 um eine Endlosschleife handelt, in der keine Wartezustände auftreten (bei analoger Ein-/Ausgabe wird direkt das entsprechende Interface angesprochen), besteht (bei Tasks gleicher Priorität) für das PEARL-System keinen Anlaß zum Taskwechsel. Somit wird die Starttask nie wieder aktiviert.

Programmbeispiel:

```
        :
        :
DECLARE
     :
     Sollwert_a    SEMA PRESET(1)
     :
START_Regelung : TASK;
     :
   ACTIVATE TASK2;
   WHILE Versuchsdauer GT Dauer
   REPEAT
     :
     REQUEST Sollwert_a;
     Beschleunigungssollwert:= .....
     RELEASE Sollwert_a;
     :
     AFTER T1 RESUME;
     Dauer := NOW-Versuchsbeginn;
   END;
   TERMINATE TASK2;
```

```
TASK2 : TASK;
DECLARE
     :
     laufende Berechnung BIT(1)
                    INITIAL('1'B);
WHILE laufende_Berechnung
REPEAT
  :
  REQUEST Sollwert_a;
  Stellgröße := ........
  RELEASE Sollwert_a;
  :
  SEND Roh_Stellgröße TO
          Stellgrößenausgang;
END;
```

Das Problem wird gelöst, indem man die Tasks unterschiedlich priorisiert. So erhält die Starttask eine höhere Priorität als Task2. Sobald die Wartezeit in der Starttask abgelaufen ist, wird der Prozeß in der Task2 unterbrochen. Der Positions- und Geschwindig-

keitsregelalgorithmus berechnet den neuen Beschleunigungssollwert. In der verbleibenden Zeit (T1 - Rechenzeit des überlagerten Reglers) arbeitet die Task2.

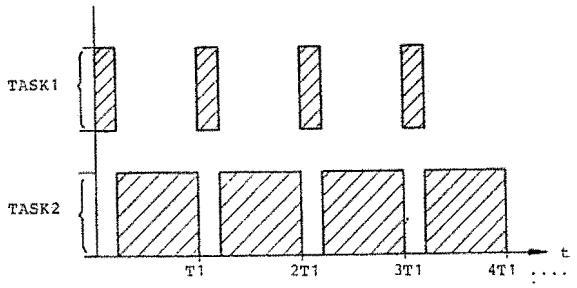


Bild 8 Zeitschema der Taskbearbeitung

Infolge des hohen Verwaltungsaufwandes kann bei diesem Betriebssystem ein Taskwechsel nur alle 2o ms stattfinden. Damit die reale Abtastzeit nicht variiert, wird T1 als ein Vielfaches von 2o ms gewählt. Nachteilig ist ebenfalls die mangelnde Vorausberechenbarkeit der Abtastzeit des inneren Regelkreises (= Rechenzeit der Task2).

2.2.2 Protokollierung und Auswertung

Sinnvoll ist es neben dem Programm die Protokollierung und Auswertung vorzunehemen. Dabei wird die Auswertung mit niedriger Priorität während der Wartezeit der Reglertask durchgeführt.
Zur Zeit erfolgt die Auswertung nach der simulierten Fahrt. Im Anschluß einer Fahrt werden die Messwerte geplottet. Dazu benötigt man Standard-Grafiksoftware der Fa. HP. Hierbei wird von der Möglichkeit der Einbettung externer Routinen (mit Schnittstellenprüfung) in das vorhandene PEARL-System Gebrauch gemacht.

3 Bechreibung der Schnittstellen

Die Schnittstelle des Prozeßrechners HP1000 besteht aus einer D/A-Karte mit 4 Analogausgängen á 8 Bit (negative Spannungen nicht möglich), 2 A/D-Karten mit je 12 Bit Analogeingängen und einer Binärschnittstelle (DAS) mit 16 Bit Ausgängen. Über die Binärschnittstelle wird der Hybridrechner gesteuert und das Vorzeichen der Stellgröße ausgegeben. Zusätzlich können auch Steuersignale für einen am Hybridrechner angeschlossenen X-Y-Schreiber übertragen werden.

4 Verwendung der Programmiersprache PEARL bei der Realisierung der Aufgaben auf der HP1000

4.1 Anforderungen an die Programmiersprache

Infolge der Aufgabenstellung, d.h. laufende Änderung der Regelalgorithmen, durfte die Programmiersprache keine aufwendige Programmarbeit erfordern, wie es bei Assemblersprachen der Fall ist.
Die Kommunikation mit dem Hybridrechner und eventuelle Änderungen der Ein- und Ausgangsbelegung sollte einfach sein. Die Echtzeitprogrammierung sollte ein nachvollziehbares (und berechenbares) Timescheduling haben, damit Schlüsse für die Anforderungen an ein später einzusetzendes Mikroprozessorsystem gezogen werden können.

4.2 Voraussetzung für die Benutzung der Sprache PEARL

Der Autor besaß Programmierkenntnisse in den Sprachen Pascal, FORTRAN (Großrechenanlagen) und der Assemblersprache für den Motorola 6802. Bezüglich einer Echtzeit-Prozeßsprache war hier echtes Neuland zu betreten. Vorteilhaft war, daß vom gleichen Institut auf ähnliche Weise vor diesem Projekt eine Simulation einer Regelung eines Blockheizkraftwerkes durchgeführt wurde. Die Erfahrungen, die hierbei mit PEARL und dem Prozeßrechner in Verbindung mit dem Hybridrechner gemacht wurden, konnten jedoch nur teilweise übernommen werden, da das Herstellerbetriebssystem der HP1000 vor Beginn dieses Projektes geändert wurde und der Betrieb sich darauf erst wieder stabilisieren mußte.

4.3 Vorteile und Nachteile von PEARL als Echtzeit-Programmiersprache

Die zur Verfügung stehende Literatur suggerierte die perfekte Erfüllung der Anforderungen an die Programmiersprache durch PEARL. Mängel stellten sich erst bei der Benutzung heraus.

4.3.1 Vorteile

Bestechend ist die einfache Handhabung der Sprachkonstrukte in PEARL, die sich bei der Programmierung positiv auswirkt. Bekannte Elemente von Hochsprachen (besonders von Pascal) mischen sich mit der logisch nachvollziehbaren prozeßeigenen Sprachwelt.

- Schnittstellenbehandlung
  Die Kommunikation der I/O-Schnittstellen ist einfach. Physikalische I/O-Änderungen werden nur im Systemteil des Programms durchgeführt. (Zeitersparnis und Gewinn an Übersicht).
- Programmaufbau
  Der strukturierte Programmaufbau und die fast unbegrenzte Namensgebung für Variable und Schnittstellen ist selbstdokumentierend und macht Programme auch nach längerer Zeit nachvollziehbar.
- Modultechnik
  Dieses Hilfsmittel ermöglicht die Einzelerprobung der Programme. Zusätzlich kann man mit wenigen Änderungen Programmteile für unterschiedliche Programmabläufe verwenden.
- Echtzeitsynchronisation
  Vernachlässigt man die Schnittstellenlaufzeit, so können Ereignisse auf einfache Art (fast) zeitgenau gesteuert oder registriert werden.
- Die Fehlermeldungen des Compilers sind ausreichend, man muß jedoch unnötige Warnungen von wichtigen unterscheiden lernen.
- Die Verwendung des Hersteller-Standardpaketes für Grafik durch Einbindung externer Routinen in PEARL stellte eine zwingend notwendige Erweiterung des Sprachumfanges dar.

4.3.2 Nachteile

Für den Benutzer suggeriert PEARL Großrechenkomfort. In Wirklichkeit ist PEARL auf einem Kleinrechner implementiert, dessen Betriebssystem den gewünschten Komfort nicht bieten kann.

- Sprachumfang
  Der Prozeßrechner wird im Einprozessor-Multitasking-

Betrieb gefahren, was bei einer Echtzeitprogrammierung zu erheblichen Nachteilen führt.Bei der Abarbeitung z.B. zweier gleichpriorisierter Prozesse muß der eine erst beendet sein,bevor der andere beginnen kann. Das kann die Zeitsynchronisation mit dem Hybridrechner zerstören (s.Bsp.Kaskadenregelung).
Für die Auslegung des Reglers ist eine Abschätzung der Rechenzeit zur Compile-Zeit notwendig, z.B. per Sprachkonstrukte. Es gibt bei PEARL im Gegensatz zu den Assemblersprachen nicht einmal eine Zeitangabe für die Dauer der Operationen je Herstellerimplementierung. Man muß sich mit der Abtastzeit regelrecht "herantasten". Änderungen der Regelalgorithmen können den gefundenen Rechenzeitwert wieder zunichte machen.

Warum gibt es bei PEARL keine Sprachelemente für die Grafik?
Die selbsterstellte Grafik sowie die Userroutine für die Rechnersystemzeit sind in anderen Sprachen realisiert und müssen beim Bindevorgang explizit angegeben werden.

- Laufzeit-Umgebung
  Verwirrend ist, daß es mehrere Ebenen gibt, aus denen Fehlermeldungen dem Benutzer erscheinen können.
  1) Fehlersignale werden vom PEARL-Laufzeitsystem angezeigt(magere Ausstattung).
  2) Fehlermeldungen können zusätzlich vom Hersteller-Betriebssystem kommen(Resourcenverwaltung).

  Ein Monitoring der PEARL-Laufzeit wird vermißt. Es wird nicht angezeigt, ob ein Prozeß soeben gelaufen ist und wie oft er im vergangenen Zeitraum gelaufen oder eingeplant ist.
  Unklarheiten bestanden bei der Analyse, warum Prozesse "abgestürzt" sind (mangelhafte Fehlermeldung).

- Programmentwicklungsumgebung
  Sehr Zeitaufwendig ist der Binde-Vorgang , besonders bei Verwendung von mehreren Moduln. Sehr nützlich wurde die Verwendung von Makros empfunden, die jedoch die gleiche Benennung der Objekt- und Bindefiles voraussetzt.

- Notwendig ist auch die Information des Zeitverlustes bei Taskwechsel. Laufen ineinander geschachtelte Programmteile mit unterschiedlichen Taktzeiten (s.Kaskadenregelung mit unterschiedlicher Abtastung für den Beschleunigungs- und Wegregelkreis), so darf der Taskwechsel die zeitgenaue Abarbeitung der Reglertasks nicht zerstören. Beim Multitasking-Betrieb hängt die Zeit des Taskwechsels noch zusätzlich von der Belastung des Rechners und der gesetzten Priorität der Task ab.

5 Ausblick

Der Autor hat sich bei seiner Problemlösung für die Methode der Echtzeit-Simulation entschieden. Hierbei wurde PEARL als Echtzeit-Simulationssprache eingesetzt. Sämtliche Problemstellungen können in PEARL vorteilhaft und zeitsparend implementiert werden.
Kritisiert wird, daß in PEARL keine Echtzeitzusicherungen vorhanden (Assertions) sind
Kritisiert wird, daß in PEARL keine Echtzeitzusicherungen (Assertions) vorhanden sind, mit denen Echtzeitanforderungen durch den Compiler überprüfbar wären.
Mangelnde Fehlermeldungen machten eine intensive Betreuung seitens der Spezialisten des Prozeßrechenzentrums notwendig. Besonders bei Laufzeit-Fehlersignalen ist eine stärkere Differenzierung der Fehlerquellen notwendig.
Eine Simulation zieht eine ausreichende Dokumentation nach sich. Deshalb ist es zwingend, daß die verwendete Simulationssprache Sprachkonstrukte für eine Grafik zur Verfügung stellt.

Literaturverzeichnis

Systematisches Programmieren mit PEARL
Herbert Brinkkötter,Klaus Nagel
Herbert Nebel und Klaus Rebensburg
Studien-Text, Ak.Verlagsges.Wiesbaden

Regelungstechnik I
Unbehauen
Vieweg Braunschweig/Wiesbaden

Abtastregelung
J.Ackermann
Springer-Verlag 1983

Concurrently Executable Modules

Ein einheitlicher Ansatz der Systementwicklung

Dr. W.K. Epple *, H. Spandl
Universität Karlsruhe

Zusammenfassung

Dieser Beitrag stellt einen einheitlichen Ansatz zur Software Entwicklung für verteilbare Systeme vor. Dabei wird versucht, die Konzepte der Datenabstraktion zu erweitern und für die Spezifikation, den Entwurf und die Implementierung von verteilbaren Systemen nutzbar zu machen.

Zunächst wird das sogenannte Π – **Paradigm** (sprich: Pi – Paradigm) vorgestellt. Dieses besteht aus einer **Methode**, einer **Sprachfamilie** mit den darauf abgestimmten **Werkzeugen** sowie einem gemeinsamen (Modellierungs–) **Konzept**: den *C*oncurrently *E*xecutable *M*odules, kurz *CEM*s genannt. Zum Schluß wird anhand eines kleinen Beispiels die Anwendung der Methode und des CEM Konzeptes aufgezeigt. Dieses Beispiel veranschaulicht, wie die Techniken der Datenabstraktion erfolgreich zur Entwicklung von verteilten Systemen eingesetzt werden können.

Schlüsselwörter: Datenabstraktion, Software Engineering, Modulares Programmieren, Parallelität, Verteilbare Systeme, Verteilte Systeme

Abstract

This paper presents the concept of a uniform approach to software development of distributed systems. The aim is to extend and apply what has been learned about the power of data abstraction to the specification, design and implementation of distributed systems.

First the socalled Π – **Paradigm** (pronounce: P – Paradigm) will be described. This paradigm consists of a **method**, a set of adopted **languages** and **tools**, and a common (modeling) **concept**: a Concurrently Executable Module (CEM). The paper finally shows the application of the method and the CEM concept to an example. This demonstrates how data abstraction techniques can be successfully used for the development of concurrent systems.

Key Words: data abstraction, software engineering, modular programming, concurrency, distributable systems, distributed systems

1. Einleitung

Über die letzten Jahre konnte ein ansteigender Bedarf für verteilte (Rechner–, Software–) Systeme beobachtet werden. Dieser Bedarf wurzelt in der Tatsache, daß der Mensch oftmals nicht mehr in der Lage ist, mit der zunehmenden Komplexität der heutigen Systeme Schritt zu halten. Hinzu kommt, daß in einer technischen Umgebung ein hohes Maß an

- Sicherheit
- Zuverlässigkeit und
- Flexibilität

gefordert wird. Zudem erfordert der Mangel an ausgebildetem und erfahrenem Personal in der Softwareindustrie die Entwicklung von wiederverwendbaren Software Komponeneten.

Aufgrund dieser Anforderungen entstand ein großer Forschungsschwerpunkt um den Software – Entwicklungsprozeß zu unterstützen und zu standarisieren. Das Ziel all dieser Forschungsaktivitäten ist es, das Produkt Software in kürzerer Zeit und mit höherer Qualität zur Verfügung zu stellen. Obwohl heutzutage bereits eine Vielzahl von Werkzeuge für die Entwicklung von kommerzieller Datenverarbeitungssoftware existiert, insbesondere für die Design–, die Implementierungs– und die Testphase, besteht immer noch ein großer Bedarf an Methoden und Werkzeugen, die alle Phasen der Entwicklung von integrierten und verteilbaren Computersystemen abdecken. In [11] wird berichtet, daß je weiter die Phase, die ein Werkzeug unterstützt, von der Kodierungs– und Testphase entfernt ist, desto unwahrscheinlicher wird es, daß es überhaupt Anwendung findet. Für die Phase der Anforderungsdefinition wird in der Praxis gegenwärtig quasi kein Hilfsmittel verwendet.

Dieser Beitrag stellt einen einheitlichen Ansatz der Softwareeinwicklung für verteilbare Systeme vor, indem er versucht, die Konzepte der Datenabstraktion zu erweitern und für die Spezifikation, den Entwurf und die Implementierung von verteilten Systemen nutzbar zu machen.

Grundlage ist das sogenannte Π – **Paradigm**, bestehend aus einer **Methode**, einer **Sprachfamilie** und dazugehörigen **Werkzeugen** sowie einem gemeinsamen (Modellierungs–) **Konzept**: Gleichzeitig Ausführbaren Modulen (Concurrently Executable Modules, kurz *CEM*s genannt). Der Name Π – Paradigm steht für Peacock – Paradigm, wobei Peacock wiederum der Name des ESPRIT Forschungsprojektes ist, in dessen Rahmen der hier vorgestellte Ansatz näher erforscht wird.

2. Das Π – Paradigm

Ein heutzutage oft beschrittener Weg der Systementwicklung zerlegt ein System in einem ersten Schritt in disjunkte Einheiten, die eine bestimmte Funktion ausführen. In einem zweiten Schritt werden die Informationen identifiziert, die zwischen den

Einheiten ausgetauscht werden. Man bezeichnet diese Vorgehensweise als funktionalen Ansatz. Diese Technik wird weitgehend von den traditionellen Programmiersprachen unterstützt, die Konzepte wie Prozeduren oder parallele Prozesse zur Verfügung stellen. Der von uns verfolgte Ansatz stellt dagegen die Verwaltung einer Dateneinheit und die auf den Dateneinheiten vorgenommenen Operationen in den Vordergrund. Das allgemeine Ziel ist es, die Konzepte der Datenabstraktion auf die Spezifikation, den Entwurf und auf die Implementierung verteilter Systeme anzuwenden.

Um ein verteiltes System erfolgreich entwickeln zu können, genügt es nicht, lediglich eine gute Programmiersprache zur Verfügung zu stellen. Selbst das Vorhandensein einer kompletten Entwicklungsumgebung, die eine Spezifikationssprache, eine Implementierungssprache und Werkzeuge zur Manipulation oder Analyse einer Systembeschreibung enthält, genügt nicht, wenn nicht ein Paradigma der Systementwicklung zugrundeliegt.

Dieses Paradigma muß Antworten auf folgende Fragen bereithalten (siehe auch Bild 1):

- Wie sieht ein System auf den unterschiedlichen Abstraktionsebenen aus?
- Wie kann der Systementwickler ein System auf den verschiedenen Abstraktionsebenen beschreiben?
- Welche Hilfen werden dem Systementwickler geboten, damit er sich zwischen den verschiedenen Ebenen bewegen kann, um so das System aufzubauen oder zu verändern?

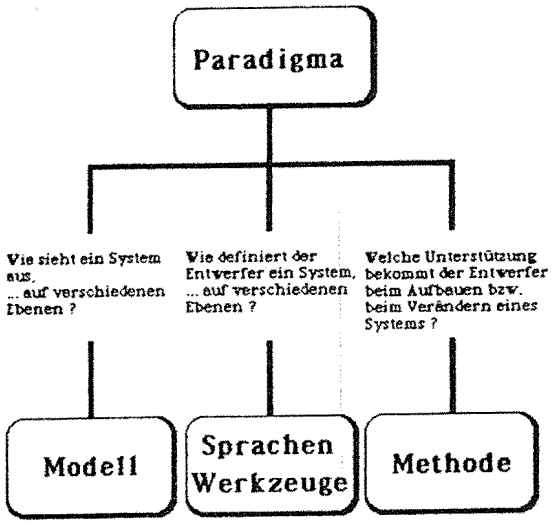


Bild 1: Bestandteile eines Paradigmas

Zur Beantwortung der ersten Frage wird ein **konzeptuelles Modell** der Struktur und Organisation eines Systems zur Verfügung gestellt, das definiert, mit welchen Komponenten der Entwickler arbeiten kann und wie diese Einheiten miteinander verknüpft bzw. wie sie erzeugt werden können. Das Modell umfaßt dabei alle Abstraktionsebenen der Systementwicklung, von der niedrigsten Stufe des lauffähigen Systems bis zur höchsten Spezifikationsstufe und in die Ebene der Anforderungsdefinition hinein. Die elementaren Komponenten des hier vorgestellten Ansatzes sind die gleichzeitig ausführbaren Module: CEMs.

Sprachen erlauben die Beschreibung von CEMs auf den verschiedenen Ebenen des Systementwurfs wie Anforderungserfassung, Design und Implementierung, wobei der Übergang von einer Beschreibung in die nächste von Werkzeugen zu unterstützen ist.

Die **Methode** basiert auf der Datenabstraktion [3], bei der die Details der Repräsentation einer Datenstruktur und die Details der auf dieser Datenstruktur operierenden Funktionen "versteckt" werden und damit beim Entwurf zunächst nicht berücksichtigt werden. Zunächst werden mögliche Datenmanipulationen nur benannt.

Ein gutes **Paradigma** ist charakterisiert durch seine Klarheit –es existiert nur **ein** konzeptuelles Modell, **eine** Entwicklungsmethode und **eine** Sprachfamilie. Damit ein Paradigma auch effektiv angewandt werden kann, muß es selbstverständlich durch entsprechende (rechnergestützte) Werkzeuge unterstützt werden.

3. Concurrently Executable Modules

Das dem Π – Paradigm zugrundeliegende Modell entstammt einer Synthese der Konzepte aus den Gebieten der Datenabstraktion und dem der parallelen Prozesse. Daraus resultiert eine einzige Einheit modularer Struktur: Gleichzeitig Ausführbare Module, oder Concurrently Executable Modules, kurz CEMs genannt.

Der Name spiegelt die Tatsache wieder, daß auf der untersten Ebene einer Systemdefinition ein System aus Modulen in dem Sinn besteht, daß jedes Modul eine Einheit der Datenabstraktion darstellt. Jede dieser Einheiten unterstützt die konkurrierende Ausführung der exportierten Operationen wo immer dies möglich ist. Dabei ist zu beachten, daß die **lokale Parallelität** innerhalb eines Moduls aus der Sicht untergeordneter Module zu einer **globalen Parallelität** wird. Auf der untersten Definitionsebene besteht ein System dann aus kommunizierenden sequentiellen Prozessen [4].

Die höheren Ebenen einer Systemdefinition abstrahieren von den verschiedenen Aspekten eines lauffähigen Systems. So hat zum Beispiel jedes Modul einen assoziierten Datentyp, da es ja eine Einheit der Datenabstraktion darstellt. Dies geschieht auf einer Ebene der Systemdefinition, auf der alle mit Speicher und Paralletität zusammenhängenden Details weggelassen werden. Andere Ebenen der Systemdefinition erhält man, indem lediglich von der möglichen Parallelität oder dem erforderlichen Speicherplatz eines lauffähigen Systems abstrahiert wird.

Der Systementwickler kann mit Hilfe von CEMs das System auf verschiedenen Abstraktionsebenen repräsentieren. Darüberhinaus kann die Darstellung eines Systems bei Verwendung von CEMs in den beiden zusätzlichen Dimensionen

- Formalität einer Beschreibung
  (So kann zum Beispiel eine erste Spezifikation lediglich die Systemkomponenten aufzählen, während ihre semantische Bedeutung informell dargestellt wird.)
- Vollständigkeit einer Beschreibung
  (In dem Sinne, daß auch eine unvollständige Spezifikation möglich ist und als von Wert betrachtet wird.)

beeinflußt werden. CEMs erlauben die Darstellung eines Systems in jeder "Ecke" des durch die drei Dimensionen

- Abstraktionsgrad,
- Formalitätsgrad und
- Vollständigkeitsgrad

aufgespannten Raumes (Bild 2).

Ein lauffähiges Programm hätte z.B. die "Koordinaten" 100% vollständig, 100% formal und 0% abstrakt. Eine Darstellung abstrakter Datentypen wie man sie gewöhnlich in der einschlägigen Literatur findet, hätte die Merkmale 50–70% vollständig, 100% formal und 100% abstrakt.

Durch das hier vorgestellte Modell wird der Systementwerfer gezwungen, ein System als eine aus CEMs gebildete Struktur zu betrachten. CEMs sind die Grundlage des Modells, sie sind die einzige Einheit zur modularen Stukturierung in der Sprachfamilie. Die Methode der Systementwicklung beinhaltet die Definition bzw. die Veränderung von Definitionen von CEM – Strukturen entlang der drei Freiheitsgrade.

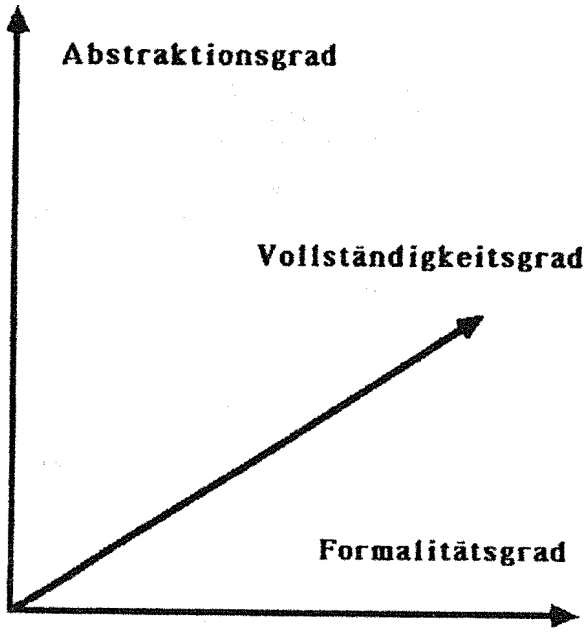


Bild 2: Freiheitsgrade des Entwurfs

4. Ein kleines Beispiel

Das vorgestellte Konzept wird anhand des Beispiels "Liftsystem" etwas konkretisiert.

4.1 Problemstellung

Gegeben sei ein Gebäude mit m Stockwerken und n Aufzügen. Jeder Lift besitzt innen für jedes Stockwerk einen Leuchttaster, der aufleuchtet, sobald er betätigt wird. Des weiteren existiert ein Alarmknopf, mit dessen Hilfe dem Hausmeister ein Notfall angezeigt werden kann. Auf jedem Stockwerk gibt es Tasten, mit denen man einen Mitnahmewunsch für eine Aufwärts– bzw. eine Abwärtsfahrt signalisieren kann. Auch hier leuchtet der Taster auf, sobald er betätigt wird. Wird ein bestimmter Fahrauftrag erledigt, so werden die entsprechenden beleuchteten Tasten wieder zurückgesetzt (Bild 3).

Es ist nun ein Steuerprogramm zu entwickeln, bei dem sichergestellt ist, daß alle Aufträge innerhalb einer endlichen Zeitspanne erledigt werden.

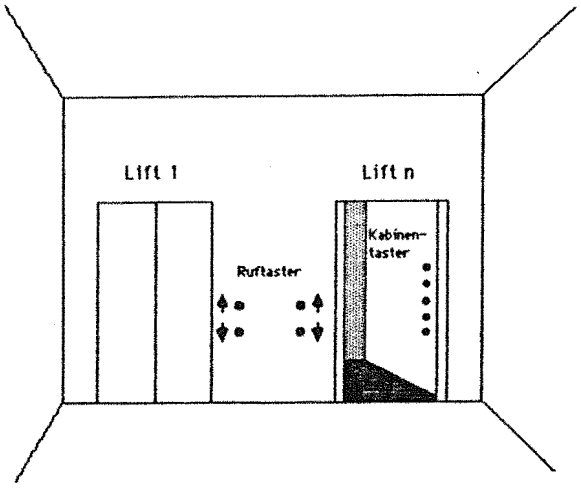


Bild 3: Das Liftsystem

4.2 Skizze einer Problemlösung

4.2.1 Systementwurf

Das Liftsystem besteht aus zwei Teileinheiten, den *Aufzügen* selbst und den *Stockwerken*. In dem System können mehrere Lifts gleichzeitig existieren, die Stockwerke sind jedoch allen Lifts gemeinsam.

Die gewählte Zerlegung spiegelt diese Struktur wieder, indem folgende Module (CEMs) zur Verfügung gestellt werden (siehe auch Bild 4):

- ein Modul zur Modellierung der Stockwerke (Floor–CEM). Die Hauptaufgabe ist die Verwaltung der Mitnahmeanforderungen.
- ein Modul zur Modellierung eines Aufzugs (Lift–CEM), welches wiederum aus folgenden Teilmodulen besteht:
  - einem Modul zur Verwaltung der innerhalb der Kabine getätigten Aufträge.
  - einem Kontrollmodul, das entscheidet, welche Aktion(en) auszuführen sind.
  - einem Modul, welches den Lift real bewegt.

Der Stockwerke CEM ist prinzipiell eine passive Einheit, die Anforderungen entgegennimmt und Informationen über noch zu erledigende Aufträge an die einzelnen Aufzüge weitergibt.

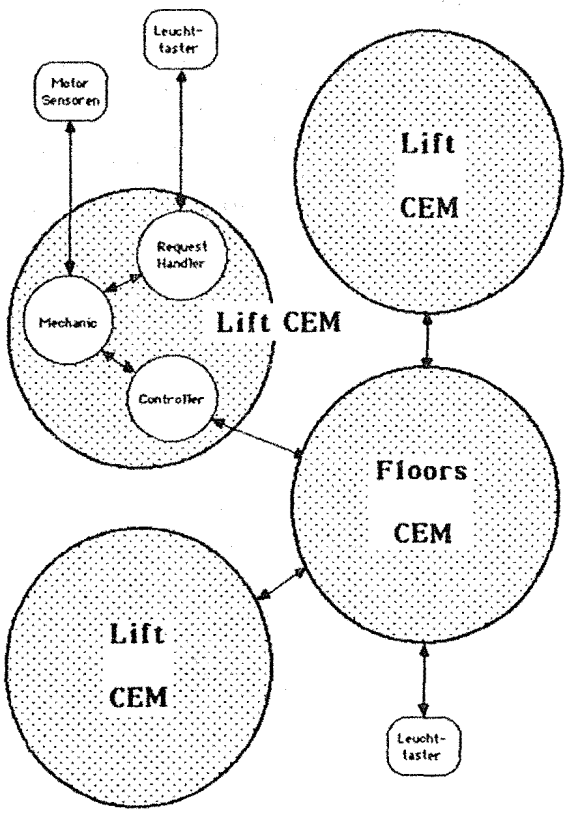


Bild 4: Struktur des Lift-steuerungs Systems

Jeder Lift–CEM kontrolliert seine Bewegung selbst. Die verwendete, relativ einfache Strategie versucht, möglichst in der momentanen Bewegungsrichtung weiterzufahren. Wenn ein Stockwerk erreicht wurde, wird überprüft, ob irgendwelche Aufträge vorliegen, die diese Etage betreffen. Ist dies der Fall, so werden sie bedient. (Dabei wird vorausgesetzt, daß die Überprüfung in Realzeit ausführbar ist.) Falls noch unerledigte Aufträge in der aktuellen Bewegungsrichtung vorliegen, so wird diese beibehalten. Im anderen Fall wird versucht, die Richtung umzudrehen, indem überprüft wird, ob Aufträge für die Gegenrichtung vorliegen. Wenn ja, so setzt sich der Aufzug dorthin in Bewegung. Sollte jedoch kein Auftrag vorliegen, so verbleibt der Lift auf dem aktuellen Stockwerk und "pollt" solange die Etagen über bzw. unter sich (inklusive dem momentanen Flur), bis ein neuer Fahrwunsch signalisiert wird.

Die hier vorgestellte Strategie ist sicherlich nicht die optimalste, sie erfüllt aber die Anforderungen und ist zur Veranschaulichung des CEM Konzeptes ausreichend.

4.2.2 Design

Als nächstes werden die abstrakten Datentypen definiert. Dazu wird hier exemplarisch der Lift–CEM behandelt. Dieser besteht aus mehreren Untermodulen, die alle getrennte Aufgabenbereiche wahrnehmen. Die verwendete Schreibweise lehnt sich an die Programmiersprache Pascal an.

Ein Lift setzt sich aus folgenden Teilen zusammen:

```
MODULE request_handler;
Sorts:      floors
Operations:
   request_floor                  : floor
   any_request_for_current_floor  :        ⇒ BOOLEAN
   any_request_above              : floor ⇒ BOOLEAN
   any_request_below              : floor ⇒ BOOLEAN
Semantics:      ...
```

Der Request_handler ist zuständig für Fahrwünsche, die innerhalb der Aufzugskabine getätigt werden. Er hat auch dafür zu sorgen, daß die entsprechenden Schalter beleuchtet werden. Die einzelnen Funktionen haben folgende Aufgabe:

- request_floor
  Nimmt eine Anforderung für Etage <floor> entgegen.
- any_request_for_current_floor
  Gibt an, ob das aktuelle Stockwerk besucht werden muß.
- any_request_above und
  any_request_below
  Wahr, falls der Lift noch Aufträge über bzw. unter der Etage <floor> erledigen muß. Die Sonderfälle oberstes bzw. unterstes Stockwerk müssen dabei natürlich entsprechend berücksichtigt werden.

Ein wichtiger Teil ist auch der "controller". Er ist ein aktiver Modul (CEM), der die im vorherigen Kapitel beschriebene Bewegungsstrategie verwirklicht. Hinzu kommt ferner der "mechanic"–CEM, der für die Motorsteuerung des Lifts zuständig ist und der außerdem die verschiedenen Sensoren, Taster etc des Aufzugs überwacht.

4.2.3 Eine mögliche Implementierung

Das CEM Konzept stellt eine Reihe von Anforderungen an die potentielle Implementierungssprache. Im einzelnen muß es möglich sein

- Datentypen zu definieren,
- Objekte eines bestimmten Typs zu generieren,
- die erlaubte Parallelität zwischen den Operationen eines Objekts zu definieren,
- parallele Objekte zu verknüpfen und ihre Kommunikation zu beschreiben,
- die (mögliche) Verteilung der Module auf verschiedene Knoten zu beschreiben, sowie
- (parallele) Objekte dynamisch zur Laufzeit zu erzeugen bzw. zu vernichten.

Eine optimal auf das CEM Konzept abgestimmte Implementierungssprache existiert zur Zeit noch nicht. Unter den existierenden Sprachen gibt es jedoch einige, die Teile der oben aufgeführten Anforderungen erfüllen und so gegebenenfalls als Zielsprache doch in Frage kämen. Es sind dies unter anderen (in alphabetischer Reihenfolge):

- Ada [1]
- Conic [6]
- Modula–2 [10]
- Nil [7]
- Occam [5]
- Pascal–Plus [8]

- Pearl [9]
- Smalltalk [2]

In einer (Pearl ähnlichen) Pseudo Programmiersprache könnte die Aufzugsteuerung folgendermaßen implementiert werden:

```
PROGRAM lift_system;
:
:
CEM floors;
   MONITOR CEM request_handler;
   PROCESS CEM io_handler;
      :
   BEGIN
      (* Initialisierungscode ... *)
      ACTIVATE io_handler;
   END;
CEM lift;
   MONITOR CEM request_handler;
   MONITOR CEM mechanic;
   PROCESS CEM controller;
      :
   BEGIN
      (* Initialisierungscode ... *)
      ACTIVATE controller;
   END;
:
:
INSTANCE lifts : ARRAY [1..n] of lift];
         house : floors;
BEGIN
   (* ''Hauptprogramm'' *)
END
```

Dabei wird hier davon ausgegangen, daß zu jeder CEM Beschreibung ein Anweisungsblock definiert ist, der bei der Generierung eines Objekts des entsprechenden Type durchlaufen wird. Dieser Block kann der Initialisierung dienen, oder auch z.B. im Falle der Prozeß CEMs die Prozeßanweisungen in Form einer Endlosschleife enthalten.

Neben der oben aufgeführten Implementierungsskizze sind sicherlich noch andere Lösungen vorstellbar, man denke nur an Objekte, die über Nachrichtenaustausch kommunizieren und nicht über gemeinsamen Speicher wie es beim Monitoransatz der Fall ist. Auch wurde nichts über eine mögliche Verteilung der CEMs ausgesagt (wie z.B., ein Lift–CEM residiert in einem lokalen Steuerrechner im jeweiligen Aufzug etc). Zur Illustration sollte dies jedoch genügen.

5. Zusammenfassung

Der Beitrag zeigt, wie unter Verwendung des einheitlichen Ansatzes des II – Paradigms auf der Basis von Datenabstraktion verteilbare Systeme entwickelt werden können. Es ist zu erkennen, daß diese Vorgehensweise weitgehend unabhängig von der endgültigen Realisierungssprache angewandt werden kann. Dazu muß lediglich gewährleistet sein, daß die Implementierungsprache eine Reihe von Mindestanforderungen erfüllt.

Der Vorteil der hier vorgeschlagenen Methode liegt darin, daß der Systementwerfer ein System in allen Phasen der Entwicklung als eine aus CEMs gebildete Struktur sieht. CEMs sind die Grundlage des Modells und sie sind die einzige Einheit zur modularen Strukturierung in der Sprachfamilie. Die Entwicklungsmethode beinhaltet lediglich die Definition bzw. die Veränderung von Definitionen von CEMs entlang der drei Freiheitsgrade: Abstraktionsgrad, Formalitätsgrad und Vollständigkeitsgrad.

Anerkennung

Der hier vorgestellte Ansatz entstand im Rahmen des Projektes *PEACOCK*, das von der Europäischen Gemeinschaft im Rahmen des ESPRIT Programms gefördert wird. An diesem Projekt sind folgende Institutionen beteiligt: Plessey Electronics Systems Research (Großbritanien), Eurosoft Systems (Frankreich), The Hatfield Polytechnic (Großbritanien), sowie die Universitäten Dortmund und Karlsruhe (Deutschland). Die dargestellten Ideen und Konzepte entstanden in enger Zusammenarbeit aller Partner. Die Arbeiten wurden am Institut für Informatik III, Forschungsgruppe Prozeßrechentechnik, Prof. Dr.–Ing. U. Rembold durchgeführt.

Literatur

1 Ada
Reference Manual for the Ada Programming Language
ANSI / MIL–STD 1815A, 1983

2 A. Goldberg, D. Robson
Smalltalk–80 – The Language and its Implementation
Addison–Wesley, 1983

3 J. Guttag, J.J. Horning
Formal Specification as a Design Tool
Xerox PARC CSL-80-1

4 C.A.R. Hoare
Communicating Sequential Processes
Communications of the ACM, August 1978

5 Inmos Ltd
Occam Reference Manual
Prentice Hall 1984

6 J. Kramer, J. Magee, M. Sloman, K. Twidle, N. Dulay
The Conic Programming Language Version 2.4
Research Report DOC 84/19
Imperial College, London, 1984

7 R.E. Strom, S. Yemini
NIL: An intergrated Language and System for Distributed Programming
Sigplan 83 Symposium on Programming Languages Issues in Software Engineering

8 J. Welsh, D.W. Bustard
Pascal–Plus – Another Language for modular Multiprogramming
Software – Practice & Experience, Vol 9, 1979

9 H. Werum, H. Windauer
Pearl
Vieweg Verlag, 1978

10 N. Wirth
Programming in Modula–2
Springer 1983, 2nd Ed.

11 M. Zelkowitz, R.T. Yeh, R. Hamlet, J. Cannon, V. Basili
State-of-Practice Survey of Software Industry
IEEE Computer, 1983

Anschrift der Autoren

Dr. Wolfgang K. Epple
BMW AG, Abt. EG–26
Knorr – Straße 146
Postfach 400240
8000 München 40, F.R.G.

Horst Spandl
Institut für Informatik III
Prof. Dr.–Ing. U. Rembold
Universität Karlsruhe
Postfach 6380
7500 Karlsruhe, F.R.G.

UNIX Mail: spandl@uka.UUCP
(alte Form: ...!unido!uka!spandl)

Ein Werkzeug für die Systemprogrammierung von Realzeitsystemen auf der Basis von MODULA-2

Dipl.-Ing. Jürgen Stoll, München

Zusammenfassung

MODULA-2 ist eine Programmiersprache die auf Grund ihrer Konzeption auch sehr gut für die Systemprogrammierung geeignet ist. Allerdings stellt das in MODULA-2 verwendete Coroutinen-Konzept für die Programmierung von Realzeitsystemen keine befriedigende Lösung für die Steuerung paralleler Rechenprozesse dar. In diesem Bericht wird die Ersetzung des Coroutinen-Konzepts durch ein Rechenprozeßmodell vorgestellt, wobei das Rechenprozeßmodell mit einem Realzeitbetriebssystemkern realisiert wird. Nach der Vorstellung des Coroutinen-Konzepts werden die Schnittstellen im MODULA-2 Cross-Kompilierer und im Realzeitbetriebssystemkern analysiert und dann eine Realisierung ausführlich besprochen. Das Ergebnis stellt sich als ein Satz von Prozeduren dar, wobei die Prozedurdefinitionen (Definition-Part in MODULA-2) so zweckmäßig entworfen werden, daß es möglich ist, ohne weiteres andere Betriebssysteme einzusetzen und zwar "nur" durch Schreiben eines neuen Implementation - Parts. Es eröffnet sich somit - unterstützt durch MODULA-2 - die Möglichkeit, einheitliche Betriebssystemschnittstellen (Syntax und Semantik) ohne Beteiligung der verschiedenen Betriebssystemhersteller zu bekommen.

Summary

MODULA-2 is a programming language which is also suitable for system level programming. However for programming embedded realtime systems, the coroutines as a concept for synchronization do not fulfill our requirements. In this paper the replacement of the coroutine model by a task model is described. After the introduction of the model of coroutines we are going to analyse the interfaces of the MODULA-2 cross-compiler and the realtime operating system kernel. A detailed description of the realization follows. The result is a set of procedures. The definitions of the procedure headers (Definition-Part in MODULA-2 terms) are constructed so that it is possible to adapt another operating system by merely writing a new Implementation-Part. With this approach we have a chance to get with the support of MODULA-2 a uniform operating system interface (syntax and semantics) without the participation of the various producers of operating systems.

Schlüsselworte

Real- bzw. Echtzeitprogrammierung, MODULA-2, Coroutinen, Rechenprozeßmodell, Systemprogrammierung, Realzeit-Betriebssystemkern, hierarchisches verteiltes Prozeßrechnernetz.

Keywords

realtime programming, MODULA-2, Coroutines, task model, system programming, realtime operating system kernel, hierarchical distributed industrial control net.

1. EINLEITUNG

1.1. Motivation

Am Institut für Systemorientierte Informatik an der Universität der Bundeswehr München beschäftigen wir uns mit dem Thema "Gewährleistung einer eingeschränkten Funktionsfähigkeit von verteilten Realzeitsystemen beim Auftreten von Fehlern im Betrieb". Dazu wurde ein hierarchisch verteiltes Realzeitsystem <Obenhuber, Rzehak 83> auf der Basis des MC 68000 aufgebaut.

Bei der Auswahl der Sprache für die Systemprogrammierung dieses experimentellen Realzeitsystems war folgende Randbedingung zu beachten: Die Programmentwicklung für das verteilte Realzeitsystem soll auf einem Host-System (Perkin-Elmer 3240, Sprachen: FORTRAN, PASCAL, PEARL in Vorbereitung) mit Cross-Software durchgeführt werden.

1.2. Systemprogrammierung

Die Systemprogrammierung war bislang die Domäne der Assemblerprogrammierung mit all ihren Nachteilen.

Die Charakteristika bei der Systemprogrammierung sind:

(C1) Zugriff auf physikalische Adressen
(C2) Einzelbitverarbeitung
(C3) Aufbau von Datenstrukturen
(C4) Algorithmen für die Bearbeitung bzw. Verwaltung dieser Datenstrukturen

In Assembler kann man die Punkte (C1) und (C2) leicht beherrschen. Die Punkte (C3) und (C4) sind in Assembler sehr schwerfällig zu handhaben. Umgekehrt verhält es sich bei der Verwendung von Hochsprachen (z.B. PASCAL)

wobei die Punkte (C1) und (C2) manchmal gar nicht zu erfüllen sind.

Genau diese Lücke schließt MODULA-2. Um den häufig beobachteten Effizienzverlust beim Verwenden von Hochsprachen zu vermeiden, werden im Rahmen des LILITH-Projektes <Wirth 84> Vorschläge zur besseren Anpassung der Hardware an Hochsprachen gemacht.

1.3. Das Projekt LILITH

Die Programmausführungszeit für einen Algorithmus der in PASCAL kodiert ist, kann gegenüber der Ausführungszeit für den gleichen Algorithmus in FORTRAN kodiert um ein Mehrfaches größer sein. Gegenüber der Kodierung in Assembler kann dieser Faktor noch größer sein. Gleichzeitig ist der vom Kompilierer für eine Hochsprache erzeugte Kode wesentlich länger als der von einem geübten Assembler-Programmierer.

Diese Zahlen, die man messen und prüfen kann, müssen nun interpretiert werden. Man muß klar unterscheiden zwischen einer Programmiersprache und ihrer Implementierung.

Eine Programmiersprache ist ein Formalismus, der auf Grund der kompakten und formalen Definition seiner Konstrukte und deren eindeutigen Bedeutung sehr geeignet ist um Algorithmen und Datenstrukturen zu beschreiben. Von diesem Standpunkt aus ist der Begriff 'Sprache' sehr unglücklich und führt zu Mißinterpretationen. Eine Implementierung ist ein Mechanismus für die Interpretation von Algorithmen, welche mit einem Formalismus beschrieben sind. Daraus zog Wirth den Schluß, daß der Verlust an Effektivität nicht inhärent im Konzept der strukturierten Hochsprache, sondern in deren inadäquaten Implementierung begründet liegt. Die Zielsetzungen beim LILITH Projekt waren:

Es sollte ein Arbeitsplatzrechner mit einem geschlossenen Hardware- und Software - Ansatz entwickelt werden <Ohran 84>.

(A1) Auf dem Rechner soll nur eine Sprache (Formalismus) implementiert werden. Alle Programme (Algorithmen) sollen in dieser Sprache formuliert werden, ohne Ausnahme!

