Mehrrechner-PEARL

Erfahrungen und notwendige Konsequenzen. Von Dr. **H. Steusloff**, Karlsruhe

1. Merkmale von Mehrrechnersystemen in der Prozeßautomatisierung

Steigende Anforderungen an die Führung technischer Prozesse haben heute eine Komplexität der Prozeßautomatisierungssysteme bewirkt, die mit den bislang üblichen assembler-programmierten Einzelrechnersystemen nur noch schwer beherrschbar ist [1]. Die gerätetechnische Antwort auf diese Situation ist die Entwicklung verteilter Mehrrechnersysteme für die Prozeßautomatisierung, wie sie in jüngster Zeit in zunehmend schneller Folge auf dem Markt erscheinen; Beispiele sind in [2, 3, 4, 5] genannt. Wesentliche unterscheidende Merkmale solcher Systeme sind gegeben durch

- den Grad der Annäherung an die allgemeine Polyederstruktur (B i l d 1),
- die Art und Aufgabenverteilung der Verarbeitungseinheiten (Prozessoren),
- das Prinzip der Kommunikation zwischen den Verarbeitungseinheiten und seine Realisierungsform [6],
- die einsetzbaren Methoden zur Erhöhung der Fehlertoleranz [7].

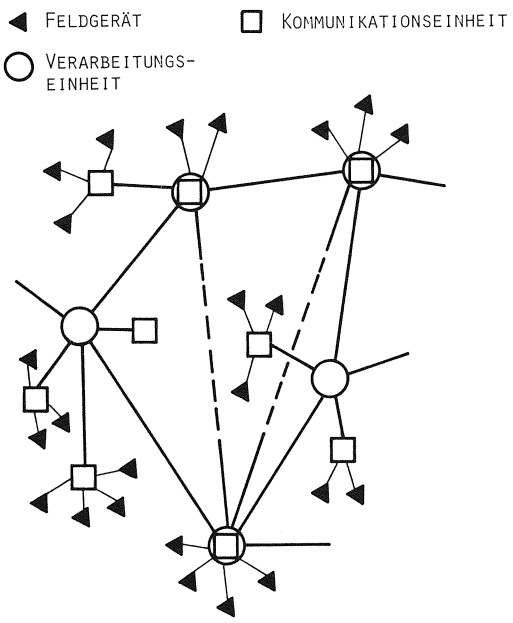

Bild 1. Polyeder-Systemstruktur

Am ersten Beispiel eines im IITB entwickelten [8] und industriell eingesetzten [9] verteilten Mikrorechnersystems (RDC = Really Distributed Computer Control System) seien diese Merkmale erläutert.

Der Ansatz des IITB ging aus von der Verwendung der Mehrrechnersysteme bei der Automatisierung von Anlagen, deren Struktur und Eigenschaften während des Betriebes an neue Anforderungen leicht anpaßbar sein müssen (open-ended design). Systemkonfiguration und Fehlertoleranzmaßnahmen sind daher so zu gestalten, daß sie leicht und ggf. online veränderbar sind. Eine vorübergehende Leistungsdegradation wird aus Wirtschaftlichkeitsüberlegungen zugelassen; es ist daher dynamische, sog. funktionsbeteiligte Redundanz zugelassen. Die dynamische Redundanz sowie eine einsatzbedingt, u.U. logisch enge Kopplung der verschiedenen Verarbeitungseinheiten bei räumlicher Verteilung, bedingen eine schnelle, gesicherte Kommunikation innerhalb des Mehrrechnersystems.

Die Ausführung des räumlich verteilten IITB-Mehrrechnersystems RDC zeigt B i l d 2 . Mehrere Prozessorstationen sind durch ein ringförmiges, seriell übertragendes, dezentral gesteuertes und on-line erweiterbares Kommunikationssystem verbunden. Die Prozessorstationen enthalten jeweils einen Verarbeitungsprozessor PμP, einen Kommunikationsprozessor LμP und die Prozeß-Ein/Ausgabe. Kernstück ist ein Busschalter BSU mit integrierter Stationsüberwachung. Abhängig vom Betriebszustand der Station rekonfiguriert der Busschalter die drei genannten Funktionseinheiten und veranlaßt eine systemweite Bekanntmachung des Betriebs- oder Fehlerzustandes. Die Rekonfiguration einer Station kann z.B. die Ein-/Ausgabe-Einheiten direkt mit dem Kommunikationsprozessor verbinden, so daß bei defektem PμP die Prozeßdaten (auch Interrupts!) in anderen Stationen über das Kommunikationssystem verfügbar sind. Diese Gerätekonfiguration, zusammen mit einer betriebszustandsabhängigen Programmkonfiguration, ermöglicht die Fehlertoleranzmaßnahmen innerhalb des Mehrrechnersystems.

