BENUTZERHILFSMITTEL FÜR PEARL IM EINSATZ AUF MEHRRECHNER-SYSTEMEN

E. Fahr / G. Barbian Dornier System

U. Bügel / G. Bonn IITB

Zusammenfassung

Bei der Programmierung von Mehrrechner-Systemen in der Prozessdatenverarbeitung fehlen heute bei fast allen bekannten Programmiersprachen geeignete Sprachmittel und unterstützende Benutzerhilfsmittel. Dies gilt auch für PEARL nach DIN 66253.
Im folgenden wird ein Konzept vorgestellt, das synchrone und asynchrone Mechanismen des Botschaftenaustausches auf Basis eines Port-Konzeptes verwendet. Weiterhin werden Beschreibungselemente für Hardware- und Software-Konfiguration eingeführt, die in einer von den Ablaufteilen separierten Liste zusammengefaßt werden. Die Hardware-Konfigurationsbeschreibung umfaßt die Prozessorbeschreibung, die Peripheriebeschreibung (Systemteil) und die Beschreibung der physikalischen Verbindungen zwischen den Prozessorstationen. Die Software-Konfigurationsbeschreibung besteht aus der Initialkonfiguration, der logischen Verbindungsstruktur zwischen den Port's und der Rekonfigurationsbeschreibung. Als rekonfigurierbare Einheiten werden Modulgruppen (Collection's) eingeführt, die miteinander ausschließlich durch Botschaftenaustausch über Port's kommunizieren können. Mit dem vorgestellten Konzept können Systeme realisiert werden, die das Austauschen von Softwareteilen bei weiterlaufendem Gesamtsystem erlauben.

Schlüsselwörter:

Mehrrechner-Systeme, verteilte Programmsysteme, Botschaftenaustausch, Port-Konzept, Konfigurationsbeschreibung

Summary

Today almost all well-known programming languages don't provide suitable constructs and supporting useraids for the programming of distributed real-time systems. This is also valid for PEARL (DIN 66253). In the following, a concept will be shown, which uses synchronous and asynchronous mechanism for the message transfers based on the port-concept. Furtheron description elements for hardware- and software-configuration will be introduced, which are concentrated in a list separated from the program themselve. The processor description, the peripheral description (system part) and the description of the physical connections between the processor stations, are part of the hardware configuration description. Components of the software configuration description are the initial configuration, the logical connection structure between the ports and the reconfiguration description. Groups of modules (collections) for dynamic configuration will be shown, which exclusively communicate by message-transfer using ports. With the introduced concept systems can be realized to permit on-line changes.

Keywords:

multi computer systems, distributed program systems, message-transfer, port-concept, configuration description