(A2) Das Betriebssystem soll für den Single-User-Betrieb konzipiert werden.

Durch diese Entscheidung umgeht man solche Probleme wie

- Rechenkernvergabe
- Schutzmechanismen
- Betriebsmittelverwaltung
- Abrechnung

(A3) Für den Arbeitsplatzrechner soll ein Prozessor verwendet werden der leistungsfähig genug ist, um die Rasteroperationen auf einem graphikfähigen Bildschirm und die Benutzerprogramme auszuführen.

1.4. MODULA-2

Aus der ersten Bedingung (A1) ergab sich folgendes Anforderungsprofil an die zu entwickelnde Sprache (MODULA-2).

Die Sprache muß sowohl die Formulierung von Algorithmen auf einem hohen Abstraktionsniveau als auch Operationen der Hardware direkt unterstützen.

MODULA-2 ist eine strukturierte blockorientierte Hochsprache, mit der man auf einem hohen Abstraktionsniveau Algorithmen und Datenstrukturen beschreiben kann. Mit den sogenannten 'Low-Level-Facilities' ist es in MODULA-2 möglich auf physikalische Adressen zuzugreifen und Einzelbitverarbeitung zu betreiben <Paul 84>, <Gutknecht 84>.

Durch das Modulkonzept mit kontrollierbaren Import- und Export-Möglichkeiten (Information-Hidding) und die getrennt übersetzbaren Moduln (Entwicklung größerer Programmpakete) wird der Systemprogrammierer aus der Sicht des Software-Engineering wirklich sehr gut unterstützt <Pomberger 84>.
Zusammenfassend gilt:

MODULA-2 ist im wesentlichen eine Kombination der hervorragenden Merkmale der drei Sprachen

(M1) PASCAL (Syntax weiter systematisiert) <McCormack, Gleaves 83>, <Summer, Gleaves>

(M2) MODULA-1 (schachtelbare Module mit kontrollierbaren Import/Export-Möglichkeiten) <Wirth 77>

(M3) MESA (getrennt übersetzbare Moduln mit Implementation- und Definition-Part) <Mitchell, Mayburg, Sweet 78>

1.5. Echtzeit-Systemprogrammierung

Neben der Programmierung von hardwarenahen Elementarfunktionen stellen die Echtzeit - oder Realzeitsysteme (Bild 1) eine weitere Aufgabenklasse im Bereich der Systemprogrammierung dar.

Die zusätzlichen Charakteristiken für die Systemprogrammierung von Einprozessor-Realzeitsystemen sind:

(C5) Verwaltung und Quasi-simultane Ausführung von parallel ablauffähigen Rechenprozessen.

(C6) Reaktion auf asynchrone Ereignisse in einer vorgegebenen anwendungsabhängigen Zeitspanne.

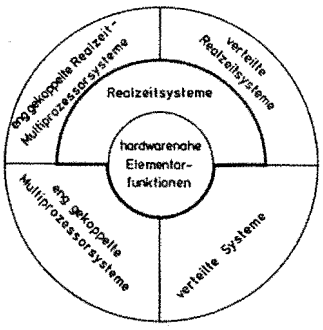


Bild 1: Klassen in der Systemprogrammierung

Für diese Punkte (C5) und (C6) stellt die Syntax von MODULA-2 keine Sprachkonstrukte zur Verfügung. Wegen (A2) war es auch nicht notwendig irgendwelche Annahmen über ein Betriebssystem - das normalerweise die in (A2) ausgeklammerten Punkte organisiert - zu machen <Wirth 83>.

Vielmehr wurden die Coroutinen als Werkzeug für die Lösung der Aufgaben der Punkte (C5) und (C6) im Laufzeitsystem (Modul SYSTEM) zur Verfügung gestellt.

2. Das Coroutinen-Modell

2.1 Allgemeine Vorstellung

Eines der Hauptmerkmale in der Prozeßautomatisierung ist die Steuerung und Regelung gleichzeitg ablaufender technischer Prozesse. Entsprechend benötigt man auf der Programmierseite adäquate Modelle und Prinzipien für die Abbildung dieser parallelen technischen Prozesse auf parallele Rechenprozesse.

Bekannte repräsentative Modelle sind:

- Coroutinen
- FORK- und JOIN-Anweisungen
- cobegin-Anweisungen
- PROCESS Deklarationen

Im folgenden wird das Coroutinen-Konzept (Kooperierende Routinen) näher erläutert <Andrews, Schneider 83>, <Dal Cin, Lutz, Risse 84>.

Coroutinen sind vergleichbar mit Unterprogrammen. Sie unterscheiden sich jedoch in der Art der Weitergabe der Kontrolle. Während bei Unterprogrammen der Kontrollfluß streng hierarchisch abläuft, ist die Kontrollvergabe bei Coroutinen symmetrisch organisiert <Conway 1963>. Beim Aufruf eines Unterprogramms beginnt dessen Ausführung immer mit der ersten Anweisung; es wird dann völlig abgearbeitet bis zur RETURN-Anweisung. Coroutinen können dagegen auch nur teilweise abgearbeitet werden. Ein erneuter Aufruf bewirkt, daß mit ihrer Ausführung am Unterbrechungspunkt fortgefahren wird.

Mit einer Anweisung, die hinsichtlich der Bedeutung der RESUME-Anweisung entspricht, wird die Kontrolle zwischen Coroutinen weitergegeben.

Wie bei einem Unterprogrammaufruf übergibt die Anweisung RESUME(Coroutine-name) die Kontrolle an die benannte Coroutine und speichert so viel Information der rufenden Coroutine ab, um bei einer späteren Kontrollübernahme an der Anweisung nach RESUME wieder fortzufahren. Diesen Vorgang nennt man Quasi-Nebenläufigkeit. Nach der Kontrollübergabe bleiben - anders als bei einem Unterprogramm - die Werte aller lokalen Objekte der Coroutine erhalten. Die Abarbeitung der Coroutine wird sozusagen am Unterbrechungspunkt eingefroren; man sagt auch, daß die Coroutine suspendiert wird.

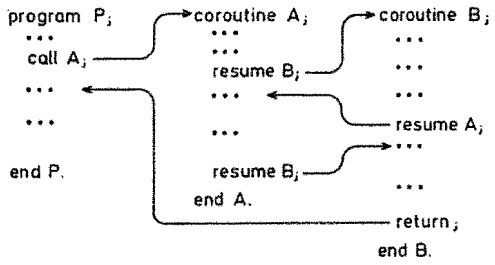


Bild 2:

Prinzip der Benutzung von Coroutinen aus <Andrews, Schneider 83>

Beim ersten Aufruf einer Coroutine wird mit der Abarbeitung am Beginn der Coroutine begonnen, d.h. Initialisierung der Speicherstruktur für die Statusinformation mit den Startwerten der Coroutine. Die Symmetrie der Kontrollübergabe oder die Gleichberechtigung der Coroutinen untereinander erkennt man daran, daß mit der RESUME-Anweisung jede Coroutine jede andere Coroutine aufrufen kann. Die Ko-operation der Routinen muß jedoch der Anwender festlegen.

Jede Coroutine kann als Implementierung eines Rechenprozesses betrachtet werden. Die Synchronisation der Rechenprozesse erfolgt mit der Ausführung von RESUME-Anweisungen. Für echte Parallelität in der Abarbeitung von Programmen (mehr als 1 Prozessor) sind Coroutinen nicht geeignet, da die Semantik der Coroutinen nur die Abarbeitung von einer Routine zur gleichen Zeit erlaubt.

Zusammengefaßt kann man sagen: Coroutinen sind Rechenprozesse bei denen die Kontrollvergabe fest vom Anwender vorgegeben wird und nicht dem zufälligen Eintreffen externer Ereignisse überlassen ist. Es ist möglich die einem Problem inhärente Parallelität auf der Programmebene zu modellieren, die Abarbeitung muß jedoch quasi-simultan erfolgen.

2.2. Umsetzung und Verwendung von Coroutinen in MODULA-2

Da in MODULA-2 die Coroutinen zu den 'Low-Level Facilities' gezählt werden, müssen die zugehörigen Typen (ADDRESS, PROCESS) und die entsprechenden Prozeduren (NEWPROCESS, TRANSFER, IOTRANSFER) vom Modul SYSTEM importiert werden. Mißverständlicherweise wird mit PROCESS eine Coroutine bezeichnet.

Mit der Prozedur NEWPROCESS werden Coroutinen kreiert. Der Aufruf hat die Form:

```
PROCEDURE NEWPROCESS
    (procid: PROC; a: ADDRESS; n: CARDINAL;
     VAR processid: PROCESS; ip: CARDINAL);
```

procid steht für den Bezeichner einer Prozedur vom Typ PROC. Es sind nur parameterlose Prozeduren, die nicht in einer anderen Prozedur enthalten sein dürfen, zugelassen. Diese Prozedur enthält den Kode für die neu erzeugte Coroutine.

a steht für die Startadresse des Arbeitsspeichers (workspace)

n steht für die Größe des lokalen Speichers einschließlich der Kontext-Information.

processid steht für den Bezeichner der Coroutine. Durch den Aufruf wird dieser Variable ein Zeiger in den Arbeitsspeicher der neu kreierten Coroutine zugewiesen. Gleichzeitig wird der Status der Coroutine so initialisiert, daß bei einer ersten Kontrollübergabe die Ausführung am Anfang der Coroutine beginnt.

ip steht für die Interruptebene der Coroutine (vgl. Kapitel 3.2.)

Durch die Trennung von Kode und Daten (workspace) ist es möglich mehrere Coroutinen mit demselben Kode zu kreieren.

Mit einem Aufruf der Prozedur TRANSFER wird die Kontrolle von einer Coroutine zu einer anderen übergeben.

```
PROCEDURE TRANSFER
(VAR form-processid, to-processid: PROCESS);
```

from-processid steht für den Bezeichner der Coroutine, die die Kontrolle abgibt.

to-processid steht für den Bezeichner der Coroutine, die die Kontrolle erhält.

Die rufende Coroutine wird angehalten. Für eine spätere Fortführung - unmittelbar nach der TRANSFER-Anweisung - wird die dazu notwendige Statusinformation im Arbeitsbereich der rufenden Coroutine abgelegt. Die gerufene Coroutine wird entsprechend der abgelegten Statusinformation in ihrem Arbeitsbereich fortgeführt. Dies ist entweder die erste Anweisung nach der zuletzt ausgeführten TRANSFER-Anweisung oder aber bei einer erstmaligen Kontrollübernahme die erste Anweisung innerhalb der Coroutine. In MODULA-2 ist das Hauptprogramm als Coroutine anzusehen.

Da Coroutinen explizit mit einer TRANSFER-Anweisung gestartet werden, müssen sie auch durch solch eine TRANFER-Anweisung beendet werden. Wird bei der Programmausführung auf das Ende einer Coroutine gelaufen, so wird das ganze Programm mit einer Fehlermeldung abgebrochen.

Ein weiteres charakteristisches Merkmal in der Echtzeitprogrammierung ist die Reaktion auf unerwartete Ereignisse (Interrupt). Um in der MDOULA-2 Terminologie zu sprechen; der Punkt der Kontrollübergabe ist bei der Verarbeitung von Interrupts a priori nicht bekannt.

Mit Hilfe der Prozedur IOTRANSFER wird eine 'unprogrammierte' Kontrollübergabe (Interrupt) zwischen Coroutinen vorgenommen.

```
PROCEDURE IOTRANSFER
          (VAR from-processid, to-processid:
           PROCESS; iva: ADDRESS);
```

from-processid steht für den Bezeichner der Coroutine, die die Kontrolle abgibt.

to-processid steht für den Bezeichner der Coroutine, die die Kontrolle erhält.

iva steht für die Interrupt - Vektor-Adresse die einem Gerät zugeordnet ist.

Der Aufbau einer Interrupt-Service-Routine als Coroutine sieht dann folgendermaßen aus:

```
Procedure InterruptHandler;
...
Beginn  (* InterruptHandler *)
  Loop
    IOTRANSFER
        (from-processid, to-processid, iva);
      .
      .
      .
  END
END InterruptHandler .
```

Durch den Aufruf von IOTRANSFER wird die Startadresse des Interrupt-Handler (Geräte - Prozeß in MODULA-2 Terminologie) in die Interrupt-Vektor-Tabelle eingetragen. Danach wird die Kontrolle an die in der Parameterliste angegebene Coroutine übergeben. Löst nun ein Gerät einen Interrupt aus, so wird die gerade aktive Coroutine angehalten und die dem Interrupt zugeordnete Coroutine (Interrupt-Handler) nach der IOTRANSFER-Anweisung fortgeführt. Bedingung ist allerdings, daß die Geräte-Priorität höher ist als die der vor Auslösung des Interrupts aktiven Coroutine. Der Interrupt-Handler ist in einer Endlosschleife eingeschlossen, d.h. IOTRANSFER wird wieder ausgeführt. Offen bleibt nun ob die unterbrochene Coroutine weiter abgearbeitet wird, oder ob in die Coroutine gesprungen wird, deren Bezeichner (to-processid) in der Parameterliste von IOTRANSFER steht.

Um den Programmablauf zu verstehen ist es wichtig zu wissen, daß nur beim <u>ersten</u> Aufruf von IOTRANSFER zur Coroutine to-processid verzweigt wird. Bei allen weiteren Aufrufen von IOTRANSFER welche durch Interrupts initiiert werden, wird in die unterbrochene Coroutine zurückgekehrt.

2.3. Coroutinen-Konzept und Rechenprozeß-modell im Vergleich

Im folgenden werden virtuelle Maschinen bestehend aus den Komponenten "Speicher" und "Prozessor" betrachtet. Die Prozessoren der virtuellen Maschine können reale Prozessoren oder auch Rechenprozesse des unterliegenden Betriebssystems sein <Persch, et.al. 84>.

Für Einprozessorsysteme werden drei virtuelle Maschinen unterschieden:

(VM1) Synchrone Einprozessormaschine:

Alle Rechenprozeßaktivitäten werden in einer Coroutinenumgebung ausgeführt. Die Übergabe der Kontrolle wird nur durch <u>explizite</u> Synchronisationsanweisungen erreicht. (Bsp. aus PEARL: ACTIVATE, TERMINATE, SUSPEND, ...) .

(VM2) Zeitscheibengesteuerte Einprozessormaschine:

Die Rechenkernvergabe wird durch das Ende einer Zeitscheibe oder durch explizite Synchronisationsanweisungen angestossen.

(VM3) Asynchrone Einprozessormaschine:

Die Rechenkernvergabe wird durch asynchrone Ereignisse oder durch explizite Synchronisationsanweisungen angestossen.

Die Ereignisse können sein:

- Geräteunterbrechungen (Interrupts)
- Zeitgeberunterbrechungen bei Betriebssystemen mit Time-Share-Betrieb, d.h. (VM2) wird durch (VM3) mit erfaßt
- Aufruf von Funktionen des Betriebssystems (Traps) (Ein-/Ausgabe, Änderung der Priorität eines Rechenprozesses, warten auf ein bestimmtes Ereignis, ...)

Es findet also eine asynchrone Rechenprozeßverdrängung entsprechend den momentanen Prioritätsverhältnissen und den Stati der Rechenprozesse statt. Im allgemeinen weiß der Benutzer nicht, welcher Rechenprozeß gerade aktiv ist.

Unser Ziel ist also eine virtuelle Maschine vom Typ (VM3). Was wir in MODULA-2 vorfinden ist zunächst eine virtuelle Maschine vom Typ (VM1). Durch IOTRANSFER kann jedoch das Eintreffen eines Interrupts zum Synchronisationsereignis erklärt werden, an dem die Kontrolle an eine andere Coroutine abgegeben wird.

Ein (sequentieller) Rechenprozeß besteht aus privaten Daten und einem sequentiellen Programm. Ein paralleles Programm besteht aus mehreren sequentiellen Rechenprozessen, die zeitlich (quasi) parallel ablaufen können und sich gegenseitig in einer definierten Weise beeinflussen. Die Beeinflussungsmöglichkeiten bestehen im wesentlichen im geordneten Datenaustausch untereinander und in der Möglichkeit, den zeitlichen Ablauf zu beeinflussen <Hansen 77>.

Coroutinen sind ein Hilfsmittel für die Beschreibung und Modellierung der einem Problem inhärenten Parallelität durch Sequentialisierung. D.h. Coroutinen als Beschreibungsmittel für parallele Rechenprozesse eignen sich ausschließlich für Einprozessorsysteme, eine Einschränkung die im Rechenprozeßmodell nicht vorhanden ist.

Das Coroutinen-Konzept ist auf einer sehr niedrigen Ebene angesiedelt. Daraus ergibt sich eine gewisse Schwerfälligkeit in der Handhabung (Beispiel in <Dal Cin 84>, S. 282-287). Der Programmierer muß sich für jede Applikation die Synchronisation neu überlegen, auch wenn eine Synchronisation gar nicht notwendig wäre, weil die Rechenprozesse unabhängig sind. Jeder Kontrollwechsel muß explizit zur Programmerstellungszeit statisch angegeben werden. Beim Rechenprozeßmodell wird die "Verzahnung" der parallel ablauffähigen Rechenprozesse implizit durch das Betriebssystem vorgenommen, nur bei einer notwendigen Synchronisation (wechselseitiger Ausschluß, Kooperation) muß man Angaben machen. Dies ist ein wichtiger Punkt der bei der Synchronisation der Coroutinen beachtet werden muß, denn dadurch können sehr leicht "Deadlocks" programmiert werden.

Aufbauend auf dem Coroutinen-Konzept ist die Implementierung höherer Konzepte zur Verwaltung paralleler Rechenprozesse möglich. Die dazu erforderlichen Prozeduren müssen vom Programmierer selbst geschrieben werden (Beispiel in <Wirth 83>). Die Einschränkung des möglichen Gewinns durch Mehrprogrammbetrieb wird aber auch durch die höheren Konzepte nicht beseitigt, denn die explizite Kontrollübergabe bleibt bestehen. Durch die Verdrängungsmöglichkeit in (VM3) ergibt sich eine bessere Auslastung des Prozesses.

Prioritäten in MODULA-2 werden nicht im Sinne von "bevorzugt zu bedienen" verstanden. Die Angabe einer Priorität bei der Modul-Deklaration dient vielmehr zur Deklaration eines Monitors.
Dadurch kann man für Geräte-Rechenprozesse "wechselseitigen Ausschluß" garantieren. Es wird also die Interruptebene festgelegt, bis zu der (einschließlich) Interrupts mit niedriger Priorität ausgesperrt werden.

Da jetzt Interrupts auf gleicher Ebene sich nicht gegenseitig unterbrechen können, wurde als Ausweg die Prozedur LISTEN eingeführt (<Dal Cin 84; S. 282-287>, <Spector>). Durch den Aufruf LISTEN wird die Interruptebene einen Befehl lang auf 0 gesetzt, dadurch können auch Interrupts mit niedriger Priorität abgearbeitet werden.

Mit der Prozedur IOTRANSFER wird gewissermaßen die Zuordnung einer Interrupt-Quelle zu einer Interrupt-Service-Routine hergestellt (CONNECT-Funktion). Die komplementäre Prozedur (DISCONNECT-Funktion) gibt es nicht.

Wie bereits unter Kapitel 2.2 ausgeführt wurde, wird nur beim ersten Aufruf von IOTRANSFER in einem Interrupt-Handler zur Coroutine to-processid verzeigt, während bei allen nachfolgenden Aufrufen zur unterbrochenen Coroutine zurückgekehrt wird. Der dynamische Ablauf entspricht an dieser Stelle nicht der statischen Programmbeschreibung.

Dies erklärt sich aus der Tatsache, daß die Abbildung der Interrupt-Verarbeitung auf Coroutinen im Prinzip ein Verstoß gegen das Coroutinen-Konzept ist.

Die Nachteile des Coroutinen-Konzepts für Realzeitprogrammierung sind also

(N1) Beschränkung auf Einprozessorsysteme

(N2) Schwerfälligkeit in der Handhabung

(N3) Einschränkung des möglichen Gewinns bei Mehrprogrammbetrieb

(N4) Fehlende Möglichkeit der Vergabe von Prioritäten im Sinne von "bevorzugt zu bedienen"

(N5) Mangelnde Flexibilität bei der Interrupt-Verarbeitung

(N6) Diskrepanz bei der Interrupt-Verarbeitung zwischen Programmbeschreibung und Programmausführung

3. Verfügbare MODULA-2 (Cross-)Kompilierer

3.1. Eine Liste mit Adressen

MODULA-2 ist inzwischen für mehrere Prozessoren und Betriebssysteme verfügbar.

Eine Liste (Stand Oktober 1985) kann vom Institut für Systemorientierte Informatik der Universität der Bundeswehr bezogen werden. Diese Liste kann natürlich nicht vollständig sein, da ständig neue Implementierungen vorgestellt werden <Anderson 84>, <Vergleich 85>.

3.2. Das Programmpaket SMILER-2

SMILER-2 steht als Synonym für die drei Cross-Kompilierer-Systeme

- SMILER (PDP-11 / LSI-11)
- SMILERM (MC 6809)
- SMILERX (MC 68000)

Bei allen drei Cross-Kompilierern ist das Hauptprogramm SM2 und die Pässe PASS1, PASS2, PASS3, SYMPASS, LSTPASS und ERRPASS identisch. Entsprechend den Zielmaschinen unterscheiden sich die Pässe PASS4 und PASS5 (Kodegenerierung). Die Originalversion ist auf den CDC-Rechensystemen lauffähig.

Im folgenden wird nur die Version für den MC 68000 - bezeichnet als SMILERX - betrachtet.

Die Portierung auf unseren 32-Bit Rechner war auf Grund

- der unterschiedlichen Hardware
- den verschiedenen Betriebssystemen
- und der unterschiedlichen Mächtigkeit des implementierten PASCAL-Sprachumfangs

nicht ganz einfach. Besondere Probleme waren:

- Real-Zahlen Darstellung
- Abbildung des SET-Konstrukts
- Verwendung von mehr als 32 Bit in einem CDC-60 Bit Maschinenwort
- File-Handling

Auf Grund der gemachten Erfahrungen sollte die Wirtsmaschine für eine erfolgreiche Portierung folgende Forderungen erfüllen:

- minimale Maschinenwortlänge 32 Bit
- Hauptspeicherausbau > = 1 MByte
- Externspeicher > = 10 MByte
- PASCAL-Kompilierer sollte/muß getrennt übersetzbare Moduln unterstützen

4. Ersetzung der Coroutinen durch ein Betriebsystem mit Prozeßmodell

Um den Implementierungsaufwand niedrig zu halten, wird das Prozeßmodell mit Hilfe des auf dem Markt käuflich erhältlichen Realzeitbetriebssystemkern MTOS-68K[1] (Multi-Tasking/Multi-Processor-Operating-System für den Prozessor MC 68000) realisiert <User's Guide>, <Installations Guide>.

4.1. MTOS - Ein Realzeit-Betriebssystemkern

MTOS ist ein Realzeitbetriebssystemkern, der gedacht ist für den Einsatz in sogenannten "Embedded Systems" ist und für mehrere Prozessoren verfügbar:

- Intel 8086
- Intel 8080/85
- Motorola 6800
- Motorola 6809
- Motorola 68000

Wichtige Eigenschaften von MTOS-68K sind:

- Echtzeit-Multi-Tasking/Multi-Prozessor Betriebssystemkern
- Prozessorvergabe gemäß Prioritäten/ Round-Robin
- Interruptgesteuerte Ein-/Ausgabe
- ROM-fähig
- Ein-/Ausgabetreiber, standard- und benutzerdefiniert
- Dynamische Debugger
- Anwendungsschnittstelle im RAM

Diese Anwendungsschnittstelle wird im nächsten Punkt näher untersucht.

4.2. Die Schnittstellen

4.2.1. Anwendungs- und Systemschnittstelle in MTOS

Da MTOS ein ROM-fähiger Realzeitbetriebssystemkern ist, muß die Verbindung zwischen Kern und Anwendung (Anwenderrechenprozesse) über einen Schreib-/Lese-Speicher geschehen (Bild 3). Die Systemschnittstelle hat dabei die Form

```
Marke 1   DS x
Marke 2   DS y
    .
    .
    .
```

also eine Platzhalterfunktion.

Die Anwenderschnittstelle stellt ein Konfigurationsprogramm dar. Es ist bisher in MTOS nicht möglich Rechenprozesse dynamisch zu kreieren. Entsprechend muß der Anwender in einem Konfigurationslauf die Anzahl der Rechenprozesse, die Anzahl der Gerätetreiber, die dazugehörigen Startadressen, usw. angeben. Mit Hilfe von Makros werden dann die entsprechenden Kontrollstrukturen aufgebaut.

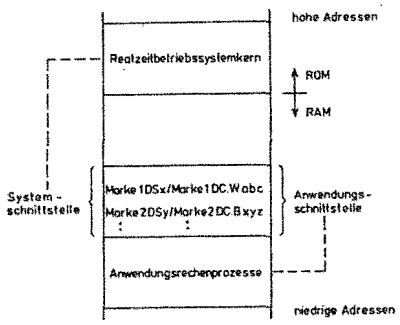


Bild 3: Schnittstelle zwischen dem MTOS-Kern und den Anwendungsrechenprozessen

---

[1] MTOS-68K ist ein eingetragenes Warenzeichen der Industrial Programming Inc.

Das uns interessierende Teilergebnis eines Konfigurationslaufes hat die Form

```
Marke 1   DC.W abc
Marke 2   DC.B xyz
   .
   .
   .
```

Diese beiden Bereiche müssen adressgleich im Speicher liegen. So werden also die Anwendungsparameter dem Systemkern übergeben.

4.2.2. Das Modul SYSTEMX im SMILER

Im Pseudo Modul SYSTEM sind die zielprozessor abhängigen Dinge für die "Low-Level-Facilities", die Routinen zur Unterstützung des Kompilierers und die Laufzeitroutinen enthalten. Im original MODULA-2 Kompilierer <Wirth 83> muß dieses Modul für jeden Übersetzungslauf dem Kompilierer zur Verfügung stehen. Durch die Transkription von MODULA-2 nach PASCAL gab es da Probleme. Die Lösung sah dann so aus: In Pass 1 werden die reservierten Worte für das Pseudo Modul SYSTEM ins Namensbuch des Kompilierers eingetragen und die Realisierung wird im Modul SYSTEMX vorgenommen (X für 68000). Das Modul SYSTEMX wird zur Bindezeit wie ein normales getrennt übersetzbares Modul dazugebunden. Durch diese Lösung hat man hier die Möglichkeit in das Laufzeitsystem (NEWPROCESS, TRANSFER, IOTRANSFER) von MODULA-2 einzugreifen, ohne daß man den ganzen Kompilierer neu übersetzen muß.

4.3. Realisierung

Zunächst könnte man im Modul SYSTEMX alle Kontrollstrukturen für MTOS entsprechend einer MODULA-2 Applikation aufbauen und die Anwendungsschnittstelle entsprechend ausfüllen. Diese Lösung ist wohl sehr elegant aber dafür sehr aufwendig. Da es nicht unser ursprüngliches Ziel war Coroutinen durch ein Prozeßmodell zu ersetzen, entschieden wir uns für folgende Lösung.

Es wird in MTOS eine Dummy Applikation implementiert (z.B. 10 Rechenprozesse und 2 oder 3 Standard-Gerätetreiber). Benutzerspezifische Treiber müssen allerdings vorher in MTOS integriert werden. Da der Benutzer die Startadresse der Schnittstelle (Kern-Anwendung) in MTOS selbst festlegen muß, ist die Startadresse bekannt, und somit die Adressen der Rechenprozesskontrollblöcke (Startadresse des Rechenprozesses) und die Stelle wo die aktuelle Anzahl der Rechenprozesse stehen muß.

Beim Aufruf von NEWPROCESS zur Laufzeit, wird die Bezugsadresse des Arbeitsspeichers einer Variable vom Typ PROCESS zugewiesen. In diesem Arbeitsspeicher steht unter anderem auch die Startadresse der Prozedur, die zum Rechenprozeß umgewandelt wurde. Die Reservierung des Speicherplatzes und die Vorbelegung mit den Startwerten wurde bereits zur Übersetzungs- und Bindezeit erledigt.

Die Anzahl der Aufrufe von NEWPROCESS entspricht der aktuellen Anzahl der Rechenprozesse. NEWPROCESS muß so abgeändert werden, daß bei einem Aufruf dem Prozeßdeskriptor eine MTOS-Rechenprozeßnummer zugeordnet wird (Tabelle) und die Startadresse des nächsten freien Dummy-Rechenprozesses durch die Startadresse des MODULA-2-Rechenprozesses ersetzt wird.

Mit den beiden Zugriffsfunktionen Gib-Rechenprozeßnummer(Prozeßname) und Gib-Prozeßname(Rechenprozeßnummer) wird die Konsistenz der Tabelle gewährleistet. Daß bei einer tatsächlichen Anwendung nicht alle Dummy-Rechenprozesse einen entsprechenden MODULA-2 Prozeß zugeordnet bekommen, hat keine weiteren Auswirkungen. Die restlichen Dummy-Rechenprozesse sind im Zustand dormant.

Die Prozedur IOTRANSFER wird gestrichen, denn es darf kein Interrupt am Betriebssystem "vorbei" abgearbeitet werden <Wirth 83>. TRANSFER entfällt ebenfalls und wird durch entsprechende MTOS-Dienste ersetzt (Tabelle 1).

MTOS-68K USER'S GUIDE

APPENDIX 1: SUMMARY OF MTOS SUPERVISOR SERVICES

SVC	SDB MACRO	FUNCTION	DESCRIPTION
1	STRSDB	START	start task
2	GIASDB	GETIA	get initial argument
3	TRMSDB	TRMT	terminate task without timed restart
4	TRRSDB	TRMR	terminate task with timed restart
5	CPRSDB	CPTY	change task priority
6	PAUSDB	PAUSE	pause for given interval
7	CANSDB	RESUME	resume (cancel pause of) another task
8	EIOSDB	EIO	reset/immediately set/test event flags
9	LRSSDB	LRS	*reset/immediately set EFs of given task
$A	CEFSDB	CEF	*copy event flags
$B	SEFSDB	SEF	set event flags after given time
$C	WEFSDB	WEF	wait until event flag are set
$D	MBSSDB	MBS	send a message to a mailbox
$E	MBRSDB	MBR	receive a message from a mailbox
$F	ALOSDB	ALOC	allocate a block of memory from a pool
$10	DALSDB	DALOC	return (de-allocate) memory to a pool
$11	STISDB	STIME	set ASCII date and time
$12	GTISDB	TIME	get (read) ASCII date and time
$13	SYNSDB	SYN	synchronize to ASCII ("wall") time
$14	DIOSDB	DIO	perform discrete I/O
$15	PIOSDB	PIO	perform peripheral I/O
	RSVSDB	RESERVE	reserve unit
	RLSSDB	RELEASE	release unit
	REDSDB	READ	read with default prompt
	WRISDB	WRITE	write
	REPSDB	READPT	read with given prompt
	ROCSDB	READ1C	read one character w/o prompt
$16	RSFSDB	RSF	release a semaphore
$17	WSFSDB	WSF	wait for SF to be free
$18	.	.	[spare]
$19	MIOSDB	MIO	* perform (local) memory I/O
$1A	CITSDB	CIT	* convert task number to TCB address
$1B	CTISDB	CTI	* convert TCB address to task number
$1C	.	.	** perform Debugger support
$1D	CDISDB	CDI	* copy discretes

* = introduced for Debugger, but may be used by any task
** = may be used ONLY by Debugger tasks

Tabelle 1: MTOS - Dienste aus <User's Guide>

Es ist nun zu klären, wie MTOS-Dienste in MODULA-2-Programmen benutzt werden können.

Wie heute allgemein üblich, werden auch in MTOS die Systemdienste mit Hilfe eines TRAPS aufgerufen. Die Adresse des Parameterblokkes, der genaue Spezifikationen des Dienstes und den Rückkehrstatus enthält, wird im Stack übergeben.

Die Ausführung eines TRAPS kann mit Hilfe der im Cross-Kompilierer schon vorhandenen Prozedur SYSCALL erledigt werden.

```
FROM SYSTEM IMPORT SYSCALL;
SYSCALL (Service: CARDINAL;
         StartAdrParaBlock: ADDRESS;
         VAR Reply: CARDINAL);
```

Die Aufbereitung des Parameterblockes muß in

dienstorientierten Prozeduren erledigt werden.

Einige Details bleiben noch zu klären.

Der Arbeitsspeicher (Workspace) eines Rechenprozesses (Bild 4) muß nach wie vor angelegt werden. Denn zum einen verwaltet der Kompilierer selbst Informationen im Arbeitsspeicher (Dijkstra Display's, statischer Link für Zugriff auf globale Variable aus einer Prozedur) und zum anderen benötigen wir Informationen (Initialwerte der Register) für den Anschluß an MTOS.

```
(*  THE STRUCTURE OF A PROCESS WORKSPACE :                          *)
(*  --------------------------------------                          *)
(*                                                                  *)
(*  THE VALUE OF A VARIABLE OF TYPE 'PROCESS' (CALLED               *)
(*  PROCESS-VARIABLE) IS A POINTER TO THE REFERENCE                 *)
(*  LOCATION INDICATED BY --->>> (PD).                              *)
(*  ALL OFFSETS (IN BYTES) RELATIVE TO THE PROCESS BASE             *)
(*  --->>> OF THISPD.                                               *)
(*  THE PICTURE SHOWS A PROCESS WORKSPACE GENERATED                 *)
(*  AT NEWPROCESS TIME.                                             *)
(*                                                                  *)
(*    HIGH MEMORY : LWA + 1 OF WSP IS AT 52(PD).                    *)
(*                                                                  *)
(*                                                                  *)
(*         52(PD) LWA + 1 OF WSP                                    *)
(*                ==========================                        *)
(*         48(PD) I  RESERVE                I   1 LONGWORD          *)
(*         44(PD) I      ...                I   1 LONGWORD          *)
(*         40(PD) I      ...                I   1 LONGWORD          *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*         16(PD) I  INTERRUPTCODE          I   8 LONGWORDS         *)
(*         12(PD) I  FIRSTELEMENTSIZE       I   1 LONGWORD          *)
(*          8(PD) I  FIRSTFREEELEMENT       I   1 LONGWORD          *)
(*          4(PD) I  STACKLIMIT = FWA WSP   I   1 LONGWORD          *)
(*    --->>> (PD) I  PROCESS BASE THISPD    I   1 LONGWORD          *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*        -36(PD) I  DIJKSTRA DISPLAY'S     I   9 LONGWORDS         *)
(*        -40(PD) I  INITIAL A7 = SP        I                       *)
(*        -44(PD) I  INITIAL A6 = MP        I                       *)
(*        -48(PD) I  INITIAL A5 = PD        I                       *)
(*        -52(PD) I  INITIAL A4             I                       *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*                I      ...                I                       *)
(*                I      ...                I   TOTAL 16            *)
(*                I      ...                I   LONGWORDS           *)
(*                I      ...                I   FOR ALL             *)
(*       -100(PD) I  REGISTERS D0 - A7     I   REGISTERS           *)
(*       -104(PD) I  PROCESS TERMINATES    I   1 LONGWORD          *)
(*      FOPOS(PD) I  FORMAT/OFFSET(MC68010)I  1 WORD               *)
(*      PCPOS(PD) I  INITIAL PC           I   1 LONGWORD          *)
(*      SRPOS(PD) I  INITIAL SR           I   1 WORD              *)
(*                I                       I                       *)
(*                I                       I                       *)
(*                I                       I                       *)
(*                I                       I                       *)
(*        FWA WSP I                       I                       *)
(*                ==========================                      *)
(*                                                                  *)
(*                                                                  *)
(*    LOW MEMORY  : THE VALUE OF STACKLIMIT IS                      *)
(*                  = FWA OF WSP.                                   *)
```

Bild 4: Struktur des Arbeitsspeichers einer Coroutine

In SYSTEMX haben drei Adressregister eine feste Bedeutung

A7 = SP aktiver Stack Pointer

A6 = MP Mark Pointer (Stack Marker)
Zeiger auf die lokalen Variablen und Parameter einer Coroutine (bzw. Rechenprozess)

A5 = PD Process Description
Zeiger auf den aktiven Prozeßbezeichner

Diese Register müssen bei jedem Rechenprozesswechsel gerettet werden. Dies wird in MTOS durch Retten aller Register erledigt.

Ein Sonderfall ist noch zu klären. In MODULA-2 wird das Hauptprogramm wie eine Coroutine behandelt. Analog gibt es in MTOS einen INITIAL-Rechenprozeß (Rechenprozeß Nummer 0) der vom MTOS-System gestartet wird und der alle anderen Rechenprozesse startet. Die Verbindung zwischen MTOS-INITIAL-Rechenprozeß und MODULA-2 MAIN-Programm muß im Modul-Rumpf von SYSTEMX hergestellt werden.

5. Stand und Ausblick

Die Portierung vom SMILERX auf unsere Host - Maschine (PE 3240) ist erfolgreich abgeschlossen <Bosse, Heine, Kotschi 84>, <Westenkirchner 85>. Ein Bibliotheksverwaltungsverwaltungsprogramm <Hafner 85> ist ebenfalls verfügbar.

Nach der vorgestellten Analyse scheint die Ersetzung des Coroutinen-Konzepts durch ein Prozeßmodell in Form einer Diplomarbeit durchführbar. Die Arbeiten könnten bis Ende 1986 abgeschlossen sein.

Da in MTOS die Parameterstruktur nicht einheitlich ist, scheint es nicht sinnvoll die verschiedenen Parameterblockstrukturen als Typen zu vereinbaren und dann zu exportieren. Vielmehr wollen wir einen Satz von parametrisierten Prozeduren erarbeiten und diese in Form von einem getrennt übersetzbaren Modul (Definition- und Implementation-Part) bereitstellen.

Bei der Entwicklung des Definition-Parts soll nicht nur der vorhandene Echtzeitbetriebssystemkern betrachtet werden. Ziel ist vielmehr zu versuchen, die Prozedurvereinbarungen so universell zu gestalten, daß auch andere Betriebssysteme duch einen entsprechenden Implementation-Part angeschlossen werden können.

In diesem Zusammenhang sollen insbesondere die nachfolgenden Entwicklungen betrachtet werden:

(E1) RTOS (realtime operating system) vom EWICS TC 8

(E2) MODEB (MODULA-2 Echtzeitbetriebssystem) basierend auf dem Vorschlag des EWICS TC 8

(E3) MOSI (Microprocessor Operating Systems Interface) vom IEEE 825, Revision 4.0

Jeder, der schon größere Programmpakete portiert hat, weiß, daß die unterschiedlichen Betriebssystemschnittstellen und -dienste große Hindernisse bei der Portierung darstellen. MODULA-2 bietet hier die Möglichkeit durch einen Satz von Prozedurdefinitionen (Definition-Part) eine Anpassung an die unterschiedlichen Betriebssysteme zu ermöglichen. Hierdurch könnte eine Vereinheitlichung geschaffen werden, ohne daß vorhandene Betriebssysteme geändert werden müssen.

6. Literatur

<Anderson 84>

Anderson, T.L.: Seven MODULA-2 Compilers Reviewed.
Journal of PASCAL, ADA & MODULA-2, März/April 84

<Andrews, Schneider 83>

Andrew, G.R.; Schneider, F.B.: Concepts and Notation for Concurrent Programming.
Computing Surveys, Vol. 15, No. 1, March 83

<Bosse, Heine, Kotschi 84>

Bosse, J.; Heine, P.; Kotschi, R.: Portierung eines MODULA-2 Cross-Systems auf dem Prozeßrechner PERKIN ELMER 3240.
Diplomarbeit an der Hochschule der Bundeswehr München, Fachbereich Informatik, 1984

<Conway 1963>

Conway, M.E.: Design of a separable transition-diagram comipler.
Communication of the ACM 6, 7 (July 1963), pp 396-408

<Dal Cin, Lutz, Risse 84>

Dal Cin, M.; Lutz, J.; Risse, T.: Programmierung in MODULA-2.
Teubner Verlag, Stuttgart, 1984

<Gutknecht 84>

Gutknecht, J.: Tutorial on MODULA-2.
Byte, August 1984, pp 157-176

<Hafner 85>

Hafner, U.: Portierung und Erweiterung eines Bibliotheksverwaltungsprogramms für ein MODULA-2 Programmiersystem.
Diplomarbeit an der Universität der Bundeswehr München, Fakultät für Informatik, 1985

<Hansen 77>

Hansen, P.B.: The Architecture of Concurrent Programs.
Prentice-Hall, INC. Englewood Cliffs, New Jersey 07632

<Installation Guide>

MTOS-68K Installation Guide.
Industrial Programming, Inc. Jericho, New York 11755

<Levi 81>

Levi, P.: Betriebssysteme für Realzeitanwendungen.
Datakontext-Verlag, Köln 1981

<McCormack, Gleaves 83>

McCormack, J.; Gleaves, R.: MODULA-2 - A Worthy Successor to PASCAL.
Byte 1983, pp 385-395

<Mitchell, Mayburg, Sweet 78>

Mitchell, J.G.; Mayburg, W.; Sweet, R.: MESA Language Manual.
Report CSL-78-1, Xerox, Palo Alto, California, 1978

<Obenhuber, Rzehak 83>

Obenhuber, H.; Rzehak, H.: Die Anpassung eines Mikroprozessorsystems an den CAMAC-Standard zum Aufbau eines hierarchischen lokalen Rechnernetzes.
Bericht Nr. 8309 der Hochschule der Bundeswehr München, Fachbereich Informatik, Oktober 1983

<Ohran 84>

Ohran, R.: LILITH and MODULA-2.
Byte, August 1984, pp 181-192

<Paul 84>

Paul, R.J.: An Introduction to MODULA-2.
Byte, August 1984, pp 195-210

<Persch, et.al. 84>

Persch, G.; Jansohn, H.-S.; Landwehr, R.; Uhl, J.; Dausmann, M.; Kirchgässner, W.: A Protable Ada Tasking System for Single Processors
German Chapter of the ACM - Berichte, Teubner Verlag, Stuttgart, 1984

<Pomberger 84>

Pomberger, G.: Softwaretechnik und MODULA-2.
Hauser-Verlag, München, Wien, 1984

<Spector>

Spector, D.: Ambiguities and Insecurities in MODULA-2.
MS 108-17-3, Prime Computer, Inc. 500 Old Connecticut Path Framingham, Massachusetts 01701

<Summer, Gleaves>

Summer, R.T.; Gleaves, R.E.: MODULA-2 - A Solution to PASCAL's Problems.
Volition Systems, P.O. Box 1236, Del Mar, CA 92014

<User's Guide>

MTOS-68K User's Guide.
Industrial Programming, Inc. Jericho, New York 11753

<Vergleich 85>

Drei MODULA-Compiler im Vergleich.
Computer Persönlich, Ausgabe 14, 26.6.85, pp 50-54

<Westenkrichner 85>

Westenkrichner, H.: Überarbeiten der Ein-/Ausgabe und testen spezieller Sprachkonstrukte eines MODULA-2 Cross-Kompilierers.
Diplomarbeit an der Universität der Bundeswehr, Fakultät für Informatik, Institut für Systemorientierte Informatik, 1985

<Wirth 77>

Wirth, N.: "MODULA: A Language for Modular Multiprogramming Language."
Software-Practice and Experience, Vol. 7, pp 3-35, 1977

<Wirth 83>

Wirth, N.: Programming in MODULA-2.
Second Edition, Springer-Verlag, Berlin, Heidelberg, New York 1983

<Wirth 84>

Wirth, N.: History and Goals of MODULA-2.
Byte, August 1984, pp 145-152

Verfasser:

Jürgen Stoll
Universität der Bundeswehr München
Fakultät für Informatik
Institut für Systemorientierte Informatik
Werner-Heisenberg-Weg 39
D-8014 Neubiberg
Tel. (089) 6004-2404

Einsatz des offenen Echtzeit-Datenbanksystems BAPAS-DB in einer industriellen Anwendung mit hohen Datenraten

J. Geidies, Dipl.-Ing., Lüneburg

Zusammenfassung

Für die Überwachung der Fertigung des Mercedes 190 in Bremen wurde ein großes Software-System in PEARL auf Basis des offenen Echtzeit-Datenbanksystems BAPAS-DB realisiert.

Die Datenrate aus dem Prozeß, die mit Hilfe des Datenbanksystems verarbeitet wird, beträgt bis zu 200.000 Datenblöcke à 20 Bytes Nutzinformation pro Tag (16 Stunden). Zusätzlich sind parallele Zugriffe auf die Datenbank über Dialoge von 20 Terminals realisiert. Die Zugriffe erfolgen im Rahmen eines verteilten Rechnernetzes sowohl zentral als auch dezentral.

Dieser Vortrag beschreibt zum einen die Strukturen des Anwendungssystems, zum anderen aber auch die Eigenschaften des Echtzeit-Datenbank-Systems, die zur Erfüllung der Anforderungen des Anwendungssystems erforderlich waren.

Summary

The open realtime data base system BAPAS-DB is used for monitoring the production process for Mercedes 190 cars.

One essential feature of the process is the high data rate of up to 200.000 blocks of data, each with 20 bytes per day, processed by the realtime data base system.