Eine andere Klasse von Mehrrechnersystemen tritt im Bereich der Automatisierung von komplexen, räumlich aber wenig ausgedehnten Gerätesystemen auf (embedded systems). Hier

---

Dieser Beitrag berichtet über Arbeiten, die mit Mitteln des Bundesministeriums für Forschung und Technologie (DV-Programme) gefördert wurden. Die Verantwortung für den Inhalt liegt allein beim Verfasser.

liegt die Struktur des Mehrrechnersystems fest. Fehlertoleranzmaßnahmen sind bestimmt durch die Notwendigkeit, die volle Leistung des Mehrrechnersystems auch bei Fehlerzuständen zu gewährleisten; daher findet man oft statische Redundanz. Wegen der räumlichen Nähe der Verarbeitungseinheiten sind schnelle parallele Busverbindungen für die Kommunikation möglich; andererseits vermeidet man komplizierte Kommunikationsverfahren und sorgt für eine schwache Kopplung der einzelnen Verarbeitungseinheiten. Ein solches Mehrrechnersystem liegt dem Beitrag von H.-J. Schneider (im vorliegenden Band) zugrunde [10].

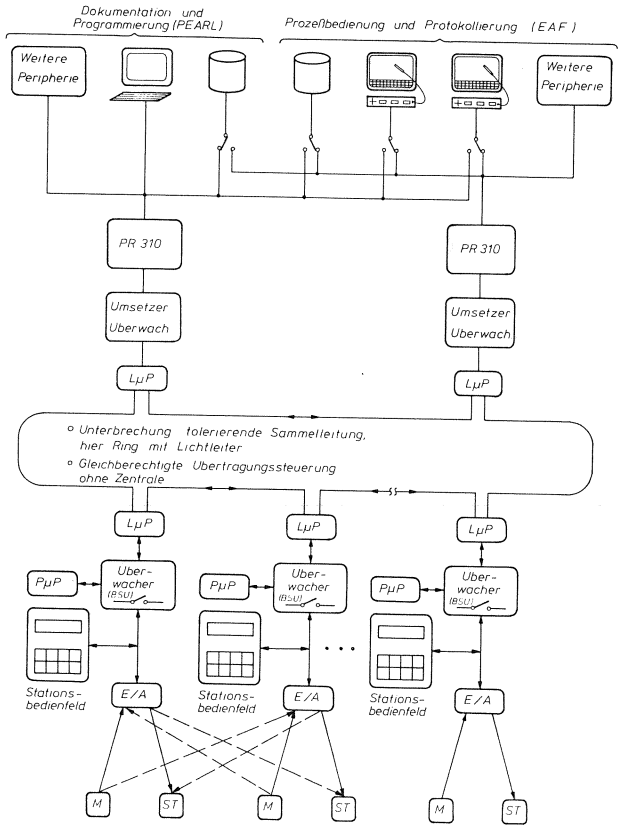

Bild 2. Struktur des RDC-Systems (IITB)

Es sei betont, daß die eben genannten Mehrrechnersysteme lediglich verschiedene Ausprägungen der allgemeinen Polyederstruktur nach B i l d 1 sind. Es sollte daher möglich sein, ein gemeinsames Konzept für eine Programmiersprache zur Beschreibung der programmtechnisch zu realisierenden Funktionen bei solchen Mehrrechnersystemen zu finden.

2. Struktur und Komponenten einer höheren Echtzeitprogrammiersprache für Mehrrechnersysteme

Die Programmierung von Mehrrechnersystemen nach Abschnitt 1 ist mit dem Hilfsmittel "Assembler" unwirtschaftlich und fehleranfällig. Es ist daher eine höhere Sprache einzusetzen, die nach Möglichkeit sowohl zur Systemprogrammierung als auch zur Anwendungsprogrammierung geeignet ist.

Die Struktur und notwendigen Komponenten einer solchen Sprache hat der Verfasser in [11] über system- und entscheidungstheoretische Ansätze abgeleitet; das Ergebnis zeigt B i l d 3 . Dieser Sprachentwurf ist aufgrund seiner Ableitung aus der allgemeinen Polyederstruktur nach B i l d 1 als allgemeingültig für Mehrrechnersysteme anzusehen. Er enthält sowohl die bekannten funktionsbeschreibenden als auch strukturbeschreibende Sprachelemente. Die in heute üblichen Programmiersprachen nicht oder nur in Ansätzen vorhandenen Sprachelemente sind in B i l d 3 durch einen vollen oder unterbrochenen Strich doppelt eingerahmt. Nicht aus der Polyederstruktur ableitbar sind die Sprachelemente zur Behandlung von Fehlerzuständen, die aber explizit in der Sprache vorhanden sein sollten und eine transparente Beschreibung des Fehlertoleranzverhaltens eines Systems zulassen müssen.

Die Realisierung dieses Sprachenentwurfs sollte - wenn möglich - auf der Basis einer existierenden Sprache erfolgen, um bewährte algorithmische Sprachelemente übernehmen zu können. Über die Auswahl dieser Basisprache und deren Erweiterung für Mehrrechnersysteme berichtet der folgende Abschnitt.

3. PEARL für Mehrrechnersysteme (MEHRRECHNER-PEARL)

Anhand einer Wertanalyse, wie sie in [12] beschrieben ist, fiel die Entscheidung zugunsten der Sprache PEARL (Process and Experiment Automation Realtime Language), die zwischenzeitlich als Vornorm bzw. Normvorschlag DIN 66253 vorliegt [16, 17]. Die Vorzüge von PEARL gegenüber anderen höheren Programmiersprachen wie FORTRAN, PASCAL oder auch der Systemimplementierungssprache Ada liegen in

- dem Strukturierungsansatz gemäß B i l d 3 aufgrund der Trennung von SYSTEM- und PROBLEM-Teil,
- dem Vorhandensein von strukturbeschreibenden Sprachelementen in Form der Datenwegbeschreibung des SYSTEM-Teils,
- der Existenz eigentlicher Sprachelemente für die Formulierung paralleler Prozesse, Echtzeitbedingungen und Prozeßdaten-Ein-/Ausgabe.