1. Randbedingungen zum Projekt

Im wehrtechnischen Bereich werden in Zukunft verstärkt höhere Programmiersprachen eingesetzt, wobei eine Beschränkung auf wenige Sprachen - PEARL und ADA - notwendig ist. Parallel zu dieser Entwicklung zeichnet sich in modernen Waffen- und Führungssystemen ein vermehrter Einsatz von Mehrrechnersystemen ab.
Es ist naheliegend, daß auch die Programmierung von Mehrrechner-Systemen durch höhere Programmiersprachen erfolgen muß, um die heute vorliegenden Anforderungen bezüglich Wirtschaftlichkeit und Zuverlässigkeit der Programmsysteme zu erfüllen.
Die derzeit hierfür besonders geeignete Echtzeitprogrammiersprache ist PEARL.
Jedoch fehlen auch bei PEARL nach DIN 66253 gewisse Sprach- und Benutzerhilfsmittel für den Einsatz auf Mehrrechner-Systemen.
Aus diesem Grunde führt die Dornier System GmbH Friedrichshafen in Zusammenarbeit mit dem IITB (Fraunhofer-Institut für Informations- und Datenverarbeitung) Karlsruhe im Auftrag des BWB eine Studie über "Benutzerhilfsmittel für PEARL im Einsatz auf Mehrrechner-Systemen" durch. Ziel der Studie ist das Aufzeigen von Wegen, wie anhand vorliegender Erfahrungen von Dornier System und dem IITB ein einheitliches, in seinen Leistungen und Eigenschaften abgestuftes, an verschiedene Einsatzbereiche anpaßbares Mehrrechner-PEARL entworfen und erstellt werden kann. Dabei soll durch Aufzeigen abgestufter Ergänzungen zu PEARL nach DIN 66253 unter Berücksichtigung der vorhandenen Implementationen das Hilfsmittel Mehrrechner-PEARL für den Verteidigungsbereich allgemein verfügbar gemacht werden. Dem Anwender sollen Hilfsmittel bzw. Werkzeuge zur Verfügung gestellt werden, die es ihm erlauben, die Beschreibung der Systemkonfiguration sowie die Steuerung der Rekonfiguration des Gesamtsystems in Abhängigkeit vom Betriebszustand des Rechnersystems durchzuführen. Ebenso muß es möglich sein, die Prozeßkommunikation und die Synchronisation von Programmen in verschiedenen Prozessoren eindeutig zu beschreiben bzw. zu steuern.
Die Ergebnisse der Studie werden in Form einer Sprachbeschreibung und einer Anwendungsanleitung (Handbuch), die auch zur Vorlage bei Normungsgremien gedacht ist, zur Verfügung gestellt. Der vorliegende Beitrag gibt den gegenwärtigen Stand (Okt. '84) der Studie wieder. Bei allen Arbeiten sind, soweit verfügbar, einschlägige Ergebnisse anderer Forschungs- und Entwicklungsstellen zu berücksichtigen. Ebenso wurde bei den Studienarbeiten ein reger Meinungsaustausch mit der Hochschule der Bundeswehr, H. Prof. Dr. Rzehak, geführt, dem wir an dieser Stelle danken möchten. Für das vom PEARL-Verein entgegengebrachte Interesse an dieser Studie, sowie für die vielen Anregungen des zum ersten Mal zusammengetretenen PEARL-Sprachpflegeausschusses möchten wir ebenfalls an dieser Stelle herzlich danken.

2. Einsatzerfahrungen und Konsequenzen

2.1 Erfahrungen mit dem Dornier-Mehrrechner-PEARL

Für die Konzeption von PEARL-Spracherweiterungen für verteilte Systeme wurde bei Dornier System von den Anforderungen aus militärischen Anwendungen (z.B. Bordrechner, Führungssysteme, Feuerleitrechner) ausgegangen. Bei diesen Systemen handelt es sich um komplexe, räumlich jedoch wenig ausgedehnte Gerätesysteme (embedded systems). Die räumliche Nähe der Verarbeitungseinheiten erlaubt die Verwendung schneller paralleler Busverbindungen für die Inter-Prozessor-Kommunikation. Um die strengen Echtzeitanforderungen in solchen Systemen zu erfüllen, wurde auf komplizierte Kommunikationsverfahren verzichtet und eine lose Kopplung zwischen den einzelnen Verarbeitungseinheiten gewählt. Der softwaretechnische Ansatz für das Arbeiten mit verteilten Systemen war das Einführen "netzglobaler" Größen, die im Programm durch das Attribut "GLOBAL NET" gekennzeichnet werden.

Als netzglobal deklariert und spezifiziert werden können:

- Variable,
  Netzglobale Variable wurden nach dem Prinzip "send if changed" implementiert. Bei diesem Ansatz werden netzglobale Variable in jedem Prozessor gehalten. Lesezugriffe auf diese Variablen erfolgen ohne Aktivierung der Netzkommunikation. Bei einer Wertänderung einer netzglobalen Variable wird die Netzkommunikation automatisch aktiviert und die Änderung bei allen Prozessoren eingetragen.

- Tasks,
  Der Task-Control-Block einer netzglobalen Task wird nur in dem Prozessor geführt, in dem die Task auch deklariert ist. Auf netzglobal deklarierten Task's können von allen anderen Prozessoren aus alle Taskinganweisungen aufgerufen werden.

- SEMA's,
  Netzglobale Sema's werden, wie netzglobale Variable auch, in allen Prozessoren gehalten. Von allen Prozessoren aus können auf Sema's die Funktionen RELEASE und REQUEST angewandt werden.