This report describes the structure of the software system and requirements necessary for the use of realtime data base systems.

1. Aufgabenstellung

In dem neuen Werk von Daimler Benz in Bremen wird die Baureihe Mercedes 190 in drei riesigen Hallen, den Fertigungsbereichen Rohbau, Lackierung und Montage, produziert. Dabei kommen ca. 700 automatische Fertigungsanlagen (z.B. Schweißroboter, Förderstrecken) sowie ca. 5.000 Hilfsbetriebe (Beleuchtung, Energie, Lüftung) zum Einsatz.

Mitte 1984 wurden für die Steuerung, Überwachung und Instandhaltung dieser Fertigungsanlagen und Hilfsbetriebe drei zentrale Produktionswarten mit neuer Technologie in Betrieb genommen. Das System wurde in enger Zusammenarbeit der Firmen Daimler-Benz, Siemens, Werum und des Fraunhofer-Instituts IITB realisiert /4/.

In diesem Vortrag sollen die Komponenten Überwachung und Instandhaltung vorgestellt werden.

Aus den Anforderungen des Anwenders kristallisierten sich die folgenden tragenden Funktionen heraus, die das Anwendungssystem kennzeichnen (Bild 1):

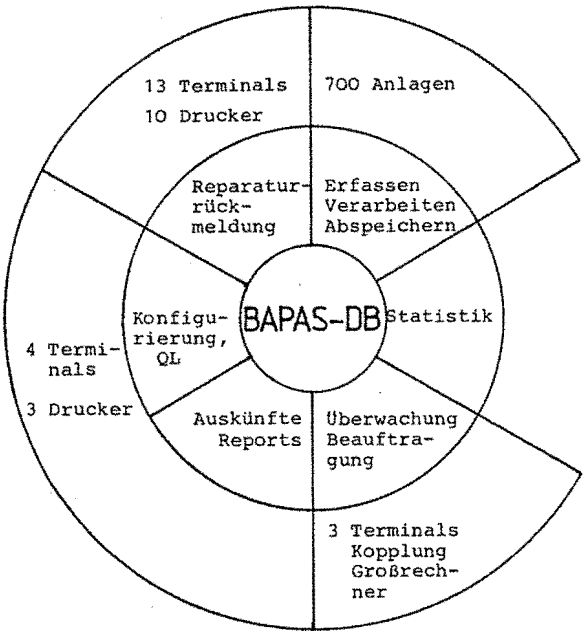


Bild 1. Struktur des Anwendungssystems

Anlagenüberwachung:
Alle Anlagen aus den 3 Fertigungsbereichen sowie alle Hilfsbetriebe sind an das System angeschlossen. Sie liefern im Dauerbetrieb ständig Nutzinformationen wie Taktzeiten, Taktzeitüberschreitungen, Maschinenlaufzeiten, Stückzahlen. Diese Daten sowie die Statusmeldungen (Störungen, Stillstände, Betriebswechsel) werden erfaßt, gespeichert und ausgewertet.
Im einzelnen hat die Anlagenüberwachung folgende Funktionen:

- Erfassung und Auswertung von Unterbrechungen (Stillstände, Störungen) für die Störungsüberwachung
- Ermittlung von Anlagen-Kennzahlen (Bereitschaftszeiten, Taktzeiten, mittlere Laufzeiten (MTBF), mittlere Störzeiten (MTTR), Pufferfüllstände, Verfügbarkeiten, Nutzungsgrade, Stückzahlen) zur Anlagen-/Prozeßüberwachung
- Statistische Auswertungen (mittlere Takt- und Maschinenlaufzeiten, Taktzeitverteilungen, Störungs- und Stillstandsklassifizierung) zur vorbeugenden Schwachstellenanalyse
- Langzeitarchivierung und -auswertung für Langzeitanalyse und -überwachung (3 Tage bis zu mehreren Monaten).

Dialogfunktionen:
Alle oben aufgeführten Anlagenüberwachungsdaten stehen ständig online mit schnellen Responsezeiten auf 20 räumlich zentral und dezentral verteilten Bildschirmen zur Verfügung. Dabei kann der Benutzer eine von zwei Zugriffsarten wählen:

- Informationen über Masken und Reports, die vom Anwender vordefiniert wurden. Hierbei kann grob unterschieden werden in Listen mit folgendem Inhalt:
  - Ständig sich aktualisierende Störungs- und Stillstandsübersicht
  - Listen aller aufgetretenen Störungen mit Unterscheidung in kleine und große Störungen
  - Anlagenspezifische Informationen wie Störungen, Anlagen-Kennzahlen und statistische Auswertungen
  - Bereichsübergreifende Informationen wie Stückzahlübersichten (Verhältnis der Soll- zur Iststückzahl)
  - Tagesbezogene Gesamtübersichten
  - Übersichten über Füllstandsverteilung und mittlere Füllstände von Puffern.

  Fast alle Listen sind für verschiedene Zeiträume (aktuell, Schicht, Woche, Monat) anwählbar.
- Online-Zugriffe über die Query Language von BAPAS-DB mit frei verknüpfbaren Zugriffskriterien.

Instandhaltungssystem:
Ausgehend von Störungen im Betriebsablauf kann von den Zentralwarten die Instandhaltung disponiert und per Terminaldialog der Reparaturdienst beauftragt werden. Der Techniker vor Ort quittiert die durchgeführten Arbeiten und ergänzt die Aufträge um weitergehende Informationen, die dem System neben der Personaldisposition auch eine Schwachstellenanalyse für die vorbeugende Wartung und Ersatzteilhaltung sowie eine automatische Kostenabrechnung ermöglichen. Die einzelnen Funktionen sind:

- Beauftragung von Instandhaltungsmaßnahmen
- Rückmeldungen
  - Technische Erledigung
  - Arbeitsumfang zur internen Abrechnung (Lohnrückmeldung)
- Datenaustausch mit dem Rechenzentrum.

Datenbank:
Aus den beschriebenen Funktionen heraus ergaben sich folgende Zugriffspfade auf die Datenbank, die konkurrierend zueinander erfolgen können durch

- Abspeichern aller Prozeß- und Archivierungsdaten
- Online-Zugriffe parallel von 20 Terminals
- Zugriffe von Auswertungsprogrammen
- Dezentralen Zugriff von den intelligenten Unterstationen
- Online-Zugriffe über die Query Language (Datenbank-Abfragesprache).

2. Systemstruktur

2.1 Hardware

Die räumliche Verteilung (3 Hallen auf einem Gebiet von rund einem halben Quadratkilometer), die Vielzahl der Signale sowie die notwendige Rechnerkapazität waren Gründe, die Anwendungsfunktionen auf ein fehlertolerantes Mehrrechnersystem zu verteilen (B i l d 2).
Zehn Vorrechner (Intelligente Unterstationen) erfassen und verarbeiten die Prozeßsignale einer jeweils zugeordneten Gruppe von Anlagen und wickeln den Dialog mit dem Reparaturdienst vor Ort ab. Über einen schnellen, doppelten Lichtleiterbus, der von dem Fraunhofer-Institut IITB in Karlsruhe realisiert wurde, haben sie Zugriff auf die Datenbank, die auf einem 11. Rechner installiert ist. Dieser Rechner erfüllt die restlichen, zuvor aufgeführten Aufgaben inklusive einer Kopplung zur Großrechenanlage der zentralen EDV.
Einige Zahlen, die den Umfang des Systems charakterisieren:

Datenvolumen	:	700 automatisch überwachte Anlagen 5.000 Hilfsbetriebe
Datenbank	:	50 MB Daten auf externem Speicher 400 KB Daten resident
Datenfluß	:	ca. 200.000 Datensätze à 20 Bytes Information pro Tag (16 Std.)
Dialoge	:	20 Bildschirme mit parallelem Dialogbetrieb (7 zentral, 13 über Unterstationen)
Rechner	:	Zentral: Siemens R30 mit 1 MW Hauptspeicher Dezentral: Siemens R10/R30 mit bis zu 1 MW Hauptspeicher. (Insgesamt 14 Rechnereinheiten)

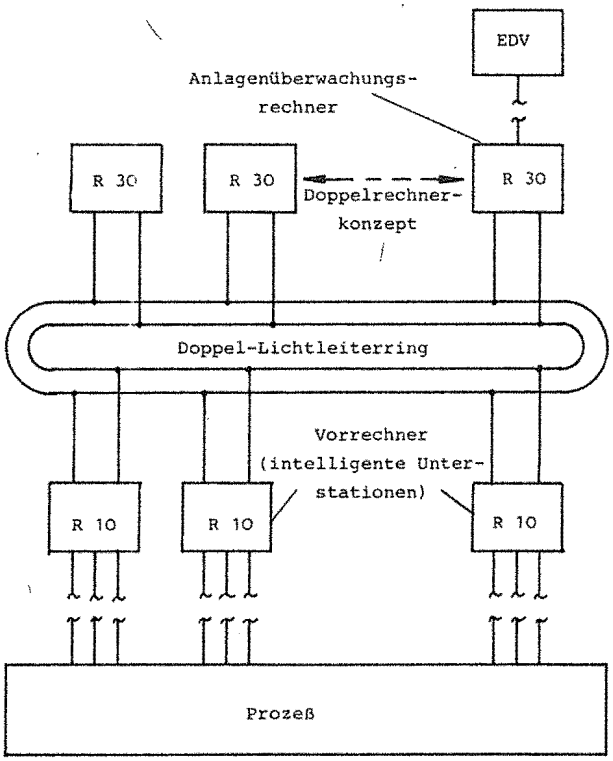


Bild 2. Systemübersicht der Hardware

2.2 Software

Die Struktur des Anwendungssystems wird aus Bild 1 deutlich. Kern ist das Datenbanksystem BAPAS-DB mit der sogenannten DBV-Schnittstelle für die Anwenderprogramme. Es kann grob in 5 Bereiche unterteilt werden:

- Erfassung, Verarbeitung, Abspeicherung
- Statistik
- Überwachung und Beauftragung, Kopplung zum Großrechner
- Auskünfte und Konfigurierung
- Reparaturrückmeldung.

Da nur ein Adressierungsbereich von maximal 64 KW zur Verfügung stand, wurde das Programmsystem in logische Einheiten, die sogenannten PEARL-Welten, aufgeteilt. Diese können sich entweder über einen gemeinsamen Common-Data oder lokale Kommunikation miteinander verständigen.

Zur Verteilung der Anwendungsfunktionen auf dem Mehrrechnersystem wurde die in Bild 3 beschriebene Softwarestruktur gewählt. Es wird von den intelligenten Unterstationen über Kommunikation auf die Datenbank im Anlagenüberwachungsrechner zugegriffen.

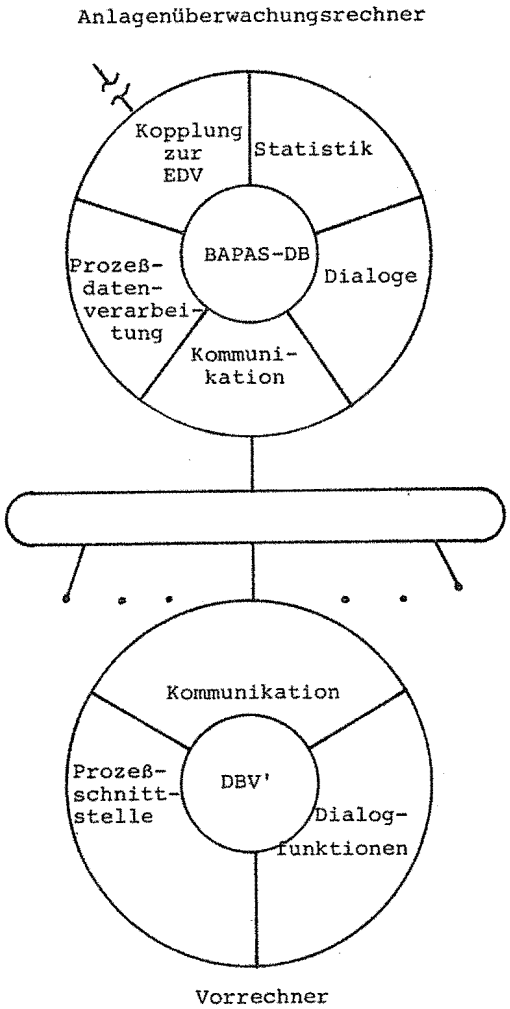


Bild 3. Verteilung der Software-Funktionen.

Zur Erreichung einer hohen Verfügbarkeit wurde ein Doppelrechner-Konzept gewählt. Bei Ausfall eines der beiden Rechner wird dies automatisch erkannt. Unter Lastabwurf von weniger wichtigen Aufgaben übernimmt der noch laufende Rechner Funktionen des ausgefallenen Partners.

Nun noch einige Zahlen, die den Umfang des Anwendungspakets charakterisieren:

Anzahl Module	:	ca. 90
Anzahl Tasks	:	ca. 70
Anzahl Quellzeiten	:	ca. 100.000
Größe des Codes	:	ca. 800 KB
Größe der Daten	:	ca. 400 KB

Folgende Produkte von Werum wurden eingesetzt:

- PEARL-Programmiersystem
- Datenbanksystem BAPAS-DB
- Maskenlaufzeitsystem MMC .

3. Anforderungen an das Echtzeit-Datenbanksystem BAPAS-DB

Die Realisierung eines solchen Systems stellt eine Reihe von Anforderungen an das zugrundeliegende Datenbanksystem:

- Das Datenbanksystem muß minimale Programm-Zugriffszeiten für wichtige Prozeßdaten garantieren, auch wenn an 20 Terminals gleichzeitig gearbeitet wird. Dazu müssen Datenbankzugriffe prioritätsgesteuert auf Satzebene abgearbeitet werden.
  Das bedeutet, daß Tasks mit hoher Priorität andere mit niederer Priorität während einer Datenbank-Transaktion überholen können. Daneben muß es möglich sein, pro Datei die für die jeweilige Anwendungsfunktion optimale Zugriffsstrategie einzusetzen.
- Das Anwendungssystem muß pro Minute zwischen 200 und 300 Prozeßdaten verarbeiten, d.h. ihrer Verwendung gemäß in unterschiedliche Datenbankmengen abspeichern und berechnete Fertigungsdaten in den Anlagenübersichten aktualisieren. Das ergibt im Mittel 17 Datenbankzugriffe pro Sekunde, also rund 50 - 60 msec pro Datenbankzugriff. Wenn man nun die Auskunftsfunktion noch hinzurechnet, ist ein rein plattenorientiertes Datenbanksystem bei einer mittleren Plattenzugriffszeit von 50 msec überfordert. Diese Datenmengen können nur dann verarbeitet werden, wenn das Datenbanksystem hauptspeicherresidente Dateien ermöglicht.
- Die Anwendung verlangt eine verlustfreie Datenspeicherung bis zu mehreren Monaten. Dabei ist eine manuelle Reorganisation von Datenbeständen nicht zulässig, was reorganisationsfreie Zugriffsstrategien voraussetzt.
- Der operatorfreie 24h-Betrieb setzt Sicherungs- und Recovery-Maßnahmen wie Checkpointing, Before Image, After Image voraus, die jederzeit Datenkonsistenz und -integrität, auch bei parallelen Schreibvorgängen, ermöglichen müssen.
- Mögliche, bisher noch nicht vorgesehene Auswertungen müssen parallel zum laufenden Betrieb interaktiv über die Query Language angefordert werden können.
- Mehrrechnerfunktionen - wie Logbuch, automatisches Update von gespiegelten Platten - müssen verfügbar sein.
- Das Datenbanksystem muß verteilte Zugriffe ermöglichen.
- Automatische Restartfähigkeit.

Die Anforderungen aus der Anwendung und die daraus resultierenden Eigenschaften eines Datenbanksystems sind in B i l d 4 zusammengefaßt.

Anforderungen	Eigenschaften von BAPAS-DB
Ereignisgesteuertes Abspeichern wichtiger Prozeßdaten mit garantierten Zugriffszeiten.	Prioritätsgesteuerte Multitasking-Zugriffe auf Datenbank.
Hohe Prozeßdatenrate (200.000 pro Tag) plus akzeptable Responsezeiten.	Hauptspeicherdateien pro DB-Datei Zugriffsverfahren wählbar.
24-Stunden-Betrieb Daten über Monate verfügbar. Automatische Löschung alter Daten.	Reorganisationsfreie Zugriffsverfahren.
Datenintegrität und Konsistenz auch bei Hardwareausfall. Automatischer Wechsel auf Notsystem.	Erkennen von Fehlerzuständen. Checkpointing, Before- und After Image.
Schnelles Prototyping.	Offenheit von BAPAS-DB. Zugriffsverfahren sind anpaßbar und austauschbar.
Additive Realisierung der Funktion. Erweiterbarkeit des Systems.	Implizite Synchronisation von Anwendertasks in der Datenbank. Sperrung nur bei gleichzeitigem Update-Wunsch eines einzelnen Satzes.

B i l d 4. Gegenüberstellung von Anforderungen und Eigenschaften

Das Echtzeit-Datenbanksystem BAPAS-DB, als Kern des Anwendungssystems eingesetzt, erfüllt alle diese oben definierten Anforderungen (B i l d 5).

Genauere Informationen über BAPAS-DB sind in /1/ und /3/ enthalten.

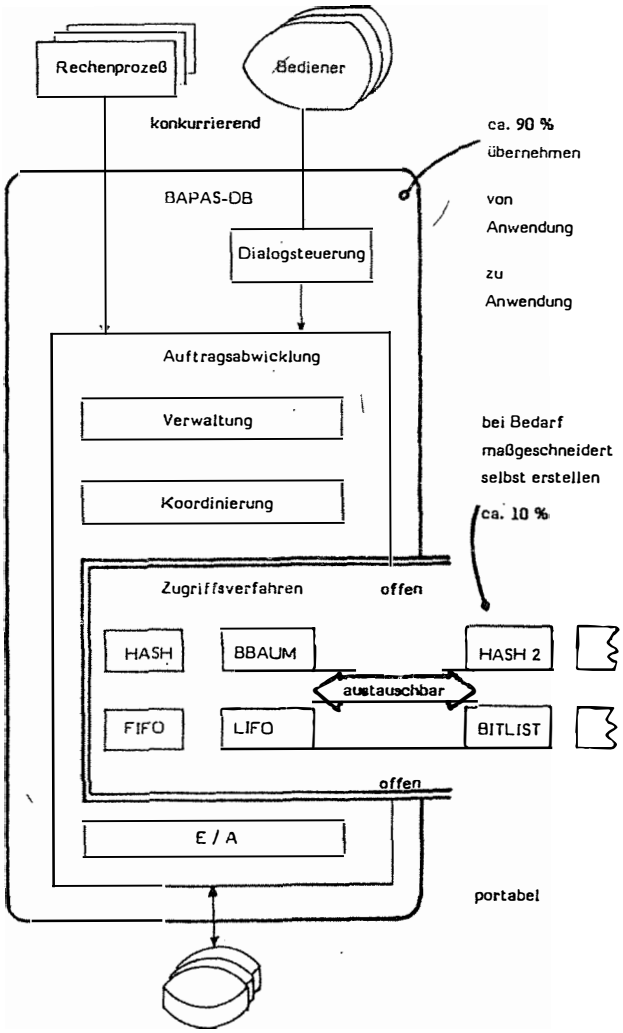


Bild 5. Strukturüberblick über die Komponenten von BAPAS-DB

Literatur

/1/ G o e d e, K.; L a n d w e h r, K.: BAPAS-DB - Ein portables offenes Datenbanksystem für Prozeßrechner. Informatik-Fachberichte 39, Springer-Verlag Berlin, Heidelberg 1980, S. 443 - 452.

/2/ L o c k e m a n n, P.M.; H ä r d e r, T.: Datenhaltung in Echtzeitsystemen. Universität Karlsruhe 1981.

/3/ L a n d w e h r, K.; W i n d a u e r, H.: Database Management System for Process Control. Computer in der Industrie, Proceedings of the European Workshop on Industrial Computer Systems, April 1983, Oldenbourg-Verlag, S. 37 - 46.

/4/ L u i t h l e, J.: Integrierte Fertigungssteuerung durch bereichsübergreifende Datenkommunikation und benutzergerechte Informationsaufbereitung. FhG-Berichte 2 - 85, 1985, S. 32 - 36.

Anschrift des Autors :

Dipl.-Ing. Jürgen Geidies
Werum Datenverarbeitungssysteme GmbH
Glogauer Straße 2 A
D-2120 Lüneburg
Tel.: 04131/53066

Echtzeitdatenbank mit PEARL-Schnittstelle

1 SYSTEMÜBERSICHT

1.1 Generelle Zielsetzung

Das Datenhaltesystem dient dazu, die Führung technischer Prozesse unter harten Realzeitbedingungen und Sicherheitsanforderungen im 24-Stunden-Betrieb zu unterstützen. Dazu gehört neben einer sicheren Datenbestandswartung und -Pflege und einer zugriffsorientierten Ablage der Daten insbesondere ein abgesicherter laufender Betrieb.

1.2 Prinzipielles zu Aufbau und Wirkungsweise

Das System gliedert sich in die Funktionsbereiche:

- Datenbestandswartung und -Pflege,
- Datenbestandsaufbereitung und
- Datenbestandsnutzung im lfd. Betrieb

DATEN-BESTANDS-WARTUNG UND PFLEGE	DATEN-BESTANDS-AUFBEREITUNG	DATEN-BESTANDS-NUTZUNG
INTERPRETER	PEARL	PEARL
DATENZUGRIFFSSYSTEM		

Abb. 1 Funktionsbereiche

Durch Benutzung des KAE-Botschaftenkonzepts als Schnittstelle zwischen den Systemkomponenten ist die Möglichkeit gegeben, je nach Aufgabenstellung und Anlagenkonfiguration, eine Verteilung dieser Funktionen auf mehrere lose gekoppelte Rechner (EPR 1300/ MPR 1300) vorzunehmen (siehe Abb. 2).

Das Datenhaltesystem arbeitet auf der Basis von

- Dateien (DATASETS) und
- Sichten auf Dateien (WORKSETS).

Die Struktur von Dateien ist einfach und übersichtlich, was insbesondere für eine problemlose Wartung und Pflege der Datenbestände von Bedeutung ist (siehe Abb. 3).

Um eine betriebsgerechte Nutzung dieser relativ einfach gehaltenen Dateien zu unterstützen, wird ein Verfahren zur

- Vorauswahl,
- Verknüpfung und
- Ableitung mit optionaler
- Schlüssellistenerzeugung

im Rahmen automatischer Datenbestandsaufbereitung bereitgestellt. Dabei ist wesentlich daß diese Maßnahmen dateiübergreifend angewendet werden können.

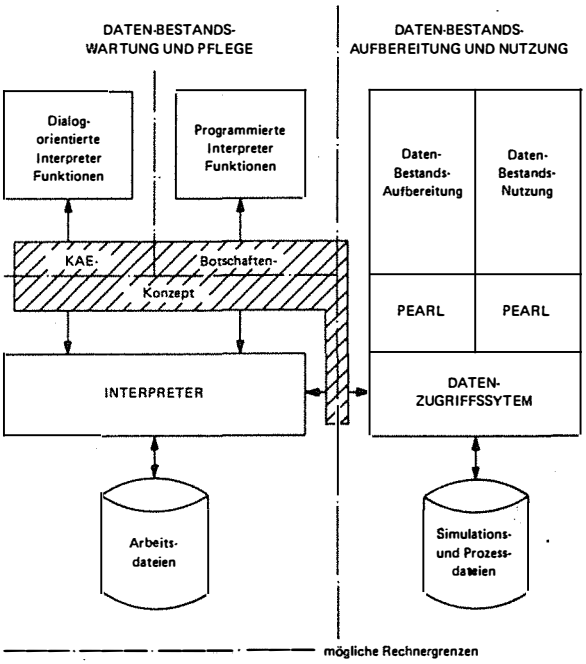


Abb. 2 Darstellung der Systemkomponenten

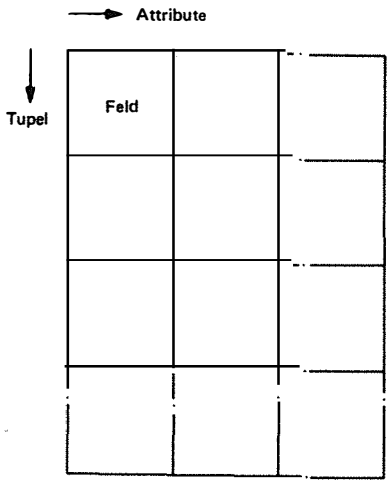


Abb. 3 Aufbau einer Datei

Bei Änderung von Dateien im Rahmen der Datenbestandswartung und -Pflege werden die so erzeugten, verarbeitungsgerechten Sichten auf die Dateien automatisch aktualisiert.

Für den laufenden Prozeß stehen mit solchen Sichten also Zugriffsdateien zur Verfügung, die eine

- realzeitgerechte Datenmanipulation auch für
- komplexe Zugriffe

erlauben (siehe Abb. 4).

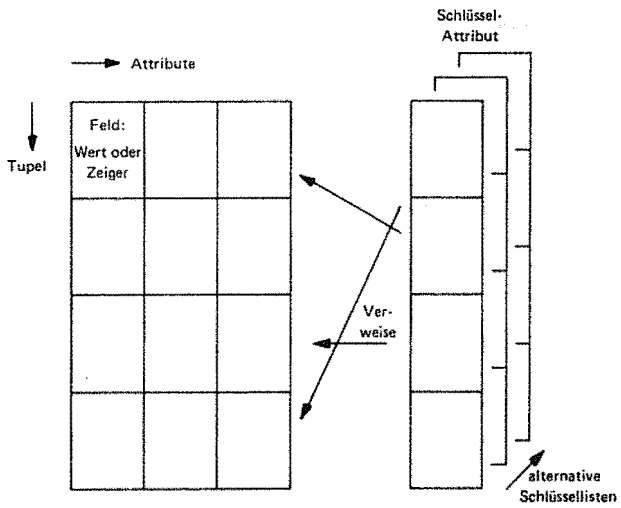


Abb. 4 Aufbau einer Sicht

Bei Aufbau oder Änderung und Erweiterung des Datenmodells hat der Anwender die Möglichkeit,

- Dateitypen

zu benennen und diese hinsichtlich ihrer Struktur zu beschreiben. Dateien desselben Typs werden nachfolgend als Instanzen dieses Dateityps bezeichnet. Sie sind die realen Träger der Datenbestände, welche im Dialog bzw. im Rahmen programmierter Abläufe erzeugt werden können.

In ähnlicher Weise werden bei der Datenaufbereitung

- Sichttypen

benannt und strukturell beschrieben. Darüber hinaus werden die Bezüge zu den Attributen der über diesen Sichttyp erreichbaren Dateien festgelegt. Außerdem werden die optional zu generierenden Schlüssellisten definiert. Bei Sichten desselben Typs wird auch hier von Instanzen eines solchen Sichttyps gesprochen.

Danach beinhaltet die Datenbasis das

- Typenverzeichnis für Dateien und Sichten,

sowie deren

- Instanzen (siehe Abb. 5).

Um den reibungslosen Test- und Simulationsbetrieb parallel zur laufenden Prozeßführung sicher durchführen zu können, wird die Adaption des Datenmodells an den realen Prozeß nach einem abgestuften Konzept vorgenommen:

- **Stufe 1:**
  Hierbei handelt es sich um
  - Datenbestandsmanipulation mit
  - Protokollierung und
  - Plausibilitätsprüfungen,

  welche über
  - Arbeitsdateien auf einem
  - separaten Arbeitsbereich

  vorgenommen werden.
  Dazu können bei Bedarf Daten aus der vorhandenen Datenbasis kopiert werden.

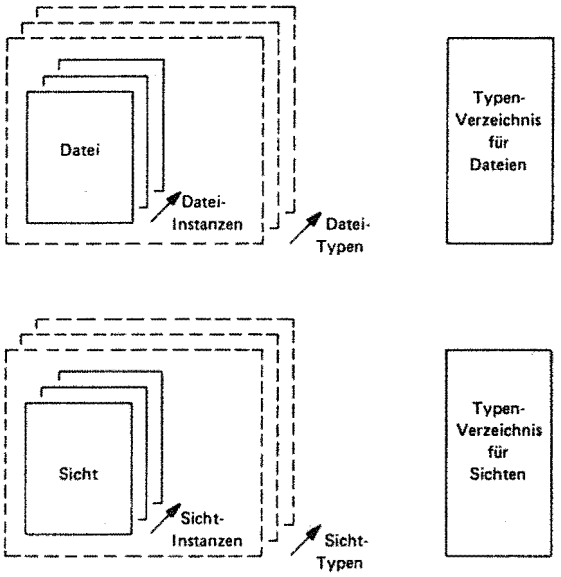


Abb. 5 Aufbau der Datenbasis

- **Stufe 2:**
  Diese beinhaltet das Einbringen von vorgeprüften Arbeitsdateien in die
  - Simulationsebene

  der Datenbasis, ohne die entsprechende
  - Prozeßebene

  zu beeinflussen.
  Dieses wird durch die parallele Führung der Instanzen für Dateien und Sichten auf
  - Prozeß-Ebene und
  - Simulations-Ebene

  ermöglicht (siehe Abb. 6).
  Durch Einführung der Simulationsebene besteht die Möglichkeit, statisch vorgeprüfte Datenbestandsveränderungen im Sinne eines Probebetriebes weiter abzusichern.

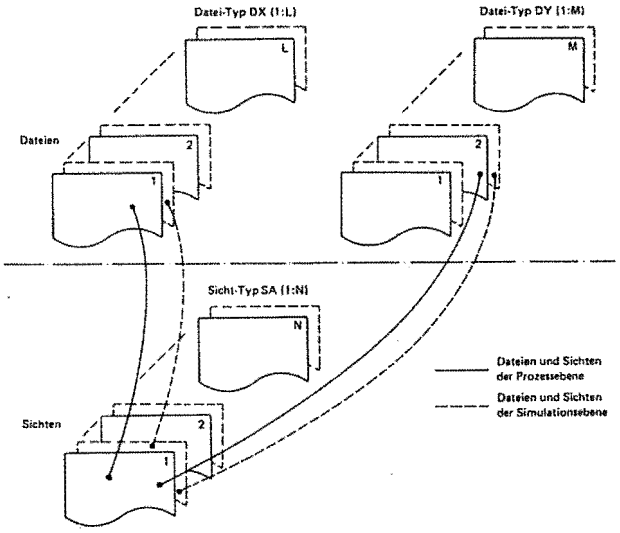


Abb. 6 Dateien und Sichten der Simulations- und Prozeßebene

- **Stufe 3:**
  Mit dieser Stufe wird nach erfolgreichem Probebetrieb die Übernahme der Dateien und Sichten in die Prozeßebene vorgenommen. Dabei werden automatisch die aktuellen Prozeßdaten in die korrespondierenden Felder der neuen Dateien kopiert (siehe Abschnitt 4.1).

Die realzeitgerechte Nutzung der Datenbasis seitens des Anwenders wird durch die Integration von Datenbankanweisungen in die Realzeitsprache PEARL ermöglicht. Hiermit werden die Vorzüge von PEARL bzgl.

- Parallelität,
- Synchronisationsmöglichkeit und
- konsistenter Datenzugriffe

wirksam.

Um dieses Schnittstellenkonzept durchgängig und damit benutzerfreundlich zu halten, wurden die Funktionen der Datenbestandsaufbereitung ebenfalls in PEARL eingebracht.

2 DATENBESTANDSWARTUNG UND -PFLEGE

Die Datenbestandswartung und -Pflege wird über eine Interpreterschnittstelle abgewickelt. Diese kann manuell bzw. über Interpreterkommandos genutzt werden (siehe Abb. 7). Sie umfaßt die Erzeugung, Veränderung, Erweiterung und Protokollierung von Dateien. Von besonderer Bedeutung ist hier die Aufteilung in Prozeß-, Simulations- und Arbeitsdateien. Gemäß dem Sicherheitskonzept dürfen strukturelle und inhaltliche Veränderungen zunächst nur in den Arbeitsdateien erfolgen. Zu diesem Zweck können Kopien von Prozeß- bzw. Simulationsdateien als Arbeitsdateien angelegt werden. Eine Zurücküberführung in die Simulationsdateien ist mittels entsprechender Transportbefehle möglich (siehe Abb. 8).

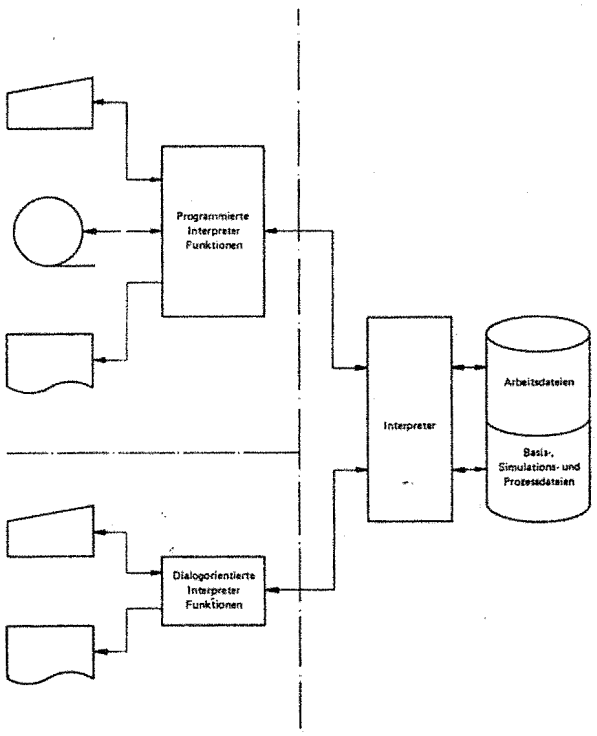
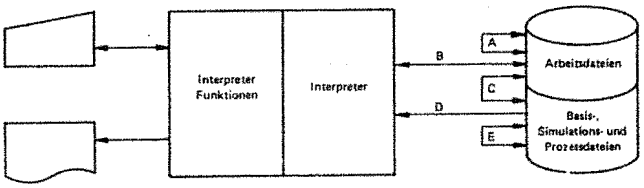


Abb. 7 Datenbestands-Wartung und Pflege



1. Überführen in den lfd. Betrieb (E)
2. Inhaltsverzeichnisse erstellen (B, D)
3. Tupel auslisten (B, D)
4. Datei-Transporte (A, C)
5. Modifikationen von Arbeits-Dateien (A, B)

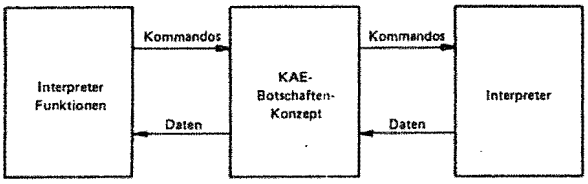
Abb. 8 Datenwege

Hieran kann sich die Simulations- und Testphase anschließen. Nach erfolgreicher Simulation können dann die Simulationsdateien in die entsprechenden Prozeßdateien überführt werden.

Die Interpreter-Funktionen bieten folgende Möglichkeiten:

- Auskunftsfunktion
- Steuerung der Betriebsphasen
- Transport von Dateien
- Listen von Dateien
- Ändern von Dateien
- Entfernen von Dateien
- Definieren von Dateitypen
- Entfernen von Dateitypen
- Setzen von Filtern.

Die Schnittstelle zum Interpreter wird mit dem KAE-Botschaften-Konzept realisiert (siehe Abb. 9). Somit besteht die Möglichkeit, die Interpreteransteuerung von einem separaten Rechner aus zu tätigen (siehe Abb. 2). Die Anweisungen werden in Textform dem Interpreter übergeben und ausgeführt. Die durch diese Anweisungen angeforderten Tupel und Inhaltsverzeichnisse werden binär transportiert. Notwendige Konvertierungen erfolgen durch die Dialogschnittstelle bzw. in den Programmen, welche die programmierbaren Interpreter-Funktionen bedienen.



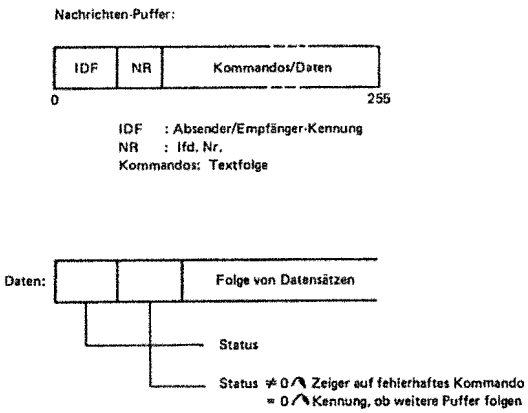


Abb. 9 Datenübertragung

2.1 Dialogbetrieb (Menütechnik)

Die dialogorientierte Benutzung der Interpreterfunktionen erfolgt über Menütechnik (siehe Abb. 10).

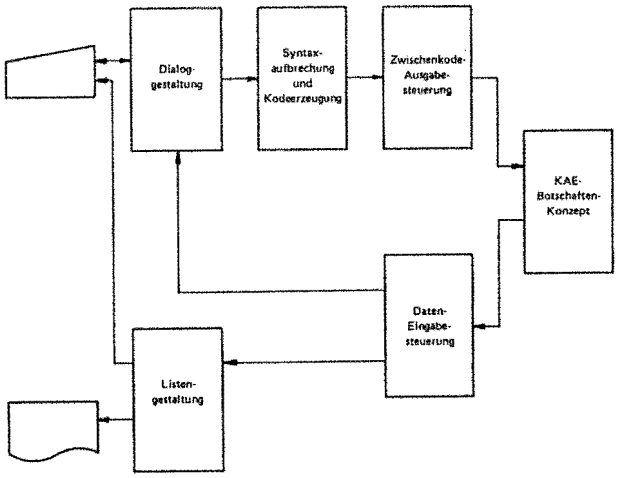


Abb. 10 Dialogorientierte Interpreter-Funktionen

Für die Manipulation von Dateien gilt folgendes:

- Die Manipulation im Sinne von Löschen, Hinzufügen und Modifizieren von Tupeln kann auf Arbeitsdateien, Simulationsdateien und Basisdateien erfolgen.
- Für die Manipulation kann eine beliebige Quelle als Ursprung explizit angewählt werden.
- Wähend einer solchen Manipulation kann per Anwahl auf
  - andere Prozeßdateien,
  - Prozeß- und Simulationsdateien bzw.
  - Prozeß- und Simulationssichten

  lesend zugegriffen werden.

Menü-Bedienung

Die Bedienung des Dialogprogramms wird in einer separaten Beschreibung ausführlich dargestellt.

2.2 Programmierbare Interpreter-Funktionen

Bei der Erstellung und Pflege von Dateien ist es erforderlich, betriebliche Belange zur Konsistenzsicherung der Dateien einfließen zu lassen, die über den datenbanktechnischen Aspekten liegen. Aus diesem Grund wird zusätzlich zu den über Menütechnik bedienbaren Basis-Interpreter-Funktionen eine Programmschnittstelle geschaffen (siehe Abb. 11).

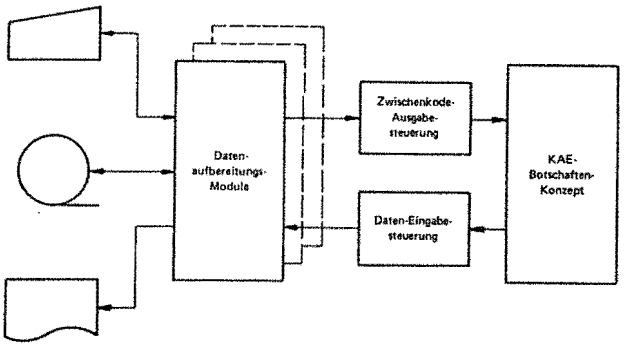


Abb. 11 Programmierte Interpreter Funktionen

Prozeduren in höheren Sprachen (z.B. PEARL) können diese inhaltlichen, zum Teil dateiübergreifenden Überprüfungen vornehmen und nutzen dabei den Anweisungsvorrat, der die Schnittstelle zum Interpreter darstellt.

Neben der Erzeugung und Wartung von Dateien können Prozeduren über diese Schnittstelle auch Protokolle erzeugen. Dialogform und Protokollformate sind dadurch losgelöst von der Datenhaltung in den Prozeduren frei wählbar.

Die hierzu erforderliche Schnittstelle zum Interpreter wird beschrieben in der 'Detailspezifikation für die Datenbestands-Wartung und -Pflege'.

3 DATENBESTANDSNUTZUNG UND -AUFBEREITUNG IN PEARL

3.1 Zielsetzung

Die PEARL-Schnittstelle hat das vorrangige Ziel, Funktionen der Datenhaltung harmonisch in das vorhandene Realzeit-Sprachkonzept einzugliedern. Dabei werden im Interesse des PEARL-Anwenders nicht verträgliche Erweiterungen vermieden. Der Ansatz lautet somit:

- Erhaltung von PEARL-Syntax und -Semantik in allen nicht unmittelbar betroffenen Bereichen und
- Modellierung der Datenhaltefunktionen unter Nutzung des durch PEARL vorgegebenen Sprachkonzepts.

Deshalb beschränken sich die sprachlichen Erweiterungen auf:

- Spezifikation von Dateien und Sichten,
- sowie Deklaration von Ports für Dateien und Sichten,
- Zugriffsfunktionen bezogen auf Dateien und Sichten und
- Blöcke zur Abgrenzung und Koordinierung von Zugriffen.

Die Schnittstelle berücksichtigt die in Abschnitt 1 erläuterte funktionale Trennung von:

- Datenbestandsaufbereitung und
- Datenbestandsnutzung.

Im folgenden wird der Aspekt Datenbestandsnutzung vorangestellt, da dieser aus der Sicht des Anwenders von vorrangigem Interesse ist.

Der Aspekt Datenbestandsaufbereitung, welcher die Formulierung von generativen Abläufen zur Erzeugung neuer Sichten auf vorhandene Datenbestände beinhaltet, betrifft einen kleineren Anwenderkreis (administrative/organisatorische Aufgaben) und wird im Anschluß daran erläutert.

3.2 Datenbestandsnutzung in PEARL

Der zentrale Begriff bei der Nutzung von Datenbeständen ist die Sicht, welche nachfolgend mit dem in PEARL neu eingeführten

- Schlüsselwort WORKSET

angesprochen wird. Vor der Benutzung eines WORKSETS ist dieser gemäß PEARL-Semantik zu spezifizieren. Diese Spezifikation wird gegenüber der entsprechenden globalen Deklaration, welche der Datenbasis entnommen wird, geprüft.

3.3 Spezifikation von Sichten

WORKSET-Typen werden in den PEARL-Modulen durch eine Spezifikation für die weitere Benutzung bekannt gemacht.

```
Beispiel:  SPC wsname( ) WORKSET
                    [ na   KEY CHAR(8),   nb FIXED,
                      nc   KEY FIXED,     nc FLOAT ] ;
```

Die runden Klammern hinter dem Namen von WORKSET-Typen erlauben eine indizierte Auswahl aus der Menge der generierten Instanzen dieses WORKSET-Typs. Die Instanzen eines WORKSET-Typs haben identische Struktur, können aber durch unterschiedliche Anzahl von Tupeln repräsentiert werden.

Die Attribute werden in Form von Strukturkomponenten aufgelistet und müssen einfachen PEARL-Objekten oder Strukturen entsprechen. Das Schlüsselwort KEY gibt an, für welche Attribute Schlüssellisten generiert wurden.

3.3.1 DEKLARATION VON DATAPORTS

Der Datentransport aus den Dateien bzw. Sichten in die PEARL-Prozeduren wird über Puffer organisiert, die Gegenstand der Datendeklaration in PEARL sind und die Schnittstelle zu den externen Datenbeständen darstellen. Diese Datenpuffer, im folgenden Ports genannt, sollen ein Tupel einer Datei aufnehmen und für die Ansprache durch PEARL zugänglich machen. Damit muß die Struktur eines Dataports der Struktur der korrespondierenden Datei oder Sicht entsprechen.

```
DCL port_name WORKSET_PORT [ komponents ];
```

Die Deklaration von Worksetports in Programmen der Datenbestandsnutzung ist aus Gründen der Datensicherheit nur auf Task-, Prozedur- oder Block-Ebene zugelassen.
Bei Eintritt in den Transaktionsblock wird die Verbindung zwischen Port und Datei bzw. Sicht hergestellt.
Alle weiteren Aktionen innerhalb einer Transaktion beziehen sich dann ausschließlich auf Ports.

3.4 Benutzung von Sichten

der Aktionsrahmen, innerhalb dessen in PEARL auf Objekte eines WORKSETS zugegriffen werden kann, ist klar abgesteckt. Der Mechanismus der Abgrenzung erfolgt nach dem PEARL-Blockkonzept. Die Blöcke können geschachtelt werden. Dabei gelten die allgemeinen Regeln aus PEARL für die Schachtelung von Blöcken bezüglich der Zugriffe auf Daten und der Kontrollflüsse. Diese Blöcke können jedoch nicht mit einer GOTO-Anweisung verlassen werden.
Der Block-Eintritt wird markiert durch die alternativen Anweisungen:

```
ENTER_WORKSET (port_name, ws_name(nr)) EXIT label;
TRY_WORKSET (port_name, ws_name(nr)) EXIT label;
TIMEOUT_WORKSET (port_name, ws_name(nr), msec) EXIT
label;
```

Die Benutzung der alternativen Eintrittsformen ist im Zusammenhang mit Synchronisation von WORKSET-Benutzung zu sehen. Normalerweise ist die Benutzung eines WORKSETS jederzeit möglich. Es gibt jedoch Ausnahmesituationen, in denen eine zeitweilige Sperrung von WORKSETS aus Konsistenzerhaltungsgründen notwendig ist.