PEARL enthält damit die in B i l d 3 gestrichelt eingerahmten Sprachelemente. Die in B i l d 3 durch Vollstriche doppelt umrahmten Sprachelemente fehlen aber auch bei PEARL und sind - je nach den Erfordernissen der Einsatzumgebung gemäß Abschnitt 2 - geeignet zu ergänzen. Im folgenden wird die im IITB realisierte Lösung eines MEHRRECHNER-PEARL zur Programmierung des in Abschnitt 2 skizzierten RDC-Mehrrechnersystems (B i l d 2) vorgestellt. H.-J. Schneider hat in seinem Beitrag [10] über eine andere Ausprägung von PEARL für Mehrrechnersysteme berichtet, die aber ohne Schwierigkeit in den allgemeinen Sprachentwurf nach B i l d 3 einzuordnen ist. Beide Lösungen ergänzen sich daher. Arbeiten zu einer - möglichst normbaren - Sprache MEHRRECHNER-PEARL, die eine einheitliche, gestufte Erfüllung der

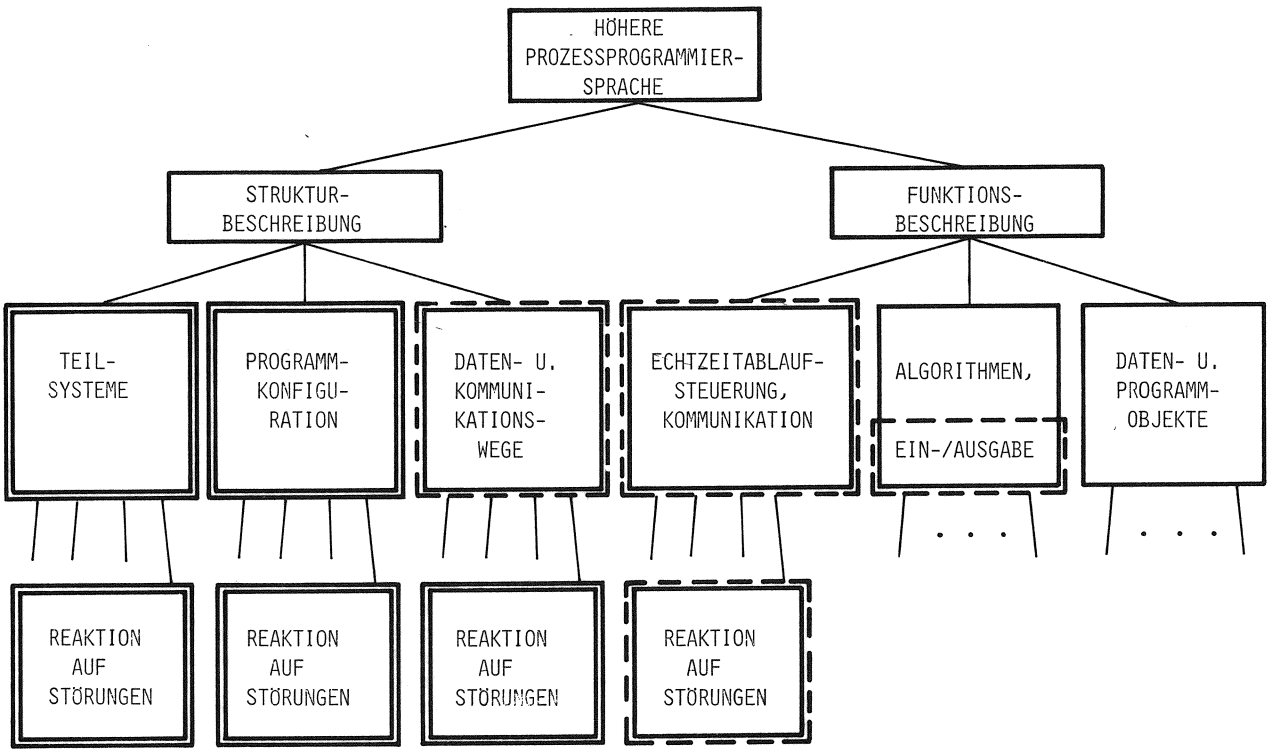

Bild 3. Grundstruktur einer Prozeßprogrammiersprache für verteilte Automatisierungssysteme

Anforderungen aus unterschiedlichen Mehrrechnersystemen gewährleistet, sind im Anlaufen.

MEHRRECHNER-PEARL des IITB ist in [8] ausführlich beschrieben. Diese PEARL-Erweiterungen um einen

- STATIONS-Teil zur Beschreibung der Gerätekonfiguration, einen
- LOAD-Teil zur Beschreibung der Programmkonfiguration, sowie
- eine Ergänzung des SYSTEM-Teils zur Beschreibung alternativer Datenwege

seien daher hier nur durch Beispiele vorgestellt. Dabei sind die Sprachmittel zur Fehlerbehandlung, d.h. zur Formulierung von Fehlertoleranzmaßnahmen, einbezogen.