Beim Einsatz in Projekten hat sich das Mehrrechner-PEARL von DORNIER bewährt. Das Echtzeitverhalten wurde durch die Netzkommunikation nur unwesentlich beeinflußt; es entstanden dadurch keine zeitlichen Probleme. Die Spracherweiterungen und die zur Verfügung gestellten Hilfsmittel konnten von den Anwendern zur Lösung des Problems effizient genutzt werden. Es zeigten sich beim Arbeiten mit diesem System aber auch einige Punkte, die berücksichtigt werden mußten, bzw. die neu überdacht werden sollten.
Aus Speicherplatzgründen und um das Echtzeitverhalten so wenig wie möglich zu beeinflussen, wurden bei Anwendungsprojekten netzglobale Variable und Tasks sehr restriktiv gehandhabt.
Netzglobale SEMA's haben sich für die Synchronisation von Tasks, die auf unterschiedlichen Prozessoren laufen, als nicht geeignet erwiesen. Hier sollten Lösungsansätze erarbeitet und implementiert werden, die eine netzglobale Synchronisation erlauben. Ein möglicher Ansatz bietet hier das Botschaftenkonzept.
Da die Konfigurationsbeschreibung bei jedem Binderlauf vom System im Dialog neu angefordert wird, können durch Fehleingaben falsche Konfigurationen festgelegt werden. Man sollte die Möglichkeit schaffen, daß eine korrekte Beschreibung in einer Liste abgespeichert wird und später beliebig oft wieder bearbeitet werden kann. Diese Konfigurationsbeschreibung sollte in einer PEARL-ähnlichen Sprache erfolgen. Bedingt durch die Tatsache, daß bei Bordrechnern die Programme auf PROM's gespeichert sind, waren die Voraussetzungen für eine dynamische Rekonfiguration nicht erfüllt.
Bei Führungs- und Feuerleitsystemen besteht jedoch die Anforderung der dynamischen Rekonfigurierbarkeit. Die Komplexität dieser Systeme macht es notwendig, eine Geräte- und Verbindungsbeschreibung einzuführen.

2.2 Erfahrungen mit dem Mehrrechner-PEARL des IITB

Zur Bewältigung der programmtechnischen Komplexität verteilter fehlertoleranter Mehrrechnersysteme für den Einsatz in der Prozeßautomatisierung wurde im IITB die Sprache MEHRRECHER-PEARL definiert, die zugehörigen Programmerzeugungs- und Ablaufsysteme entwickelt und mit dem verteilten Prozeßautomatisierungssystem RDC (HE 79) in mehreren Anwendungen in der Stahlindustrie erfolgeich eingesetzt.

Für die Verteilung eines PEARL-Programms in einem Mehrrechnersystem wurde der Modul als kleinste (konfigurierbare) Einheit gewählt. Durch die modulweise Aufteilung werden Schnittstellen zwischen den in verschiedenen Rechnerstationen lokalisierten Modulen auf die mit dem Attribut "GLOBAL" gekennzeichneten PEARL-Objekte beschränkt. Die Adreßbezüge zu globalen Objekten müssen dynamisch hergestellt werden, der Zugriff erfolgt über das Kommunikationssystem. Darüberhinaus wurde ein Botschaftensystem für taskspezifischen Datenaustausch mit den Synchronisationsvarianten "Rendezvous" und "not wait send" eingesetzt (Bo 80). Eine sprachliche Integration dieser Mechanismen mittels DATIONs war geplant. Über diese zunächst skizzierte Abbildung von Standard-PEARL auf Mehrrechnersysteme hinaus besteht jedoch der Wunsch, deren Redundanz zur Erhöhung der Ausfallsicherheit zu nutzen und dafür adäquate programmiersprachliche Ausdrucksmöglichkeiten bereitzustellen.
Dafür schlägt Steusloff (St 77) eine erweiterte Strukturbeschreibung aus folgenden drei Komponenten vor:

- Ein Stationsteil beschreibt für das Programmerzeugungssystem relevante gerätetechnische Eigenschaften der verschiedenen, in einem heterogenen Mehrrechnersystem verbundenen Rechnerstationen und führt Bezeichner für deren Betriebszustand kennzeichnende Statusvariable ein.

- Ein Ladeteil, der zu jedem Modul gehört, gibt an, in welche Stationen der Modul in Abhängigkeit von welchen Statusbedingungen (Werte der o.g. Statusvariablen) zu laden ist und erlaubt somit die übersichtliche Formulierung von Rekonfigurationsstrategien, z.B. im Sinne einer funktionsbeteiligten Redundanz.