Hierbei wird das Synchronisationsverhalten wie folgt beeinflußt:

- ENTER_.. bewirkt implizites Warten auf Freigabe der Sperre,
- TRY_.. bewirkt sofortige Rückkehr zum laufenden Rechenprozeß

und

- TIMEOUT_.. bewirkt zeitlich begrenztes Warten auf Freigabe der Sperre.

Alle diese Anweisungen, die einen Blockbeginn kennzeichnen, müssen mit einem EXIT-Befehl abgeschlossen werden. Das LABEL muß dabei außerhalb dieses Blockes liegen. Der EXIT-Befehl entspricht einem GOTO-Befehl, er sorgt nur dafür, das Transaktionsblöcke, die mit dieser Anweisung verlassen werden, durch implizite LEAVE-Anweisungen geschlossen werden.
Die EXIT-Anweisung kann auch innerhalb von Transaktionsblöcken wie GOTO-Anweisung verwendet werden.

Treten Fehler bei dem Versuch der WORKSET-Zuteilung auf, so wird der Programmfluß nicht innerhalb des Blockes fortgesetzt, sondern an der Stelle, die durch das LABEL markiert ist. Das gleiche gilt für den Mißerfolg bei dem Befehl TRY_.. oder einer Zeitüberschreitung bei dem Befehl TIMEOUT_.. . Über eine Statusinformation kann der Benutzer klären, ob die letzte Aktion auf einen WORKSET erfolgreich war oder nicht.

```
DB_STAT (port_name);
```

Diese Funktion liefert ein Ergebnis vom Typ FIXED(15).
War die letzte Aktion auf einen WORKSET erfolgreich, so liefert die Funktion den Wert 0.
Wird auf eine Task, die sich in einem solchen Transaktionsblock befindet, eine TERMINATE-Anweisung ausgeführt, so wird diese Beendigung der Taskaktivität zurückgestellt, bis der Transaktionsblock verlassen wird. Selbstterminierung in einem solchen Block ist nicht zugelassen.
Das Ende eines solchen Transaktionsblockes wird markiert durch die Anweisung:

```
LEAVE_WORKSET ;
```

3.4.1 LESENDER ZUGRIFF AUF DATEIOBJEKTE ÜBER SICHTEN

Soll in einem solchen Transaktionsblock auf die Daten eines WORKSETS nur lesenderweise zugegriffen werden, so sind die oben genannten Anweisungen für den Blockeintritt ausreichend. In diesen Blöcken können Tupel ausgewählt und Daten gelesen, aber nicht verändert werden. Damit ist aber nicht automatisch sichergestellt, daß die Daten, auf die man sich bezieht, für die Dauer der Bearbeitung im laufenden Transaktionsblock konsistent sind. Um den Zeitrahmen für Sperren nicht unnötig groß werden zu lassen, wird dem Anwender die Möglichkeit gegeben, sich zu informieren, ob innerhalb des Transaktionsblockes in den Dateiattributen Veränderungen vorgenommen worden sind (passives Sperrkonzept).

Bei Eintritt in einen Transaktionsblock wird der Änderungsstand jedes Attributes der Bezugsdateien eines WORKSETs kopiert. Die Funktion

```
CHECK_IMAGE (port.attribute [ , port.attribute .. ] )
```

vergleicht nun die augenblicklichen Änderungsstände mit denen, welche bei Blockeintritt kopiert wurden und liefert als Ergebnis eine Bit(1)-Variable (True/False), mit Hilfe derer eine Kontrollflußeintscheidung getroffen werden kann.

Hat eine Änderung stattgefunden, so kann, ohne den laufenden Block zu verlassen, wieder aufgesetzt werden. Dazu dient die Funktion:

```
DEFINE_IMAGE ( port.attribute [ , port.attribute .. ] );
```

welche die Änderungsstände der aufgeführten Attribute neu festlegt.
Verlangt die Anweisung, daß der Rahmen der Konsistenzsicherung weiter gespannt wird, als über einen Transaktionsblock, so können die Änderungszähler der Dateiattribute auch PEARL-Objekten zugewiesen werden.

```
fixedobj = IMAGE_COUNT (port.attribute);
```

Bei Anwendung dieser Funktion muß das dynamische Verhalten des Systems berücksichtigt werden.

3.4.2 ÄNDERN VON DATEIOBJEKTEN

Sollen Datenobjekte in einer Datei verändert werden, so sind die

Attribute, in denen Änderungen vorgenommen werden, bei Blockbeginn anzugeben.

ENTER_WORKSET (port, ws(nr)) UPDATE (attributes) EXIT label;

Schlüsselattribute können nicht verändert werden.

Die Änderungen werden als übliche Zuweisungen zu den Strukturelementen in dem algorithmischen Teil eines Transaktionsblockes formuliert. Die Änderung des entsprechenden Feldes in der Datei wird jedoch erst vorgenommen, wenn der Block verlassen wird, oder wenn eine Positionierung auf ein anderes Tupel des WORKSETs stattfindet. Damit werden überflüssige Systemaufrufe vermieden.

Wird eine vorzeitige Aktualisierung der Bezugsdateien erforderlich, so kann dieses über die Anweisung:

UPDATE_TUPEL (port);

explizit erfolgen.

Ist die Änderung eines Feldes abhängig von gesicherten Daten in anderen Attributen, so ist eine Aktion erforderlich, die Konsistenzüberprüfung und Feldzuweisung in einer geschlossenen Anweisung erlaubt:

CHECK_IMAGE_UPDATE (port.attributes);

Diese Anweisung bewirkt, wie die Anweisung CHECK_IMAGE(..), die Konsistenzüberprüfung der angegebenen Attribute. Hat diese Aktion ergeben, daß Veränderungen in dem Zeitrahmen der Transferblockbearbeitung nicht stattgefunden haben, so findet ein 'update' der Dateien mit den veränderten Werten des aktuellen Tupels statt.

3.4.3 SPERREN VON DATEIATTRIBUTEN

Neben dieser passiven Form des Erkennens von Datenmanipulationen ist auch eine aktive Form des Schutzes vorgesehen. Diese Methode birgt aber Risiken.

ENTER_WORKSET (port, ws(nr)) EXCLUSIV (attributes) EXIT label;

Die EXCLUSIV-Benutzung eines Attributes umfaßt die Benutzungsart UPDATE.

Die Sperre beschränkt sich nicht auf die Attribute der Tupel eines WORKSETS, sondern gilt für die betroffenen Attribute der Datei. Das heißt, daß die Bearbeitung von Transaktionsblöcken zweier Tasks sequentialisiert wird, falls auf dasselbe Attribut einer Datei in der Form UPDATE und EXCLUSIV zugegriffen werden soll.

3.4.4 ZUGRIFFSVERHALTEN

Wegen der großen Datenmengen muß davon ausgegangen werden, daß alle Dateien auf einem Massenspeicher (Plattenspeicher) gehalten werden. Bei der Anwahl eines Tupels wird dieses im normalen Arbeitsmodus also von dem externen Speichermedium gelesen. Damit muß für einen solchen Vorgang die mittlere Zugriffszeit in Anrechnung gebracht werden. Dieses Zeitverhalten ist nicht in allen Arbeitsphasen akzeptabel. Für die Beschleunigung des Zugriffs auf Dateiobjekte der Betriebsdateien wird nun ein abgestuftes Konzept bereitgestellt.

Zunächst kann durch ein günstiges Verhältnis von Blocklänge zu Tupellänge die Plattenaktivität verringert werden, und damit das Zeitverhalten verbessert werden.

Dann kann das Datenverwaltungssystem dazu veranlaßt werden, innerhalb eines Transaktionsblockes die Tupel schon im Vorgriff bereitzuhalten. Ob dieses Verfahren einen positiven Einfluß hat, hängt sehr stark vom Algorithmus ab. Wird ein WORKSET sequentiell nach der Anordnung seiner Tupel bearbeitet, können die jeweils nächsten Tupel bereitgestellt werden. Dazu ist die Anweisung für den Blockeintritt zu erweitern:

ENTER_WORKSET .. .. PREFETCH EXIT label;

Orientiert sich der Algorithmus an der aufsteigenden Folge in einer Schlüsselliste, so müßte die Anweisung lauten:

ENTER_WORKSET .. .. PREFETCH (attribute) EXIT label;

Als umfassendste Möglichkeit kann einer Instanz eines Dateityps oder Worksettyps in den Prozeßdateien das Attribut 'RESIDENT' gegeben werden. Diese Entscheidung kann auf das ganze Systemverhalten von so großer Bedeutung sein, daß dies nicht als Programmierhilfe angesehen werden kann, sondern beim Systemdesign berücksichtigt werden muß. Dieses Attribut kann nur über die Dialogschnittstelle vergeben werden (siehe auch 4.)

3.5 Orientierung in Sichten

Der physikalische Transfer von Daten in den Port findet statt, wenn ein bestimmtes Tupel des WORKSETS ausgewählt wurde.

Die Anwahl der Tupel kann erfolgen über

- den Platz im WORKSET
- den Platz in der Schlüsselliste
- den Wert eines Attributes.

3.5.1 DIE ORDNUNGSZAHL ALS AUSWAHLKRITERIUM

Die Erstellung von WORKSETS ist ein prozeduraler Vorgang, bei dem die Tupel nach einem vorgegebenen Algorithmus – dem Generator-Programm – aus den Tupeln von Bezugsdateien abgeleitet werden. Die dabei entstehende Anordnung von WORKSET-Tupeln ist solange gültig, solange die zugrundeliegenden Bezugsdateien sich im Rahmen von Datenbestandswartung und -Pflege nicht strukturell ändern.

Die Platznummer eines Tupels in dieser Anordnung kann als Auswahlkriterium in einem Transaktionsblock verwendet werden. Die Platznummern sind definiert von 0 bis CARDINAL_NUMBER(..) – 1.

SELECT_TUPEL (port, oz);

Existiert ein Tupel der Ordnungszahl 'oz', so steht dieses Tupel auf der PEARL-Ebene zur Verfügung. Existiert kein Tupel dieser Ordnungszahl, so findet kein Datentransfer statt, und der Status dieser Aktion ist ungleich Null.

Um in Schleifen alle Tupel eines WORKSETS erreichen zu können, ohne einen fehlerhaften SELECT_TUPEL ausführen zu müssen, kann die Anzahl der Tupel eines WORKSETS mit Hilfe der folgenden Anweisung abgefragt werden:

cnr = CARDINAL_NUMBER (ws(nr));

Diese Funktion liefert ein Ergebnis vom Typ FIXED(15).

3.5.2 SUCHE BESTIMMTER TUPEL

Neben der Anwahl eines Tupels über die Ordnungszahl kann auch nach einem bestimmten Wert in einem Attribut eines WORKSETS gesucht werden.

FIRST_TUPEL (port.attrib, x);

Dieser Suchvorgang kann erheblich beschleunigt werden, wenn für das Attribut eine Schlüsselliste existiert. Ist der Wert 'x' in dem Attribut des WORKSETS gefunden, so steht dieses Tupel zur Verfügung wie in der Funktion SELECT_TUPEL.

Von der aktuellen Position im WORKSET aus, die durch die letzte erfolgreiche Tupelauswahl bestimmt ist, kann in dem Rest des WORKSETS nach einem Wert gesucht werden.

NEXT_TUPEL (port.attrib, x);

Ist die Auswahl eines Tupels nicht über die Ordnungszahl erfolgt, so kann man die Ordnungszahl im WORKSET ermitteln:

nt = TUPEL_NR (port);

Existieren Schlüssellisten zu einem WORKSET, so kann man neben der Ordnungszahl des aktuellen Tupels auch die Ordnungszahl, die dieses Tupel in einem Schlüssel einnimmt, abfragen.

nk = TUPEL_NR (port.attrib);

Dabei muß 'attrib' ein Schlüsselattribut sein.

Mit dieser Möglichkeit kann man sich über eine Schlüsselliste bewegen wie über einen WORKSET.

SELECT_TUPEL (port.keyattrib, nk);

Damit wird das Tupel ausgelesen, das an der Stelle ;nk' in der sortierten Schlüsselliste steht.

3.6 Die Generierung von Sichten

3.6.1 MODULE ZUR SICHTEN-GENERIERUNG

Die Generierung von WORKSETS ist in die Gastsprache PEARL eingebettet. Ziel ist es, die Datenmengen zu reduzieren und verarbeitungsgerecht zu strukturieren.

PEARL-Module, die die Prozeduren zur WORKSET-Erstellung beinhalten, sind besonders gekennzeichnet. Sie enthalten in der Kopfzeile einen Zusatz:

MODULE wsg DB_GEN;

So gekennzeichnete Module dürfen nicht in normale PEARL-Programmpakete eingebunden werden. Die WORKSET-Generierung ist ein sequentieller Prozeß, deshalb darf nur eine Task daran beteiligt sein: die Main-Task.

3.6.2 BEZUG AUF DATEIEN

Der Bezug zu den Dateien wird, ähnlich wie die Peripherie- und System-Verbindungen im Systemteil, in einem besonderen Deklarationsblock behandelt.

DATA_BASE;

Darauf folgt die Spezifikation der vorhandenen Dateien, die für die Generierung der WORKSETS benötigt werden:

SPC ds_name DATA_SET [komponents..];

Die Attribute der Datei werden in der Schreibweise aufgelistet wie sie für Strukturkomponenten üblich ist. Die Reihenfolge ist durch die Anordnung der Attribute bei der Generierung der Dateien vorgegeben. Die Komponenten können vom Datentyp FIXED, FLOAT, BIT, CHAR, CLOCK, DURATION oder STRUCT sein.
Bei der Definition von Dateitypen werden die Attribut-Zugriffstypen beschrieben. Ist ein Attribut auf dieser Ebene als MAINKEY oder KEY definiert worden, so muß in der Spezifikation und der entsprechenden Deklaration des dazugehörenden Ports dieses Attribut als KEY spezifiziert bzw. deklariert werden.

Die Deklaration der Ports für die Dateibenutzung hat folgende Form:

DCL port_name DATASET_PORT [komponents];

3.6.3 DEKLARATION VON SICHTEN

Die WORKSETS, die generiert werden sollen, werden durch eine Deklarationsanweisung bekannt gemacht. Die WORKSETS sind Arbeitsdateien, die sich dem Anwender darstellen wie normale Dateien, und die Attribute eines Tupels werden deshalb auch ähnlich beschrieben wie in der Spezifikation der Originaldateien. Um Konsistenz in den Datensätzen zu gewährleisten, wird jedoch ein Unterschied gemacht, denn einzelne Felder der Bezugsdateien können in verschiedenen WORKSETS vorkommen. Sind diese Felder vom Zugriffstyp 'variant', so müssen an Stelle der Werte Zeiger auf diese Felder in die WORKSETS übernommen werden.

DCL ws_name WORK_SET [attribute, ...];

Die Attribute sind 'pointer' oder 'values'. Die 'values' werden beschrieben wie Strukturkomponenten, die jedoch zusätzlich das Attribut KEY enthalten dürfen.

att_name KEY FIXED

Solche Attribute werden in einer separaten Schlüsselliste abgespeichert. Nach Beendigung der WORKSET-Generierung werden die Objekte dann entsprechend ihrer binären Darstellung aufsteigend sortiert.

Die 'pointer' werden mit einem Pfeil auf das Attribut der Datei beschrieben wie z.B.

att_name –> dsb.name CHAR(12)

Beispiel: DCL WSA WORK_SET [NR KEY FIXED,
NAM –> DSA.NAME CHAR(8),
BTS –> DSB.BTS FLOTAT];

Die Deklaration der Datenports für die WORKSETS wurde im Kapitel 3.4.1 beschrieben.

3.6.4 ORIENTIERUNG IN EINER DATEI

Um einen WORKSET aus einer oder mehreren Dateien aufbauen zu können, muß man sich in einer Datei orientieren können und auf die Objekte ihrer Tupel zugreifen können.

Zunächst müssen Beginn und Ende der Benutzung einer Datei abgesteckt werden:

ENTER_DATASET (ds_port, ds[(nr)]) EXIT label;
LEAVE_DATASET ;

Die Semantik dieser Blockanweisungen entspricht der aus dem Abschnitt 3.4. Auf die Synchronisationsvorsätze TRY_.. und TIME-OUT_.. kann verzichtet werden.

Die Zugriffsfunktionen, wie sie im Abschnitt 3.4 erklärt wurden, gelten hier mit wenigen Einschränkungen auch. Da es keine Schlüssellisten für die Dateien gibt, sind die Funktionen für die Tupelauswahl folgende:

select_TUPEL (ds_port, tnr);	Auswahl nach Ordnungszahl
FIRST_TUPEL (ds_port.att, val);	Erstes Tupel mit dem Wert 'val' im betreffenden Attribut

NEXT_TUPEL (ds_port.att, val); Nächstes Tupel mit dem Wert 'val' im betreffenden Attribut, vom aktuellen Tupel aus.

3.6.5 AUFFÜLLEN EINER SICHT

Um Tupel in einem WORKSET ablegen zu können, müssen durch das System Verwaltungsblöcke angelegt werden. Dies geschieht durch die Anweisung

CREATE_WORKSET (ws_port, ws_name [(nr)]) EXIT label;

Sind mehrere Instanzen für diesen WORKSET vorgesehen, so gibt 'nr' die Nummer dieser Instanz wieder.

Das Ende der Generierphase wird durch die Anweisung

CLOSE_WORKSET ;

markiert. Nach Ausführung dieser Anweisung können keine Tupel mehr zu diesem WORKSET hinzugefügt werden.

3.6.6 FEHLERERKENNUNG BEI DER GENERIERUNG

Für alle Dateizugriffe, die für einen WORKSET veranlaßt wurden, gibt es Statusrückmeldungen. Dieser Status steht dann durch die Funktion

value = DB_STAT (ws_port);

zur Verfügung. Es wird nur der Status der letzten Aktivität gehalten. Die Funktion liefert einen Wert vom Typ FIXED(15).

3.6.7 ERSTELLUNG EINES SICHTEN-TUPELS

Felder, die in der Form der 'values' deklariert wurden, werden durch Zuweisungen vorbelegt, wie sie in PEARL üblich sind, d.h. es muß Typverträglichkeit gewährleistet sein.

WSPORT.ATTRX = DSPORT.ATTRY;

Felder, die als 'pointer' deklariert wurden, sollen auf ein bestimmtes Feld der Bezugsdatei zeigen. Das aktuelle Tupel in der Datei, das gerade im Zugriff ist, liefert die Tupelnummer für diese Zuordnung:

port.nam = TUPEL_NR (ds_port);

Ist das Tupel auf diese Weise erstellt, wird es mit der Anweisung

ACCEPT_TUPEL (ws_port);

in die fortlaufende Position im WORKSET abgespeichert.

3.6.8 DATEIEN ALS SICHTEN

Will man in PEARL eine Datei als Ganzes im Zugriff haben, so muß man nicht tupelweise kopieren, um schließlich einen WORKSET zu erhalten, der ein identisches Abbild der Datei ist. Diese Redundanz wird vermieden, es muß aber ein systemgerechter Verwaltungsblock für z.B. Synchronisations- und Sperrmechanismen angelegt werden. Ist jedoch mindestens ein Attribut der Datei zusätzlich als Schlüssel deklariert worden, so müssen hier natürlich auch die Schlüssellisten gebildet werden. In jedem Fall reduziert sich die WORKSET-Generierung auf eine Anweisung:

IDENTICAL_WORKSET (ws(nr), ds(nr));

Diese Anweisung ersetzt das CREATE_WORKSET ( ) ...; und CLOSE_WORKSET. Für diesen Vorgang erübrigt sich die Deklaration von Ports.

3.6.9 EINTRAGUNGEN IN DATEIEN

Es kann im Rahmen der WORKSET-Generierung zweckmäßig sein, auf Attribute der Bezugsdateien schreibend zuzugreifen.

Dieses ist nur für solche Felder gestattet, welche per WORKSET-Benutzung nicht weiter verändert werden können. Diese tragen das Zugriffsattribut INVARIANT (siehe Abschnitt 1) und stellen automatisch erzeugte Schlüsselinformationen dar.

Als Beispiel hierfür läßt sich die automatische Markierung der Teilnetzzugehörigkeit eines Betriebsmittels im Rahmen einer topologischen Analyse der Betriebsmitteldatei für ein elektrisches Netz anführen.
Die Änderungen können ähnlich wie bei WORKSET-Benutzung über die Anweisungen:

ENTER_DATASET (ds_port, ds) UPDATE (attribute) EXIT label;

SELECT_TUPEL (ds_port, oz);
ds_port.att = expression;
.....
LEAVE_DATASET;

erfolgen.

Die Attribute, die als KEY deklariert wurden, können nicht verändert werden.

3.6.10 ÄNDERUNGSZUSTAND EINER DATEI

Ist auf der Ebene der Datenbestandswartung und -pflege in einem Dateityp eine Änderung vorgenommen worden, so führt dies zu einem Anlauf der WORKSET-Generatoren, die diesen Dateityp behandeln. Da der Lauf eines WORKSET-Generators das Neuerstellen aller WORKSET-Instanzen beinhaltet, wird unter Umständen ein überflüssiger Aufwand getrieben, wenn die Änderung sich nur auf eine Instanz bezieht. Mit der Funktion

MODIFICATION_LEVEL (ds(nr))

wird dem Programmierer die Möglichkeit geboten, den Kontrollfluß in seinem Generator so zu beeinflussen, daß unnötige Generierungen entfallen.
Die Funktion liefert ein Ergebnis vom Typ FIXED(15).

Der Funktionsaufruf liefert bei der Angabe einer existierenden Dateiinstanz folgende Werte:

0 – die Instanz ist seit dem letzten Generatorlauf unverändert geblieben.

1 – die Instanz wurde verändert.
Der Datensatz der Instanz ist unverändert erhalten geblieben, es wurden nur Tupel am Ende der Instanz hinzugefügt.

2 – die Instanz wurde verändert.
Es wurden beliebig Tupel verändert oder hinzugefügt oder gelöscht.

3.7 Beispiel für Erzeugung und Benutzung einer Sicht

Um ein Gefühl dafür zu vermitteln, wozu und in welcher Weise Sichten auf Dateien erzeugt und verwendet werden, sind im folgenden die zwei PEARL-Moduln aufgeführt:

- GSS zur Erzeugung einer Sicht (Datenbestandsaufbereitung)
- CSS zur Nutzung dieser Sicht (Datenbestandsnutzung)

Das Beispiel geht davon aus, daß in einer Betriebsmittelliste 'DBM' für alle Betriebsmittel eines elektrischen Netzes (Leistungsschalter, Kabeltrenner, Kabelerder, ...) folgende Attribute aufgeführt sind:

- TYP des Betriebsmittels
- KNOTEN1 als topologische Ortsangabe
- KNOTEN2 als topologische Ortsangabe
- STATUS als laufender Betriebszustand.

Um für die Aufgabenstellung:

- Überwachung aller Sammelschienen

eine benutzergerechte Sicht auf die umfangreiche Datei zu schaffen, wird der

- WORKSET WSS als Vorauswahl aller Betriebsmittel vom Typ Sammelschiene

generiert, welche dann zur zyklischen Überwachung im 10 Sekundenrahmen genutzt wird.

MODULE: GSS DB_

```
MODULE: GSS DB_GEN;  /* SAMMEL-SCHIENEN-WORKSET-GENERATOR */

/* GENERIERT SICHT 'WSS' AUF BETRIEBSMITTEL DES TYPS 'SAMMELSCHIENE'
   AUS DER BETRIEBSMITTELDATEI 'DBM' MIT DEN KOMPONENTEN:
      . KNOTEN DER SAMMELSCHIENE ALS WERT
      . STATUS DER SAMMELSCHIENE ALS ZEIGER IN DIE BETRIEBSMITTELDATEI
*/

DATA_BASE; /* ANSCHLUESSE AN DAS DATENHALTESYSTEM */

SPC DBM DATASET A ( TYP, KNOTEN1, KNOTEN2 ) KEY FIXED, STATUS FIXED Ü;

DCL WSS WORKSET A KNOTEN FIXED, STATUS -> DBM.STATUS FIXEDÜ;

PROBLEM;   /* PEARL PROBLEM-TEIL */

MAIN: TASK;

DCL ( OK, SAMMELSCHIENE )  FIXED INIT( 0, 1 );

DCL BM DATASET_PORT A ( TYP, KNOTEN1, KNOTEN2 ) KEY FIXED, STATUS FIXED Ü;

DCL SS WORKSET_PORT A KNOTEN FIXED, STATUS -> DBM.STATUS FIXEDÜ;

CREATE_WORKSET ( SS, WSS ) EXIT ERROR;
ENTER_DATASET  ( BM, DBM ) EXIT ERROR;
FOR I FROM 0 TO (CARDINAL_NUMBER(BM) -1) REPEAT

  SELECT_TUPEL ( BM, I );

  IF DB_STATUS ( BM ) EQ OK THEN  IF BM.TYP EQ SAMMELSCHIENE
                                  THEN SS.KNOTEN = SSKNOTEN ( BM );
                                       SS.STATUS = TUPEL_NR ( BM );
                                       ACCEPT_TUPEL ( SS );
                                  FIN;
                            FIN;
END;

LEAVE_DATASET;
CLOSE_WORKSET;

ERROR: /* Fehler z.B. ueber das Queueing-System melden */ ;

END;
PROC: SSKNOTEN

( BM DATASET_PORT A ( TYP, KNOTEN1, KNOTEN2 ) KEY FIXED, STATUS FIXEDÜ )

  RETURNS (FIXED);

DCL DUMMY FIXED INT ( -1 );

IF BM.KNOTEN1 NE DUMMY THEN RETURN ( BM.KNOTEN1 );
                       ELSE RETURN ( BM.KNOTEN2 );
                       FIN;
END;
MODEND;
```

Abb. 12 Programmbeispiel, Teil 1

```
MODULE: CSS;          /* SAMMEL-SCHIENEN-UEBERWACHUNG */

/* BENUTZT SICHT 'WSS' AUF BETRIEBSMITTEL DES TYPS 'SAMMELSCHIENE'
   AUF DIE BETRIEBSMITTELDATEI 'DBM' MIT DEN KOMPONENTEN:
      . KNOTEN DER SAMMELSCHIENE ALS WERT
      . STATUS DER SAMMELSCHIENE ALS ZEIGER IN DIE BETRIEBSMITTELDATEI
*/

PROBLEM;   /* PEARL PROBLEM-TEIL */

SPC WSS WORKSET A KNOTEN FIXED, STATUS FIXED U;

SPC SSERROR ENTRY ( FIXED, FIXED, FIXED );

CHECK: TASK;

DCL OK FIXED INIT( 0 );

DCL SS WORKSET_PORT A KNOTEN FIXED, STATUS FIXED U;

REPEAT AFTER 10 SEC RESUME;

  ENTER_WORKSET ( SS, WSS ) EXIT ENTERROR;

  FOR I FROM 0 TO (CARDINAL_NUMBER(SS) -1) REPEAT

    SELECT_TUPEL ( SS, I );

    IF SS.STATUS NE OK THEN CALL SSERROR ( I, SS.KNOTEN, SS.STATUS );

                          FIN;

  END;

LEAVE_WORKSET;

END;

ENTERROR: /* Fehlerreaktion */ ;

END;

MODEND;
```

Abb. 13 Programmbeispiel, Teil 2

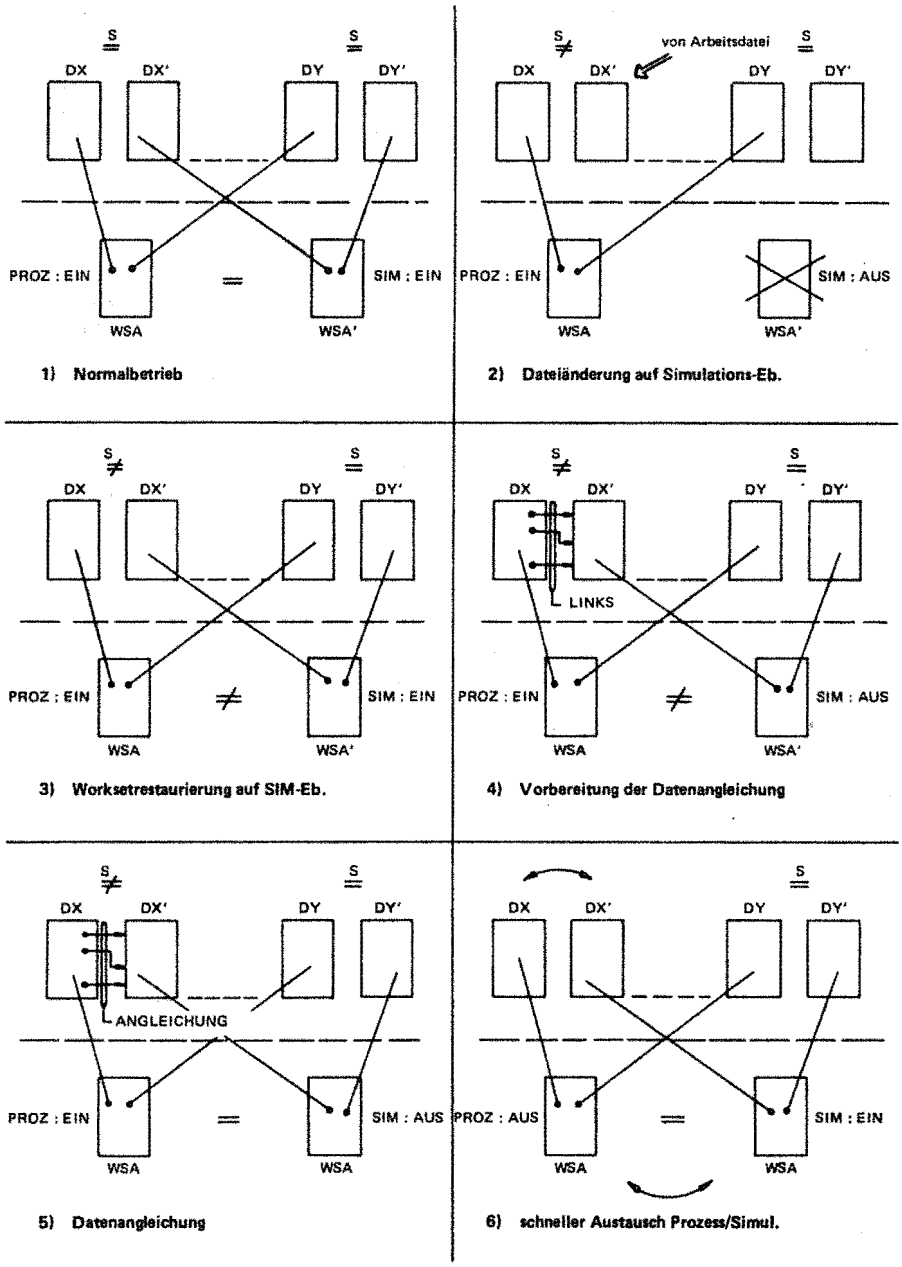


Abb. 14 Phasen der Datenmodell-Anpassung

4 REALISIERUNGSASPEKTE

Im laufenden Abschnitt werden einige wesentliche Aspekte bezüglich der Realisierung von Verfahren und Schnittstellen erklärt, ohne auf die Details der Implementierung einzugehen.

4.1 Einbringen struktureller Änderungen in die Datenbasis

Um das in Abschnitt 1 angesprochene Verfahren in Bezug auf seine dynamischen Aspekte zu verdeutlichen und insbesondere das

- stoßfreie Einbringen von Veränderungen in die Prozeßebene

zu demonstrieren, werden die Phasen dieses Ablaufes unter Bezug auf Abb. 14 im einzelnen dargestellt:

1) Der WORKSET WSA, welcher auf Instanzen der Dateitypen DX bis DY bezogen ist, existiert in identischer Form sowohl für die Prozeßebene als auch für die Simulationsebene. Die Dateiinstanzen sind strukturell gleich, d.h. sie besitzen dieselben Tupel, können aber unterschiedliche Inhalte bezüglich variabler Attribute haben. In diesem Zustand ist sowohl Prozeßführung als auch Simulation möglich.

2) Nach Sperren der Simulation wird eine der Dateiinstanzen, auf welche sich die WSA bezieht, durch ein im Rahmen der Datenbestandswartung und -Pflege strukturell verändertes Image ausgetauscht. Damit ist WSA für die Simulationsebene unbrauchbar geworden. Die Prozeßebene wird jedoch nicht gestört.

3) Durch Anforderung per Dialog wird die automatische WORKSET-Generierung auf Simulationsebene vorgenommen. Danach kann die Simulationsebene wieder benutzt werden. Während der Phase der WORKSET-Regenerierung wird die Prozeßebene nicht blockiert.

4) Zur Vorbereitung des Datenangleichs der Simulationsebene an die Prozeßebene bezüglich der strukturell geänderten Instanz von DX wird ohne Blockierung der Prozeßebene eine Verbindung der korrespondierenden Tupel aus DX und DX' hergestellt.

5) Unter Sperren der Simulationsebene kann jetzt die Datenangleichung vorgenommen werden. Diese geschieht auf der Basis
   - routinemäßiges Kopieren aller korrespondierenden Tupelinhalte

   und
   - Mitführen aller laufenden Datenveränderungen in korrespondierenden Tupeln der Simulationsebene.

   Eine Blockierung der Prozeßebene liegt nicht vor.

6) Austausch von WSA mit WSA' und DX mit DX' im Sinne einfacher Zeigervertauschung. Dieser kurze Augenblick wird unter Sperren aller WORKSETS der Prozeßebene, welche sich auf DX beziehen, abgesichert.

4.2 Residente Dateien

Im Abschnitt 3 wurden Möglichkeiten angesprochen, das Zeitverhalten des Systems in der Phase der Prozeßführung positiv zu beeinflussen. Als durchgreifendstes Mittel wurde dabei die Möglichkeit vorgestellt, eine Datei als ganzes resident zu machen. Da mit dieser Eigenschaft, je nach Umfang der Dateien, große Teile des Arbeitsspeichers gebunden werden, muß dieses Attribut mit Bedacht vergeben werden. Mit Hilfe einer speziellen Benutzerkonfiguration kann über die Dialogschnittstelle die Instanz einer Prozeßdatei oder eines Worksets in den Arbeitsspeicher kopiert werden. Diese 'residency' bleibt erhalten, bis sie über die gleichen Schnittstellen wieder abgewählt wird, auch wenn ein Austausch einer Dateiinstanz vorgenommen werden muß.

4.3 Ausfall der Datenbasis

Durch Installation einer zweiten Datenbasis auf einem zweiten Rechnerwird es ermöglicht, bei Ausfall des betriebsführenden Rechner z.B. einen Notbetrieb auf dem zweiten Rechner zu führen. Die Daten-Konsistenz zwischen den Datenbasen obliegt dem Benutzer, um z.B. einen parallelen Notbetrieb mit einem reduzierten Datenmodell durchzuführen. Die Anwahl der Datenbasis erfolgt über die Interpreter-Schnittstelle. Durch entsprechende Anwahl können also alle Nachführungen des Datenbestandes bei laufendem Betrieb der Prozeßebene sowohl auf dem betriebsführenden Rechner als auch auf dem Betriebsrechner erfolgen.

Autor:

Klaus Odenwald, Dipl. Inf. und Dipl. Ing.
Krupp Atlas Elektronik GmbH
Sebaldsbrücker Heerstraße 235
2800 Bremen
Telefon (04 21) 4 57-29 88

Ein rechnendes Gedächtnis als Prozessrechner

Helfried Broer, Braunschweig

Zusammenfassung

Dieses Papier beschreibt eine Version des Rechnenden Gedächtnisses. Die Rechnenden Gedächtnisse sind Mitglieder einer breiten Klasse von berechnungsuniversellen Rechenanlagen, die man Lebensräume für Aktoren nennen kann. In einem Lebensraum für Aktoren erscheint die Hardware in einem neuen Licht: Die Hauptaufgabe der Hardware besteht in einem Lebensraum für Aktoren erstens aus der Aufgabe, Material für die Konstruktion und die Isolierung von Aktoren bereit zu stellen, und zweitens aus der Aufgabe, ein Medium für die Kommunikation der Aktoren zur Verfügung zu stellen. Auf diese Weise eröffnet ein Lebensraum für Aktoren neue Aussichten auf parallele Aktivitäten und auf dynamische Konfigurierbarkeit.

Schlüsselwörter: Architektur, Homogenität, Parallelität, Konfigurierbarkeit

Summary

This paper describes one Computing Memory version. Computing Memory is one member of a broad class of universal computers which might be called Living Spaces for Actors. In a Living Space for Actors hardware appears in a new light: The main mission of hardware is composed firstly of the task to provide material for the construction and the isolation of Actors and secondly of the job to erect a communication medium to enable communications between different Actors. This way a Living Space for Actors opens new prospects for parallel activity and dynamic configurability.

Key words: architecture, homogeneity, parallelism, configurability

Einleitung

In sehr vielen Anwendungsbereichen sind heute Echtzeitproblemstellungen vorhanden. Echtzeit-Datenverarbeitungsanlagen unterscheiden sich in einer ganzen Reihe von Punkten von den traditionellen Datenverarbeitungsanlagen, die in der allgemeinen Business-Informatik eingesetzt werden. Die wichtigsten Unterscheidungsmerkmale ergeben sich durch die spezifischen Betriebsbedingungen der Echtzeitsysteme. Diese spezifischen Betriebsbedingungen schlagen sich zum Beispiel nieder in

- der hohen Anforderung an Zuverlässigkeit und Sicherheit.
  Ein wesentliches Charakteristikum (um nicht zu sagen: das wesentlichste Charakteristikum) aller Echtzeitsysteme ist die im Vergleich zu den Datenverarbeitungsanlagen im konventionellen Rechenbetrieb notwendige hohe Anforderung an Zuverlässigkeit und Sicherheit. Dies wird besonders deutlich, wenn zum Beispiel an die über die Existenz der gesamten Zivilisation entscheidenden rechnergestützten Verteidigungssysteme gedacht wird.

- der Zeitabhängigkeit der Programmabläufe.
  Im Gegensatz zu den Datenverarbeitungsanlagen, die im konventionellen Rechenbetrieb eingesetzt werden, arbeiten Echtzeitsysteme nur dann korrekt, wenn alle Programme ihre vorgegebene Funktion innerhalb eines festgelegten Zeitintervalls durchführen. Eine besondere Schwierigkeit stellen dabei die Zeitverfälschungen dar, die sich durch

das Eigenzeitverhalten der Betriebssysteme ergeben.

- der Parallelität der Schnittstelle Rechner / technisches System.
  Die Parallelität dieser Schnittstelle ist auf die Unabhängigkeit der verschiedenen Funktionskomponenten des technischen Systems zurückzuführen. Sie äußert sich in erster Linie durch die Unabhängigkeit der Anforderungen der verschiedenen Ein-/Ausgabeeinheiten. Um die simultane Bedienung aller Einheiten zu unterstützen, muß in dem Echtzeitrechner eine konkurrente Programmstruktur verwendet werden.

- der anwendungsabhängigen Rechnerkonfiguration.
  Im Gegensatz zum konventionellen Rechenzentrumsbetrieb wechseln beim Rechnereinsatz in technischen Systemen mit jeder Rechneranwendung gewöhnlich auch Art und Umfang der Rechnerkonfiguration in einem erheblichen Maße.

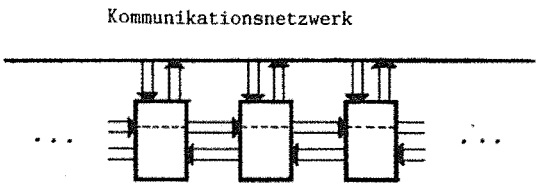
In dieser an /Nehmer 84/ angelehnten Auflistung der Besonderheiten des Echtzeitbetriebs sind nur einige der spezifischen Betiebsbedingungen genannt, die die Echtzeitsysteme von den Datenverarbeitungsanlagen der Business-Informatik abgrenzen. Sie allein reichen aber schon vollkommen aus, um die Schwierigkeiten zu verdeutlichen, die beim Aufbau eines wirtschaftlich einsetzbaren Universalprozeßrechners bewältigt werden müssen. In der Tat haben sich die dabei auftretenden Schwierigkeiten als so massiv erwiesen, daß bis heute noch keine allseits befriedigende Lösung gefunden wurde.

In diesem Beitrag soll ein neues hochparalleles, mikroprogrammierbares Universalrechnerkonzept vorgestellt werden, welches sich aus einer ganzen Reihe von Gründen vorzüglich zum Aufbau eines solchen Systems eignen sollte. Dieses neue Rechnerkonzept wird das Rechnende Gedächtnis genannt. Es wurde Anfang der achtziger Jahre von V.S. Cherniavsky an der Technischen Universität Braunschweig entwickelt /Cherniavsky 80, 81/. Es zeichnet sich in erster Linie durch seine minimale Hardwarestruktur aus. Verglichen mit dem von-Neumannschen Rechnerkonzept ist die Grenze zwischen der Hardware und der Software in dem Rechnenden Gedächtnis deutlich in Richtung der Software verschoben.
Mit diesem Papier sollen die ersten Schritte in der Richtung "Verwendung eines Rechnenden Gedächtnisses in dem Bereich der Echtzeitanwendungen" unternommen werden. Diese Tatsache deutet schon darauf hin, daß mit diesem Beitrag nicht der Anspruch eines bis in alle Details ausgereiften Konzeptes erhoben werden soll oder kann. Verhindert wird dies auch dadurch, daß bis heute bereits mehrere unterschiedliche Versionen des Rechnenden Gedächtnisses entwickelt wurden, die sich durch verschiedene, zum Teil sogar stark voneinander abweichende Eigenschaften auszeichnen, und daß es nicht von vornherein klar ist, welche Eigenschaften für die vorliegende Aufgabe am vorteilhaftesten sind. Zu guter Letzt kommt hinzu, daß es durch die vorhandenen Platzbeschränkungen unmöglich ist, die große Vielzahl der anders- oder neuartigen Elemente des Rechnerkonzeptes in angemessener Weise zu präsentieren. Aus allen diesen Gründen kann hier, insgesamt gesehen, nicht viel mehr als ein erster Eindruck von einer radikal andersartigen Rechnerhardware vermittelt werden.

In diesem Beitrag wird zunächst (in der gebotenen Kürze) eine Version des Rechnenden Gedächtnisses beschrieben. Im Vordergrund stehen dabei die Hardwarestrukturen der Version. Dann folgt die Beschreibung der wesentlichen Eigenschaften dieses Rechnenden Gedächtnisses. Auch hierbei müssen viele Details (und sogar einige interessante Eigenschaften der Version) unterdrückt werden. Bevor der Beitrag dann mit einem Ausblick abschließt, wird noch einmal kurz auf die spezifischen Betriebsbedingungen der Echtzeit-Datenverarbeitungsanlagen eingegangen.

Kurze Beschreibung eines Rechnenden Gedächtnisses

Das Rechnende Gedächtnis, das hier kurz beschrieben werden soll, besteht wie alle anderen bisher entwickelten Versionen aus einer linearen Anordnung von gleichartigen Gedächtniszellen. Diese Gedächtniszellen, die hier auch kürzer einfach Zellen genannt werden, sind durch lokale Nachbarschaftsleitungen und durch ein Kommunikationsnetzwerk untereinander verbunden. Global betrachtet bietet das Rechnende Gedächtnis das folgende Bild:



Innerhalb der Hardware des Rechnenden Gedächtnisses gibt es, anders als in dem traditionellen von-Neumannschen Rechnermodell, keine Rechen- oder Steuer-

einheiten. Nichtsdestoweniger ist das Rechnende Gedächtnis eine berechnungsuniversale Rechenanlage. Sie kann, wie jede andere berechnungsuniverselle Rechenanlage auch, in üblichen Programmiersprachen unter Verwendung von üblichen Programmiermethodiken programmiert werden, obwohl auf der anderen Seite die Besonderheiten des Rechnenden Gedächtnisses besondere Programmiersprachen und Programmiermethodiken nahe legen. Dies wird in den folgenden Abschnitten sicherlich noch deutlicher werden.

Abgesehen von den beiden physikalischen Randzellen ist jede Zelle des Rechnenden Gedächtnisses mit ihren zwei direkten Nachbarzellen und mit dem globalen Kommunikationsnetzwerk verbunden. Alle Zellen sind vollkommen gleichartig aufgebaut: Jede Zelle besteht aus zwei Teilen, die Kontrollteil und Datenteil genannt werden. In diesen Teilen sind neben verschiedenen Schaltelementen (Gattern) mehrere Bitregister enthalten, die entsprechend ihrer Zugehörigkeit zu den Teilen Kontroll- oder Datenbitregister genannt werden. Die Kontroll- bzw. Datenbitregister einer Zelle bilden in ihrer Gesamtheit das Kontroll- bzw. Datenregister der Zelle.