3.1 Teilsystem-Beschreibung (STATIONS-Teil)

Dieser eigenständige Programmiersprachenteil dient der Beschreibung der statischen Struktur eines verteilten Rechnersystems. Für jeden Einzelrechner (Verarbeitungseinheit) sind drei Komponentengruppen beschrieben, nämlich der Prozessor, die Prozeß-Ein/Ausgabegeräte und die Funktionen des lokalen Betriebssystems (B i l d 4). Diese Angaben sind erforderlich, da im allgemeinen unterschiedliche Prozessoren und den lokalen Erfordernissen angepaßte Geräteausstattungen vorliegen.

Eine Teilsystem-Beschreibung wird eingeleitet durch den Namen des Teilsystems und eine Stationsnummer. Es folgt die Charakterisierung des Prozessortyps. Das Schlüsselwort WORKSTOR kennzeichnet die Notation der Arbeitsspeichergrenzen. Wichtig ist die Verknüpfung von Statusbezeichnungen mit Codes für den Betriebszustand (Statuscodes):

STAIPR möge einen Prozessorausfall, STA1COM den Ausfall der Teilsystemkommunikation bedeuten. Über diese Statuscodes wird der Einsatz der funktionsbeteiligten Redundanz gesteuert.

Es folgen Gerätebeschreibungen als Aufzählung von Eigenschaften und Registeradressen. Die Gerätenamen sind innerhalb eines Gesamtsystems eindeutig; obwohl grundsätzlich beliebige Namen möglich sind, ist zur Verbesserung der Lesbarkeit von Programmen eine aussagekräftige Namensgebung zu empfehlen (z.B. ANIN56Ø: Analog Input, Station Nr. 56. Gerät Nr. Ø).

Die Beschreibung der in einem Teilsystem vorhandenen Betriebssystemfunktionen durch die korrespondierenden PEARL-Schlüsselworte schließt eine Teilsystem-Beschreibung ab.

```
STATIONS;
/*** ERSTE PROZESSORBESCHREIBUNG */
NAME              : STA1,56;
PROCTYPE          : IITB;
WORKSTOR          : 560000,5649142;
STATEID           : (STA1PR:H'188'),(STA1COM:H'430');
   .
   .
/*** GERAETEBESCHREIBUNG */
DEVICE ANIN560    : ANINO,IN,WORD,FIXED(15),
                    H'F101',H'F102',H'F103',
                    NONE,NONE,H'F104',NONE;
   .
   .
/*** BESCHREIBUNG BETRIEBSSYSTEM */
OPSYS             : (ACTIVATE,TERMINATE,SUSPEND),
                    (AT,ALL,UNTIL,WHEN),
                    (ENABLE,DISABLE,TRIGGER),
                    NONE,NONE,NONE;
/*** BESCHREIBUNG DER NAECHSTEN STATION */
NAME              : STA2,57;
   .
STAEND;
```

Bild 4. Beispiel für die Formulierung von Teilsystem-Beschreibungen (STATIONS-Teil)

Die statische Beschreibung des gesamten verteilten Rechnersystems ergibt sich aus der Zusammenfassung aller Teilsystem-Beschreibungen durch die Schlüsselworte STATIONS und STAEND. In ihr sind damit auch die als dynamische Redundanz im Gesamtsystem verfügbaren Betriebsmittel enthalten.

3.2 Programmkonfiguration und -rekonfiguration (LOAD-Teil)

Ladeanweisungen (B i l d 5) bestehen aus der Angabe eines Teilsystems, aus einer sogenannten Ladepriorität, aus einer Betriebszustandsbedingung und aus optionalen Zusätzen. Die Ladepriorität gibt die Wichtigkeit eines Programmoduls relativ zu anderen Moduln innerhalb eines Teilsystems an (Funktionspriorität); sie ist zu unterscheiden von der Ablaufpriorität einzelner Teilprogramme. Der Betriebszustand INITIAL bezeichnet den Urlade- und Normalzustand, während die in Klammern eingeschlossenen logischen Ausdrükke nicht normale Betriebszustände mit Hilfe der Statusbezeichner aus dem STATIONS-Teil beschreiben. Die optionale Angabe einer Startnummer für den Initialzustand beschreibt die Urstart-Reihenfolge eines Programmsystems. Das optionale Attribut RES[IDENT] bewirkt ein Urladen von Programmen in Ersatz-Teilsysteme, um dort auch bei Ausfall der Programmladeeinrichtung verfügbar zu sein oder bei zeitkritischen Programm-Rekonfigurationsfällen den Ausfall von Verarbeitungsfunktionen zu verkürzen.

```
MODULE MOD1;
LOAD;
TO STA1 LDPRIO 5 INITIAL STARTNO 1;
TO STA2 LDPRIO 5 ON (STA1PR.AND..NOT.STA3PR) RES;
TO STA3 LDPRIO 5 ON (STA1PR.AND.STA3PR);
SYSTEM;
   .     BESCHREIBUNG VON DATENWEGEN,DATENKORREKTUR,
   .     ERSATZDATEN.
   .
PROBLEM;
   .
   .     BESCHREIBUNG VON ALGORITHMEN.
   .
MODEND;
```

Bild 5. Beispiel für die Formulierung eines Ladeteils (LOAD-Teil)

3.3 Prozeßdaten-Ersatzwege und -Ersatzwerte (SYSTEM-Teil)

Die Sprachelemente zur Beschreibung von Prozeßdaten-Ersatzwegen und -Ersatzwerten bei Störungen sind im PEARL-SYSTEM-Teil angeordnet, um die Beschreibung von Datentransport und Datengewinnung konsequent von den Verarbeitungsalgorithmen zu trennen (B i l d 6).