- Der auch in Standard-PEARL enthaltene Systemteil ist erweitert um Möglichkeiten zur Beschreibung alternativer Datenwege.

Die Betriebserfahrungen mit RDC-System und MEHRRECHNER-PEARL beziehen sich insbesondere auf den Betrieb einer Pilotanlage zur Steuerung von 28 Tieföfen in einem Blockwalzwerk der Fa. THYSSEN AG (St. 81). Redundanz wird in der Weise erreicht, daß jeweils zwei Stationen so zusammenarbeiten, daß bei Ausfall einer der beiden die jeweilige Partnerstation die Funktion der ausgefallenen mit übernimmt. In einer Reihe weiterer Aufgaben der Prozeßautomatisierung wie
- Automatisierung der Feinlegierungsprozesse in einem Stahlwerk,

- Geräuschanalyse und -klassifikation,
- Erdgasvertragsüberwachung, Gasverbundsystem,
- Roboterregelung,

wurde ausschließlich MEHRRECHNER-PEARL für die Anwendungsprogrammierung erfolgreich eingesetzt.

Die Betrachungsweise der Einsatzerfahrungen konzentriert sich auf die effiziente Implementierbarkeit und erzielbare Robustheit (netz-)globaler PEARL-Objekte.

- Globale Prozeß-E/A
  Das Betreiben nicht stationslokaler Prozeß-E/A-Werke einschließlich Interpretation von Interrupts läßt sich sehr transparent formulieren und mit den bereitgestellten Protokolldiensten des RDC-Systems äußerst effizient implementieren.
- Globale Datenobjekte
  Die Kommunikation zwischen Teilnehmern über im Mehrrechnersystem verteilte globale Datenobjekte führt zu einer Reihe von Implementationsschwierigkeiten, insbesondere für Referenzvariable und IDENT-Prozedurparameter.
- Taskkooperation
  Die globale Steuerung mittels Taskkooperation stellt keine besonderen Anforderungen an das Ablaufsystem. Ein Vergleich "verteilte Semaphore" versus "Rendezvousmanöver" fiel bezüglich erzielbarer Robustheit eindeutig zugunsten des Botschaftenprinzips aus. Die unstrukturierte Nutzbarkeit von Semaphoren (nicht paarweise) verhindert praktisch eine fehlertolerante Implementation.

Aus diesen Betriebserfahrungen heraus wird in (Bo 81) gefordert:

"Zusammenfassung mehrerer Module zu einer Modulgruppe, die als nur geschlossen lade- und rekonfigurierbare Einheit nur einen Ladeteil besitzt und innerhalb derer bezüglich des GLOBAL-Attributs die Regeln von Standard-PEARL gelten. Zwischen solchen Modulgruppen existieren Verbindungen nur noch über Systemteil-definierte Objekte einschließlich der Kommunikations-DATIONs."

3. Beschreibungsmittel von Mehrrechner-PEARL

In den heute bekannten Prozeßprogrammiersprachen stehen Beschreibungsmittel für algorithmische Abläufe in vielfältiger Art zur Verfügung. PEARL ist den Sprachen zuzuordnen, die Sprachelemente zur Beschreibung von Strukturmerkmalen zentraler Prozeßautomatisierungssysteme zur Verfügung stellen. Für verteilte Prozeßrechner-Systeme fehlen jedoch auch bei PEARL nach DIN 66253 solche Beschreibungsmittel völlig. Auf den folgenden Seiten sollen nun Beschreibungsmittel für

- die Gerätekonfiguration
- die Programmkonfiguration
- das Botschaftenkonezpt

vorgestellt werden.

3.1 Gerätekonfiguration

Der Hardware-Anteil eines Mehrrechner-Systems besteht aus:

- mehreren, möglicherweise unterschiedlichen Prozessoren,
- einer evtl. umfangreichen Peripherie,
- und den Verbindungen, die die Prozessoren untereinander bzw. mit den Peripherie-Geräten aufweisen.