Die Belegung des Kontrollregisters einer Zelle bestimmt das Verhalten dieser Zelle, das heißt die Art und Weise, wie diese Zelle auf die Signale reagiert, die auf den Kommunikationsleitungen an der Zelle eintreffen. In Anlehnung an die Bezeichnungsweise in der Automatentheorie wird diese Belegung auch manchmal der Zustand der Zelle genannt, obwohl es der Funktionsweise der Zellen in dem Rechnenden Gedächtnis angemessener ist, stattdessen von der Rolle der Zelle zu reden /Broer 84/.

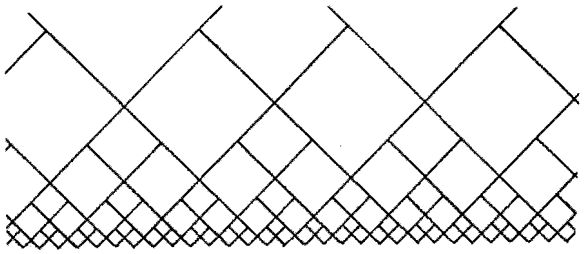
In Abhängigkeit von der Rolle, die eine individuelle Zelle zu einem bestimmten Zeitpunkt zu spielen hat, kann der Kontrollteil unterschiedliche Instruktionen an den Datenteil dieser Zelle senden. Durch eine solche Instruktion kann z.B. der Kontrollteil verlangen, daß der Datenteil

- den Datenregisterinhalt entsprechend den Signalen auf den Leitungen von dem Kommunikationsnetzwerk oder von den Nachbarzellen überschreibt,
- den Datenregisterinhalt auf die Leitungen zu dem Kommunikationsnetzwerk oder zu den Nachbarzellen ausgibt,
- den Datenregisterinhalt zyklisch rotiert oder daß er
- den Datenregisterinhalt mit bestimmten Signalen auf den Leitungen von dem Kommunikationsnetzwerk vergleicht.

Anders als die Zellen in einem Direktzugriffsspeicher, die ja bekanntlich nur über ihre fixierten Platznummern (Koordinaten) in dem Speicher angesprochen werden können, können die Zellen in dem Rechnenden Gedächtnis sowohl über ihre Datenregisterinhalte, als auch über ihre relative Lage zueinander angesprochen werden. Vorgreifend sei schon an dieser Stelle darauf hingewiesen, daß ein assoziatives Adressierungsverfahren für die Konfigurierbarkeit (Teilbarkeit) eines Rechnenden Gedächtnisses von allergrößter Wichtigkeit ist. Denn anders als das Adressierungssystem in einem Direktzugriffsspeicher ist ein assoziatives Adressierungssystem teilbar: Zerschneidet man ein assoziatives Adressierungssystem in einzelne Teile, so erhält man mehrere assoziative Adressierungssysteme. Ebenfalls vorgreifend sei schon an dieser Stelle darauf aufmerksam gemacht, daß die Nachbarschaftsleitungen zwischen den Zellen eine fundamentale Bedeutung haben: Erstens wäre es in dem Rechnenden Gedächtnis ohne die Nachbarschaftsleitungen unmöglich, auf eine Zelle über die relative Lage dieser Zelle zu einer anderen Zelle zuzugreifen. Zweitens, und dies ist weitaus wichtiger, könnten in dem Rechnenden Gedächtnis ohne die Nachbarschaftsleitungen zwischen den Zellen keine Mikroprogramme, und somit auch keine (Makro-) Programme, ausgeführt werden. Drittens schließlich basiert die hardware-gesteuerte automatische Speicherverwaltung auf der Anwesenheit eben dieser Kommunikationsleitungen.

Doch zurück zu der Beschreibung der Version. Wie bereits ausgeführt, sind alle Zellen des Rechnenden Gedächtnisses mit einem Kommunikationsnetzwerk verbunden. Dieses Kommunikationsnetzwerk wurde in das Konzept der Rechnenden Gedächtnisses aufgenommen, um den Einfluß der immanenten Verzögerungszeiten der Halbleiter-Schaltelemente auf ein Mindestmaß zu reduzieren. Details zu diesem Thema sind z.B. in /Broer85a/ zu finden.

Das selbst-organisierende Kommunikationsnetzwerk ist eine potentiell unendliche Struktur. Es besteht aus Knoten, die in einer einheitlichen Weise untereinander verbunden sind. In der folgenden Abbildung ist ein Ausschnitt des Kommunikationsnetzwerkes dargestellt:



Die Linien in der Abbildung repräsentieren die Kommunikationskanäle des Netzwerkes. Sie bestehen aus paarweise entgegengesetzt gerichteten Leitungen. An den Kreuzungspunkten der Linien befinden sich die Knoten des Netzwerkes. Die Zellen des Rechnenden Gedächtnisses sind in der Abbildung nicht dargestellt; sie müssen an dem unteren Rand der Abbildung (pro Dreiecksspitze eine Zelle) gedacht werden.

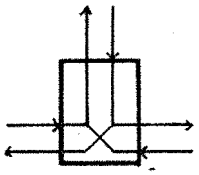
entsprechend den zwei unterschiedlichen Kreuzungspunkten in der Abbildung müssen beim Aufbau des Netzwerkes zwei unterschiedliche Knotentypen verwendet werden. Sie werden in Abhängigkeit von der Zahl der Anschlußstellen Vereinigungsknoten (wenn drei externe Anschlüsse vorhanden sind) oder Kreuzungsknoten (wenn vier vorhanden sind) genannt.

Die externen Anschlüsse eines Knotens, sie werden hier auch Ports genannt, bestehen aus einem Eingabe- und einem Ausgabesockel. Entsprechend der natürlichen Bezeichnungsweise werden die drei Porte eines Vereinigungsknotens der linke, der rechte und der obere Port und die vier Porte eines Kreuzungsknotens der linke, der rechte, der obere linke und der obere rechte Port genannt.

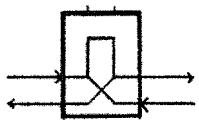
Die Knoten des Kommunikationsnetzwerkes leiten die an dem Knoten eintreffenden Signalströme in Abhängigkeit von dem jeweils verwendeten Operationsmodus in unterschiedlicher Weise weiter. Die dabei in Betracht kommenden Verhaltensweisen sollen nun getrennt für die beiden Knotentypen kurz beschrieben werden:

a) Ein Vereinigungsknoten kann zwei unterschiedliche Verhaltensweisen an den Tag legen:

In dem Normalmodus vereinigt ein Vereinigungsknoten die Signalströme, die an den Eingabesockeln des linken und des rechten Ports eintreffen, zu einem gemeinsamen Signalstrom. Verwendet wird bei dieser Vereinigung die logische Oder-Regel. Der so gebildete Gesamtsignalstrom wird dann zu dem Ausgabesockel des oberen Ports weitergeleitet. Simultan dazu wird ein zweiter Signalstrom in die entgegengesetzte Richtung transportiert: Die Signale, die an dem Eingabesockel des oberen Ports eintreffen, werden zu den Ausgabesockeln des linken und des rechten Ports weitergeleitet. Die folgende Abbildung illustriert diese Arbeitsweise:

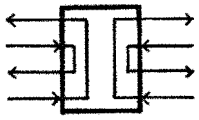


Auch in dem Reflektormodus vereinigt ein Vereinigungsknoten die Signalströme, die an den Eingabesockeln des linken und rechten Ports eintreffen, zu einem Gesamtstrom. Dieser Gesamtsignalstrom wird dann aber nicht, wie in dem Normalmodus, zu dem oberen Port weitergeleitet, sondern zurück "reflektiert" zu den Ausgabesockeln des linken und des rechten Ports. Die folgende Abbildung verdeutlicht die Arbeitsweise:



b) Entsprechend der größeren Zahl der Ports kann ein Kreuzungsknoten auch mehr als nur zwei unterschiedliche Verhaltensweisen aufzeigen:

In dem Passiermodus behandelt ein Kreuzungsknoten vier unterschiedliche Signalströme: Der Signalstrom, der an dem Eingabesockel des rechten (linken) Ports an dem Knoten eintrifft, wird zu dem Ausgabesockel des oberen rechten (oberen linken) Ports weitergeleitet. Gleichzeitig werden die entsprechenden entgegengesetzt gerichteten Signalströme in der umgekehrten Richtung transportiert. Die folgende Abbildung verdeutlicht diese Arbeitsweise:

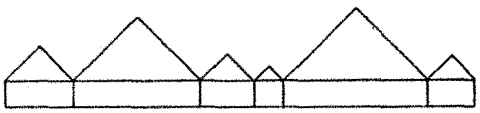


Der Passiermodus wird auch als Normalmodus des Kreuzungsknotens bezeichnet. Neben dem Passiermodus sind von einem Kreuzungsknoten auch Verhaltensweisen möglich, die alle unmittelbar vergleichbar sind mit den Arbeitsweisen eines Vereinigungsknotens: Bei Vernachlässigung der Signale an dem oberen linken oder dem oberen rechten Port kann von dem Kreuzungsknoten durch Berücksichtigung der drei verbleibenden Ports ein Vereinigungsknoten nachgebildet werden. In diesen Fällen übernimmt dann der obere rechte oder der obere linke Port die Aufgabe des oberen Ports in dem simulierten Vereinigungsknoten. Darauf soll hier nicht näher eingegangen werden.

Soweit zu der Beschreibung der Hardwarestruktur. In dem nächsten Abschnitt werden einige fundamentale Eigenschaften dieser Version beschrieben.

Lebensräume für Aktoren

In dem oben beschriebenen Rechnenden Gedächtnis können jeweils Gruppen von benachbarten Zellen zu einer Station zusammengefaßt werden. Stationen sind in sich abgeschlossene Einheiten, die aus den betreffenden benachbarten Zellen und einem spezifischen Teil des Kommunikationsnetzwerks bestehen. In der folgenden Abbildung ist eine mögliche Aufteilung eines Rechnenden Gedächtnisses in einzelne Stationen dargestellt:



Alle Stationen sind durch isolierende Grenzlinien voneinander und von dem Rest des Rechnenden Gedächtnisses getrennt. Diese Stationsgrenzlinien werden automatisch von der Hardware errichtet, sobald die Grenzzellen der Station durch entsprechende Bitmuster gekennzeichnet werden. Gleichzeitig mit der Errichtung dieser Grenzlinien wird in dem durch die Stationsgrenzlinie in der Station eingeschlossenen Teil des Kommunikationsnetzwerkes eine baumförmige Kommunikationsstruktur aufgebaut. Aus Platzgründen können diese Selbstorganisationsfähigkeiten des Netzwerkes hier nicht beschrieben werden. Mehr Informationen darüber sind z.B. in /Broer 85a, 85b/ enthalten.

In den Stationen des Rechnenden Gedächtnisses können Aktoren residieren. (Aus Einfachheitsgründen soll hier davon ausgegangen werden, daß pro Station maximal ein Aktor verwendet werden kann.) Die Aktoren sind durch Mikroprogramme definierte programmausführende "Wesen". Sie übernehmen innerhalb eines Systems spezifische Aufgaben wie z.B. die Verwaltung und/oder Bedienung einer Ein-/Ausgabeeinheit oder die Verwaltung einer Datenstruktur.

Die Fähigkeiten eines Aktors können in weiten Grenzen variieren: Denkbar sind zum Beispiel Aktoren, die nur bestimmte Ereignisse registrieren und zählen. Denkbar sind aber auch Aktoren, die sich z.B. wie mikroprogrammierbare von-Neumannsche Rechenanlagen verhalten. Letztere standen in den hisherigen Mehraktivitätsversionen des Rechnenden Gedächtnisses im Vordergrund des Interesses. Dort wurden die Stationen mit den Aktoren, die sich wie (mikroprogrammierbare) von-Neumannsche Rechenanlagen verhielten, Aktivitätszonen genannt.

Das Verhalten der Aktoren wird durch maßgeschneiderte Mikroprogramme definiert. Die Mikroprogramme, die das Verhalten eines individuellen Aktors bestimmen, sind ebenso wie die (Makro-) Programme und die Daten in der Station dieses Aktors enthalten. Daher kann sich ein Aktor zum Beispiel auch dynamisch selbst verändern.

Anmerkung:
Die prinzipielle Möglichkeit, in einem Rechnenden Gedächtnis Aktoren verwenden zu können, gründet sich auf die Homogenität des Gedächtnisses und auf die Fähigkeit der Struktur, Mikroprogramme ausführen zu können. Diese Fähigkeit wiederum basiert auf der Verteilung der Kontrollogik auf alle Zellen des Gedächtnisses und, darauf wurde schon hingewiesen, auf der Anwesenheit der Nachbarschaftsleitungen zwischen den Zellen. Diese beiden Voraussetzungen ermöglichen es auch, daß die Mikroprogramme an Ort und Stelle ausgeführt werden können, d.h. daß bei der Ausführung der Programme keine Mikroprogrammbefehle transportiert werden müssen. Dabei spiegelt sich allerdings die aus dem von-Neumannschen Rechnerkonzept bekannte Erscheinung der Befehlstransporte in einer abgewandelten Form wieder: Während es bei der Ausführung eines Programmes in dem von-Neumannschen Rechnerkonzept notwendig ist, die Befehle des Programmes nacheinander zu dem Befehlsregister zu transportieren, ist es in dem Rechnenden Gedächtnis erforderlich, eine Kontroll-

information über die Nachbarschaftsleitungen von Zelle zu Zelle durch das Programm weiterzuleiten. Nicht zuletzt wegen der fundamentalen Bedeutung dieser Informationstransporte wird von der Ausbreitung einer Informationswelle über die Nachbarschaftsleitungen geredet. Details zu diesem Thema können z.B. in /Cherni 80, 81/ oder /Broer 84/ gefunden werden.

In einem Rechnenden Gedächtnis können, solange Zellen in ausreichender Zahl vorhanden sind, beliebig viele Stationen - und damit auch beliebig viele Aktoren - erzeugt werden. Alle diese Aktoren können, da sie sich durch die isolierenden Stationsgrenzlinien nicht gegenseitig stören können, echt gleichzeitig tätig sein. Die Aktoren können über das Kommunikationsnetzwerk miteinander kommunizieren, sie können sich in dem Rechnenden Gedächtnis bewegen, sie können andere Stationen und Aktoren erzeugen und sie können sich selbst auflösen, sobald sie ihre Arbeit erledigt haben.

Dies sind die Gründe, weshalb man die Hardware eines Rechnenden Gedächtnisses in Anlehnung an die Bezeichnungsweise in der Biologie auch einen Lebensraum für Aktoren nennen kann. Denn in einem Lebensraum für Aktoren kommt der Hardware eine Aufgabe zu, die unmittelbar mit der Aufgabe der Umwelt in den biologisch definierten Lebensräumen (Biotopen) vergleichbar ist: Die Hauptaufgabe der Hardware in einem Lebensraum für Aktoren besteht erstens aus der Aufgabe, Material für die Konstruktion und Isolierung von Aktoren bereitzustellen, und zweitens aus der Aufgabe, ein Medium zur Ermöglichung der Interaktionen zwischen den verschiedenen Aktoren zur Verfügung zu stellen. Verglichen mit der traditionellen Betrachtungsweise erscheint die Rechnerhardware in einem Lebensraum für Aktoren daher in einem völlig neuen Licht /Broer 85a, 85b/.

Diese neue (und im Anbetracht der Möglichkeiten der VLSI-Technologie zeitgemäß zu nennende) Hardwaresicht ermöglicht vollkommen neue Lösungen. Hier seien beispielhaft nur die extrem schnellen Prozeduraufrufe und Returns oder die hardware-gesteuerte Speicherselbstverwaltung genannt /Broer 85b/. Neben diesen neuen Möglichkeiten können mit Hilfe der Aktoren auch alte Konzepte wie zum Beispiel die Monitorkonzepte /Hoare 74/, die capability-orientierten Adressierungskonzepte /Fabry 74/ oder andere verwandte Modulkonzepte /Buzzard 85, Goldberg 83/ unmittelbar unterstützt werden. Ein Ende dieser Möglichkeiten ist nicht einmal in Sicht.

Echtzeitsysteme

Hier sollen noch einmal die eingangs erwähnten spezifischen Betriebsbedingungen des Echtzeitbetriebs unter die Lupe genommen werden, um die Aussage zu untermauern, daß sich ein Rechnendes Gedächtnis beim Aufbau von Echtzeitsystemen geradezu anbietet. In der Einleitung wurde festgestellt, daß sich die spezifischen Betriebsbedingungen zum Beispiel niederschlagen in

- der hohen Anforderung an Zuverlässigkeit und Sicherheit.
  In einem Rechnenden Gedächtnis bieten sich insbesondere durch die Homogenität der Hardware die bekannten Techniken zur Erhöhung der Redundanz geradezu an. Daneben können aber auch vollkommen neue Wege beschritten werden: Denkbar sind z.B. Diagnosestationen, die sich durch andere Stationen bewegen können, um etwa die Funktionsfähigkeit der einzelnen Zellen zu überprüfen. Diese Überprüfung kann ohne eine nennenswerte Störung der betreffenden Stationen geschehen. (Als Vorbild für diese Diagnosestationen können die freien Zellen dienen, die die Speicherselbstorganisation ermöglichen. Auch sie können sich durch eine arbeitende Station bewegen, ohne die Arbeit der Station zu stören.) Dies sind nur einige der Gründe, weshalb es möglich sein sollte, extrem zuverlässige Rechnende Gedächtnisse aufzubauen.

- der Zeitabhängigkeit der Programmabläufe.
  In der Einleitung wurde ausgeführt, daß es bei der Planung der Zeitabläufe besonders schwierig ist, das Eigenzeitverhalten der Betriebsysteme zu berücksichtigen. In einem Rechnenden Gedächtnis können die Betriebssysteme durch die mögliche Verteilung der Aufgaben auf spezialisierte Stationen bis auf kleine Rudimente beseitigt werden. Daher kann es auch kaum zu einer starken Zeitverfälschung der Anwenderprogramme durch das Eigenzeitverhalten kommen.

- der Parallelität der Schnittstelle Rechner / technisches System.
  Die sich durch die Unabhängigkeit der verschiedenen Ein-/Ausgabeeinheiten der Schnittstelle bemerkbar machende Parallelität ist für ein Rechnendes Gedächtnis nur in einer stark abgeschwächten Form eine Herausforderung. Denn innerhalb eines Rechnenden Gedächtnisses können

ja angepaßt an den Bedarf so viele (gleichzeitig arbeitende) Aktoren erzeugt werden, wie sie benötigt werden.

- der anwendungsabhängigen Rechnerkonfiguration. In einem Rechnenden Gedächtnis ist die Grenzlinie zwischen der Hardware und der Software deutlich in Richtung der Software verschoben. Mit der Verschiebung dieser Grenze bewegt sich auch die Unterscheidungslinie zwischen verschiedenen Rechnerkonfigurationen in den Bereich der Software hinein. Mit anderen Worten: Dank der Homogenität des Rechnenden Gedächtnisses fällt die Anpassung des Rechners an verschiedene Situationen wesentlich leichter.

Ausblick

Faßt man die hier beschriebenen Teilergebnisse zusammen, so muß man nach meiner Meinung zu dem Schluß kommen, daß sich ein Lebensraum für Aktoren vorzüglich zum Aufbau eines Echtzeitsystems eignet. Dies täuscht natürlich nicht darüber hinweg, daß bis zum ersten Einsatz eines Rechnenden Gedächtnisses in dem Bereich der Echtzeitdatenverarbeitung noch sehr viel Arbeit geleistet werden muß. Und auch wenn die ersten Schritte in dieser Richtung an der Technischen Universität Braunschweig in die Wege geleitet wurden, so bleibt dennoch ein Berg von Aufgaben ungelöst liegen. Dies liegt insbesondere auch daran, daß es durch die schlechte finanzielle Lage immer nur möglich ist, kleine und kleinste Schritte zu unternehmen.

Acknowledgement

Besonders herzlich danken möchte ich an dieser Stelle Prof. V.S. Cherniavsky, ohne den dieses Papier nicht hätte geschrieben werden können, sowie den Herren U. Hafermann, G. Pogrzeba und P. Tillert, die mir bei der Erstellung dieses Papieres hilfreich und anregend zur Seite gestanden haben.

Literaturliste

(Diese Liste beinhaltet auch Hinweise auf einige Arbeiten, die in dem Forschungsprojekt "Rechnendes Gedächtnis" entstanden sind, auf die hier aber nicht direkt Bezug genommen wurde)

/Buzzard 85/ G.D. Buzzard T.N. Mudge
Object-Based Computing and the Ada PL.
Computer
March 1985 pp. 11 - 19

/Broer 83a/ H.E. Broer
Informations-transformationen in Homogenen Rechnenden Strukturen
Informatik-Bericht Nr. 8302
Technische Universität Braunschweig, 1983

/Broer 83b/ H.E. Broer V. S. Cherniavsky
Ein Rechnerkonzept mit hoher Parallelität: Das Rechnende Gedächtnis (Teil I und II)
Proceedings Parallel Computing 83
M. Feilmeier, J. Joubert and U. Schendel
North Holland, 495 - 504, 1983

/Broer 84/ H.E. Broer
Eine Einführung in das Konzept der Rechnenden Gedächtnisse
Informatik-Bericht Nr. 8406
Technische Universität Braunschweig, 1984

/Broer 84b/ H.E. Broer
Baumartige Kommunikationsstrukturen
Informatik-Bericht Nr. 8409
Technische Universität Braunschweig, 1984

/Broer 85a/ H.E. Broer
A Computing Memory with a Self-Organizing Communication Network
erscheint in:
Proc. Int. Conf. Parallel Computing 85
Joubert et. al. (ed.)

/Broer 85b/ H.E. Broer
Computing Memory: A Living Space for Actors
Informatik-Bericht (in Vorbereitung)
Technische Universität Braunschweig, 1985

/Cherni 80/ V.S. Cherniavsky
Computing Memory Part I:
The One-Dimensional One-Activity Version
Informatik-Bericht Nr. 8006
Technische Universität Braunschweig, 1980

/Cherni 81/ V.S. Cherniavsky
Computing Memory Part II:
The Multi-Activity System
Informatik-Bericht Nr. 8102
Technische Universität Braunschweig, 1981

/Cherni 82/ V.S. Cherniavsky
Computing Memory: A Non - Traditional
Computer Architecture
Proc. 5th Hawaii ICSS
Honolulu 1982 pp. 343 - 349

/Cherni 83/ V.S. Cherniavsky H.E. Broer
A New Homogeneous Microprogrammable
Computer Architecture
Proceedings Int. Conf. Microcomputing II
W. Remmerle, H. Schecher (Hrsg.)
German Chapter of the ACM
Berichte Nr. 17, pp. 159 - 177, 1983

/Cherni 84a/ V.S. Cherniavsky P. Ruckmann
Simulation Homogener Rechnender Strukturen
Informatik-Bericht Nr. 8404
Technische Universität Braunschweig, 1984

/Cherni 84b/ V.S. Cherniavsky
Das Rechnende Gedächtnis
Eine Homogene Rechnende Struktur und ihre
Auswirkungen auf Programmiersprachen
Informatik-Bericht Nr. 8405
Technische Universität Braunschweig, 1984

/Fabry 74/ R.S. Fabry
Capability - Based Addressing
Communication ACM
Vol. 17 1974 pp. 403 - 411

/Goldb 83/ A. Goldberg D. Robson
Smalltalk-80:
The Language And Its Implementation
Addison Wesley, Reading, Mass. 1983

/Haferm 84/ U. Hafermann
Eine vollassoziative Einaktivitätsversion
des Rechnenden Gedächtnisses
Informatik-Bericht Nr. 8403
Technische Universität Braunschweig, 1984

/Hoare 74/ C.A.R. Hoare
Monitors:
An Operating System Structuring Concept
Communication ACM
Vol. 17 1974 pp. 549 - 556

/Nehmer 84/ J. Nehmer
Systemarchitektur von Realzeitsystemen
Informatik Spektrum
Band 7 Heft 2 1984 pp. 65 - 72

/Pogrze 85/ G. Pogrzeba
Vergleich des Rechnenden Gedächtnisses mit
dem von Neumannschen Rechnerkonzept
Informatik-Bericht (in Vorbereitung)
Technische Universität Braunschweig, 1985

/Ruckma 83/ P. Ruckmann
Eine Mehraktivitätsversion des Rechnenden
Gedächtnisses: Version R
Informatik-Bericht Nr. 8303
Technische Universität Braunschweig, 1983

Broer, Helfried

Abt. für Mathematische und Experimentelle Informatik
Institut für Theoretische und Praktische Informatik
Technische Universität Braunschweig
Gaußstraße 11
3300 Braunschweig

Tel. 0531-391-3107

UNIX und PEARL in Echtzeitrechnernetzen

Dipl.-Phys. Ulrich Scholtze / München

Zusammenfassung

Ein standardisiertes Konzept für Echtzeitsysteme in Dispositions- u. Steuerungsanwendungen wird beschrieben. Bausteine sind gleichartige Mikrocomputer, die unter UNIX und PEARL laufen und über Ethernet gekoppelt werden. Die Vorteile werden beschrieben und Konfigurationsbeispiele gegeben. Eine Anwendung des Konzeptes wird vorgestellt.

Schlüsselworte: Echtzeitnetzwerk, UNIX, PEARL, Ethernet

Summary

A standardized concept for real time systems in scheduling and controlling applications is described. Building blocks are homogeneous Microcomputers running under UNIX and PEARL connected via Ethernet. Advantages are discussed and configuration examples included. An application of this concept is described.

Keywords: real time network, UNIX, PEARL, Ethernet.

1. Probleme bei Echtzeitanwendungen

Dispositions- und Steuerungssysteme werden auch heute noch überwiegend maßgeschneidert und damit kostenintensiv erstellt. Solche Systeme sind gekennzeichnet durch:

- Steuerungsebene:
  Hard- und Software sind Teil eines abgrenzbaren technischen Prozesses, z.B. in der Produktionsmittelsteuerung, und operieren nach einer statisch oder zur Laufzeit vorgegebenen Zielfunktion
- Dispositionsebene:
  Diese Ebene dient der Koordination mehrerer technischer Prozesse, unterliegt häufig weniger engen Realzeitanforderungen und erfordert mehr Mensch-System-Interaktion

Der Anwender fordert folgende Eigenschaften dieser Systeme:

- kostengünstig
- zuverlässig
- angemessen
- erweiterbar
- integrierbar

Für den Systemarchitekten bedeuten diese Forderungen:

- schnelle Realisierung mit preisgünstigen Komponenten bei häufig kurzer Konzeptphase
- Einführung von Redundanz wegen in der Praxis unzuverlässigen Komponenten und / oder schnelles Wiedererlangen der Funktionalität
- genaue Beschreibung der Benutzeranforderungen und klare Abstimmung mit dem Kunden, u.a. durch "prototyping", formale Anforderungssprachen, Simulation etc. bei guter Kenntnis der Eigenschaften der Realisierungskomponenten
- leichte Wartung und Anpassbarkeit an wachsende, sich ändernde Benutzeranforderungen
- Anschluss an vorhandene und kommende Rechnersysteme, z.B. in der Produktionsleitebene

Diese Randbedingungen sind teils widersprüchlich und daher nur als Kompromiß lösbar. Gefundene Kompromisse lassen sich dann nur schwierig auf Standard-Komponenten (Hard- u. Software) abbilden und führen zur Maßschneiderei mit selbstgefertigten Komponenten bzw. von Projekt zu Projekt wechselnden Hardware- u. Betriebssystem-Umgebungen.

2. Lösungsansatz

Als Lösungsansatz bietet sich der Einsatz vorhandener Hard- und Software-Bausteine an. Die vorgeschlagenen Bausteine sind:

1) Vernetzbarer Mikro-Rechner mit 1 MB Hauptspeicher und ca. 1 MIPS Prozessorleistung
2) Time-sharing-Betriebssystem UNIX
3) Echtzeit-Hochsprache PEARL u. PEARL-Betriebssystem

Für diese Bausteine muß gelten:

- UNIX u. PEARL sind auf gleicher Hardware lauffähig
- UNIX u. PEARL sind kommunikationsfähig

Diese Bausteine können zur PEARL-Engine zusammengefaßt werden (B i l d 1). Das Konzept beruht auf den Ergebnissen eines BMFT-Förderprojektes, das gemeinsam von dem IRT Institut für Rundfunktechnik GmbH, München, Werum Datenverarbeitungssysteme GmbH, Lüneburg, und PCS GmbH, München, durchgeführt wurde ([1], [2]).

Vorteile dieses Konzeptes sind:

A günstiges Preis/Leistungsverhältnis durch Mini-Leistung zum Mikro-Rechner-Preis

B Einheitliche Basis-Hardware

C spezialisierte Betriebssysteme für abgrenzbare Aufgabenbereiche Disposition u. Steuerung

D langjährig u. vielfach bewährte Komponenten

E Erweiterbarkeit in kleinen Stufen über Netz

F Ingegrierbarkeit über Netz oder gateway-Rechner

G Redundanz durch Mehrfacheinsatz gleicher Bausteine am Netz

H Einheitliche Umgebung in Entwicklungs- u. Betriebsphase (Bild 2)

I Aufgabenoptimierte Konfiguration u. Anpassbarkeit (Bild 3)

J UNIX-Werkzeuge und Anwendungspakete auch für Echtzeit-Anwendungen

3. Konfigurationsbeispiele

Mit diesen Bausteinen läßt sich ein breiter Anwendungsbereich von dedizierten low-cost-Lösungen (Bild 1) über redundante Systeme (Bild 4) bis hin zu multifunktionalen Systemen (Bild 3) abdecken. Übliche Echtzeitsystem-Strukturen wie Hierarchie oder Stern werden dabei nicht mehr hardwaremäßig, sondern durch Anwendungs-Software realisiert. Dadurch steigt die Sicherheit in der Projekt-Realisierung für Anwender und Systemarchitekten. Spätere Anpassungen sind leichter durchführbar.

4. Anwendungsbeispiel

Als Beispiel dient ein Anwendungsfall, der zur Zeit von Dornier System GmbH als Generalunternehmer mit PCS GmbH als Unterauftragnehmer realisiert wird. Die Arbeitsgemeinschaft der Rundfunkanstalten Deutschlands (ARD) benötigt ein System, mit dem Hörfunksendungen vom Sternpunkt Hessischer Rundfunk/ Frankfurt/ Main disponiert, zwischen den Rundfunkanstalten geschaltet und übermittelt werden können. Es handelt sich also um ein wide-area-network (WAN), dem als Knoten im Sternpunkt ein lokales Netzwerk zugrunde liegt. Als Verfügbarkeit wird bereits im Probebetrieb 99,5 % gefordert. Die Sternpunkt-Konfiguration ist in Bild 5 dargestellt.

Die Anwendung wird genauer im Beitrag von Herrn Frank / Dornier System GmbH dargestellt.

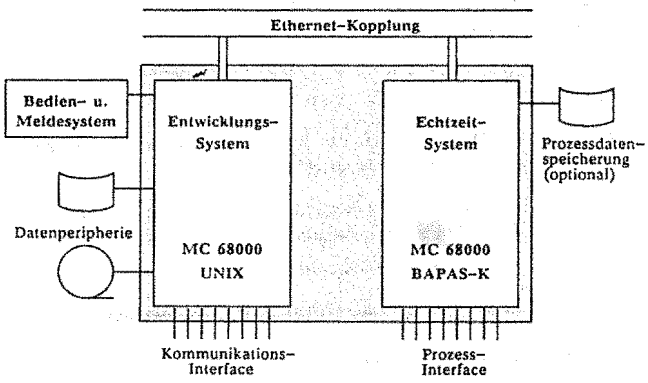


Bild 1. PEARL-Engine

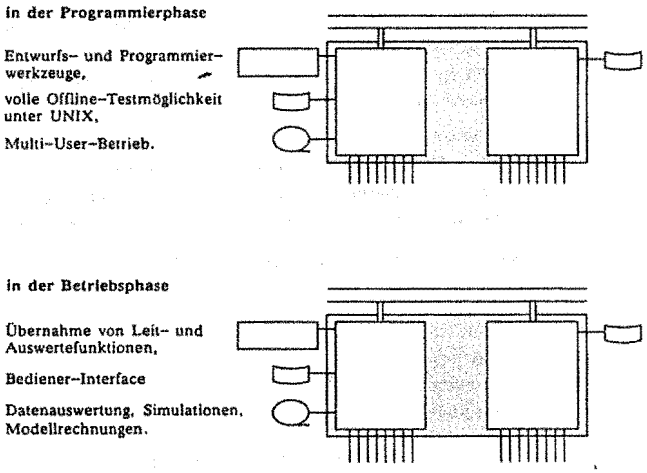
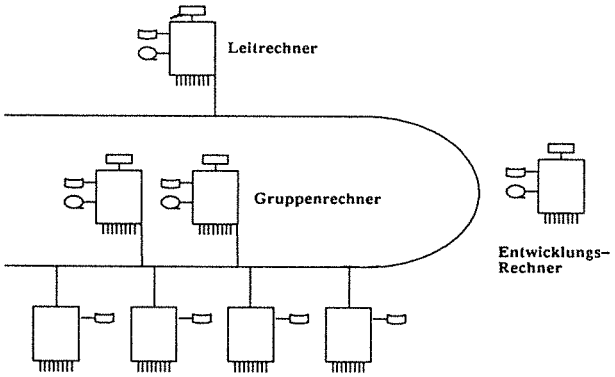


Bild 2. Programmier- und Betriebsphase

Durch die Ethernet-Kopplung ergibt sich eine Öffnung der Technologie zu integrierten anpassungsfähigen Echtzeit-Systemen.



Echtzeitsysteme als Prozess-Server am Ethernet-Netz

Bild 3. **Aufgabenoptimierte Konfiguration**

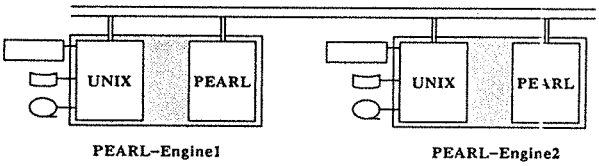


Bild 4. **Redundantes System**

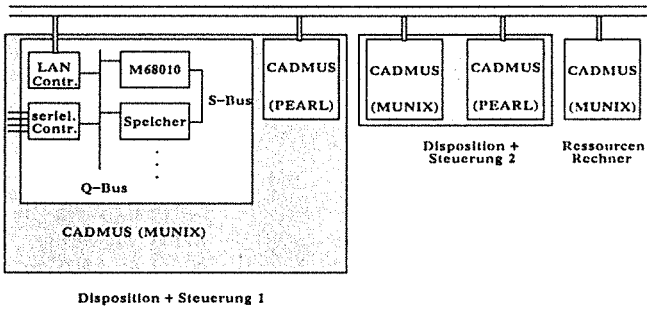


Bild 5. **Sternpunkt-Konfiguration**

Literatur

[1] Färber, G.: PEARL Engine 68000: Hardware
Tagungsband des Workshops "PEARL in der Rundfunktechnik",
29.09.1983, Institut für Rundfunktechnik, München.

[2] Sauter, D.; Windauer, H.: PEARL-Engine: Software
Tagungsband des Workshops "PEARL in der Rundfunktechnik",
29.09.1983, Institut für Rundfunktechnik, München.

Anschrift des Autors

Scholtze, Ulrich
PCS GmbH Pfälzer-Wald-Str. 36
D-8000 München 90
Tel.: 089/68004-0

Anwendung von PEARL bei der Erneuerung des ARD Hörfunksterns

R.J. Frank Friedrichshafen

Zusammenfassung

Im Vorhaben der "Erneuerung des ARD-Hörfunksystems", wird ein Rechner-Verbundsystem bestehend aus einer Zentrale und 15 Satellitenrechner zum Einsatz kommen. Moderne 32-Bit-Rechner auf MC 68000 - Basis (PEARL-Engine), eine Echtzeitdatenbank BAPAS-DB, ein Mensch-Maschine-Kommunikations-System, das LAN ETHERNET zur Rechnerkopplung und die Programmiersprache PEARL sind die Säulen des zu entwickelnden operationellen Systems.

Mit modernen Software-Werkzeugen, EPOS, VICO und einem PEARL-Testsystem, soll eine durchgängige Begleitung und Unterstützung der Entwicklung ermöglicht werden.

Die Möglichkeit der Verbindung dieser Software-Werkzeuge und die angestrebte Realisierung wird aufgezeigt.

Schlüsselwörter

PEARL-Engine, Software-Werkzeuge, Software-Produktionsumgebung.

Summary

In a project, the "renewal" of the ARD star-meshed broadcasting system", an interconnected computer system, consisting of a host and 15 satellite computers will be employed. Modern 32-bit-computers using the MC 68000 (PEARL-Engine), a realtime database BAPAS-DB, a man-machine-communication-system, the LAN ETHERNET for connecting the computers and the programming-language PEARL are the major elements of the operational system.

Modern software-tools like EPOS, VICO and a PEARL testsystem will provide continous support for the development from the start to the finish.

The possibility of a juncture of these software-tools and the desired realization will be indicated.

Keywords

PEARL-Engine, software-tools, software-production-environment.

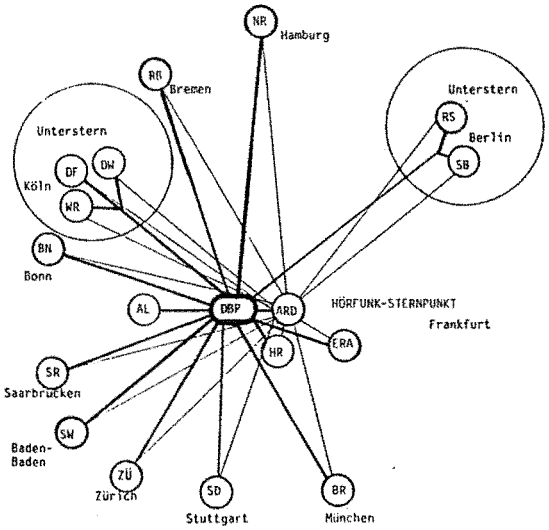
1. Das Projekt "Erneuerung des ARD-Hörfunk-Sternsystems

Für das Gemeinschaftsprojekt der ARD "Erneuerung des seit 1974 in Betrieb befindlichen ARD-Hörfunk-Sternsystems" ist die Dornier System GmbH Generalunternehmer. Die Anforderungen an das neue System wurden von der ARD-Arbeitsgruppe "Erneuerung Hörfunkstern-System" unter Mitwirkung der Deutschen Bundespost erarbeitet [9]. Über das Hörfunk-Sternsystem und sein Dauerleitungsnetz wird der Programmaustausch zwischen den einzelnen Rundfunkanstalten und dem Ausland abgewickelt (B i l d 1). Ein in den wesentlichen Teilen redundant ausgelegtes Rechner-Verbundsystem soll dabei u.a. folgende Funktionen erfüllen:

- Bestellung der Übertragung eines Hörfunkbeitrags durch jede angeschlossene Anstalt.
- Disposition und Belegungsoptimierung der vorhandenen Leitungen.
- automatische Schaltung und Überwachung der Verbindungen.

Der Auftrag umfaßt die komplette Erneuerung der Steuerungstechnik und schließt alle ton- und studiotechnischen Einrichtungen, die zu einer automatisierten Betriebsabwicklung erforderlich sind, mit ein.

Zu den wichtigsten Unterlieferanten zählen die Firmen Neumann, Berlin, PCS in München und WERUM, Lüneburg. Das neue ARD-Sternsystem soll 1987 den Betrieb aufnehmen.



Abkürzungen:

ARD	Arbeitsgemeinschaft der Rundfunkanstalten Deutschlands	RB	Radio Bremen
AL	Ausland	RS	RIAS-Berlin
BN	Bonn	SB	Sender Freies Berlin
DBP	Deutsche Bundespost	SD	Süddeutscher Rundfunk
DF	Deutschlandfunk	SR	Saarländischer Rundfunk
DW	Deutsche Welle	SW	Südwestfunk
HR	Hessischer Rundfunk	WR	Westdeutscher Rundfunk
NR	Norddeutscher Rundfunk		Datenleitungen
ERA	Ersatz Rundfunkanstalt		Modulationsleitungen

Bild 1. Das Tondauer- und Datenleitungsnetz des ARD-Hörfunk-Sternsystems

Bild 2. Prinzipielle Übersicht der Rechnersysteme im ARD-Hörfunk-Sternsystem

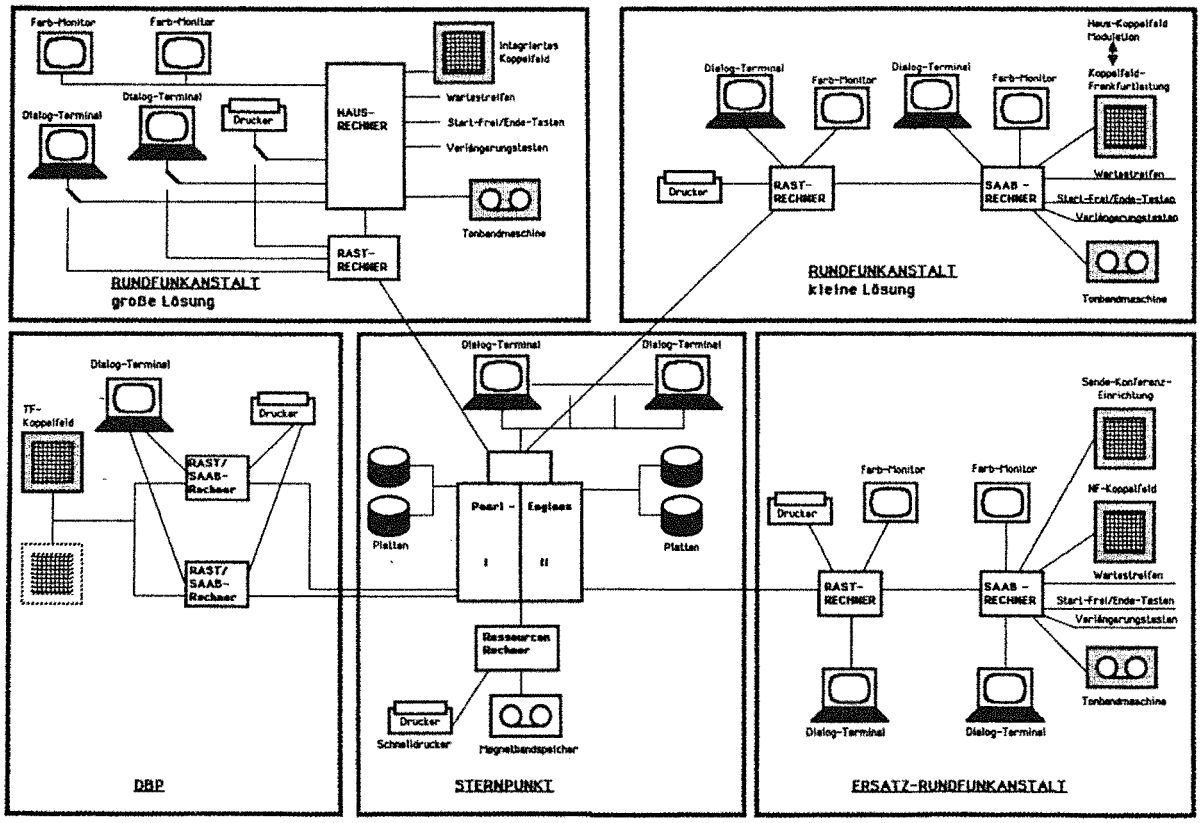
Kennzeichnend für das System sind 3 verschiedene Rechnertypen (B i l d 2) [1]:

- der Zentralrechner Sternpunkt (STP)
- ein Rundfunkanstalten-Sternpunkt Rechner (RAST)
- ein Schaltauftragsabwicklungs-Rechner (SAAB)

Die Aufgabenverteilung im System ist getrennt nach

- Sternpunkt
  - o "Master" für die Kommunikation zwischen den Rechnern
  - o Disponent für das Tonleitungsnetz und damit zentrale Datenbank für alle Bestellvorgänge
  - o Versorgen der RAST-Rechner mit Schaltbefehlen
  - o Langzeitspeicherung von Überspieldaten
  - o Überwachung des Überspielbetriebs bzgl. Tonqualität und Ausführung von Schaltungen
  - o Bereitstellen einer zentralen Konferenzanlage
  - o und eines zentralen Überspieltonträgers zur Zwischenspeicherung von Beiträgen.

BILD 2: ARD-HÖRFUNKSTERN-SYSTEM (PRINZIPIELLE ÜBERSICHT)



- RAST-Rechner
  - o Kommunikation mit dem Sternpunkt-Rechner
  - o Verwaltung des 24-Stunden-Schaltbefehls-vorrates.
  - o Anzeige des Schaltvorrates auf Monitoren
  - o Eingabestation für Tonleitungs-Bestellungen
  - o Abfragestation bzgl. Bestell- und Schalt-daten des Sternpunktes
  - o Kommunikation mit dem SAAB-Rechner.
- SAAB-Rechner
  - o Schalten der internen Tonleitungsverbindungen
  - o Steuerung von Tonbandmaschinen
  - o Ergänzen von Schaltbefehlen um "interne" Informationen
  - o Kommunikation mit dem RAST-Rechner

Bei der Erneuerung des Systems sollen folgende Aspekte besonders beachtet werden:

- Eine Mensch-Maschine-Schnittstelle mit den Eigenschaften menuegesteuert, selbsterklärender Darstellung, Bedienfehler prüfend und änderungsfreundlich.
- Einsatz einer höheren Programmiersprache (PEARL) und Realisierung einer modular strukturierten Software.
- Einsatz ausbau- und entwicklungsfähiger Hard- und Software durch standardisierte Hardwarekomponenten, 32-Bit-Architektur, Einsatz einer Datenbank, eines Dialogprogrammsystems und einer Standardnetzkoppelung (ETHERNET).