Der Name TEMP einer Prozeßdatenquelle möge eine einzulesende Temperatur bezeichnen, deren Meßfühler standardmäßig mit dem Gerät ANIN56Ø (siehe Teilsystembeschreibung), Kanal 8, einem Analogeingang, verbunden sei. Alternative Verbindungen sollen bestehen zu den Geräten ANIN571 und ANIN572, ggf. auch von zusätzlichen (redundanten) Meßfühlern her. Bei diesen Alternativen können Korrekturalgorithmen zur Anpassung der eingelesenen Werte an unterschiedliche Geräte- und Datenwegeigenschaften angegeben werden. Die letzte Alternative bezeichnet einen Ersatzwert, in derselben Zeile stehen auch die Prüfkriterien für Plausibilitätsprüfungen der gewonnenen Meßwerte. Die Bearbeitung einer solchen Datenwegbeschreibung erfolgt in der Reihenfolge ihrer Notation, abhängig vom Betriebszustand der Datenwege und dem Ergebnis der Plausibilitätskontrolle. Sind alle Datenwege gestört oder liefern sie nicht plausible Werte, so gilt der Ersatzwert als Meßgröße. Damit steht für die mit Prozeß-Ein/Ausgabedaten korrespondierenden Variablen der algorithmischen Programmanteile (PEARL-PROBLEM-Teil) immer ein gültiger Wert zur Verfügung, ohne die Verarbeitungsalgorithmen mit Reaktionen auf Störungen zu belasten.

```
SYSTEM;

TEMP : -)  ANIN560 * 8
 /*$    -)  ANIN571 * 5,
            CORR: TEMP=TEMP/3-16
       -)  ANIN572 * 5,
            CORR : CALL ADJUST (TEMP,2)
       -)   REP  : TEMP=1300,
            PLAUS: (HI=1600,LO=100,DELTA=10)*/;
```

Bild 6. Beispiel für die Formulierung von Prozeßdaten-Ersatzwegen und -Ersatzwerten (erweiterter SYSTEM-Teil)

3.4 Pilotimplementierung

Der MEHRRECHNER-PEARL-Compiler des IITB wurde auf der Grundlage eines transportablen PEARL-Compilers der Firma Werum, Lüneburg realisiert [13]. Während die SYSTEM-Teil-Erweiterungen in diesen Compiler eingearbeitet sind, ist er für LOAD-Teil-Anweisungen transparent; diese Anweisungen werden durch einen eigenen Übersetzer in Bedingungstabellen umgesetzt. Ein weiterer Übersetzer wandelt die Angaben des STATIONS-Teiles in strukturbeschreibende Tabellen um.

Das Ablaufsystem DISPOS ist in allen Prozessorstationen resident und umfaßt

- einen Kern mit Fehlerdiagnose und Zeitbearbeitung,
- ein PEARL-Betriebssystem,
- ein Netz-Betriebssystem und
- das Rekonfigurationssystem, bestehend aus den verteilten lokalen Betriebszustands-Beobachtern und dem globalen Lader.

Der globale Lader ist auch Bestandteil des zentralen Programmerzeugungssystems, das außerdem die genannten Übersetzer, ggf. unterschiedliche Codegeneratoren bei heterogenen Prozessoren sowie einen statischen Binder enthält. Die dynamisch auszuführenden Bindevorgänge bei der Verlagerung von Modulen besorgt das Rekonfigurationssystem.

4. Erfahrungen mit MEHRRECHNER-PEARL

Die Erfahrungen des IITB mit MEHRRECHNER-PEARL gründen sich auf mehrere Automatisierungsprojekte im Bereich der eisenschaffenden Industrie. Die Projekte umfassen die Automatisierung einer Tiefofenanlage [14], einer Anlage zur Zusammenstellung von Feinlegierungen, einem Datenverbundsystem sowie ein Programmsystem zur Geräuschanalyse und -klassifikation. Insgesamt wurden ca. 30.000 Zeilen Programmanweisungen und ca. 10.000 Zeilen Datendefinitionen mittels

MEHRRECHNER-PEARL erstellt. Alle Funktionen in den genannten Projekten konnten vollständig in PEARL implementiert werden, d. h. ohne den Einsatz von Assemblerprogrammen.

Bei den Erfahrungen mit MEHRRECHNER-PEARL sollen vor allem die global wirkenden Sprachelemente betrachtet werden, d. h.

- die globale Datenkommunikation allgemein,
- der globale E/A-Datenaustausch,
- globale Tasking-Operationen,
- globale Synchronisation sowie
- die Programmrekonfiguration.