Für diese Elemente benötigen wir folgende Beschreibungsmittel:

- Prozessorbeschreibung
- Peripheriebeschreibung
- Beschreibung der physikalischen Verbindungen

Während die Prozessorbeschreibung und die im Systemteil enthaltene Peripheriebeschreibung prozessorlokale Informationen enthalten, besitzt die Verbindungsbeschreibung prozessorübergreifenden Charakter. Deshalb soll hier der Ansatz für eine Zusammenfassung aller notwendigen Beschreibungen in einer eigenen übergeordneten Liste ausgebaut werden. Diese Liste sollte als passives Beschreibungsteil (Textfile), das über dem Programm steht, gesehen werden. Um dem Programmierer den Aufbau der Liste zu erleichtern, sollte sie sich nach Möglichkeit an der PEARL-Syntax orientieren.

Sie ist aber nicht Bestandteil des eigentlichen Programmes.

Da sowohl der Compiler als auch der Linker/Lader auf die darin enthaltene Information zugreifen, sollte dieser Beschreibungsteil per Programm in eine für die Weiterverarbeitung geeignete Form gebracht werden. Dabei kann gleichzeitig die Vollständigkeit der Beschreibung überprüft werden.

3.1.1 Prozessorbeschreibung

Da wir davon ausgehen müssen, daß wir ein Netzwerk mit unterschiedlichen Prozessoren (inhomogenes System) haben und folglich die Module auf die entsprechende Zielmaschine (Station) angepaßt übersetzt werden müssen, brauchen wir gewisse Informationen über die Zielmaschine. So braucht der verwendete Übersetzer (Compiler) zu jedem Modul entsprechende Angaben über Prozessor-Typ, Verarbeitungsbreite und Befehlssatz. In einem fehlertoleranten Mehrrechner-System werden weiterhin Bezeichner für Betriebszustände benötigt, die vom Rekonfigurations-System ausgewertet werden. Diese Informationen sollten, in einer Stationsbeschreibung zusammengefaßt, in die Liste integriert sein.
Ein Beispiel der sich hieraus ergebenden Sprachmittel zeigt Bild 1.

```
STATIONS;
        NAME:       STA1;
        PROCTYP:    8086;
        WORKSTORE:  8K-64K;
        STATEID:    (PR: '188'84: '01FF'84);
        ...
        ..
        NAME:       STA2;
        ...
        ..
STAEND;
```

Bild 1. Prozessorbeschreibung

3.1.2 Peripheriebeschreibung

Beim Übergang zu Mehrrechner-Systemen kann zur Beschreibung der Peripherie das Systemteil weiterhin verwendet werden. Aus Gründen der Übersichtlichkeit und der Programmiersicherheit empfiehlt sich jedoch die Auslagerung der Systemteile aus den Moduln und die Verwendung nur eines Systemteils pro Station. Dieser kann dann der Prozessorbeschreibung zugeordnet werden. Eine Erweiterung des Systemteils bei der Ansprache von E/A-Geräten auf abgesetzten Stationen ist möglich. Dies kann durch Verwendung des Stationsidentifikators (Bild 1) in den Systemteil-Anweisungen geschehen. Eine Notwendigkeit dieser Erweiterung ist z.B. bei der Verlagerung von Programmteilen (Rekonfiguration, Kapitel 3.2.2) gegeben.

3.1.3 Verbindungsbeschreibung

Ein Mehrrechner-System besteht aus mehreren Stationen (Knoten), die durch Datenleitungen miteinander verbunden sind. Je nach Distanz und gewünschter Übertragungsgeschwindigkeit sind die Verbindungen unterschiedlich (V24, HDLC, IEC-BUS, Ethernet, etc.). Nicht jeder Knoten im Netz muß direkt mit allen anderen verbunden, jedoch muß jeder erreichbar sein. Die Kopplung kann durch dedizierte Leitungen direkt oder durch bus- und ringartige Verbindungen, bei denen die Zugriffskonflikte durch ein Kopplungssystem geregelt werden, erfolgen. Sämtliche Mischformen dieser Topologien treten in der Praxis auf. Um in einem derart vermaschten Netz einen Datenaustausch durchführen zu können, braucht das System exakte Informationen zur Wegfindung. Die einzelnen Verbindungen sollten zur besseren Unterscheidung mittels einer Kennung (Name) ansprechbar sein. Das folgende Kapitel befaßt sich mit Ansätzen, die Beschreibung der Verbindungen unter den einzelnen Stationen in einer PEARL-orientierten Syntax durchzuführen.
Schauen wir uns dazu ein Beispiel einer Verbindungsbeschreibung in PEARL an:

```
SYSTEM;

SCHALTER(15:31): <- DIGEIN(1) * (0:15);
```

In diesem Fall werden dem im Programm ansprechbaren Feld 'SCHALTER(15:31)' die Signale '0:15' des Eingabegerätes 'DIGEIN(1)' zugeordnet.
In dieser Form läßt sich auch eine Verbindung zwischen zwei Stationen beschreiben.

```
LINE#1:  STA1 --> K1 * ... * Kn * STA2;
```

Hier ist mit 'LINE#1' die bidirektionale Verbindung der Station 'STA1' über die Koppelglieder 'K1 - Kn' nach Station 'STA2' definiert.