Um diesen Forderungen gerecht zu werden, wird als Rechnerhardware (B i l d 3) die 32-Bit PEARL-Engine II an der Standardperipherie moderne Tintenstahldrucker PT88 und PT90 und als Schnittstellen zur Prozeßperipherie EMULEX-Interfaces eingesetzt.

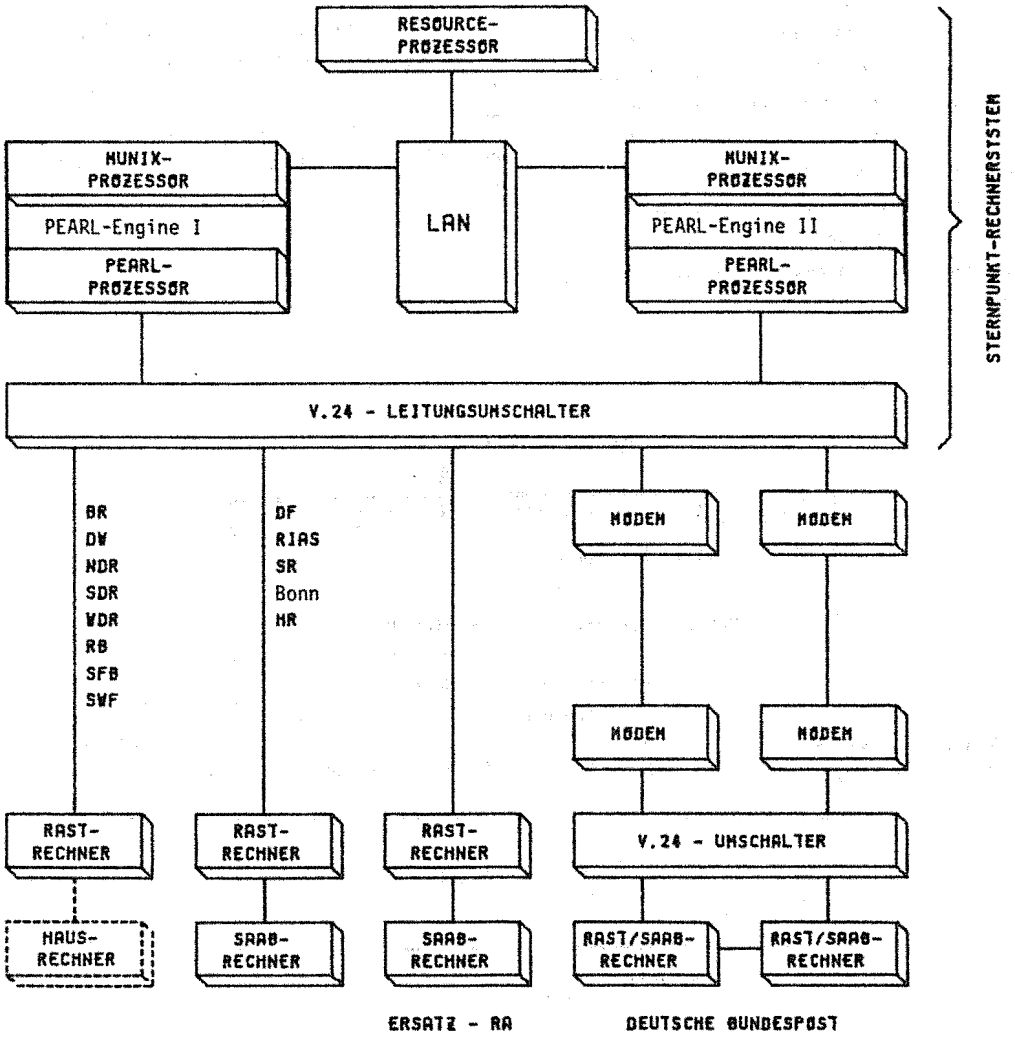


Bild 3. Rechner-Konfiguration des ARD-Hörfunk-Sternsystems

Für die Kopplung in der Zentrale wird als LAN ETHERNET verwendet. In den Satellitenrechner wird der PEARL-Teil einer PEARL-Engine als plattenloser Systemknoten verwendet. Die Software wird dort von einem Streamer geladen.

An der Prozeßperipherieseite wird zur Erneuerung ein Meßautomat (Sender- und Empfänger) und eine neu zu entwickelnde, prozessorgesteuerte ARD-Einheitstontruhe eingesetzt.

2. Struktur der operationellen Software

In jedem Rechner des Systems werden folgende Komponenten zum Einsatz gelangen (B i l d 4):

- PEARL-Anwendungsprogrammsystem lauffähig in einem
- PEARL-Laufzeitsystem [6]
- Echtzeitdatenbanksystem BAPAS-DB [7]
- Mensch-Maschine-Kommunikationssystem MCC [8]

PEARL-Compiler und Laufzeitsystem, Datenbank und das Mensch-Maschine-System sind Produkte der Fa. WERUM; die Anwendungssoftware wird von DORNIER SYSTEM und WERUM gemeinsam erstellt.

Die Entscheidung für PEARL basiert darauf, daß es eine Sprachempfehlung der Rundfunkanstalten gibt, PEARL einzusetzen, damit der Software-Austausch erleichert und die Software-Entwicklungszeit verkürzt werde. Außerdem empfiehlt sich PEARL unter den verfügbaren Sprachen wegen der Möglichkeit, alles in einer einzigen Sprache und mit einer anwendungsorientierten Datenstruktur-Definition zu realisieren.

3. Software-Entwicklungsumgebung und ihr Einsatz

3.1 Software Werkzeuge

Da die Software-Werkzeuge, Spezifikationssprachen, Verfahren und Methoden "in allen Tätigkeitsbereichen und Phasen" des Projektes angewendet werden sollen, muß eine "solche Kombination von Hilfsmitteln mit Rechnerunterstützung eine Software-Produktionsumgebung" 2 genannt werden.

Als Entwicklungswerkzeuge werden eingesetzt:

- EPOS Entwicklungs- und projektmanagementorientiertes Spezifikations-System
- Editor MED
- PEARL-Generator
- VICO Versions,- Schnittstellen- und Konfigurationskontrolle
- SPRAM System Performance Report and Maintenance
- PEARL-Testsystem
- Betriebssystem MUNIX
- PEARl-Compiler für PEARL-Engine

Der Einsatz erfolgt in den Projektphasen wie es in B i l d 5 dargestellt ist.

Alle Werkzeuge werden auf dem MUNIX-Teil der PEARL-Engine eingesetzt mit Ausnahme von EPOS während der Pflichtenheftphase. Das Pflichtenheft wird mit EPOS auf einer IBM 370 unter VM|CMS erstellt.

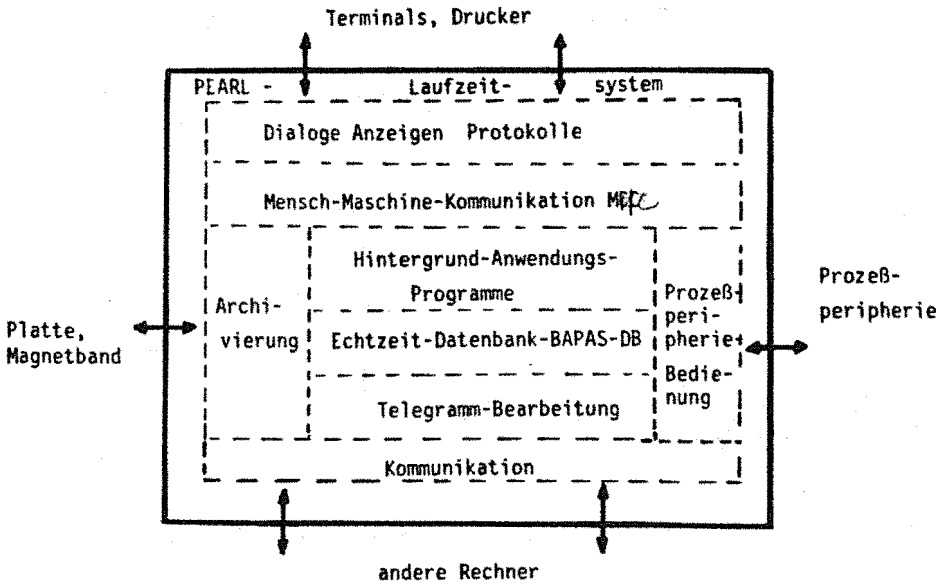


Bild 4. Operationell Software-Komponenten

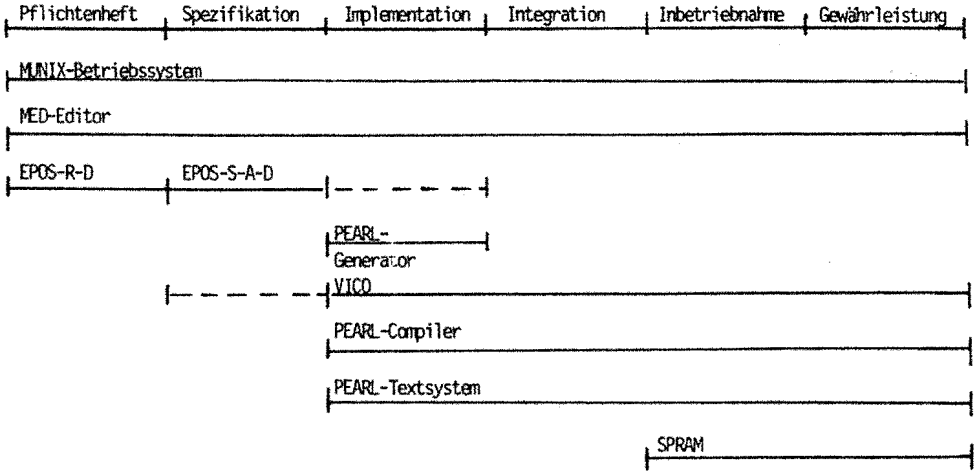


Bild 5. Software-Werkzeuge in den Projektphasen

3.2 Verbindung der Software-Entwicklungswerkzeuge zu einer Software-Produktionsumgebung

Die Erstellung des Pflichtenheftes aus der Leistungsbeschreibung des Auftraggebers erfolgt mit den Textdokumentationsmöglichkeiten des Werkzeuges EPOS und den verfügbaren Texteditoren. Eine Formalisierung des Textes erfolgt über die Kennzeichnung mittels formaler Anforderungsnummern und Attributsbezeichnungen. Diese formalen Kennzeichnungen erlauben es beim Übergang in die Spezifikationsphase, die Softwarearbeitspakete leichter zu definieren und ihnen die Anforderungen zu zuordnen.

Durch logische Verknüpfung mehrere Attribute können alle Anforderungen eines Softwarearbeitspaketes gesammelt werden. Würden beispielsweise alle den Mensch-Maschine-Dialog betreffenden Anforderungen mit dem Attribut "DIALOG" und alle die Rechner bei der DBP betreffenden mit "DBP" gekennzeichnet, so kann mittels Verknüpfung "DIALOG" und "DBP" der Anforderungskatalog für den Bildschirmdialog bei der DBP herausgefiltert und dokumentiert werden. (B i l d 6)

Die ansschliessende Grob- und Feinstrukturierung der Software wird mittels der Spezifikationssprache EPOS-S für die Elemente Task's, Prozeduren, Daten und Interrupts erfolgen. Die daraus resultierende Dokumentation aus Text- und Graphikteilen bildet schon einen wesentlichen Teil der auszuliefernden Enddokumentation.

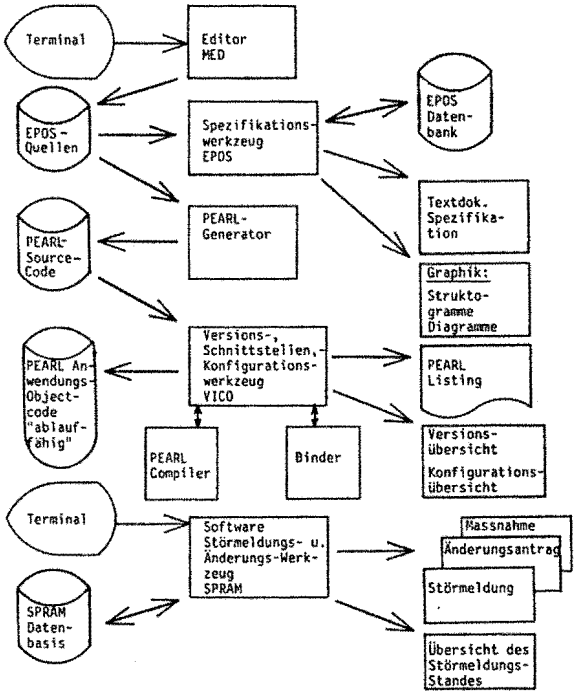


Bild 6. Ein- und Ausgaben der Software-Werkzeuge

Zur Schnittstellenabsprache zwischen den verschiedenen Softwareentwicklern eines Teams und auch zwischen verschiedenen Firmen wird die Spezifikation der Softwareschnittstellen mit dem Sprachelement MODULE der Sprache VICO-L erfolgen [3].

VICO-L ist eine Sprache zur formalen Beschreibung von Programmsystemen. Diese Beschreibungen werden analysiert und in geeigneter Form in der VICO-Datenbank abgelegt. Anschließend verfügt VICO über ein internes Abbild der Struktur eines Programmsystems, das die Grundlage der Schnittstellen,- Versions- und Konfigurationskontrolle bildt.

VICO-L unterscheidet die Analyseeinheiten
- (Programm) System (SYSTEM)
- Konfigruation (CONFIGURATION)
- Modul(familie) (MODULE)
- Implementierung (IMPLEMENTATION)

Ein System führt alle Modulfamilien auf, die zu einem Programmsystem gehören.
Eine Konfiguration beschreibt, welche Modulfamilien benutzt werden, welche Beziehungen zwischen den Modulfamilien bestehen und welche Revisionen heranzuziehen sind.
Ein Modul beschreibt die Schnittstelle und nennt die Implementierungen einer Modulfamilie. Die Schnittstelle wird definiert durch im- und exportierte Konstanten, Typen, Variablen und Prozeduren.
Eine Implementierung entspricht einem Quellprogramm in anderen Sprachen.

In der anschließenden Codierung wird durch Ausformulierung der Code-Teile in dem EPOS-S-Beschreibungsmittel ACTION die Grundlage für eine anschließende PEARL-Code-Generierung geschaffen.
Mit einem PEARL-Code-Generator wird aus den EPOS-Quelltextdateien je PEARL-Module eine PEARL-Source-Datei erzeugt.

Der erzeugte PEARL-Code enthält schon die notwendige Erläuterung des Codes als Kommentartext. Der Code ist direkt für den Compiler übersetzbar. Mit der Entstehung des Source-Code wird die weitere Verwaltung von Source-, Objekt- und ablauffähigem Code mittels des Werkzeugs VICO übernommen. Code-Formen können direkt in die VICO-Datenbank übernommen werden.

Mit den Beschreibungsmöglichkeiten KONFIGURATION und einer Generierungsfunktion COMPILE, ähnlich dem "make" des Betriebssystem UNIX, können die ladefähigen Module erzeugt werden [3].

Änderungen im Source-Code werden in VICO als Revision eingetragen und führen dazu, daß der zugehörige Bindemodul und alle Programme, die diesen Modul benutzen, nicht mehr aktuell sind und automatisch ein Übersetzungs- und Bindevorgang eingeleitet und vermerkt wird, daß die begleitenden Dokumente nicht mehr aktuell sind.

Die ablauffähigen Programme werden mit einem PEARL-Testsystem, das auf Source-Ebene agiert, getestet [4].

Das Testsystem erlaubt es, ein PEARL-Programm einschließlich des Zusammenspiels seiner Tasks auf einem Testrechner ohne angeschlossenen Prozeß interaktiv zu testen, wobei das Zeitverhalten des Programms und der zu steuernde Prozeß simuliert werden können.

Im Umgang mit dem Testsystem muß der Benutzer lediglich Kenntnisse der Sprache PEARL und der PEARL-ähnlichen Bedienungssprache des Testsystems besitzen.
Abfrage und Änderung von Objekten des PEARL-Programms erfolgen unter Verwendung der im PEARL-Programm vereinbarten Bezeichner.

Beim Ablauf des zu testenden Programms werden die folgenden Kontrollfunktionen automatisch durchgeführt:
- Überwachung der Zugriffe auf Referenzvariablen
- Indexkontrolle
- Erkennung arithmetischer Fehler
- Überwachung der Adreß-Arithmetik
- Erkennung von Aktivierungen bereits aktiver Tasks
- E/A-Kontrolle

Der Benutzer hat zudem die Möglichkeit, interaktiv im Dialog des PEARL-Programm zu überwachen und ggf. korrigierend einzugreifen.

Das PEARL-Testsystem für Rechnernetze BAPAS-TS 5 ist für das Testen und Analysieren von PEARL-Programmen im Online-Betrieb in Rechnernetzen konzipiert und entwickelt. Testfunktionen sind von einem Testsystemrechner aus über Kommunikationswege auf PEARL-Programme anwendbar, die auf anderen Rechnern im Netz ablaufen. Diese Fähigkeit ist durch den Einsatz einer "lokalen Kommunikation" zwischen PEARL-Prozessen auch auf einen Rechner übertragbar.

Für das Konfigurationsmanagement wird ergänzend zum Werkzeug VICO, das in erster Linie die Source-Verwaltung unterstützt, ein Werkzeug SPRAM (Systems Performance and Maintenance) zur Unterstützung für die formale Bearbeitung von Störmeldungen und Änderungsanträgen ab der Projektphase Inbetriebnahme eingesetzt. Störmeldungen, Änderungsanträge und Maßnahmen werden mit dem Rechner erfasst, verwaltet und protokolliert.

Der Ablauf wird über ein Menueprogramm gesteuert. Die Programmbedienung erfolgt über vordefinierte Masken.

Dabei sind folgende Funktionen möglich:

- Automatische Vergabe einer neuen Störmeldungs-Nummer
- Vergabe einer Änderungsnummer
- Stellen eines Änderungsantrags
- Formulieren von Maßnahmen
  Die Maßnahmen beziehen sich auf eine Störmeldung oder einen Änderungsantrag.
- Übersichts-Liste mit Änderungsanträgen
- Übersichts-Liste mit Störmeldungen
- Protokoll der gewünschten Liste auf Drucker oder in ein Testfile

Zusammengehörende Dokumente sind untereinander durch Verweise verknüpft.

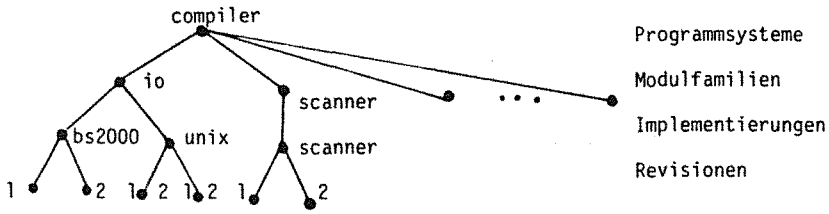
3.3 Entwicklungsaufwand für die Übergänge zwischen den Werkzeugen

Die Programme "PEARL-Generator" und "SPRAM" sind in DEC-PASCAL implementiert und müssen nach OREGON-PASCAL von PCS portiert werden. Der "PEARL-Generator", entwickelt für ATM-PEARL, ist an WERUM-PEARL zu adaptieren.
Damit eine uniforme und damit fehlerfreie Beschreibung der Softwarestruktur erreicht wird, muß das Be-Schreibungselement MODULE von VICO zur Schnittstellendefinition aus den EPOS-S-Quelltexten erzeugbar sein. (B i l d 7)

Ein dort ebenfalls vorhandenes Sprachelement ACTION MODULE läßt sich zusammen mit den in EPOS-S spezifizierten Elementen ACTION PROCEDURE und DATA für Prozeduren bzw. Daten dazu verwenden, diese Schnittstellendefinition in EPOS-S zu spezifizieren.

Ähnlich dem Konzept des "PEARL-Generators" sind in einer Analyse eines Gesamt-EPOS-Quellfile incl. des beschreibenden Elementes ACTION MODULE alle referenzierten Objekte (Daten und Prozeduren) zu suchen und mit ihren Grundinformationen (Datentyp und Länge bzw. Aufruf-Parameter) zu extrahieren.

Der Gesamtaufwand für die Realisierung dieser o.a. Übergänge zwischen den Software-Werkzeugen liegt im Bereich einiger Mann-Monate. Abschließend ist festzuhalten, daß sich das Ziel über eine gesamte Projektlaufzeit die Software auf Basis einer Uniquelle bei akzeptablem Bedienungsaufwand zu entwickeln, durch einen vertretbaren Aufwand an Zusatzentwicklung für die Unterstützung des automatischen Übergangs zwischen Software-Werkzeugen erreichen läßt.




```
MODULE io;
   EXPORT
      string = CHARACTER 14;
      open:    PROCEDURE (filename: string,...);
      read:    PROCEDURE (ch: CHAR 1);
      write:   PROCEDURE (ch: CHAR 1);
      close:   PROCEDURE;
   END;
   IMPLEMENTATION bs2000; unix END;
   END io;
```

Bild 7. Schnittstellenbeschreibungselement MODULE

Literatur

1 Lastenheft für das Gesamtsystem
"Erneuerung ARD-Hörfunk-Sternsytem"
Dornier System GmbH, November 1985.

2 EPOS-Kursbeschreibung: Darstellung der wichtigsten Eigenschaften des entwicklungs- und projektmanagements-orientierten Spezifikations-Systems EPOS.
Forschungsinstitut für Regelungstechnik und Prozessautomatisierung, Stuttgart; GPP Gesellschaft für Prozessrechnerprogrammierung mbH, Oberhaching, April 1985.

3 VICO; Versions,- Schnittstellen- und Konfigurationskontrolle.
WERUM Datenverarbeitungssysteme GmbH, Lüneburg,
Reg. 9.1.2/8506/Ex.

4 PEARL-Testsystem, Benutzerhandbuch für PCS CADMUS.
WERUM Datenverarbeitungssysteme GmbH, Lüneburg,
Reg. 2.7.3.6.1/8507/FB

5 BAPAS-TS: PEARL-Testsystem für Rechnernetze, Benutzerhandbuch.
WERUM Datenverarbeitungssysteme GmbH, Lüneburg,
Reg. 2.7.7.1/8510/FB.

6 Werum, Windauer: Introduction to PEARL.
VIEWEG Verlag, 3. Auflage 1985

7 BAPAS-DB; Das offene Echtzeitdatenbanksystem für die Prozeßautomatisierung.
WERUM Datenverarbeitungssystem GmbH, Lüneburg,
Reg. 3.1.1/8510/EX.

8 MMC-System für PEARL-Engine 68000 und PCS Cadmus 92xx unter Munix, Benutzerhandbuch.
WERUM Datenverarbeitungssysteme GmbH, Lüneburg,
Reg. 4.4.1.1/8509/EX

9 Leistungsbeschreibung Erneuerung des Hörfunkstern-Systems, Hessischer Rundfunk
6. Febr. 1984

Anschrift des Autors

Frank, Roland J.
DORNIER SYSTEM GMBH
Abtl. ZIUA
Postfach 1360
7990 Friedrichshafen - 1

Tel.: 07545/B-8327

Kommunikation zwischen und mit PEARL-Echtzeitsystemen

Dipl.-Ing. Hans-Jörg Haubner, Karlsruhe

Zusammenfassung:

Ausgehend von einem einfachen Modell zur rechnerinternen Taskkommunikation werden prinzipielle Kommunikationsmechanismen mittels der Sprachmittel globale Daten und Semaphore beschrieben. Diese Mechanismen können mit wenigen zusätzlichen Betriebssystemfunktionen und -schnittstellen auf Echtzeitsysteme über lokale Netze (LAN) gekoppelt übertragen werden. Als Beispiel dient hier ein verteiltes Automatisierungssysytem.
Prinzipiell sind aber auch Kommunikationsfunktionen unmittelbar in geeigneten Hochsprachen realisierbar. THYNET ist ein Kommunikationssystem (WAN) für den betrieblichen Einsatz, bei dem von den Datenübertragungsprozeduren (X.25/Ebene 2) bis hin zum Netzwerkmanagement alle Funktionen in PEARL realisiert wurden.

An einem letzten Fallbeispiel wird die Einbettung eines PEARL-Systems in ein herstellerspezifisches Kommunikationssystem am Beispiel von SINEC demonstriert. Dieses System löst die Aufgabe, in einem Verbund von heterogenen Rechnern Daten, die lokal erfaßt werden, global allen Verbundteilnehmern mittels einfacher Verfahren für übergreifende Auswertungen konzentriert zur Verfügung zu stellen.

Stichworte: Verteilte Systeme, PEARL, von Kommunikationsfunktionen.

Abstract

Starting with a model for interprocess communication, basic communication mechanisms are described by means of the programming constructs, global-data and semaphores. Three approaches are presented providing communication functions for programming distributed systems. These are based upon functions integrated in the programming languages, functions realized in PEARL and the use of standard systems. In case studies for LAN and WAN, specific features and similarities of these appraches are explained.

Keywords: distributed systems, PEARL, programming of communication functions.

1. Ausgangssituation

Komplexe Prozeßautomatisierungs-Systeme werden heute schon beinahe selbstverständlich als verteilte Systeme ausgelegt. Ausschlaggebend hierfür sind Vorteile wie dedizierte Rechnerleistung vor Ort, Fehlertoleranz und Modularität.

Diesen Vorteilen stehen jedoch Aufwendungen für zusätzliche Kommunikations-Komponenten in Hard- und Software gegenüber. Üblicherweise werden dabei untere Schichten der Kommunikationsarchitektur durch Hard- bzw. Firmware-Komponenten, mittlere Schichten durch Betriebssystemfunktionen und/oder Kommunikationssysteme und obere Schichten durch Anwenderprogramme realisiert.

Allen Lösungen ist jedoch gemeinsam, daß entsprechend geeignete Schnittstellen und Funktionen dem Anwender zur Kommunikation zur Verfügung gestellt werden müssen. Dies kann auf der Basis vorhandener Sprachmittel der eingesetzten Programmiersprache oder durch Erweiterungen erfolgen.

Im vorliegenden Beitrag sollen für diese Vorgehenswseise verschiedene Lösungsansätze für PEARL-Programmsysteme einander gegenübergestellt werden. Als Fallbeispiele werden verteilte Prozeßautomatisierungssysteme, die am Institut für Informations- und Datenverarbeitung der FhG in PEARL programmiert und in der Industrie als Pilotanlagen eingesetzt wurden, herangezogen.

2. Ein einfaches PEARL-Modell der Kommunikation

Das Modell sieht zwei Instanzen, die durch PEARL-Tasks realisiert sind, vor, zwischen denen eine Kommunikation erfolgt. Die eine Task stellt den Sendeprozeß, die andere den zugehörigen Empfangsprozeß dar. Das Modell ist zunächst auf einen Speicher bzw. Adreßraum, den beide Prozesse gemeinsam benutzen, beschränkt.

Die Kommunikation kann dann wie folgt modelliert werden:

```
DCL PUFFER  TELE     GLOBAL;
DCL ANZEIGE QUITTUNG GLOBAL;
DCL ES  SEMA PRESET(0) GLOBAL;
DCL AS  SEMA PRESET(0) GLOBAL;

SENDER : TASK
         DCL X QUITTUNG;
         DCL Y TELE
          :
         PUFFER:= Y; Schreiben auf globale
                     Daten
         RELEASE ES; Synchronisation
                     Empfänger
         REQUEST AS; Warten auf Quittung
         X:= ANZEIGE; Quittung lesen

END;

EMPFANG : TASK;
          DCL Y TELE;
          DCL X QUITTUNG;
          REQUEST ES; Warten auf Empfang
          Y:= PUFFER; Lesen von globalen
                      Daten
          ANZEIGE:=X; Anzeige setzen
          RELEASE AS; Senden, Synchroni-
                      sieren
END;
```

Anhand dieses Modelles kann z.B. ein rechnerinternes Auftrags- oder Nachrichtensystem zur Interprozeßkommunikation realisiert werden. Solche Systeme werden als Software-Bus bezeichnet, da sie auf einfache Weise Interaktionen zwischen einer Vielzahl von Prozessen erlauben. Die dargestellten Abläufe sind dann als Sende- bzw. Empfangsprozedur zusammengefaßt und der Empfangspuffer durch ein Warteschlangensystem ersetzt.

3. Kommunikation in verteilten Systemen

Das vorgestellte Modell läßt sich auf verteilte Systeme mit getrennten Prozessoren und Speichern übertragen. Dazu müßte die Semantik das Sprachelement GLOBAL, in PEARL im Sinne von Rechner-global definiert, erweitert werden, etwa im Sinne von System-global. Zugriffe auf Datenobjekte, die als System-global deklariert sind und sich in einem anderen Rechner befinden, würden dann Kommunikationsvor-

gänge, die im Rahmen der PEARL-System-Umgebung zur Verfügung gestellt werden müßten, beinhalten.
Dieses Verfahren wird jedoch i.a. nicht praktiziert, üblich sind die beiden im folgenden vorgestellten Vorgehensweisen.

Zum einen besteht die Möglichkeit, die Sprache PEARL um entsprechende Kommunikations-Sprachmittel zu erweitern. Entsprechende Vorschläge hierzu liegen vor /1/, /2/. Ihnen gemeinsam ist, die beiden Elementaroperationen Senden und Empfangen als Sprachmittel ergänzt um Verbindungsstrukturen zur Verfügung zu stellen:

- TRANSMIT daten TO port;
- RECEIVE daten FROM port;

Diese entsprechen prinzipiell den Modellfunktionen, wobei z.T. die Synchronisationsmechanismen wie z.B. Warten, Nichtwarten und weitere Kontrollmechanismen variiert werden.

Die zweite Möglichkeit ähnelt stark der ersten; sie umgeht jedoch das Problem der Spracherweiterung, indem vergleichbare Funktionen auf der Basis von über CALL aufrufbaren Prozeduren zur Verfügung gestellt werden:

- CALL SENDF (port, daten,...);
- CALL RECEIV (port, daten,...);

Diese Funktionen wurden, wie im 1. Fallbeispiel dargestellt, in Rahmen eines verteilten Prozeßautomatisierungsystemes ins PEARL-Betriebssystem bzw. Laufzeitsystem integriert.
Beiden Lösungen gemeinsam ist, daß in der PEARL-Systemumgebung entsprechende unterlagerte Kommunikationskomponenten in Hard- und Software zur Verfügung gestellt werden.

4. Fallbeispiel: Verteiltes Prozeßautomatisierungssystem in einem Oxygenstahlwerk

Zur Automatisierung komplexer technischer Prozesse wurde im IITB ein verteiltes Prozeßautomatisierungssystem (RDC-System) entwickelt. Ein solches System wird seit 1981 von der Thyssen Stahl AG als Leitsystem zur Steuerung und Überwachung einzelner Verfahrensschritte im Oxygenstahlwerk eingesetzt /3/.

Die Konfiguration des Systems ist in Bild 1 dargestellt. Seine Strukturierung erfolgte bereichsweise anhand des vorgegebenen "natürlichen" Produktionsablaufes der zu automatisierenden Verfahrensschritte, Konverter und Argonspüle, Stahlentgasung, Sublanze, Stahlwerksrechner ergänzt um ein zentrales Wartensystem zur Bedienung der Gesamtanlage.

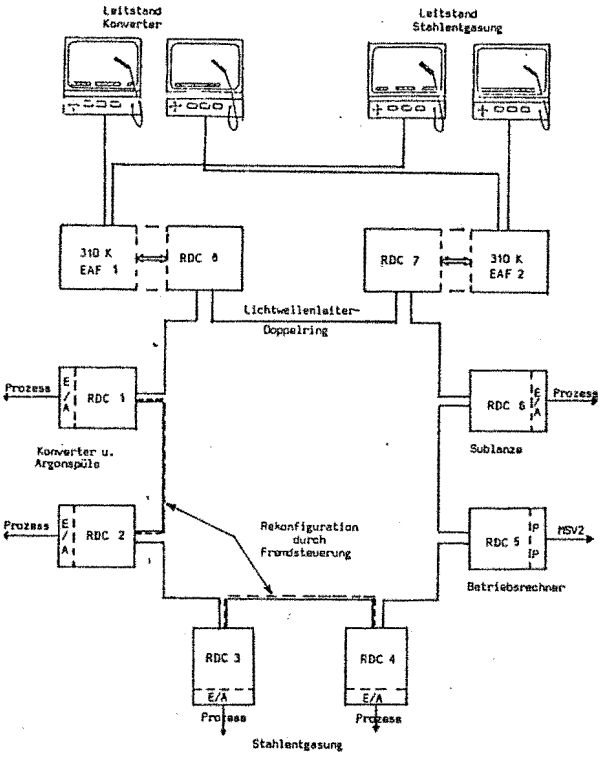


Bild 1. Verteiltes Leitsystem im Stahlwerk.

Die eigentlichen MSR-Aufgaben in den einzelnen Bereichen werden zunächst dediziert von Mikroprozessor-Systemen (Stationen), wahrgenommen. Diese sind untereinander und mit dem Wartensystem über ein lokales ringförmiges Kommunikationssystem auf der Basis eines Lichtwellenleiters miteinander verbunden.
Die Mikroprozessorstationen sind in PEARL programmiert. Als Kommunikationsfunktionen

werden SENDF, RECEIVE über das PEARL-Betriebssystem zur Verfügung gestellt, das hierzu um entsprechende Komponenten ergänzt wurde. Diese Kommunikationsfunktionen erlauben in der in Pkt. 3 beschriebenen Weise eine Kommunikation mittels Telegrammen bis maximal 128 Bytes. Auf diesen Basisfunktionen sind in PEARL geschriebene höhere Kommunikationfunktionen aufgesetzt. Es werden hiermit folgende typische Kommunikationsfunktionen wahrgenommen:

(1) Zyklische und ereignisgesteuerte Übertragung der lokal erfaßten Prozeßgrößen einschließlich Systemzuständen von den Stationen zum Wartensystem, zur Prozeßbeobachtung und -protokollierung.

(2) Übertragung von initialen Parametersätzen, Systemzuständen und Bedieneingriffen vom Wartensystem zu den Stationen zur zentralen Parametrierung und Bedienung.

(3) Zyklischer und ereignisgesteuerter Austausch von Parametersätzen, Prozeßgrößen und Systemzuständen paarweise zwischen den Stationen zur gegenseitigen Aktualisierung der Redundanzfunktionen.

Während die unter (1) und (2) beschriebenen Funktionen als "Fern-Meß- bzw. Steuerungs-Funktionen" aufgefaßt werden können, wird die unter (3) beschriebene Funktion dazu benutzt, die Stationen im Konverter- und Stahlentgasungsbereich paarweise in funktionsbeteiligter Redundanz arbeiten zu lassen. Fällt eine Station eines Paares aus, so übernimmt die Nachbarstation deren Aufgaben mit. Um hier eine weitgehend stoßfreie Übernahme zu erreichen, werden hierfür wichtige Daten wie z.B. Sollwerte u. Regelparameter von der prozeßführenden Station in der Nachbarstation ständig aktualisiert. Zur Steigerung der Effizienz wurden hier zwei weitere Basisfunktionen zur Kommunikation in das Betriebssystem integriert. Diese Funktionen HOLGM, ZUWGM ermöglichen es in einer Station direkt auf den Speicher einer anderen Station lesend bzw. schreibend zuzugreifen. Die eigentliche Empfangsfunktion entsprechend RECEIV entfällt. Zur Adressierung des "fremden" Speichers werden dabei einmalig im Anlauf der Stationen entsprechende Pointerlisten (Listen vom Typ REF) ausgetauscht, die sozusagen ein globales Adreßverzeichnis stationsübergreifender Datenobjekte darstellen.

Bei diesem System hat es sich gezeigt, daß sich selbst mit diesen einfachen Basisfunktionen komplexe Kommunikationsfunktionen aufbauen lassen. Insbesondere konnten die Nachteile verteilter Systeme, die insbesondere in einem erhöhten oft effizienzminderndem Kommunikationsaufwand und einer erschwerten oder unübersichtlichen zentralen Bedienung bestehen, vermieden werden.

5. Kommunikation mittels PEARL-Systemen

Bisher wurde gezeigt, wie Kommunikation zwischen PEARL-Systemen mit Basisfunktionen, die über die Systemumgebung zur Verfügung gestellt werden, realisiert wird. Es stellt sich die Frage, ob solche Funktionen auch mittels PEARL realisiert werden können. Dies soll am Beispiel der Ebene 2 von X.25, entsprechend HDLC, demonstriert werden.

Vorausgesetzt wird eine Hardware-Komponente am Rechner, die mechanische und elektrische Eigenschaften (Ebene 1) sowie die Parallel-Seriell-Wandlung der zu übertragenden Daten einschließlich der Flag- und CRC-Generierung und -Prüfung zur Verfügung stellt.

Diese Komponente kann über Ein- und Ausgabeoperationen von PEARL aus, betrieben werden. Hierfür sind in PEARL prinzipiell die Sprachmittel Systemteil, Dation, Read und Write vorhanden. Diese sind jedoch, wie im vorliegenden Fall, bei vielen Systemen auf Standardperipheriegeräte wie Terminal, Platte oder Prozeß-E/A beschränkt. Es ist daher sinnvoll, vergleichbare Funktionen als einfache Ein-Ausgabe-Operationen z.B. im Laufzeitsystem

zu installieren:

```
TYPE DATENBLOCK STRUCT [LAENGE FIXED,
                        DATEN(256) BIT(16)];
TYPE GERAET FIXED;
SPC ENTRY ATRANS (DATENBLOCK; GERAET);
SPC ENTRY ETRANS (DATENBLOCK, GERAET);
```

Weiterhin ist je ein Interrupt für Ein- bzw. Ausgabe notwendig, die bei den PEARL-Ein-Ausgabeoperationen implizit vorhanden sind:

```
SPC (I-IRUPT, O-IRUPT) INTERRUPT;
```

Damit können die beiden elementaren Funktionen Senden und Empfangen unter zu Hilfenahme zweier Task zur Interruptsteuerung analog zu dem im Pkt. 2 beschriebenen Modell wie folgt prinzipiell dargestellt, realisiert werden.

```
T1 : TASK;
     WHEN I-IRUPT CONTINUE;
     REPEAT;
        SUSPEND T1;
        RELEASE RECEIV-SEMA;
     END;
END;

T2 : TASK
     WHEN O-IRUPT CONTINUE;
     REPEAT;
        SUSPEND T2;
        RELEASE SEND-SEMA;
     END;
END;

T3 : TASK;
     :
        REQUEST RECEIV-SEMA; WARTEN
        CALL ETRANS (....);  LESEN
     :
END;

T4 : TASK;
     :
     CALL ATRANS (....); Schreiben
     REQUEST SEND-SEMA;  Warten
     :
END;
```

Anzeigen und Quittungen werden über entsprechende Protokollelemente von HDLC abgewickelt.

6. Fallbeispiel: THYNET, ein betriebliches Kommunikationssystem

THYNET ist ein am IITB entwickeltes paketvermittelndes Kommunikationssystem zum Aufbau von beliebig vermaschbaren Rechnernetzen. Das Gesamtsystem bildet einen heterogenen Verbund, der sich aus dem eigentlichen Kommunikationssystem und den daran angeschlossenen Teilnehmern zusammensetzt (s. Bild 2). Die Teilnehmer i.a. Rechner und Terminals sind heterogen und stellen die Datenquelle des Verbundes dar. Das Kommunikationssystem ist ein beliebig vermaschtes Netz aus Netzknotenrechnern, die über serielle Datenleitungen (Telefonleitungen) und Modems miteinander verbunden sind /4/.

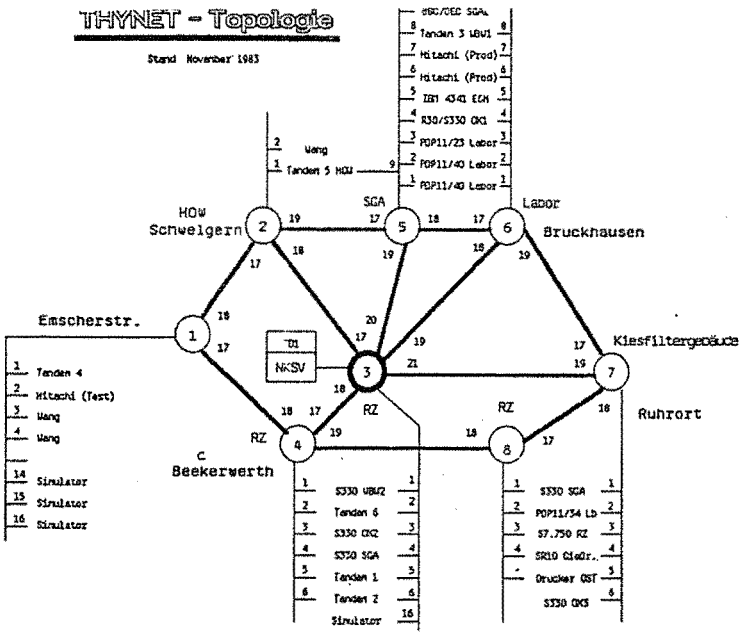


Bild 2: THYNET-Topologie.

Das Netz arbeitet nach dem Prinzip der Paketvermittlung. Übertragen werden Telegramme, deren maximale Länge 512 Bytes beträgt. Auf den Punkt-zu-Punkt-Verbindungen zwischen den Netzknoten wird einheitlich die Übertragungsprozedur HDLC eingesetzt. Die Leitungskontrolle wird dabei von sogenannten Prozedurprozessoren durch-

geführt. Diese Prozedurprozessoren werden vom PEARL-System über die beschriebenen Funktionen ATRANS und ETRANS angesteuert. Die Funktionen von HDLC werden bis auf FLAG- und CRC-Generierung über PEARL abgewickelt.

Die Ankopplung der Teilnehmer erfolgt ebenfalls über die Prozedurprozessoren. Die Netzzugangsprozedur ist mit der Ebene 3 von X.25 vergleichbar und ist ebenfalls in PEARL programmiert. Teilnehmer und Kontrolltrollinstanzen verkehren jeweils untereinander über Einzeltelegramme oder Sequenzen von Telegrammen, die über eine Wegesteuerung im Netz transparent übertragen werden.

Neben den bisher beschriebenen unbedingt notwendigen Funktionen zur Kommunikation zwischen Teilnehmern über das Netz sind im Netz weitere Funktionen zum System-Management integriert. Diese Leistungen stehen innerhalb des Netzes als Funktionsverbund zur Verfügung und können von speziellen Netzknoten aus netzweit benutzt werden. Alle Funktionen werden dabei über hierfür in den Netzknoten implementierte Kontrollinstanzen mittels spezieller Kontrolltelegramme abgewickelt.

Zum System-Management gehören folgende Teilsysteme:

- das Überwachungssystem zur zentralen Anzeige von Fehlern, Warnungen und Statusänderungen. Hiermit lassen sich z.B. Protokollereignisse und Leitungsfehler wie CRC-Fehler, Modemsignale, HDLC-Verbindungsaufbau überwachen, um schnell und zielgerichtet auf Störungen zu reagieren bzw. frühzeitig Fehlertrends zu erkennen.
- das Bediensystem zum zentralen Laden der Netzknoten (down-line-loading) und zur Umkonfigurierung und Umparametrierung von Netzknoten.
  Hiermit können im laufenden Betrieb z.B. einzelne Netzknoten nachgeladen werden bzw. bei Systemänderungen Konfigurationsparameter wie Wegetabellen modifiziert werden.
- das Ferndiagnosesystem zur Programmverfolgung (TRACE) und zum Speicherabzug (post-mortem-dump). Für Test- und Wartungszwecke lassen sich hiermit Programmabläufe zur Fehler-Lokalisierung analysieren.

Das beschriebene Programmsystem der Netzknoten ist vollständig in PEARL beschrieben. Es besteht aus 14 Modulen mit insgesamt 11 Task und benötigt ohne Betriebs- und Laufzeitsystem ca. 41 KW.

THYNET ist seit 1981 bei der Thyssen Stahl AG im Betrieb und wurde kontinuierlich auf den heutigen Umfang von 8 Netzknoten mit ca. 30 Teilnehmern erweitert.

Insbesondere bei diesem System kann gezeigt werden, daß auch Kommunikationsschichten der niedrigeren Ebenen wie z.B. Leitungssteuerung (HDLC) und Netzwerksteuerung (ähnlich X.25/3) sowie Netzwerk-Managementfunktionen mit PEARL effizient programmiert werden können.

7. Kommunikation über Standard-Kommunikationspakete

Als dritter Ansatz soll hier die Möglichkeit der Kommunikation über Standard-Programmsysteme, wie sie z.B. von Rechnerherstellern im Rahmen der Systemumgebung zur Verfügung gestellt werden, beschrieben werden.