4.1 Globale Datenkommunikation

Das Attribut GLOBAL in PEARL ist aufgrund seiner Semantik geeignet, eine beliebige Datenkommunikation zwischen MODULEs zu beschreiben, unabhängig von der Zuordnung der MODULEs zu Prozessoren. Der Compiler hat bei GLOBAL spezifizierten Objekten ein Ansprechen des Kommunikationssystems zu veranlassen, um bei Lage des korrespondierenden, GLOBAL deklarierten Datenobjektes in einer anderen Prozessorstation den Datentransport auszulösen. Dieses Verfahren ist unproblematisch bei direkter Objektanwahl. Bei indirekter Objektansprache, z. B. über REFERENCE-Variable oder IDENT-Parameter, ist jedoch nicht bekannt, ob das referenzierte Objekt GLOBAL deklariert oder spezifiziert ist, so daß hier immer ein Aufruf des Kommunikationssystems notwendig wäre und damit ein erheblicher Effizienzverlust entstünde, wenn das Objekt tatsächlich lokal ist.

Auch bei direkter Ansprache GLOBALer Objekte ist das Nachführen der Adreßtabellen mit der aktuellen Lage der Objekte aufwendig. Wir haben daher die implizierte Datenkommunikation über GLOBALe Datenobjekte nur dort zugelassen, wo diese Objekte ausschließlich direkt angesprochen werden und sich die Lage des deklarierten Objektes nicht durch Rekonfigurationsmaßnahmen ändern kann (z. B. Datenkommunikation zwischen Prozessorstationen und zentraler Wartenstation). Hier hat sich die Kommunikation über GLOBALe Objekte als sehr einfach nutzbares und effizientes Sprachmittel bewährt.

In allen anderen Fällen, insbesondere auch bei der Notwendigkeit eines synchronisierten Datenaustausches über gemeinsam genutzte GLOBALe Datenobjekte, hat sich das Botschaftenprinzip bewährt, das sich mit dem Mittel der PEARL-DATIONs realisieren läßt. Diese hauptspeicherversicherten Kommunikations-DATIONs benutzt der Programmierer in gleicher Weise wie E/A-DATIONs über READ/WRITE-Anweisungen. Die Kommunikations-DATIONs können nun mit Synchronisationseigenschaften ausgestattet werden (z. B. Rendevous-Zwang, gestützt durch time-out-Verfahren), die der Programmierer durch Verwendung solcher DATIONs explizit nutzt. Diese DATIONs können weiterhin Fehler des Kommunikationssystems behandeln und Synchronisationsdeadlocks aufgrund solcher Fehler vermeiden.

4.2 Globaler E/A-Datenaustausch

Der Sprachentwurf für MEHRRECHNER-PEARL sieht für den SYSTEM-Teil eine systemweit eindeutige Bezeichnung der E/A-Geräte vor, deren Zuordnung zu Prozessorstationen im STATIONS-Teil notiert wird. Eine zusätzliche Möglichkeit der Zuordnung ist die Angabe der Stationsnummern als Index in der Form

```
PROINT : INTRPT (n) m <-;
```

mit n = Stationsnummer und m = Anschlußnummer. Durch die so festliegende Zuordnung kann das Kommunikationssystem E/A-Daten, auch Interrupts, im System verteilen, wenn z.B. der Zugriff auf solche Daten wegen Ausfalls des Prozessors, der sie im Normalfall bearbeitet, über das Kommunikationssystem durch andere Prozessoren erfolgen muß. Diese Art der Ansprache nicht lokaler Prozeßdaten gestattet dem Programmierer ein von der aktuellen MODULE-Konfiguration unabhängiges Zugreifen zu Prozeßdaten und wird außerdem für die alternativen Datenwege im SYSTEM-Teil eingesetzt. Die Interruptreaktionszeiten im RDC-System liegen für lokale Interrupts bei ca. 600 µsec; bei nicht lokaler Interruptbearbeitung erhöhen sie sich auf ca. 1,5 msec. Nicht lokaler Zugriff auf analoge oder binäre Prozeßdaten erfordert ca. 2 msec. Damit hat sich der Zugriff auf beliebige Prozeßgrößen innerhalb einer RDC-Konfiguration als effizientes Hilfsmittel erwiesen, das für Fehlertoleranzmaßnahmen ebenso vorteilhaft einsetzbar ist, wie es die Programmierung erleichtert.

4.3 Globale Task-Steuerung

Die globale Task-Steuerung mit Ausnahme der Synchronisation durch SEMAphore, bzw. BOLTs, ist in MEHRRECHNER-PEARL unter Nutzung der gleichen Mechanismen implementierbar, wie für die implizite Datenkommunikation mittels GLOBAL deklarierter bzw. spezifizierter Datenobjekte. Das globale Datenobjekt ist hier der Task-Kontrollblock einer GLOBAL deklarierten Task, dessen aktuelle Lage im Mehrrechnersystem - insbesondere auch bei Rekonfigurationsvorgängen - bekannt sein muß. Es gelten daher dieselben Einschränkungen, wie für den impliziten globalen Datenaustausch in Abschnitt 4.1 genannt.

Abgesehen von globaler Task-Synchronisation, über die im nächsten Abschnitt berichtet wird, trat bisher in keiner der IITB-Anwendungen von MEHRRECHNER-PEARL die Notwendigkeit MODULE-übergreifender Task-Steuerung auf. Aus diesem Grunde und wegen der genannten Einschränkungen, ist die globale Task-Steuerung im MEHRRECHNER-PEARL des IITB nicht implementiert. Damit ist das Attribut GLOBAL bei TASKs nicht zulässig, sobald die Möglichkeit besteht, daß steuernde und gesteuerte Task in verschiedenen Prozessorstationen liegen.