Mit diesem Ansatz können wir zwar jetzt jedes beliebige Netzwerk beschreiben, müssen aber feststellen, daß die Beschreibung jeder einzelnen Leitung bei einem komplexen Netzwerk zu aufwendig ist. Da ein Großteil der Netzwerke auf einer Bus-, Ring- oder Sternstruktur basiert, werden für diese Verbindungsformen vereinfachte Beschreibungsmittel benötigt. Diese Beschreibung sollte sich aus einem Namen, einem Schlüsselwort für den Typ der Verbindung und einer Liste der angeschlossenen Prozessoren zusammensetzen. Auf diese

Weise ist ein einfacher Rückschluß aus dem Quelltext auf die zugrundeliegende Struktur möglich.

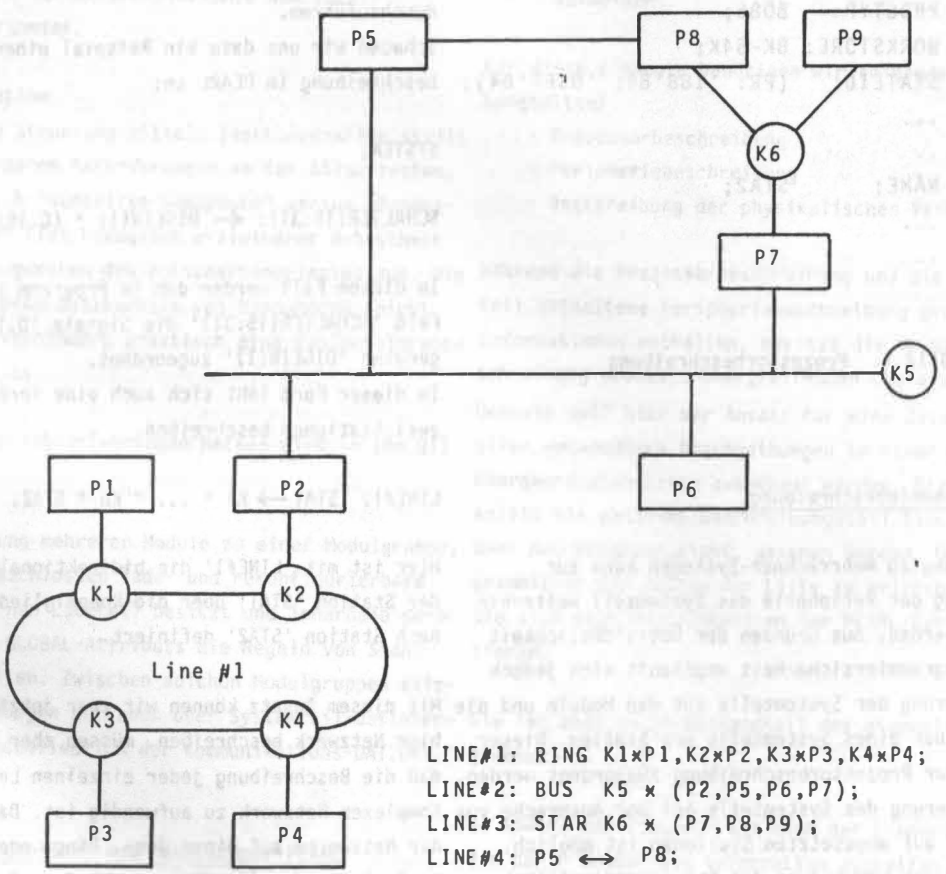
Durch Einführen der Abkürzungen BUS, STAR und RING vereinfacht sich die Beschreibung

- der busförmigen Verbindung auf
  LINE#1: BUS KO * (P1,P2,P3,P4);
- der sternförmigen Verbindung auf
  LINE#2: STAR KO * (P1,P2,P3,P4);
- der ringförmigen Verbindung auf
  LINE#3: RING K1*P1,K2*P2,K3*P3,K4*P4;

Durch Verknüpfen dieser Abkürzungen und die Benutzung der allgemeinen Leitungsbeschreibung lassen sich auch komplexe Netzwerke einfach beschreiben (Bild 2).