Solche Standardsysteme besitzen i.a. Schnittstellen für Anwenderprogramme in Form von System-Makros oder Assembler-Prozeduren. Diese müssen auf PEARL-Niveau angehoben werden. Dies bedeutet, daß eine Schnittstellenschicht zu erstellen ist, die im wesentlichen aus Assemblerprozeduren besteht. Diese werden von PEARL aus mittels CALL aufgerufen und leisten die Umsetzung der Prozedurparameter von den PEARL- auf die Assembler-Konventionen sowie den Aufruf der System-Prozeduren. Damit sind darin die Kommunikationsdienste des Standardsystems von PEARL aus verfügbar.

Der Umfang dieser Dienste deckt notwen-

digerweise die unteren Ebenen 1 bis 4 des ISO Referenzmodelles ab, höhere Ebenen sind sehr stark herstellerspezifisch. So liegen auch einheitliche Schnittstellen nicht vor.

Am Beispiel von SINEC, einem Kommunikationssystem für die Rechnerfamilie SICOMP R und M, soll hier die grundlegende Vorgehensweise erläutert werden. Zur Verfügung stehen hier u.a. Dienste wie X.25 oder ein verbindungsorientiertes Transportprotokoll.
Die elementaren Kommunikationsfunktionen sind

- Anmelden des Anwendersubsystems (SNANS)
- Sendeaufruf (SNSDT)
- Empfangsaufruf (SNSREC)
- Warteaufruf für Quittung auf Senden bzw. Empfang (WAIT)

Diese Funktionen entsprechen den Basisfunktionen des Kommunikationsmodells. Damit ergibt sich eine dem Modell entsprechende Ablaufstruktur:

Senden:

```
CALL SNANS (...);  Anmelden
:
CALL SNSDT (...);  Senden
:
CALL WAIT  (...);  WARTEN
:
```

Empfang:

```
CALL SNANS (...);  Anmelden
:
CALL SNSREC(...);  Empfangsaufruf
CALL WAIT  (...);  Warten
Daten lesen
:
```

Für das eigentliche Lesen von Daten existiert keine spezielle Funktion. Bei allen Aufrufen werden entsprechende Anzeigen zurückgegeben, deren Auswertung hier nicht dargestellt wurde.

8. Fallbeispiel: Koppelrechnersystem für den Datenaustausch im heterogenen Rechnerverbund

Im Auftrag der Badenwerk AG wird derzeit im IITB ein Koppelrechnersystem entwickelt, das einen EVU-internen wie auch übergreifenden Datenaustausch zwischen heterogenen Leitstellen- und Energiewirtschaftsrechnern ermöglicht (s. Bild 3). Die Grundidee ist, über einen sog. Koppelrechner die einzelnen Teilnehmer des Verbundes sternförmig miteinander zu verbinden /5/.

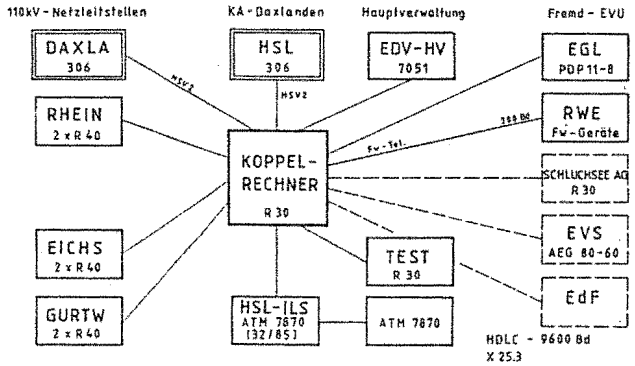


Bild 3: Konfiguration des KOPPELRECHNER-Systems.

Das System erfüllt folgende Hauptaufgaben:

- Für die Teilnehmer des Verbundes stellt der Koppelrechner ein Kommunikationssystem für den transparenten Transport von Daten zur Verfügung.

- Der Koppelrechner stellt für die Teilnehmer eine zentrale Datenbasis dar, in die bzw. aus der aktuelle Meßwerte, Zählwerte und Meldungen des elektrischen Netzes weitgehend wahlfrei von den Teilnehmern übernommen bzw. an die Teilnehmer übergeben werden.

Zur Erfüllung dieser Aufgaben wird SINEC M eingesetzt. Von SINEC werden die Subsysteme VS (Verbindungssteuerung, für Transportprotokoll), PV (Paketvermittlung für X.25), PU (Protokollunabhängige Kommunikation für einfache Kommunikation) und FV (Funktionsverbund) eingesetzt. Damit wird die Forderung nach Heterogenität weitgehend erfüllt.

Die Schnittstellen zu den Subsystemen liegen als Makros vor. Über eine in Assembler programmierte Schnittstellenschicht werden diese von PEARL aus angesprochen. Weiterhin werden die Protokoll-Unterschiede der unterschiedlichen Teilnehmeranschlüsse nivelliert, so daß der Verbund aus der Sicht eines einzelnen Teilnehmers homogen erscheint.

Bisherige Erfahrungen haben gezeigt, daß durch den Einsatz dieses Standardsubsystems weitgehende Hardware-Unabhängigkeit erreicht wird. Insbesondere können die Programme innerhalb der Systemfamilie einfach portiert werden.

9. Zusammenfassung und Ausblick

Die vorgestellten drei prinzipiellen Vorgehensweisen beruhen, wie gezeigt werden konnte, auf den gleichen Grundmechanismen des Kommunikationsmodells. Letzlich unterscheiden sie sich in der Anwenderoberfläche, die entweder von kommunikationsspezifischen Sprachmitteln eines erweiterten Sprachumfanges oder der allgemeinen Prozedurschnittstelle (CALL) gebildet wird. Bei der Konzipierung eines Systems ist die Wahl der richtigen Alternative daher stark beeinflußt von praktischen Überlegungen, wie Verfügbarkeit entsprechender Komponenten (Standardkommunikationssystem, Mehrrechner-PEARL, Betriebssystemfunktionen), Realisierungsaufwand, Effizienz, Portabilität und der zur Verfügung stehenden Hardware-Komponenten.
Weiterhin kann festgestellt werden, daß Architektur-Konzepte, wie z.B. durch das ISO-Referenzmodell für offene Systeme (OSI) definiert, auch im Bereich der Echtzeitsysteme ein geeignetes Strukturierungsmittel sind.
Von dem Standpunkt eines reinen Anwenders bietet es sich an, für die unteren Ebenen dieses Modells Standardkomponenten zu verwenden, um weitgehende Unabhängigkeit von Hardware- und Protokollnormen zu erreichen. Dabei kann es unter dem Aspekt der Vorteile höherer Programmiersprachen durchaus sinnvoll sein, solche Standards in PEARL zu realisieren. Daß und wie dies möglich ist, konnte am Fallbeispiel gezeigt werden.

Literatur

/1/ Fleischmann, A., et.al: Synchronisation verteilter Automatisierungsprogramme, Angewandte Informatik 7/83.

/2/Bügel, U., et.al: Mehrrechner-PEARL: Einsatzerfahrungen und Sprachvorschlag für Normung, FhG-Berichte 2-85.

/3/ Früchtenicht, H.W. et.al: Leitsystem zur Steuerung und Überwachung einzelner Verfahrensschritte in einem modernen Blasstahlwerk, Stahl u. Eisen (1982) Nr. 13.

/4/ Haubner, H., et.al.: THYNET ein Kommunikationssystem für den betrieblichen Einsatz, Informatik-Fachberichte (1985) 95.

/5/ Haubner, H., Räuber, H.: Datenaus-austausch im heterogenen Rechnerverbund I und II, Höhere Programmiersprachen in der Leittechnik 1984.

Dipl.-Ing. Hans-Jörg Haubner
Fraunhofer-Institut für Informations- und Datenverarbeitung
Sebastian-Kneipp-Str. 12/14
7500 Karlsruhe
Tel. 0721/6091-228

Anwendung von verteilem PEARL zur ausfallsicheren Datenerfassung.

E. Heilmeier, P. Holleczek, M. Trautner

Regionales Rechenzentrum Erlangen
der Universität Erlangen-Nürnberg

1. Einleitung

Die Erfassung unwiederbringlicher Daten erfordert ausfallsichere Rechnerkonfigurationen. Ausfallsicherheit kann durch redundante Hardware erreicht werden.

In einer Zeit der billigen Mikrorechner bietet es sich an, die zur Erfassung vorgesehenen Rechner mehrfach auszulegen und als Verteiltes System zu betrachten.

Verteiles PEARL [1] unterstützt solche Rechnerkonfigurationen und erlaubt eine bequeme Programmierung. Das soll am Beispiel der Erfassung von Telefongesprächsdaten an der Universität Erlangen gezeigt werden.

2. Steigerung der Zuverlässigkeit

Nach [2] soll unter Betriebszuverlässigkeit verstanden werden, daß der Rechner weder etwas Falsches tut, noch daß er die von ihm erwartete Funktion überhaupt nicht ausführt.

Daraus ergeben sich drei unterschiedliche Forderungen:

1. Der Rechner soll keine falschen Befehle an den Prozeß oder falsche Anweisungen an das Bedienpersonal geben.
2. Unwiederbringliche Daten sollen zerstörungssicher gespeichert werden.
3. Der Rechner soll nicht länger als eine vom Prozeß zulässige Zeit außer Funktion sein.

Der erste Punkt erfordert eine zuverlässige Programmierung mit möglichst vielen Korrektheitsüberprüfungen. Die wichtigsten Anordnungen sollten auf mehreren unabhängigen Wegen berechnet werden und nur dann erteilt werden, wenn sie in allen Fällen übereinstimmen.

Um die zweite Forderung zu erfüllen, ist es ratsam, alle wichtigen Daten auf zwei unabhängigen Datenträgern zu speichern. Probleme, die sich daraus ergeben, werden in der Literatur unter dem Begriff der "Datensicherheit" behandelt.

Redundante Systeme sind in der Lage, den Ausfall von Komponenten so aufzufangen, daß eine korrekte Arbeitsweise weiterhin aufrecht erhalten werden kann, gegebenenfalls mit verminderter Leistungsfähigkeit.

Eine Steigerung der Zuverlässigkeit ist durch

- Fehlerintoleranz und/oder
- Fehlertoleranz

zu erreichen.

Fehlerintoleranz, die Möglichkeit Fehler von vornherein zu vermeiden, wird hardwaremäßig durch Qualitätssteigerung der Technologien und bezüglich der Software durch Verfahren der Programmverifikation ermöglicht.

Techniken der Fehlertoleranz gehen von der "pessimistischen" Annahme aus, daß Fehler

dennoch nie ganz zu vermeiden sind und daher auch zur Laufzeit des Systems aufgefangen werden müssen.

Fehlertoleranztechniken mit Redundanz sind geeignet, auf Hardwareausfälle und Programmierfehler zu reagieren. Nach [3] lassen sich zwei Arten von Hardware-Redundanztechniken unterscheiden:

- statische Hardware-Redundanz und
- dynamische Hardware-Redundanz.

Statische Hardware-Redundanz gewährt eine Fehlermaskierung. Ein Systemausfall wird somit derart verhindert, daß eine ununterbrochene Fortsetzung der Ausführung gewährleistet ist. Statische Redundanz deshalb, weil trotz interner Ausfälle das Systemverhalten nach außen "statisch" erscheint.

Kennzeichen der statischen Hardware-Redundanz ist die sogenannte "funktionsbeteiligte Redundanz". Darunter versteht man eine Redundanz, bei der die zusätzlichen Mittel nicht nur ständig in Betrieb, sondern auch an der vorgesehenen Funktion beteiligt sind.

Merkmal der dynamischen Redundanz ist, daß das System bei einem Ausfall einer Systemkomponente aktiv ("dynamisch") reagiert und Ersatzleistung bereitstellt.

Bezüglich dynamischer Hardware-Redundanztechniken unterscheidet man zwischen
- "stand-by"-Verfahren und
- "fail-soft"-Verfahren.

Bei der "stand-by"-Strategie wird bei einem Ausfall des aktiven Moduls automatisch auf einen Reservemodul (engl. spare) umgeschaltet. Es handelt sich hierbei um eine sogenannte "nicht funktionsbeteiligte Redundanz", da die zusätzlichen technischen Mittel erst bei einer Störung die vorgesehene Funktion übernehmen. Nach [3] sind aber auch Varianten der "stand-by"-Technik möglich, bei denen die redundanten Module ständig mitlaufen und im Fehlerfall nur die Ausgangssignale umgeschaltet werden.

Neben dem "stand-by"-Verfahren gehört auch die "fail-soft"-Strategie zu den dynamischen Hardware-Redundanztechniken. Diese Methode ermöglicht abgestufte Fehlertoleranz, indem bei der Störung eines Moduls oder eines Prozessorausfalls bereits vorhandene Module gleichen Typs die Aufgabe des defekten mit übernehmen.

Damit in einem dynamischen hardware-redundanten System ein Ausfall toleriert werden kann, müssen folgende Punkte automatisch erledigt werden:

a) Diagnose:
   Der Ausfall eines Moduls muß erkannt und der fehlerhafte Modul lokalisiert werden.
b) Rekonfiguration:
   Ersatzleistung muß bereitgestellt und die durch den Fehler betroffenen Aufgaben müssen neu verteilt werden.
c) Wiederanlauf:
   Die Programme, die durch den Ausfall unterbrochen wurden, müssen mit konsistenten Daten wiederaufgesetzt werden.

[4] unterscheidet bezüglich Diagnoseverfahren zwischen

- Selbstdiagnose:
  Selbsttestprogramme erkennen Fehler innerhalb eines Bausteins.
- Nachbarschaftsdiagnose:
  Um eine automatische Fehlerbehandlung durchführen zu können, müssen in einem Multiprozessorsystem auch die übrigen Prozessoren über einen Ausfall informiert sein.
- Systemweite Selbstdiagnose (verteilte Diagnose):
  Das fehlertolerante Multiprozessorsystem muß in der Lage sein, fehlerhafte Systemkomponenten automatisch zu erkennen und zu lokalisieren.

Bei einem von [3] geführten Vergleich zwischen statischen und dynamischen Hardware-Redundanztechniken, bieten die statisch redundanten Systeme gegenüber der dynamischen Redundanz mehr Vorzüge. Dynamisch redundante Systeme haben den Vorteil, daß sie um eine vergleichbare Verfügbarkeit zu erzielen, weitaus weniger redundante Hardware benötigen.

3. Ausfallsichere Datenerfassung mit redundanten Rechnern

An einem konkreten Beispiel soll gezeigt werden, wie mit einfachen (Hardware-) Mitteln und einer komfortablen Programmiersprache die Zuverlässigkeit von Datenerfassungseinheiten gesteigert werden kann.

3.1. Allgemeines

Bei der Erfassung von Daten aus technischen Prozessen kann davon ausgegangen werden, daß Erfassungsaufgaben von den Verarbeitungsaufgaben getrennt auf verschiedene Prozessoren untergebracht sind. Will man sich vor Ausfällen schützen ("Datensicherheit"), besteht ein erster Schritt darin, lediglich den Erfassungsteil redundant auszulegen.

Gegen folgende Fehler will man sich dabei schützen:

- Ausfall der Verbindung zwischen Erfassungs- und Verarbeitungsrechner,
- Ausfall eines Erfassungsrechners.

Eine Maßnahme gegen den ersten Fall ist, den Erfassungsrechner mit eigener Speicherkapazität zu versehen, um den Ausfall zumindest überbrücken zu können.

Eine Maßnahme gegen den zweiten Fehler ist, den Erfassungsrechner mit seinem lokalen Speicher mehrfach auszulegen.

Selbst wenn die Umschalteinheit nur einfach vorhanden ist, kann jede der folgenden Komponenten einmal ausfallen, ohne den Betrieb zu beeinträchtigen:

- Erfassungsrechner,
- lokaler Speicher,
- Verbindung zum Verarbeitungsrechner.

3.2. Die Erfassung von Telefongesprächsdaten

Für die Abrechnung der Telefongebühren an der Universität wurden bisher die wichtigsten Kenndaten jedes Gesprächs auf Lochstreifen aufgezeichnet. Alle Telefonapparate der Universität, die Anschlüsse der Privatpatienten der Universitätskliniken eingeschlossen, sind an die Telefonzentrale, einer Einrichtung der Universität, angeschlossen. Hier wird für abgehende Ferngespräche über ca. 50 Amtsleitungen eine Verbindung zum Telefonnetz der Deutschen Bundespost hergestellt. Kennzeichnende Daten eines abgeschlossenen Telefongesprächs wie die Nummer des Anrufers, die Nummer des Empfängers, Datum und Uhrzeit des Gesprächsbeginns, Einheiten etc. wurden bisher über drei angeschlossene Lochstreifenstanzer auf Lochstreifen ausgegeben (siehe Abb. 1).

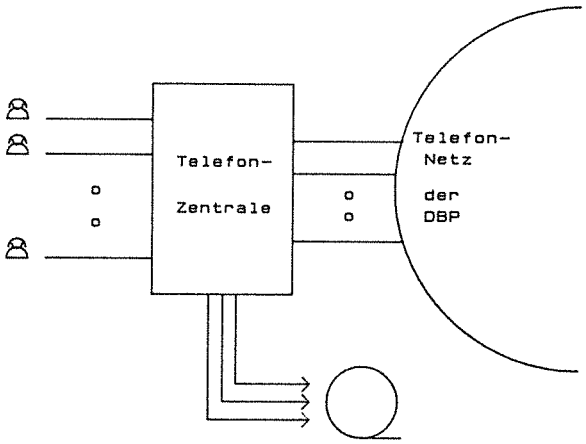


Abb. 1 Konventionelle Erfassung der Telefongespraechsdaten

Da die Verwendung eines Lochstreifens als Aufzeichnungsmittel überholt ist, bot sich als Datenträger die Diskette an. Zum anderen sollte der Datentransport automatisch erfolgen.

Die Kenndaten abgeschlossener Telefongespräche sollen somit anstatt auf Lochstreifen auf Floppy gespeichert und täglich zum Verwaltungsrechner übertragen werden. Für die Gebührenerfassung, Speicherung und Übertragung der Daten zum Verwaltungsrechner sind Mikrorechner vorgesehen.

Die Kenndaten abgeschlossener Telefongespräche kommen von drei unabhängigen Leitungen an. Bevor sie zur Speicherung auf Floppy weitergegeben werden können, müssen die parallel ankommenden Datensätze in sequentielle Form gebracht werden. Die gespeicherten Kenndaten werden automatisch zu vorgegebenen Zeitpunkten oder auf Wunsch des Bedienperso-

nals an den Verwaltungsrechner gesendet. In monatlichen oder vierteljährlichen Abständen wird am Verwaltungsrechner die Abrechnung der Gesprächsdaten durchgeführt.

Hauptziel ist, die Ausfallsicherheit des Gesamtsystems zu erhöhen, um eine kontinuierliche Erfassung und zuverlässige Speicherung der Daten zu gewährleisten.

3.2.1. Lösungsansatz

Für die "on-line"-Erfassung, Speicherung und Übertragung der Kenndaten abgeschlossener Telefongespräche ist ein Mehrrechnersystem, bestehend aus vier Mikrorechnern vom Typ Z80, eingesetzt (siehe A b b . 2). Insgesamt sind für die Aufnahme drei unabhängige Datenleitungen vorgesehen, um bei Hochbetrieb die ankommenden Daten quasi gleichzeitig erfassen und speichern zu können.

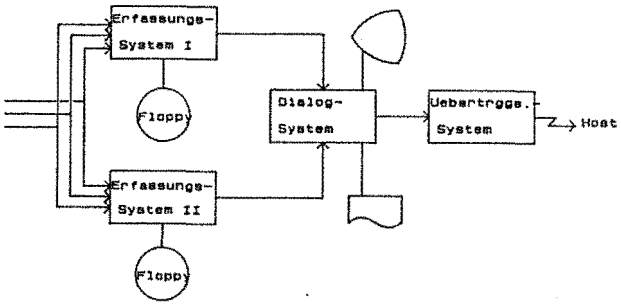


Abb. 2 Ausfallsichere Konfiguration zur Erfassung der Telephongespraechsdaten

Ein Z80-System, das speziell für die Erfassung vorgesehen ist, nimmt die ankommenden Daten entgegen und speichert diese in einer Datei auf Diskette. Gleichzeitig eintreffende Datensätze müssen vor der Weiterbearbeitung erst sequentialisiert werden.

Damit bei einem eventuellen Ausfall des Erfassungssystems (z.B. Stromausfall, Programmfehler, Spurfehler auf der Diskette) die Datenerfassung nicht zum Erliegen kommt, ist ein zweites Erfassungssystem vorhanden, das synchron zum anderen die ankommenden Daten erfaßt und auf einem eigenen Datenträger abspeichert. Mit dem Einsatz des redundanten Systems wird eine erhebliche Verbesserung der Betriebssicherheit erzielt. Eine wichtige Voraussetzung für den Anschluß einer redundanten Hardware-Einrichtung ist, daß beide Systeme von verschiedenen Stromquellen versorgt werden. Die Bedienung der Anlage ist von einer speziellen Komponente des Gesamtsystems, dem Dialogsystem, aus möglich.

Außer der Kommunikation zwischen Bedienpersonal und Gesamtsystem hat das Dialogsystem ferner die Aufgabe der Überwachung. Ein Ausfall jedes der drei restlichen Komponenten wird von diesem System erkannt und gemeldet.

Für die Übertragung der auf Floppy gespeicherten Telefongesprächsdaten zum Verwaltungsrechner der Universität ist eine eigene Systemkomponente vorgesehen.

Da die Kapazität einer Diskette nicht für die Datenaufnahme eines Monats oder länger ausgerichtet ist, wird eine Übertragung täglich automatisch gestartet.

3.2.2. Maßnahmen zur Steigerung der Zuverlässigkeit

Durch den modularen Aufbau der Funktionen jedes Teilsystems der Anlage ist ein wichtiger Grundstein für die Fehlertoleranz des Gesamtsystems gelegt.

Ein Ausfall der Übertragungskomponente, die im Normalfall nur einmal am Tag aktiviert wird, ist wegen des funktionalen modularen Aufbaus nicht weiter tragisch. Das defekte System kann in der Zwischenzeit repariert und ohne großen Datenverlust bei der Erfassung wieder dazugeschaltet werden.
Wie aus den theoretischen Ausführungen zur Fehlertoleranz im vorigen Kapitel hervorgeht, handelt es sich bei der verwendeten Methode um eine Mischform zwischen statischer und dynamischer Hardware-Redundanz mit "stand-by"-Verfahren, wobei der Trend in letztere Richtung geht.

Der einzige Unterschied zur beschriebenen "stand-by"-Strategie als dynamische Hardware-Redundanz besteht darin, daß im vorliegenden Modell nicht erst bei einem Ausfall das redundante System zugeschaltet wird, sondern, wie als Variante der "stand-by"-

Technik in [3] vorgeschlagen, bei der Datenerfassung ständig mitläuft.

Ein Ausfall der Erfassungssysteme und/oder des Übertragungssystems wird vom Dialogsystem entsprechend der Nachbarschaftsdiagnose [4] erkannt und gemeldet. Eine Störung des Dialogrechners und ein darauffolgender Ausfall eines Erfassungssystems hat zur Folge, daß der zweite Ausfall vom bereits defekten Dialogsystem nicht gemeldet werden kann. In diesem Fall wird die Störung des Dialogsystems und die des Erfassungssystems auf zwei unterschiedliche Leuchtdioden am intakten Erfassungsrechner angezeigt.

Der modulare Aufbau garantiert selbst eine kontinuierliche Datenaufnahme bei einem Ausfall aller Komponenten, bis auf einen Erfassungsrechner, der als einziges funktionsfähiges System die ankommenden Daten annimmt und abspeichert.

3.3. Das Programm

Das Programm wurde in PASS [5], [6] spezifiziert und in Verteiltem PEARL programmiert. Es umfaßt 18 Tasks mit ca. 4500 Quellzeilen und läuft auf vier gekoppelten Z80-Rechnern. Der redundante Teil des Codes beträgt ca. 1100 Zeilen.

Der Erstellungsaufwand mit Entwurf, Spezifikation und Programmierung umfaßte 3/4 Mann-Jahr.

4. Möglichkeiten von Verteiltem PEARL

Die Programmierung von verteilten Systemen stellt hohe Anforderungen an die Programmiersprache. Die Echtzeitprogrammiersprache PEARL gewährleistet als "verteiltes PEARL" [1] eine Kommunikation zwischen Prozessen, die auf demselben als auch auf verschiedenen Prozessoren laufen. Das Konzept basiert auf Botschaftsoperationen und nicht-deterministischen Kontrollanweisungen. Diese Konstrukte sind Erweiterungen von PEARL und sind im PEARL-Compiler und -Betriebssystem für Z80-Prozessoren integriert.

Mit Hilfe des Botschaftsmechanismus können Prozesse eines Prozeßsystems Botschaften austauschen und auf diese Weise miteinander kommunizieren. Im vorliegenden Fall bedeutet das einen Nachrichtenaustausch zwischen den Prozessen an den Erfassungssystemen, dem Dialog- und dem Übertragungssystem. Die Störung eines Prozessors kann insofern sofort festgestellt werden, als eine gesendete Botschaft von diesem nicht angenommen wird.

Im Zusammenhang mit dem Botschaftsmechanismus stehen die von Dijkstra [7] vorgeschlagenen nicht-deterministischen Kontrollanweisungen zur Verfügung. Mit diesen Konstrukten ist es möglich, auf das alternative oder gleichzeitige Eintreffen verschiedener Nachrichten von verschiedenen Prozessen zu warten.

Zur Überwachung der einzelnen Prozessoren ist auf jedem Rechner ein eigener Prozeß verantwortlich, der in vorgegebenen Zeitabständen eine Botschaft an den Überwachungsprozeß eines anderen Prozessors sendet bzw. erwartet. Dadurch, daß ein Prozeß auf eine Botschaft vom anderen Überwachungsprozeß und letzterer auf die Abnahme der gesendeten Nachricht wartet, ist eine gegenseitige Überwachung der Prozesse und somit der Prozessoren gewährleistet.

Durch die Einführung eigener Überwachungsprozesse, die zu fest vorgegebenen Zeitpunkten unabhängig von der Systembelastung die Aktivität aller Prozessoren überprüfen, wird eine Störung selbst dann erkannt, wenn alle Prozessoren "leerlaufen".

Bei der Erfassung der Telefongesprächsdaten werden diese an jedem der Erfassungssysteme in einer vorgesehenen Datei auf der Diskette gespeichert. Da ein Systemausfall oder das Auftreten eines Spurfehlers beim Beschreiben der Floppy nicht vorherzusehen ist, bedeutet dies einen Verlust aller erfaßten und gespeicherten Daten auf der eröffneten Datei. Um diesen Verlust an Daten möglichst gering zu halten, bietet sich als Lösung die Einführung vieler kleiner Dateien an. Jede einzelne Datei wird, sobald sie vollständig beschrieben ist, geschlossen und die Daten bleiben bei einem Fehler sicher erhalten.

Allein die Daten der zum Zeitpunkt des Fehlers eröffneten Datei sind verloren.

5. Erfahrungen

Die Programmierung verteilter Systeme in Echtzeitumgebung stellt - auch beim Einsatz einer höheren Programmiersprache - durchaus noch keine alltägliche Aufgabe dar. Insbesondere bei Problemen, die den Einsatz redundanter Hardware erfordern, sehen sich Bearbeiter vor schwierigen Entwurfsentscheidungen und Implementationshindernissen. Eine Programmiersprache mit einschlägigen Grundfunktionen, wie Botschaftsoperationen mit Zeitüberwachung (z.B. für Nachbarschaftskontrolle), Guarded Statements mit Verundung (z.B. für synchronen Gleichlauf von redundanten Prozessoren) stellt eine begriffliche Grundlage dar. Entscheidende konzeptionelle Hilfe kommt allerdings von einer auf die Möglichkeiten der Programmiersprache abgestimmten Spezifikationstechnik, wie z.B. PASS.

Literatur

[1] Fleischmann, A.; Holleczek, P.; Klebes, G.; Kummer, R.: Synchronisation und Kommunikation verteilter Automatisierungsprogramme. Angewandte Informatik 7/83, 290-297

[2] Bolch, G.: Prozeßautomatisierung mit Prozeßrechnern. Vorlesung. Erlangen 1980

[3] Maehle, E.: Fehlertolerantes Verhalten in Multiprozessoren. Untersuchungen zur Diagnose und Rekonfiguration. Dissertation. IMMD Erlangen 1982

[4] Maehle, E.: Experimente mit parallelen Programmen auf DIRMU Multiprozessor-Konfigurationen. Vortrag im Informatik-Kolloquium der Universität Erlangen-Nürnberg. 1985

[5] Fleischmann, A.; Holleczek, P.; Koch, I.; Kragl, G.: Eine Spezifikationstechnik für Verteilte Systeme. PEARL-Tagung 1984

[6] Andres, C.; Fleischmann, A.; Holleczek, P.; Hillmer, U. Kummer, R.: Eine Methode zur Beschreibung von Kommunikationsprotokollen. GI/NTG-Fachtagung Kommunikation in Verteilten Systemen 1985. Informatik-Fachberichte 95, Springer-Verlag

[7] Dijkstra, E.W.: Guarded commands, non determinacy and derivation of programs. ACM Computing Surveys. August 1975

Dr. Peter Holleczek
Regionales Rechenzentrum
Universität Erlangen-Nürnberg
Martensstraße 1
8520 Erlangen
Tel.: 09131/85-7031

Datenbanken in Realzeitumgebung am Beispiel eines Betriebsleitrechners.

Dipl.-Math. Benno Schneiders

1. ABSTRACT

Am Beispiel eines auf einem Doppelrechnersystem realisierten Betriebsleitrechner wird aufgezeigt, daß ein Datenbanksystem in einer solchen Systemumgebung sowohl sehr zeitkritische, realzeitorientierte Aufgaben zu erfüllen hat, als auch den Aufgabenbereich eines normalen Informationssystemes abdecken muß. Dazu wird zunächst die allgemeine Aufgabenstellung erläutert, ein Hard- und Softwaremodell vorgestellt und dann die speziellen Anforderungen an das Datenbanksystem näher erläutert.

Am Beispiel des Datenbanksystems PISA/DB wird aufgezeigt, wie durch spezielle Datenstrukturen und Funktionen die Forderungen nach Geschwindigkeit, Ausfallsicherheit und Verfügbarkeit erfüllt werden können. Dazu gehören z.B. die Möglichkeit des Doppelschreibens, eine komfortable Pufferverwaltung, Dateien mit direkten Zugriff und zyklische Dateien.

Zum Schluß wird dargestellt, wie auf einem Doppelrechnersystem durch einfache Maßnahmen der Wiederanlauf auf dem Standby-Rechner zu gewährleisten ist.

2. AUFGABENSTELLUNG

Aus der immer weiter fortschreitenden Automatisierung in den Produktionsbetrieben, ergibt sich die Anforderung, zwischen der schon vorhandenen Groß-EDV und der Prozeßsteuerung eine dritte Rechner-Ebene einzuschieben, den sogenannten Betriebsleitrechner (BLR). Generell kann man die Aufgabe des BLR darin sehen, beliebige Prozessvariable abzufragen, die Informationen aufzubereiten, abzuspeichern und an das Management weiterzureichen.

Im Rahmen dieses Referates, wird davon ausgegangen, daß der BLR in einem Produktionsbetrieb steht und folgende fünf Aufgabenkomplexe zu bewältigen hat.

1) Die Produktionsüberwachung
   Alle relevanten Daten der laufenden Produktionen (Zählerstände, Meßwerte, Zustände, Störmeldungen) werden automatisch gesammelt und abgespeichert. Störungen und Produktionsabweichungen werden über Bildschirme an das Management gemeldet. Sämtliche angefallene Daten zu einer Produktion werden über einen bestimmten Zeitraum aufbewahrt.

2) Statistik
   Die aus der Produktionsüberwachung anfallenden Daten werden nach Abschluß eines Produktes automatisch verdichtet und für spätere statistische Auswertungen abgespeichert. Diese Daten werden über einen längeren Zeitraum aufbewahrt. Extrakte aus den Daten werden an die Groß-EDV übertragen.

3) Disposition
   Die Disposition für die anstehenden Produktionen werden aufgrund von vorhandenen Stammdaten und standardisierten Dispositionen den aktuellen Bedürfnissen angepaßt. Die Maschinen- bzw. Produktionsleiter rufen aus den erstellten Dispositionen ihre anstehenden Aufgaben ab.

4) Stammdatenverwaltung
Hier werden die Informationen über den Maschinenpark und die Produktionspfade verwaltet und gespeichert.

5) Auswertungen
Das Betriebsmanagement hat die Möglichkeit, sich anhand der Produktionsdaten über den Zustand der laufenden Produktionen zu informieren, bzw. anhand der statistischen Daten-Aussagen über Betriebsabläufe bzw. über Schwachstellen im Betrieb zu verschaffen.

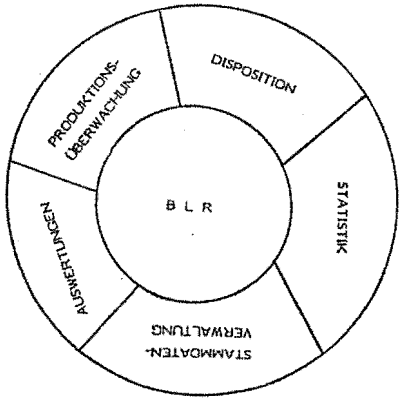


Abb. 1: Aufgaben des Betriebsleitrechners

Man sieht, daß die Aufgabenstellungen an den Betriebsleitrechner völlig unterschiedliche Anforderungen stellen. Es müssen Prozessdaten automatisch erfaßt werden. Dazu wird eine schnelle Reaktion und eine hohe Ausfallsicherheit von dem System gefordert. Andererseits dient das System als Informationssystem, d.h., die erfaßten Daten müssen so flexibel strukturiert sein, daß alle denkbaren Auswertungen in akzeptablen Zeiten durchführbar sind.

Diese komplexen Anforderungen sind natürlich nur mit einer leistungsfähigen Hard- und Software zu erfüllen.

3. HARDWARE-KONZEPT

Die Auslegung der Hardware hängt von den Anforderungen an die Ausfallsicherheit des Gesamtsystemes ab. Je sicherer das System sein soll, um so größer werden die Hardwarekosten. Allerdings sollten bei der Kostenkalkulation berücksichtigt werden, welche Unkosten durch die Einschränkung oder sogar Einstellung der Produktion entstehen können, wenn der BLR einmal ausfällt.

Die optimale Auslegung der Hardware eines BLR könnte in etwa wie folgt aussehen:

- mindestens zwei Frontendrechner als Datenkonzentrator zwischen den Prozessen und dem BLR. Diese sollten über eine beschränkte Plattenkapazität verfügen, damit bei einem eventuellen Ausfall des BLR, bzw. bei einem Stau vor dem BLR, die anfallenden Daten zwischengespeichert werden können.
- Der BLR wird als Doppelrechnersystem ausgelegt. Dabei dient ein Rechner nur als Standby und wird im Normalfall höchstens für Entwicklungs- und Testarbeiten genutzt.
- Die gesamte Peripherie ist umschaltbar von einem Rechner zum anderen.
- Die Plattenkapazität muß so ausgelegt sein, daß alle Daten doppelt geführt werden können. Auf diese Weise ist man auch gegen Plattenfehler geschützt.

Abbildung 2 stellt einen solchen optimalen BLR dar. Abhängig von der jeweiligen Anwendung können natürlich einige Komponenten einfach ausgelegt werden bzw. ganz entfallen (z.B. die Frontendrechner).

4. SOFTWAREKOMPONENTEN

In Abbildung 3 ist der generelle Aufbau des Softwaresystems auf dem betrachteten BLR dargestellt. Im Prinzip besteht das Gesamtsystem aus drei Ebenen:

- dem Betriebssystem,
- der Standardsoftware,
  dazu gehören:
  + Dialogsprache
  + Reportgenerator

+ Datenbanksystem
+ DFÜ-Software

- der Anwendersoftware
  hier also:
  + Stammdatenverwaltung
  + Disposition
  + Statistik
  + Produktionsüberwachung
  + spezielle Auswertungen z.B. Darstellung von Prozesszuständen an den Bildschirmen.

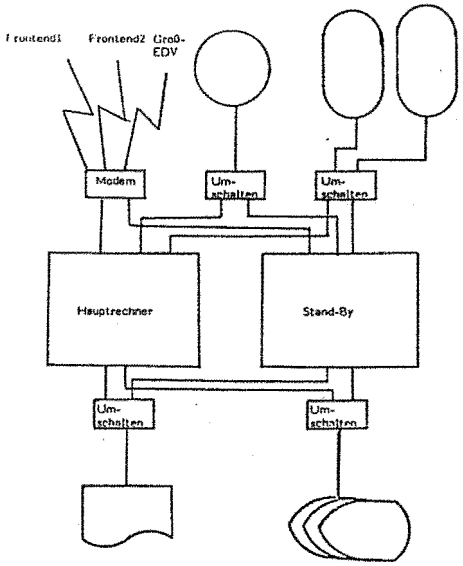


Abb. 2: Hardware eines BLR

Im Folgenden sollen die Aufgaben des Datenbanksystemes in dieser Umgebung näher betrachtet werden. Dabei geht es weniger um solche 'natürlichen' Aufgaben eines Datenbankssystemes wie z.B. die Stammdatenverwaltung bzw. die Auswertung von Daten in Listen und Formularen, sondern mehr um die besonderen Anforderungen an das DB-System, die sich aus dem Aufgabengebiet der Produktionsüberwachung ergeben.

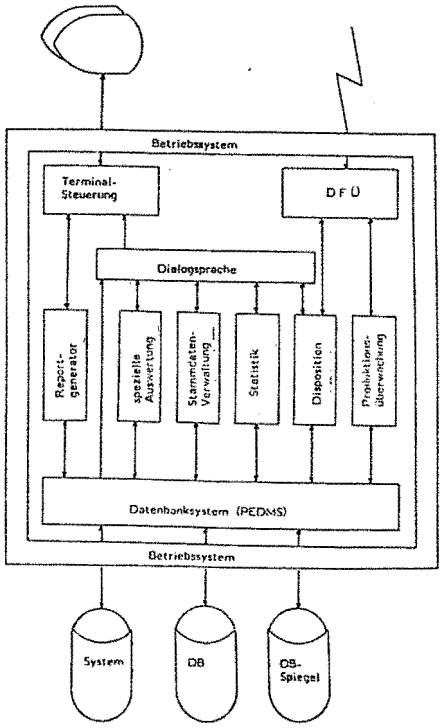


Abb. 3: Blockschaltbild der Software des BLR

5. AUFGABEN DES DATENBANKSYSTEMES

Zunächst soll kurz erläutert werden, was hier unter Produktionsüberwachung verstanden werden soll.

Der Frontendrechner fragt periodisch alle Zähler und Meßwerte der z.Z. laufenden Produktionen ab, sammelt sie zu Telegrammen und schickt diese an den BLR. Darüberhinaus empfängt er ereignisgesteuert Zustandsänderungen und Störmeldungen und gibt diese in gesonderten Telegrammen direkt an den BLR weiter.

Auf dem BLR werden die unterschiedlichen Telegramme an die Verarbeitungs-

routinen verteilt. Zunächst muß geprüft werden, ob die Meldungen nicht schon einmal empfangen wurden und eventuell schon verarbeitet sind. Dies kann z.B. bei einem Verarbeitungsstau auf dem BLR passieren -die Quittung kam nicht in einem definierten Zeitraum-, bzw. während der Synchronisation zwischen BLR und Frontend nach einem Systemausfall.

Jedes Telegramm, daß im Normalfall n-Meldungen enthält, wird in einer Datenbanktransaktion abgearbeitet. Dadurch reduziert sich der Synchronisationsaufwand zwischen BLR und Frontend auf Telegrammebene.

Wurde das Telegramm als korrekt erkannt, wird jede einzelne Meldung in der Datenbank abgespeichert und das Telegramm dem Frontendrechner quittiert.

5.1 Datenstrukturen zur Produktionsüberwachung

Um die oben erläuterten Aufgaben wahrnehmen zu können, werden im Rahmen der Produktionsüberwachung zwei unterschiedliche Datenstrukturen zur Beschreibung und Speicherung der Prozessdaten definiert:

- Statusdatei
  Die Datei spiegelt zu jedem Zeitpunkt ein aktuelles Abbild der laufenden Prozesse wieder, d.h. zu jeder Prozessvariablen wird der aktuelle Zustand und der Zeitpunkt der letzten Aktualisierung nachgehalten.
  Diese Dateien sind normalerweise nach Maschinen strukturiert und enthalten pro Prozessvariable folgende Informationen:
  + letzte Aktualisierungszeit
  + Zustand aktuell
  + Zustand normal
  + Aktionscode
  + Sonstiges (z.B. Toleranzbereiche).

  Dabei gibt der Aktionscode Auskunft darüber, was bei Änderungen von Zuständen für Sonderaktionen zu erfolgen haben.

  Die Datei wird für folgende Aufgaben benutzt:
  + zur Überprüfung, ob eingehende Meldungen schon verarbeitet wurden,
  + zur Darstellung des aktuellen Prozessabbildes auf dem Bildschirm.
- Produktionsdatei
  In dieser Datei werden alle Zustandsänderungen, Zähler- und Meßwerte in ihrem zeitlichen Ablauf festgehalten. Das heißt pro Signal/Prozessvariable werden bei jeder sie betreffenden Meldung folgende Informationen abgespeichert:
  + Signalnummer
  + Zeit
  + Zustand.

  Diese Datei ist produktionsorientiert, d.h. zeitliche Auswertungen über bestimmte Produktionen sind möglich.

5.2 Datenstrukturen und Datenbankfunktionen in PISA/DB

In den Statusdateien spiegelt sich der Aufbau des Produktionsbetriebes wieder, d.h. ihre Struktur wird sich kaum ändern. Deshalb empfiehlt es sich hier mit festen Rekordlängen zu arbeiten. Man definiert unterschiedliche Rekordtypen, z.B. die Zähler pro Maschine ergeben einen Rekordtyp.
Jeder Rekordtyp hat den gleichen formalen Aufbau:

- Maschinennummer,
- Information zu Signal Nr. 1,
- Information zu Signal Nr. n.

Der Zugriff auf diese Dateien geschieht über einen Cala-Algorithmus. Dieser Algorithmus stellt sicher, daß der Zugriff auf einen Rekord maximal einen E/A erfordert.

Da in PISA/DB durch einen komfortablen Pufferungsmechanismus die Möglichkeit besteht, speicherresidente Dateien einzurichten, bedeutet dies, ein lesender Zugriff wird immer ohne E/A erfolgen, während ein ändernder Zugriff genau einen E/A benötigt.

Im Gegensatz zu den Statusdateien hat die Produktionsdatei einen sehr variablen Aufbau. Es ist nicht vorherbestimmbar, wieviel Daten zu einer Produktion anfallen werden, andererseits sollen die zeitlichen Abläufe einer Produktion auswertbar sein, d.h. jeder Satz in der Produktionsdatei muß durch eine Produktionsnummer eindeutig einer Produktion zuzuordnen sein. Auch hier wird so verfahren, daß unterschiedliche Dateien für Zähler, Meßwerte und Zustände eingerichtet werden, aber nicht jede Änderung einer Prozessvariablen ergibt einen neuen Satz in der Produktionsdatei. Vielmehr werden hier die Möglich-

keiten der multiplen Felder in PISA/DB ausgenutzt.

Der generelle Satzaufbau sieht wie folgt aus:

- Produktionsnummer,
- Information zu Signal 1
  .
  .
  .
- Information zu Signal n.

Dabei besteht die Information zu jedem Signal aus einer Gruppe von multiplen Feldern. Beim Start einer Produktion, wird ein Satz dieses Typs mit den Ausgangswerten pro Signal abgelegt. Bei jeder Meldung zu einem Signal, wird zu jedem Feld der entsprechenden Gruppe ein Wert hinzugefügt. Erst wenn der Satz eine vorher definierte Größe überschritten hat, wird zur gleichen Produktion ein neuer Satz angelegt.

Für die Produktionsdateien wird eine weitere spezielle Funktion von PISA/DB angewendet, die sogenannten zyklischen Dateien. In den Schemadefinitionen für diese Art von Dateien kann eine Zykluszeit vorgegeben werden, d.h. alle Daten, die in dieser Datei abgelegt werden, werden während dieser Zykluszeit aufbewahrt und anschließend automatisch vom Datenbanksystem gelöscht. Der freiwerdende Platz wird sofort wieder für neue Produktionsdaten frei.

Auch für die Produktionsdaten wird die komfortable Pufferverwaltung von PISA/DB ausgenutzt. Zwar können die Dateien wegen ihrer Größe nicht vollständig speicherresident gehalten werden, aber man stellt den Produktionsdateien exclusiv einen genügend großen Pufferpool zur Verfügung. Dadurch ist einerseits sichergestellt, daß durch andere Arbeiten mit dem Datenbanksystem keine Produktionsdaten aus dem Pufferpool gealtert werden, andererseits werden die aktuell bearbeiteten Sätze ständig speicherresident sein.