Es ist jedoch möglich, über Botschaften - vermittelt über Kommunikations-DATIONs (siehe Abschnitt 4.1) - eine Task-Steuerung aufzubauen, indem lokale Task-Steueranweisungen von solchen Botschaften abhängig gemacht werden.

4.4 Globale Synchronisation

PEARL stellt für Synchronisationsaufgaben die Objekte SEMAphor und BOLT mit ihren spezifischen Operationen zur Verfügung. Sie sind insbesondere beim koordinierten Zugriff auf Daten und Betriebsmittel, zur Herstellung von mutual-exclusion-Bedingungen von wesentlicher

Bedeutung und haben sich im Standard-PEARL bei Berücksichtigung ihrer konzeptuellen Schwächen bewährt.

Im Mehrrechnersystem mit explizitem und potentiell fehlergefährdetem Kommunikationssystem haben sich SEMAphore für globale Synchronisationsaufgaben als nicht brauchbar erwiesen. Auch aufwendige Kommunikationsprotokolle können den Verlust oder die Verfälschung von Datentelegrammen nicht absolut sicher verhindern; solche Fehler würden aber bei Synchronisationsoperationen zu unkontrollierbaren Systemzuständen, z.B. dead-locks, führen. Ein zuverlässiges Hilfsmittel stellt auch hier die Kommunikation über DATIONs mit geeigneten Synchronisationseigenschaften (z.B. Rendezvous-Technik) dar.

4.5 Programm-Konfiguration und -Rekonfiguration

Der Einsatz der betriebszustandsgesteuerten Programm-Konfiguration und -Refonfiguration, formulierbar mittels des LOAD-Teils, hat sich als Fehlertoleranzmaßnahme bewährt. Messungen für den Fall residenter Ersatz-MODULEs (RESIDENT im LOAD-Teil) ergaben Rekonfigurationszeiten von ca. 20 msec. In dieser Zeit liegt das Erkennen eines Rekonfigurationsfalles sowie das kontrollierte Terminieren bzw. Aktivieren der betroffenen MODULEs (Tasks). Die für das Wiederanlaufen von Ersatz-MODULEs erforderlichen Prozeßdaten ermittelt die Start-TASK der MODULEs entweder selbst aus dem aktuellen Zustand des Prozesses oder, falls dies nicht möglich ist, aus Ersatzdatenbereichen, die in residenten Modulen liegen und über globale Datenkommunikation gemäß Abschnitt 4.1 laufend mit aktuellen Prozeßdaten gefüllt werden. Diese Möglichkeit der Aufwandsoptimierung für Fehlertoleranzmaßnahmen hat sich als wirkungsvoll erwiesen.

5. Notwendige Folgerungen

Das MEHRRECHNER-PEARL-Programmiersystem des IITB hat sich bei den bisher durchgeführten Projekten bewährt. Die Effizienz der Programmsysteme genügte allen Anforderungen. Die verfügbaren Fehlertoleranzmaßnahmen ergaben hohe Gesamt-Verfügbarkeiten der Automatisierungssysteme von ca. 99,96 % [15].

Die Erfahrungen mit der Implementation und den Betriebseigenschaften einiger Sprachelemente in Mehrrechnersystemen gibt Anlaß zu Restriktionen und konzeptionellen Ergänzungen. Dies gilt insbesondere für globale Beziehungen zwischen rekonfigurierbaren MODULEs.

Die Verwendung des GLOBAL-Attributes ist daher uneingeschränkt nur innerhalb von fest miteinander verbundenen MODULEs erlaubt, die gemeinsam rekonfiguriert werden und immer in einem zusammenhängenden Speicherbereich ablaufen. Nur in diesen Fällen können GLOBAL-Beziehungen statisch verbunden werden und im Rahmen der Fehlersicherheit eines einzelnen Prozessors sicher ablaufen bei hoher Effizienz. Als Konsequenz aus dieser Situation ergibt sich die Zuordnung eines LOAD-Teils zu mehreren PEARL-MODULEs, die zusammen eine rekonfigurierbare Einheit darstellen.

Kommunikation zwischen solchen Einheiten sollte vorwiegend über Kommunikations-DATIONs mit geeigneten Synchronisations- und Fehlerreaktionseigenschaften erfolgen, die im Programmiersystem definiert sind (SYSTEM-Teil, STATIONS-Teil). Nur wenn in Sonderfällen eine implizite, GLOBALe Objektansprache zwischen rekonfigurierbaren Einheiten sicher und effizient implementierbar ist, sollte sie erlaubt und durch ein spezielles, neues Attribut kennzeichenbar sein. In allen anderen Fällen kann dann das Programmerzeugungssystem die Verwendung GLOBALer Objektansprache zwischen rekonfigurierbaren Einheiten verhindern. Dies muß insbesondere für Synchronisationsobjekte (SEMA, BOLT) und ggf. TASKs und PROCeduren gelten.