3.2 Programmkonfiguration

In Kapitel 2.2 wurde aus den Einsatzerfahrungen heraus die Einführung von Modulgruppen, im folgenden "Collections" genannt, als konfigurierbare Programmeinheit gefordert. Auf diese Weise kann die notwendige Reduktion der Schnittstellen für konfigurierbare Einheiten erreicht werden. Dies wäre prinzipiell auch bei der Verwendung von Moduln möglich, indem die Inter-Modul-Bezüge mit den Ladeanweisungen in Relation gesetzt und verbotene Bezüge von einem Compiler erkannt werden. Eine solche Lösung ist jedoch für den Programmierer zu wenig durchschaubar und erfordert einen hohen Implementierungsaufwand. Die Schnittstellen von Collections werden in Kapitel 3.2.1 beschrieben.




```
LINE#1: RING K1*P1,K2*P2,K3*P3,K4*P4;
LINE#2: BUS  K5 * (P2,P5,P6,P7);
LINE#3: STAR K6 * (P7,P8,P9);
LINE#4: P5 <-> P8;
```

Bild 2. Verbindungsbeschreibung

Zur Beschreibung eines verteilten Programmsystems sind Konfigurationsbeschreibungen (LeB 82), (Kra 83), (Mag 83) ein geeignetes Hilfsmittel. Mit dem Ladeteil (Kap. 2.2) wurde bereits ein Weg in diese Richtung beschritten. Ein Überblick über alle Aktivitäten des Systems zur Konfiguration, d.h. der initialen, aktuellen und - bei Ausfall von Teilsystemen - reorganisierten Zuordnung von Collections zu Teilstationen, sowie das Einrichten logischer Verbindungen zwischen Collections wird durch Erweiterung des Ladeteils zu einem "Configuration-Teil" ermöglicht. Der Configuration-Teil ist eine vom verteilten Programmsystem separierte Beschreibungsliste. Seine Anweisungen werden nicht in ausführbaren Code umgesetzt, sondern dienen der Ansteuerung von Systemprogrammen des Programmentwicklungs- und des Programmablaufsystems. Deshalb werden Teile von ihnen in Laufzeittabellen (Konfigurations-/Rekonfigurationstabellen, kurz: K/R-Tabellen, Portlisten und Verbindungen) umgesetzt (Kap. 4). Die Sprachelemente zur Programmkonfigurationsbeschreibung werden nun im folgenden eingeführt.

3.2.1 Collection-Schnittstellen: Ports

Die Intertaskkommunikation innerhalb von Collections erfolgt in effizienter Weise über globale Daten und kann mit Hilfe von Semaphoren und Bolts synchronisiert werden. Inter-Collection-Kommunikation soll ausschließlich über Botschaften erfolgen. Die Adressierung der Botschaften durch direkte Ansprache von Kommunikationsobjekten (z.B. TASKs oder DATIONs) in fremden Collections schränkt die Konfigurationsflexibilität unnötig ein. Deshalb müssen Collections Interfaces besitzen, die keine fest vorgegebenen Verbindungen implizieren. Auf diese Weise sind Collections in wechselnder Umgebung einsetzbar: zur Ansprache von Anwendungsfunktionen, die durch eine Collection bereitgestellt werden, kann dynamisch eine Verbindung zu ihrem Interface hergestellt werden; auf diese Weise werden Fehlertoleranzeigenschaften, wie dynamische Rekonfiguration oder multiple Software projektierbar.