Ein weiteres wesentliches Kriterium, welches PISA/DB dazu prädestiniert in solchen Anwendungsumgebungen eingesetzt zu werden, ist die völlige Reorganisationsfreiheit der Daten, die automatische Wiederverwendung von freiwerdendem Speicherplatz und die Möglichkeit zusätzlicher Datenstrukturen parallel zur laufenden Anwendung zu definieren. Dadurch gewährleistet das Datenbanksystem einen unterbrechungsfreien 24-Stunden-Betrieb.

5.3 Datensicherheit in PISA/DB

PISA/DB bietet mehrere unterschiedliche Sicherungsverfahren an. Diese können sowohl einzeln als auch in Kombination implementiert werden. Als eine sinnvolle Kombination hat sich erwiesen:

- das Transaktion-Undo
  Dieses beinhaltet sowohl das Backout Transaktion (BOT) im laufenden Betrieb, als auch das Zurücksetzen offener Transaktionen nach einem Systemausfall,
- das Doppelschreiben
  Alle Änderungen der Daten werden vom Datenbanksystem automatisch auf zwei Platten mitgeführt. Beim Ausfall einer Platte arbeitet das System mit der verbleibenden Platte weiter. Der Anwender hat die Möglichkeit einen Drive aus dem System auszukoppeln -Erzeugen einer Kopie-, und eine frische Platte in den Drive einzulegen. Das System zieht die neue Platte automatisch auf den aktuellen Stand hoch.

5.4 Wiederanlauf

Unter Wiederanlauf soll hier der Ausfall des Hauptrechners und die Übernahme der Arbeit durch den Stand By verstanden werden. Das Datenbanksystem schreibt alle seine Restartinformationen auf die Platte. Beim Umschalten auf den Stand-By-Rechner kann das dortige Datenbanksystem also einen normalen Restart durchführen (d.h. alle offenen Transaktionen werden zurückgesetzt).

Die Anwendung, die ihre Restartinformationen auch auf der Platte vorfindet, erhält vom Datenbanksystem die Information, welche Transaktionen zurückgesetzt werden und kann sich schnell mit den Frontendrechner synchronisieren (zur Erinnerung: ein Telegramm entspricht einer Transaktion).

Schneiders, Benno
Rhonestr. 2, 5000 Köln 71
0221 70 91 233

Konzept eines verteilten Multiprozessorsystems

Clemens Kordecki Universität Karlsruhe

Kurzfassung: In diesem Bericht wird ein hardwarerealisiertes Konzept zur Kommunikation und Synchronisation von Prozessen in einem verteilten System vorgestellt. In [10] wird dargestellt wie der Aufwand zur Kommunikation und Synchronisation verringert wird, wenn die Kommunikation durch implizite Prozesse kontrollierbar wird. Ein solcher Kommunikationskanal ist asynchron. Synchrone Kommunikation ist in diesem Sinne ein Spezialfall der asynchronen Kommunikation. Die impliziten Kommunikationsprozesse, die durch Hardware realisiert werden, bilden die Basis eines verteilten Multiprozessorsystems.

Schlüsselwörter: Multiprozessorsysteme, Prozeßkommunikation, Synchronisation

1. EINFÜHRUNG

Die Trennung von funktionalen Einheiten (Modulen) hat sich in der Programmierung seit langem als vorteilhaft erwiesen. Durch das Definieren von Schnittstellen werden diese Einheiten separat test- und verifizierbar.

Als Konsequenz dieser guten Erfahrungen soll diese Trennung auch auf Betriebssystemteile und verteilte Applikationen angewandt werden. Der Bereich, der hier angesprochen wird, ist die Kommunikation und Synchronisation von verteilten Prozessen.
Ziel ist es, die Kommunikation vor dem Anwendungsprogrammierer zu verstecken und die Synchronisation implizit an die Kommunikationsobjekte zu knüpfen. In der Softwaresicht werden die Kommunikationsprozesse als Objekte modelliert, deren Benutzung eine spezielle Disziplin voraussetzt [11]. Die für die Kommunikation benutzten Datenobjekte werden Kanäle genannt. Sie können als separate Prozesse aufgefaßt werden, die für eine konkrete Kommunikation durch ihre Installationsparameter auf die jeweiligen speziellen Anforderungen abgestimmt werden. Auf Grund ihres einfachen Aufbaus realisieren wir diese Prozesse direkt durch Hardware, einem Kommunikationsspeicher innerhalb jedes Multiprozessorknotens, mit festen Zugriffsoperationen. Die Realisierung durch Hardware reduziert den Aufwand des Kommunikationsprotokolls auf einen Speicherzugriff.

Die möglichen Kommunikationsdisziplinen lassen sich durch die folgenden Pfadausdrücke beschreiben:

$$(w\ r\ )^*$$
$$(w^*r\ )^*$$
$$(w\ r^*)^*$$
$$(w^*r^*)^*$$

Prozesse dieses verteilten Multiprozessors sind Aggregationen von lokalen Daten, den Zugriffsoperationen auf diese Daten und sequentielle Aktionen. Diese Prozesse beschreiben modular und anonym zueinander die eigentliche Applikation. Ein wesentlicher Gesichtspunkt an dieser Stelle sind die lokal deklarierten Eingabe- und Ausgabe- Ports. Diese Ports stellen die Kommunikationskanäle zu anderen Prozessen dar. Eine virtuelle Verbindung dieser Ports wird während der Laufzeit eingerichtet und nach Termination der beteiligten Prozesse wieder abgebaut.

Im Gegensatz zu diesen Anwender- Prozessen sind die Kommunikationskanäle als implizite Prozesse im System integriert. Sie sind ebenso wie die Applikationsprozesse strukturiert. Lokale Daten mit den Zugriffen put und get und eigene Aktivitäten wie Init, Veto und Interrupt characterisieren diese Prozesse. Im nachfolgenden Kapitel wird auf diese Prozesse näher eingegangen.

2. KOMMUNIKATIONSKANÄLE

Grundidee dieses Verfahrens ist die Verlagerung des Kommunikationsprotokolls in einen separaten Prozeß, in dem Maßnahmen zur Synchronisation der beteiligten Prozesse parallel zu den Aktivitäten dieser Prozesse ausgeführt werden. Ergebnis dieser Separation ist eine hochgradig parallele Ausführung der beteiligten Prozesse, weil Synchronisationsmaßnahmen immer zum spätest möglichen Zeitpunkt einsetzen.
Aus der Sicht unserer geplanten Anwendung, der Steuerung und Regelung industrieller Prozesse, besteht ein verteiltes Multiprozessor-System aus einer Vielzahl einzelner Verarbeitungsknoten, die nach den Anforderungen des zu steuernden Prozesses physikalisch verteilt und miteinander vernetzt sind. Da es nicht vorhersehbar ist, welche zusätzlichen Anforderungen während des Betriebs entstehen, die neue Kommunikationswege, aber auch neue Prozesse nach sich ziehen, soll das System maximale Flexibilität bieten. Für unseren Entwurf bedeutet dies beliebige Kommmunikationswege zwischen allen Knoten und die Möglichkeit, Änderungen der Prozeßverteilung am laufenden System durchzuführen.
Jeder Verarbeitungsknoten im System ist ein Multiprozessor, der über ein globales Bussystem mit den anderen Knoten verbunden ist (Abbildung 1).

Ein Multiprozessor besteht aus folgenden Baugruppen:

- Einer Schnittstelle zum globalen Bussystem, das Fehlertoleranz und Übertragungssicherheit für das Netzwerk sichert. Diese Schnittstelle wird durch einen separaten Kommunikationsprozessor realisiert.

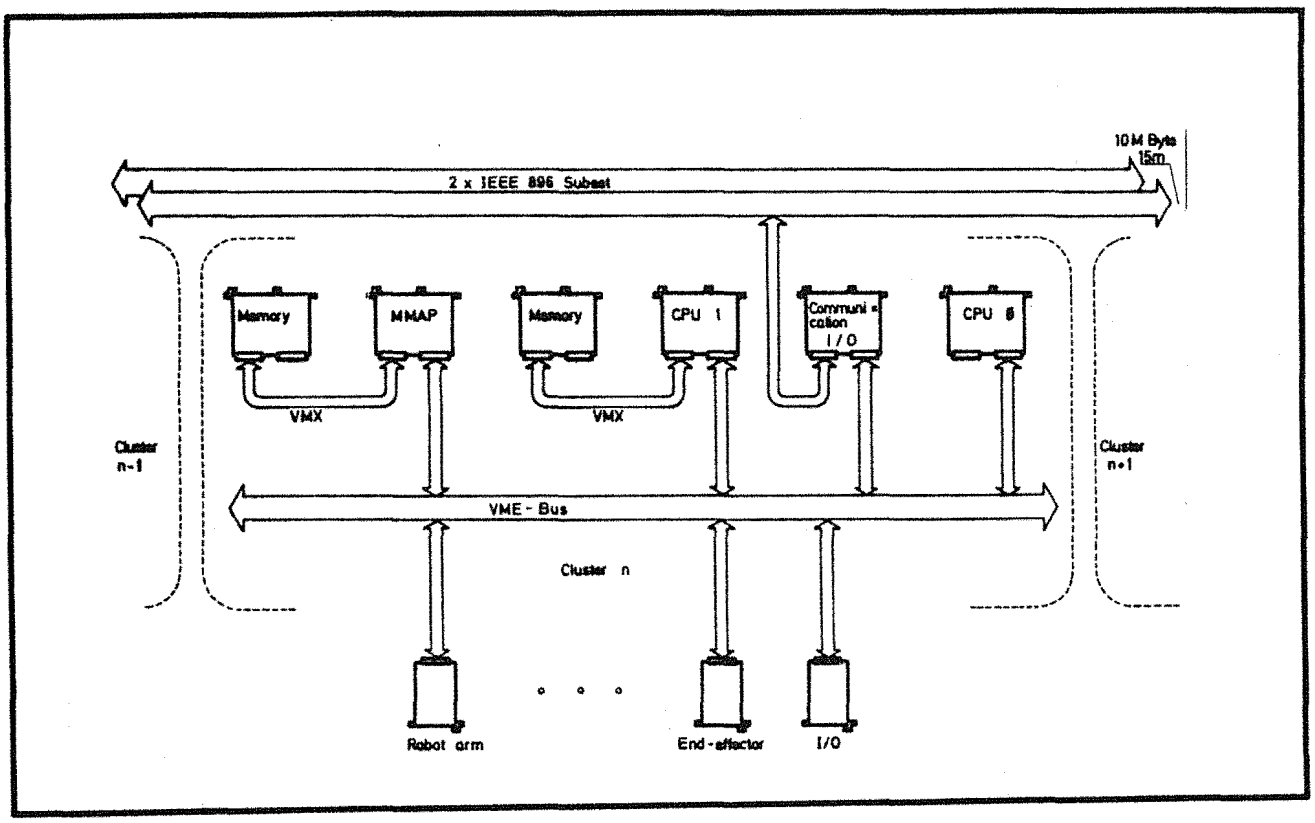


ABBILDUNG 1

- Einem Kommunikationsspeicher (MMAP) mit einer neuartigen Speicherverwaltung (MMAP Memory Map Access and Protection Unit)
- Einem Prozessor, der die Kommunikationskanäle installiert und Aktivitäten des globalen Betriebssystems durchführt.
- Mindestens einem weiteren Prozessor mit lokalem Speicher (Akteur), der die geforderte Rechenleistung erbringt. Auf allen Akteuren läuft ein lokales Betriebssystem, das um einige Systemprimitiven erweitert wurde.
- Peripherie-Anschlüssen, entsprechend den jeweiligen Knotenfunktionen, Parallele Ein- Ausgabe, A/D-Wandler, Hintergrundspeicher usw.

Die Forderungen an Kommunikationskanäle lassen sich in der folgenden Weise beschreiben:

- Es soll Speicher bereitgestellt werden, um die Kommunikation schnell, d.h. ohne Wartezyklen wie sie bei synchroner Kommunikation auftreten, in asynchronerweise abwickeln zu können.
- Es sollen unzulässige Zugriffe auf den Speicher erkannt und gemeldet werden.
- Der Kommunikationskanal soll eine Pufferfunktion ausüben, um die einzelnen Prozesse zu entkoppeln.
- Der Kommunikationskanal muß unter Beibehaltung der Applikationssicht auf die dezentralen Systeme verteilt werden können.
- Es muß die gesamte Synchronisation der beiden beteiligten Applikationsprozesse durch diesen Kommunikationsprozess überwacht werden. Ggfs. muß ein Applikationsprozess verzögert werden können bis ein Zustand eintritt, der den Zugriff ermöglicht. Die Überwachung der Synchronisation erfordert aber eine Definition der Zugriffsdisziplin, z.B. alternatives Lesen und Schreiben, für einen Kanal.

Für einen solchen Kanal lassen sich eine Reihe von Aussagen treffen:

- Dieser Kanalprozess ist der häufigste aller Prozesse.
- Es ist immer der gleiche Prozeß, nur mit unterschiedlichem Typ des Kommunikationsobjektes.
- Der Prozeß besitzt einen Zustand durch den letzten Zugriff auf das Kommunikationsobjekt.
- Er kann die Zugriffssequenz auf ein Kommunikationsobjekt überwachen und muß bei Verletzungen der Zugriffsdisziplin handeln.
- Er ist unabhängig von allen Applikationsprozessen.

3. KOMMUNIKATIONSSPEICHER UND VETO

3.1 Zugriffsdisziplinen

Der Kommunikationsspeicher ist ein gemeinsamer Speicher innerhalb eines Multiprozessorknotens. Der Zugriff auf einen Kanal dieses Speichers wird durch einen Deskriptor geschützt, dessen Inhalt Auskunft über den jeweiligen Zustand gibt. Abbildung 2 zeigt die Informationen, die der Deskriptor enthält:

B (Belegt) Der betreffende Deskriptor ist initialisiert und damit einem Kommunikationsobjekt zugeordnet.

A (Aktiv) Die aktuelle Adresse ist Null, d.h. ein Datenblock wurde vollständig übertragen oder ausgelesen.

Z (Zustand)

00 : unbenutzt, dies ist der Zustand nach der Initialisierung
01 : gelesen, der letzte Zugriff war eine Leseoperation
10 : beschrieben, der letzte Zugriff war eine Schreiboperation
11 : geschlossen, das Kommunikationsobjekt darf nicht mehr benutzt werden.

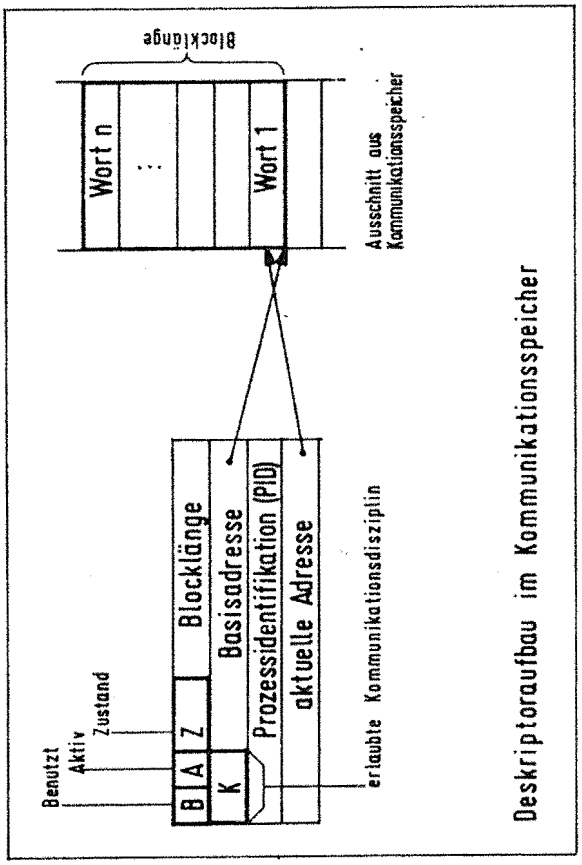


ABBILDUNG 2

K beschreibt die von den Prozessen definierte Kommunikationsdisziplin.

11 : (wr)*
01 : (w*r)*
10 : (wr*)*
00 : (w*r*)*

Basisadresse
verweist auf den Speicherbereich, der dem jeweiligen Kanal zugeordnet ist. Dieser Speicher ist von außen (vom Bus aus) nicht adressierbar.

aktuelle Adresse
verweist innerhalb des Speicherbereichs auf den Inhalt, der während eines Lese- oder Schreibvorganges als nächstes gelesen oder geschrieben wird.

PID ist die Prozeßidentifikation des Prozeßes, der aufgrund des Zustandes des Kanals unterbrochen wurde und bei einer Änderung des Zustandes erneut aktiviert wird.

Die Kopplung der beiden Prozesse ist festlegbar. Es gibt vier Abstufungen zwischen vollständig synchronisiert und vollständig unsynchronisiert. Festgelegt wird dies durch die K-bits im Deskriptor. Die vier Möglichkeiten lassen sich wie folgt beschreiben:

1. (wr)*
   Beide Prozesse sind vollständig synchronisiert. Diese Zugriffsart zwingt den Produzenten - Prozeß mit der Übertragung des Wertes zu warten, bis der Konsument den vorhergehenden Wert des Objekts vollständig gelesen hat. Genauso muß der Konsument nach dem Lesen eines Werts warten, bis der Produzenten - Prozeß wieder in einen neuen Wert übertragen hat.
2. (w*r)*
   Der Produzent darf bei dieser Zugriffsart einen neuen Wert in das Kommunikationsobjekt eintragen, auch wenn der vorher übertragene Wert noch nicht gelesen wurde.
3. (wr*)*
   Der Wert des Objekts darf vom Konsument beliebig häufig gelesen werden.
4. (w*r*)*
   Die Kopplung ist in dieser Zugriffsart am schwächsten. Die beteiligten Prozesse greifen völlig unsynchronisiert auf das Kommunikationsobjekt zu.

3.2 Der VETO Mechanismus

Veto erzwingt als Signal vom Speicher zum Busmaster das Blockieren das Prozesses. Wie aus Abbildung 2 ersichtlich, enthält der Deskriptor eine Beschreibung der Zugriffsrechte und der Zugriffssequenzen, die auf ein in diesem Speicherbereich abgelegtes Kommunikationsobjekt zulässig sind, sowie eine Identifikation des unterbrochenen Prozesses. Für jedes, zusammen mit dem Deskriptor gespeicherte Objekt, wird nun eine bestimmte Zugriffsfolge zugelassen. Die Speicherverwaltung (MMAP) überprüft während der Laufzeit die Speicherzugriffsfolge bei jedem einzelnen Zugriff. Die Folge wurde bei der Definition des Prozesses und des Kommunikationsobjektes festgelegt.

Bei einem Datentransfer zum Kommunikationsspeicher ist der am zentralen Bus aktive Prozessor der Busmaster, der die Kommunikation mit Hilfe seines lokalen Betriebssystems abwickelt. Nur er kann zu diesem Zeitpunkt Adressen anlegen und einen Datenzyklus durchführen. Alle Zugriffe auf Kommunikationskanäle werden über den Knotenbus und die MMAP abgewickelt.

Stimmt das Zugriffsrecht des Busmasters (read oder write) auf den Kanal nicht mit dessen Zustand und der vereinbarten Kommunikationsdisziplin überein, so darf der Datentransfer auf dem Bus nicht mit einer Bestätigung beantwortet und abgeschlossen werden. Vielmehr wird, initiiert vom Kommunikationsspeicher jetzt ein VETO-Signal über den Bus ausgegeben.

Dieses Veto-Signal ist ungerichtet, geht also an alle Prozessoren. Der Speicherverwaltung braucht nicht bekannt zu sein, an welchen Prozessor sich das Veto richtet, weil der jeweils aktive Busmaster das Veto automatisch auf sich bezieht und seinen gerade aktiven Prozeß blockiert.

Da die Datenleitungen zum Zeitpunkt des Vetos nicht benutzt wurden, ist der Datenbus frei. Auf den Bus wird jetzt vom aktuellen Busmaster die Prozeßidentifikation (PID) gelegt, die den am Bus aktiven Prozeß eindeutig kennzeichnet. Zeitlich etwas verzögert sendet dann der aktive Busmaster ein Veto-Acknowledge auf den Bus, das die Gültigkeit der PID-Daten anzeigt. Diese zeitliche Verzögerung beinhaltet den Übernahme-Strobe, mit dem die Prozeßidentifikation in den Deskriptorspeicher geladen wird. Damit ist dem Deskriptor des aktuellen Kanals, dessen Adresse immer noch am Bus an-

liegt, die Prozeßidentifikation des blockierten Prozesses zugeordnet, und der Datenzyklus kann abgeschlossen werden. Dieser Zyklus kann weder durch Interrupts noch durch direkte Speicherzugriffe unterbrochen werden.

Infolge anderer Prozesse oder Kommunikationsvorgänge wird das Zugriffsrecht auf den Kanal zu einem späteren Zeitpunkt verändert: entweder ist in den Kommunikationskanal inzwischen ein gültiger Wert eingetragen worden oder der Kanal ist bereit, einen neuen Wert aufzunehmen.

Dabei wird geprüft, ob der zugeordnete Deskriptor in seinem PID - Teil einen Prozeß bezeichnet ($PID \neq 0$), der auf den Kommunikationskanal wartet.

Da die Identität des unterbrochenen Prozesses durch die im Deskriptor stehende Prozeßidentifikation bekannt ist, kann dem Betriebssystem des Knotens per Interrupt-Serviceroutine dieser PID mitgeteilt werden. Dieses überführt daraufhin den wartenden Prozeß in den "ready-to-run" -Zustand. Der Prozeß wird fortgesetzt, sobald der Prozessor bereit ist, auf dem er seine Aktivität begonnen hatte.

Aus den Statusinformationen des Deskriptors lassen sich folgende Fälle ableiten, die den normalen Datentransfer unterbrechen:

1. Der Deskriptor ist aktuell keinem Kommunikationskanal zugeordnet. In diesem Fall wird ein Interrupt ausgelöst, der dem Knotenbetriebssystem einen System- oder Programmfehler anzeigt.
2. Der Deskriptor zeigt den Zustand "geschlossen", was eine Prozeßtermination, zumindest aber die Auslösung eines "Exceptions", bewirkt. Es wird der Prozeß angesprochen, der den Zugriff versucht hat.
3. Der Deskriptor ist dem entsprechenden Kanal zugeordnet, die Kommunikationsdisziplin entspricht aber nicht dem aktuellen Zugriff. In diesem Fall wird das oben beschriebene VETO ausgelöst.

Der bisherige Aufbau setzt noch einelementige Kanäle voraus. Um auch Kommunikationen mit Kanalkapazitäten von $k=o$ oder $k>1$ zu unterstützen, muß das Konzept erweitert werden.

Zur Emulation synchroner Kommunikation mit diesem Mechanismus muß der sendende Prozeß sich blockieren bis die Nachricht abgeholt wurde. Diese Betriebsart kann durch ein Statusbit im Deskriptor gefordert werden. Es wird dann durch den Veto-Mechanismus der PID eingelesen und der sendende Prozeß blockiert. Nach Verbrauch des Objekts durch den empfangenden Prozeß wird der blockierte Prozeß wieder weitergeführt. Es wird also der Empfang der Daten durch den empfangenden Prozeß implizit dadurch bestätigt, daß der sendende Prozeß fortgesetzt wird. Erreicht der empfangende Prozeß als erster den Synchronisationspunkt, so wird er wie bei der asynchronen Kommunikation blockiert. Zeitüberwachungen der Verweildauer eines Wertes im Kommunikationsspeicher kann auf Störungen im Prozeßgeschehen hinweisen.

Zur Realisierung von Kanälen mit Kapazitäten größer 1 müssen im Kommunikationsspeicher zusätzliche Zeiger auf den Speicher verwaltet werden. Die Stellung dieser Zeiger reguliert die Zugriffsdisziplin. Prinzipiell sollte diese Verwaltung natürlich mit Software realisiert werden. Dies setzt allerdings einen zusätzlichen Prozessor mit der alleinigen Aufgabe der Speicherverwaltung voraus und dürfte den bisherigen Vorteil kürzerer Kommunikationszeiten stark beeinträchtigen.

Abbildung 3 zeigt die Kommunikation über Knotengrenzen hinweg. Ein bisher nicht gelöstes Problem ist die Kopplung

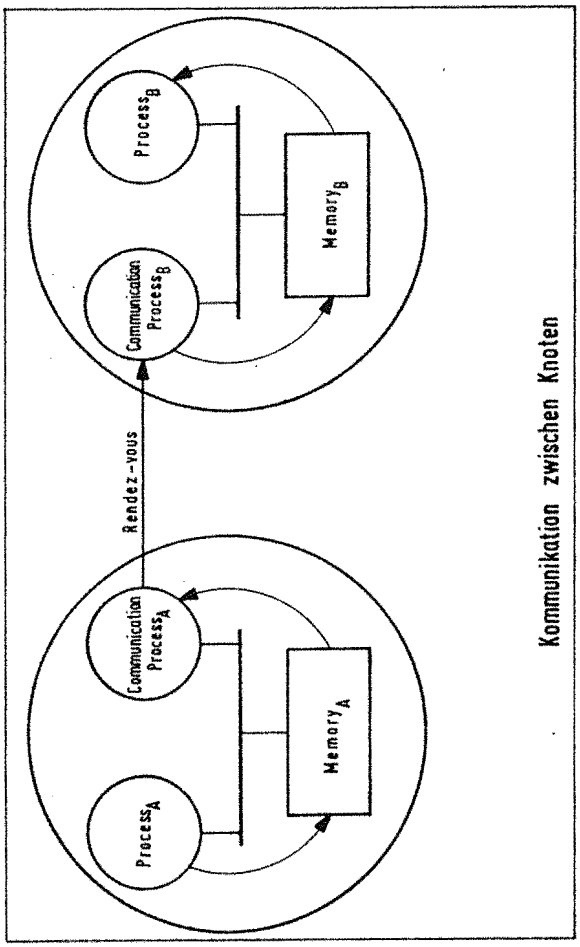


ABBILDUNG 3

von Prozessen, die auf unterschiedlichen Knoten des Systems ausgeführt werden. In unseren Überlegungen wird die Verteilung auch durch die Verteilung des Kommunikationskanals unterstützt. Dies führt zum Aufbau wie in Abb. 3, wobei zur Kommunikation die Speicher in beiden Knoten beteiligt werden.

4. ZUSAMMENFASSUNG UND AUSBLICK

Das vorgeschlagene Konzept der hierarchischen Funktionsverteilung des Betriebssystems und der impliziten Synchronisation von Applikationsprozessen weist, gegenüber den auf dem Markt befindlichen Produkten, einige Verbesserungen und Erleichterungen im Einsatz auf:

- es erhöht durch die implizite Synchronisation und das asynchrone Kommunikationsverhalten den Grad der Parallelität und läßt eine geringere Anzahl der Prozeßwechsel erwarten.
- es reduziert den Aufwand zur Beschreibung von Kommunikation in den Applikationsprozessen erheblich, wobei gleichzeitig die Belastung des Systems gesenkt wird.
- es ermöglicht eine hierarchische, auch heterogene Erweiterung, ohne Veränderung der Applikationsprozesse.

- es verbindet die Vorteile von fester Kopplung und gemeinsamen Speicherbereichen (mit der damit verbundenen schnelleren Kommunikation) mit den Vorteilen der losen Kopplung, dynamischer Rekonfigurierbarkeit, Prozeßverteilung und schnellerer Reaktion auf Eingaben.

Als Nachteile sind derzeit die nicht standardisierte Speicherverwaltung einerseits und die hinzukommenden Vetoleitungen andererseits zu nennen.

Ein oben beschriebener Aufbau wurde im September 85 fertiggestellt und erstmalig getestet. Dabei erwiesen sich die gemachten Angaben als richtig:

- die Kommunikation zwischen den Prozessen ist äußerst effizient. Sie ist nicht vom Kommunikationsprotokoll, sondern lediglich von der Zugriffszeit auf den verwendeten Bus und der Zykluszeit des Speichers abhängig. Die Busbelastung durch ein **VETO** liegt im Fehlerfall bei 700 ns.
- die Kommunikationsdisziplin führt zur Synchronisation der beteiligten Prozesse. Der Veto Mechanismus und das erneute Starten eines Prozesses nach einer Änderung des Status eines Kanals führen zur ordnungsgemäßen Synchronisation.

Als Erweiterung des vorgestellten Konzepts und der derzeitigen Realisierung ist eine Zeitüberwachung für die einzelnen Objekte geplant. Diese ermöglicht dann eine schnellere Transaktionsüberwachung. Der bereits oben beschriebene Ansatz stellt ein Prozeßsystem als eine Baumstruktur dar. Die Kontrolle dieser Struktur erfolgt durch das (globale) Betriebssystem, dessen Komponenten auf Knoten verteilt werden. Aufgaben dieses Betriebssystems in Bezug auf die Verwaltung der beteiligten Prozesse sind u.a.:

- Basisdienste bereitzustellen; z.B. Compiler und Debugger,
- die Bereitstellung der Kommunikationsprimitiven, eine Firmware integrierte Zwischensprache,
- die initiale Konfigurierung des Prozeßsystems inklusive des Ladens vom Entwicklungssystem,
- die Rekonfiguration des Systems aufgrund äußerer Einflüsse (z.B. einer Änderung des zu regelnden Systems). Es soll möglich sein, zu jeder Zeit den Zustand des Systems zu erfragen und zu ändern, also das Kommunikationsnetz abzuändern,
- die Rekonfiguration des Systems aufgrund ungünstiger Lastverteilungen. Hierzu gibt es keine bekannten Verfahren.

Die Zuordnung von Prozessen und Prozessoren unterliegt u.a. folgenden Einflüssen:

- den vom Prozeß benötigten Resourcen,
- den von den Prozessoren bereitgestellten Resourcen,
- dem Grad der Kopplung zwischen Prozessen,
- der Auslastung von Prozessoren, Bus und Peripherie,
- der topologischen Anordnung technischer Prozesse und den sie steuernden (regelnden) Prozessen,
- den Anforderungen des technischen Prozesses (z.B. an die Regelzeit).

Unsere Vorstellung der Zuordnungsstrategie ähnelt der Traversierung eines Baumes, wobei eine feste Strategie zugrunde liegt, die aber von jedem Cluster beeinflußt werden kann. Dieser Konfigurationsprozeß beginnt initial an der Wurzel des Systembaumes, bei jedem Start eines Prozesses im entsprechenden Cluster.

Anmerkungen:

Das Hardwarekonzept entstand durch intensive Diskussionen mit Dr.S.Jähnichen, GMD Forschungsstelle Karlsruhe und durch viele Gespräche mit Kollegen zum Entwurf einer Rechnerarchitektur für intelligente, sensorgeführte Robotersysteme. Die Arbeiten zur Realisierung dieser Kommunikationskonzepte wurden im Institut für Informatik III, Lehrstuhl Prof. U. Rembold durchgeführt und durch die DFG unter der Kennziffer Re 489/2 gefördert.

Literaturverzeichnis

1 Behr,P.'Entwurf eines verteilten Multicomputer-Systems, TU Berlin Dissertation 1983

2 Bowen,B.A., 'The logical design of Multiple- Microcomputer Systems', Prentice-Hall Inc. 1980

3 Böhler,E. 'Entwurf und Aufbau einer Speicherverwaltungseinheit' Diplomarbeit Uni.Karlsruhe 85

4 Brinch Hansen,P.'Operating System Principles', Prentice Hall,82

5 Dijkstra, E.W.D., 'Cooperating Sequentiell Processes', Programming Languages, Academic Press 68

6 Färber, 'Bussysteme',Oldenburg 84

7 Giloi,W. 'Rechnerarchitektur'Springer-Verlag Berlin 1981

8 Hoare, C.A.R., 'Monitors: An Operating System Structuring Concept', CACM, Vol.17, #10,74

9 Iliffe,J.K. 'Basic Machine Principles' American Elsevier 1972

10 Jähnichen ,S., 'Kommunikation und Synchronisation in verteilten Multiprozessorsystemen' Interner Bericht , GMD Forschungsstelle Karlsruhe, 1984

11 Jähnichen, S., Kordecki, C.,'Communication and Synchronization in Distributed Systems', to be published

12 Spaniol,O. Konzepte und Bewertungsmethoden für lokale Rechnernetze, Informatik-Spektrum 5(1982)

Anschrift des Autors

C.Kordecki
Institut für Informatik III
Prof.Rembold
Universität Karlsruhe
Postfach 6380
7500 Karlsruhe

Das Echtzeitsystem c´ t68000 GWK

Harald Klappa

1 Einführung

Themenstellung des vorliegenden Berichtes war zuerst allein der bei uns entwickelte und gefertigte Rechner c't 68000 GWK.

Angeregt durch die Einordnung des Vortrages in die Themengruppe "Unkonventionelle Rechnerarchitekturen" möchte ich jedoch an dieser Stelle besonders die Möglichkeiten darstellen, die sich durch den Verbund des c't 68000 mit VMEbus Systemen ergeben.

Dieser Verbund eröffnet mit seiner dualen Busstruktur, es wird zwischen globalen und lokalen Zugriffen unterschieden, Möglichkeiten in der Echtzeitverarbeitung von Messwerten und Prozessdaten, die mit vergleichbaren Microcomputer Systemen bisher nicht zu realisieren waren.

2 Das System c't 68000 GWK

2.1 Überblick

Der c't 68000 GWK ist entstanden aufgrund einer Anregung und in enger Zusammenarbeit mit der Zeitschrift c't Magazin für Computertechnik.

Er wurde konzipiert als preiswerter Universalrechner für technische und wissenschaftliche Anwendungen im Bereich der Messwerterfassung, Messwertverarbeitung und Prozesskontrolle.

Bedingt durch die Zusammenarbeit mit der Zeitschrift ist das System von der Dokumentation her sehr offen gehalten. Neben den Schalt- und Bestückungsplänen werden auch die logischen Gleichungen der verwendeten PAL Bausteine veröffentlicht.

2.2 c't 68000 GWK Beschreibung

Der c't ist ein modular aufgebauterechter Rechner, dessen einzelne Funktionsmodule über eine 16 Bit breite interne Querverdrahtung miteinander kommunizieren, die im wesentlichen die Anschluss Belegung der CPU 68000 wiederspiegelt. Diese ist jedoch nicht als anwenderverfügbarer Bus zu betrachten, da sie von ihrer Treiberleistung her nur für einen sehr begrenzten Ausbau geeignet ist.

Der eigentliche Systembus wird dem Anwender durch ein System Bus Interface (SBI) zur Verfügung gestellt. Es stehen zwei solcher Interfaces zur Verfügung. Eines mit einer externen Datenbreite von 8 Bit für den GWK EBCS Standard Bus - hier sind zahlreiche Interface und I/O Karten verfügbar - und ein zweites mit 16 Bit Datenbreite und einer zur internen Querverdrahtung des c't kompatiblen Busbelegung.

Die Module sind auf Platinen im einfach Europaformat untergebracht und gliedern sich wie folgt:

- **CPU- Modul** mit 8 MHz CPU, Systemtaktoszillator, Betriebsspannunsüberwachung, Single Step Logik, Reset- und Abortlogik, Bus Error Logik, Puffer Batterie für CMOS RAM und 8 Steckplätze, die wahlweise mit EPROM oder CMOS RAM bestückt werden können.

- **DRAM- Modul** mit wahlweise 256 K oder 1 MByte dynamischem RAM. Eigene Refreshsteuerung.

- **IOFDC- Modul** mit Floppy Disc Controller und I/O Schnittstellen für Terminal bzw. Tastatur, serielle Kommunikation und parallelen CENTRONICS Druckeranschluss. Weiterhin Timer 3 * 16 Bit und Echtzeituhr mit Batteriepufferung.

- **SBI EBCS- Modul** als Systembus Interface zum 8 Bit breiten EBCS Standard Bus. Ablaufsteuerung der 16 Bit zu 8 Bit Transformation, Interrupt Handler mit Priorisierung und Bustreiber.

- **SBI ICS- Modul** als Systembus Interface zum 16 Bit breiten ICS Bus. Interrupt Handler mit Priorisierung und Bustreiber.

- **GDP- Modul** als optionaler Graphik Prozessor. Ausgestattet mit NEC 7220 und 128 KByte Bildwiederholspeicher für 1024 * 1024 Pixel. Erweiterungsmöglichkeit für Farbdarstellung.

3 GWKVME CPU 68K

3.1 Überblick

Die GWKVME CPU 68K PCU wurde konzipiert als Prozess Steuer Karte, (Process Controll Unit PCU), die in besonderem Maße für den Einsatz in der schnellen

Messwerterfassung, der Steuerungs- und Regeltechnik sowie bei allen Aufgaben der Prozesskontrolle in Single und Multi Prozessor Systemen geeignet ist.

Sie basiert auf der mächtigen 68000 CPU und dem international genormten Industrie Standard VMEbus. Das VMEbus Interface erfüllt in allen Punkten die VME Spezification Rev. B. Es enthält den Interrupt Handler, Bus Requester und einen Four Level Bus Arbiter.

Entsprechend der Konzeption als Prozesscontroller wurde auf die üblichen Austattungsmerkmale von Standard CPU-Karten verzichtet und statt dessen Funktionen implementiert, die für Prozessanwendungen nützlich sind.

Die ausschliessliche Verwendung statischer Speicher, es stehen 10 Steckplätze wahlweise für EPROM oder CMOS RAM mit Batteriepufferung frei konfigurierbar zur Verfügung, unterstützt schnelle Echtzeitanwendungen.

Die Karte verfügt über eine batteriegepufferte Echtzeituhr, einen Timer/Counter 3 * 16 Bit, ein Status Display mit 7 Segment LED und eine serielle Schnittstelle RS 232 C.

Durch den eingebauten Watchdog mit wählbarer Intervallzeit und durch die ebenfalls auf der Karte befindliche Betriebsspannungs- überwachung in zwei Ebenen mit Power Fail Interruptauslösung wird die Störsicherheit in Langzeit Anwendungen erheblich gesteigert.

Ein Arithmetikprozessor (NS 32081) kann als optionale Ausstattung nachgerüstet werden.

Der interne Bus der CPU ist als Local Bus über den P2 Connector voll gepuffert und DMA fähig herausgeführt. Er bietet schnellen Zugriff auf Speicher und I/O Bausteine und ermöglicht den Aufbau hierarchischer Multiprozessor Systeme.

3.2 Technische Beschreibung GWKVME CPU 68k PCU

Auf eine eingehende technische Beschreibung aller oben oben aufgeführten Eigenschaften soll verzichtet werden. Lediglich die Businterfaces sollen etwas intensiver behandelt werden.

3.2.1 VMEbus Interface

Die Karte verfügt über ein in allen Punkten der VME Spezifikation Rev.B entsprechendes VMEbus Interface mit Interrupt Handler und optionaler Bus Arbitration. In ihrer Standardausführung kann sie als Bus Master in Single CPU Systemen und als Slave in Multiprozessor Systemen verwendet werden.

Bus Reqester und Arbiter sind auf einer Huckepack Platine untergebracht und können optional nachgerüstet werden. Der Arbiter bietet die Betriebsarten: Priority, Round Robin, One Level und Arbiter Off. Mit dem Requester ist der Betrieb im Release on Request Modus und im Release when done Modus möglich. Hiermit ist vollständiger Multimaster Betrieb in mehrfach Prozessor Systemen möglich.

3.2.2 Lokaler Bus

Besonderes Augenmerk verdient der über den P2 Steckverbinder herausgeführte lokale Bus.

Dieser lokale Bus führt sämtliche Systemsignale der CPU. Er ist voll gepuffert und DMA fähig. Da er nicht durch ein aufwendiges Busprotokoll belastet ist, ermöglicht er einen schnellen Zugriff nicht nur auf weitere Speicher sondern auch auf I/O Module. Wir nennen ihn ICSbus (Industrial Control System). Seine Pinbelegung ist kompatibel zum internen Bus des c't 68000 GWK.

Mit Hilfe der vom c't zur Verfügung stehenden Komponenten (DRAM, Floppy Controller, Schnittstellen), kann das ursprüngliche Zielsystem zum komfortablen Entwicklungssystem ausgebaut werden, das Entwicklung und Test der Anwendersoftware unter realen Randbedingungen ermöglicht.

In erster Linie aber ermöglicht dieser herausgeführte Lokale Bus den Aufbau hierarchischer Multiprozessorsysteme.

4 Bussysteme

4.1 Der ICSbus

Der ICSbus ist von seiner Struktur her recht einfach gehalten. Er besteht im Prinzip aus den von der CPU 68000 zur Verfügung gestellten Daten, Adress und Steuerleitungen, die gepuffert herausgeführt sind. An Stelle der CPU Signale IPL0 bis IPL2 und der Function Codes stehen auf dem Bus die Interrupt Request Leitungen für die Level 1 bis 7 und die IACKIN/IACKOUT daisy chain zur Verfügung.

Im Gegensatz zum VMXbus ist der ICSbus nicht durch ein aufwendiges Protokoll belastet und ist auch

nicht nur auf Speicherzugriffe beschränkt. Er bietet daher die Möglichkeit sehr schneller Zugriffe und Datentransfers mit Speicher- und I/O Baugruppen. Einschränkend muß bemerkt werden,daß der ICSbus nicht multiprozessorfähig ist.

4.1 Der VMEbus

An dieser Stelle soll nicht zum wiederholten Male eine Aufzählung aller Vorteile und Leistungsmerkmale des VMEbus Konzeptes stattfinden, da diese wohl als bekannt vorausgesetzt werden können. Eine Bemerkung sei mir jedoch gestattet.

Das VMEbus Konzept erfordert von allen Komponenten, seien es CPU Karten, Speichereinheiten oder I/O Module, einen wesentlich höheren Bauelementeaufwand als einfache Bussysteme. Der hierdurch verursachte höhere Preis der VMEbus Komponenten findet seine Rechtfertigung aber erst dann, wenn die eingekaufte Leistungsbandbreite auch ausgenutzt wird. Das findet in der Regel nur in Multiprozessorsystemen statt.

An diesem Punkt aber beißt sich die Katze in den Schwanz. Arbeiten nämlich auf einem Bus mehrere CPU Karten oder andere Busmaster, die auf Arbeitsspeicher, Massenspeicher oder Interface Karten zugreifen müssen, wird der Bus durch die Vielzahl der Transfers, die jeweils mit dem Protokoll belastet sind, derart blockiert, daß von der theoretisch möglichen Leistungssteigerung nur ein geringer Teil übrig bleibt.

Es wird im allgemeinen gesagt, daß in konventionell aufgebauten Systemen ab der vierten CPU eine Verbesserung des Durchsatzes nicht mehr zu erreichen sei.

4.3 Hierarchische Busstruktur

Das im vorhergehenden Abschnitt dargestellte Problem lässt sich sehr wirkungsvoll und elegant durch Einführung einer hierarchischen Busstruktur lösen.

Jede CPU hat die Möglichkeit,über ihr lokales Businterface auf lokalen Speicher und auf lokale Ein/Ausgabekanäle zuzugreifen. Das bewirkt, daß der VMEbus durch diese Transfers nicht mehr belastet wird. Er wird nur noch herangezogen zur Kommunikation der einzelnen CPU's untereinander und zum Zugriff auf globale Speicher oder I/O Geräte.

Durch die hier dargestellte unkonventionelle Systemarchitektur wird erreicht, daß Leistungsfähigkeit und Durchsatz von Multiprozessor Systemen annähernd proportional der Anzahl der eingesetzten CPU's zunehmen. Als Nebeneffekt ergibt sich die Möglichkeit, in einem solchen System weit mehr als die allein über den VMEbus zu adressierenden 16 MByte Arbeitsspeicher zur Verfügung zu stellen.

5 Software

Wie in den vorhergehenden Abschnitten verdeutlicht, sind die beiden Systemfamilien c't 68000 GWK und GWKVME sehr eng miteinander verbunden. Aus diesem Grunde ist auch bei der Softwareausstattung darauf geachtet worden, daß beide Systeme absolut Softwarekompatibel sind. Diese Kompatibilität geht soweit, daß Betriebssysteme und Anwenderprogramme ohne Änderung von einem System auf das andere übertragen werden können.

Zur Standardausstattung gehöhrt ein PEARL Compiler und das hierfür optimierte Echtzeit Betriebssystem RTOS(UH). Beide wurden am Institut für Regelungstechnik der Universität Hannover von Professor Gerth entwickelt und auf die vorgestellten Systeme portiert. Für die uns hierbei gewährte Unterstützung sei Professor Gerth und seinen Mitarbeitern an dieser Stelle gedankt.

Über RTOS und PEARL hier viele Worte zu verlieren, hieße Eulen nach Athen zu tragen. Auch die beiden anderen optional verfügbaren Betriebssysteme OS-9/68000 und CP-M/68K sollen nur namentlich erwähnt werden, da über beide schon an anderer Stelle berichtet wurde.

Sieht man vom CP-M/68K ab, das nicht als Echtzeitsystem bezeichnet werden kann, so ist es für die Funktion der weiter oben vorgestellten hierarchischen Systemarchitektur auch unerheblich, welches Betriebssystem verwendet wird.

6 Ausblick

Sinn dieses Berichtes war es, darzustellen, wie durch unkonventionelle Rechnerarchitekturen Leistungsbandbreiten geboten werden, die auf konventionelle Art mit Micros noch nicht und erst recht nicht zu vergleichbaren Preisen machbar waren.