Die Arbeiten im IITB haben gezeigt, daß ein quellsprachbezogenes, auf Mehrrechnersysteme ausgelegtes Testsystem für MEHRRECHNER-PEARL notwendig ist. Es muß möglich sein, von einer mit entsprechender Peripherie ausgerüsteten Teststation aus in den Teilrechnern (Prozessorstationen) zu testen, unter Nutzung des Kommunikationssystems, auch im Falle einer Rekonfiguration des Programmsystems. Ein solches Testsystem ist im IITB in Arbeit.

Die dargestellten Einsatzerfahrungen und deren Konsequenzen sowie Überlegungen zu den Möglichkeiten einer gestuften Erfüllung der Anforderungen an MEHRRECHNER-PEARL aus den verschiedenen Mehrrechnersystemen (siehe Abschnitt 2), sollten bald in Arbeiten zur Normung von MEHRRECHNER-PEARL einfließen. Die vorliegenden Erfahrungen bei den mit PEARL für Mehrrechnersysteme befaßten Stellen rechtfertigen die Aufnahme der Normungsarbeit.

MEHRRECHNER-PEARL trägt dazu bei, die Programmierung von Mehrrechnersystemen sicherer und wirtschaftlicher zu machen; die Einsatzvorteile von PEARL werden damit auch für Mehrrechnersysteme als durchgängiges Programmiersystem verfügbar.

6. Literatur

[1] Steusloff, H.: Strukturen von Automatisierungssystemen und ihre Konsequenzen für Programmiersprachen. Regelungstechn. Praxis (rtp), 1979, Heft 6, S. 153-158, Heft 7, S. 188-192; Oldenbourg-Verlag, München.

[2] Stein, R.: Adaptive Datenübertragung in einem dezentralen Prozeßautomatisierungssystem. Regelungstechn. Praxis (rtp), 1980, Heft 5, S. 155-158; Oldenbourg-Verlag, München.

[3] Hügle, W.: Bildschirmdialog zum Konfigurieren und Parametrieren von Prozeßautomatisierungssystemen. Regelungstechn. Praxis (rtp), 1980, Heft 4, S. 115-120; Oldenbourg-Verlag, München.

[4] Büsing, W.: Dezentrale Prozeßautomatisierungssysteme: Anforderungen und Schnittstellen. Regelungstechn. Praxis (rtp), 1980, Heft 2, S. 37-42.

[5] Borsi, L.; Pavlik, E.: Konzepte und Strukturen dezentraler Prozeßautomatisierungssysteme. Regelungstechn. Praxis (rtp), 1980, Heft 9, S. 302-309; Oldenbourg-Verlag, München.

[6] Heger, D.: Kommunikationsverfahren für Sammelleitungssysteme und deren Leistungsbeschreibung. IITB-Mitteilungen 1978, S. 41-48; Fraunhofer-Institut für Informations- und Datenverarbeitung, Karlsruhe.

[7] Syrbe, M.: Über die Beschreibung fehlertoleranter Systeme. Regelungstechnik 1980, Heft 9, S. 280-289; Oldenbourg-Verlag, München.

[8] Heger, D.; Steusloff, H.; Syrbe, M.: Echtzeitrechnersystem mit verteilten Mikroprozessoren. Forschungsbericht DV 79-01, Datenverarbeitung; Bundesministerium für Forschung und Technologie (BMFT), Bonn.

[9] Bonn, G.; Heil, W.; Kippe, J.; Saenger, F.: Selbsttest und Selbstkonfiguration von Prozeßrechnersystemen am Beispiel des RDC-Systems. IITB-Mitteilungen 1979, S. 47-54; Fraunhofer-Institut für Informations- und Datenverarbeitung, Karlsruhe.

[10] Schneider, H.-J.: PEARL-Softwaresystem für gekoppelte Klein- und Mikrorechner. PEARL-Rundschau 1980, Heft 3 (vorliegendes Heft); PEARL-Verein, Düsseldorf.

[11] Steusloff, H.: Zur Programmierung von räumlich verteilten, dezentralen Prozeßrechensystemen. Dissertation 1977, Fakultät für Informatik, Universität (TH) Karlsruhe.

[12] Steusloff, H.: Programming Distributed Computer Systems with Higher Level Languages. Distributed Computer Control Systems, Pergamon Press, Oxford New York; 1980, S. 39-50.

[13] Werum, W.; Windauer, H.: PEARL, Process and Experiment Automation Realtime Language. Programm Angewandte Information, 1978, Vieweg-Verlag, Braunschweig.

[14] Bonn, G.; Lorenz, L.: Eignung von MEHRRECHNER-PEARL zur Programmierung paralleler Prozesse; Erfahrungen und Folgerungen. Fraunhofer-Institut für Informations- und Datenverarbeitung, Karlsruhe; in Vorbereitung.

[15] Heger, D.: Dezentrales Mikroprozessorsystem mit Farbbildschirmen zentral geführt. Fachberichte Messen, Steuern, Regeln, Band 5, 1980, S.503-516; Springer-Verlag, Heidelberg.

[16] DIN 66253 Teil 1: Programmiersprache PEARL, Basic PEARL. Normentwurf, Juni 1978, Deutsches Institut für Normung (DIN), Berlin.

[17] DIN 66253 Teil 2: Programmiersprache PEARL, Full PEARL. Entwurf zur Normvorlage, April 1980, Deutsches Institut für Normung (DIN), Berlin.