Zur Realisierung des Interface werden zwei Typen von Ports eingeführt: über Ein- und Ausgangs-Ports können Boschaften in Collections eintreten bzw. sie verlassen. Der Botschaftenaustausch erfolgt über logische Verbindungen zwischen Ein- und Ausgangsports. Die Beschreibung dieser Verbindungen erfolgt im separaten Configurations-Teil, so daß bei dynamischer Veränderung der Verbindungsstruktur keine Modifikation des Ablaufteils erforderlich ist. Die Verbindungsaufnahme erfolgt durch Umsetzen der Verbindungsbeschreibung in Steuerdaten für das Netzbetriebssystem. Die Portnamen sind für die ablaufenden Programmteile Collection-lokal, nur im deskriptiven Configuration-Teil können alle Ports angesprochen werden. Bild 3 zeigt die Definition zweier Collections und die Verbindung der Port-Interfaces:

```
COLLECTION C1;
    MODULES M1, M2;
    PORTS X OUT, Y IN;

COLLECTION C2;
    MODULES M3;
    PORTS X IN , Z OUT;
LOAD C1, C2....
CONNECT    C1.X ——→ C2.X;
CONNECT    C1.Y ←—— C2.Z;
```

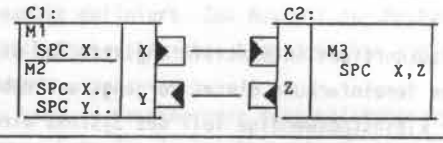


Bild 3. Collection-Vereinbarung und Beschreibung logischer Verbindungen im Configuration-Teil .

Die im Configuration-Teil vereinbarten Ports werden durch eine Spezifikation im Problemteil bekanntgemacht. Da die Verwaltung der Ports vom Netzwerkbetriebssystem übernommen wird, empfiehlt sich zwar die Herstellung des Bezugs über den Systemteil, es ginge jedoch die Namenslokalität verloren, da Systemteilobjekte stationslokale und nicht collectionlokale Namen besitzen (vgl. Kap. 3.1.2). Ports sollten deshalb reine Softwareobjekte sein.

Die Portverbindungen müssen nicht auf die 1:1-Struktur beschränkt bleiben: durch Verbinden eines Ausgangs mit n Eingängen kann eine 1→n, im umgekehrten Falle eine n→1-Kommunikationsstruktur vereinbart werden. Die Nutzung dieser Möglichkeiten wird im Kap. 3.3.3 beschrieben. Die Abbildung der logischen Verbindungsstruktur auf die physikalische (Kap. 3.1.3) wird im allgemeinen automatisch durch das Netzbetriebssystem bewerkstelligt. Es kann jedoch der Nachrichtenverkehr über bestimmte oder bevorzugte Leitungen wünschenswert sein.

Beispiel: CONNECT C1.X→C2.X VIA LINE#1;
CONNECT C1.Y←C2.Z PREFER LINE#2;

3.2.2 Dynamische Rekonfiguration

Die Angabe einer Initialverteilung der Collections kann durch einfach Ladeanweisungen im Configuration-Teil erfolgen, z.B. LOAD C1 TO STA1 STARTNO 1;
Der Stationsname STA1 muß in der Prozessorbeschreibung (Kap. 3.1.1) angegeben sein. Die Angabe einer Startreihenfolge für die Collections ist möglich. Zwei Anforderungspakete liegen der Einführung der dynamischen Rekonfiguration zugrunde:

1. Die "gewollte" Rekonfiguration:
   Gründe hierfür sind z.B.:

   - Hinweg-/Hinzunahme von Teilstationen (Wartung, geplante Modifikation)
   - Hinzufügen oder Substitution von Software-Komponenten

   In zukünftigen Automatisierungssystemen wird man eine Vereinfachung dieses Vorgangs anstreben: nur der kleinstnotwendige Teil des Systems wird gestoppt, das Restsystem läuft weiter (Kra 83). Die Substitution von Collections bei laufendem System ist durch die Definition einer Dialogumgebung möglich, die auch die inversen Anweisungen des Configuration- Teils kennt (siehe Kap. 4).

2. Rekonfiguration als Reaktion auf Störungen:
   Die Installation von Fehlerdetektionsmechanismen, die den Gesamtsystemzustand in einer solchen Weise festlegen, daß alle Teilstationen die gleiche Sicht dieses Zustandes haben, ist der Ausgangspunkt für eine konsistente Umverteilung des Programmsystems bei auftretenden Fehlern. Über die Art der Fehlererkennung und Weitergabe an das Rekonfigurationssystem werden auf Sprachebene keine Angaben gemacht. Als Konvention gilt die Verfügbarkeit einer Statusregister-Liste in jeder Station, die den Zustand aller Teilsysteme enthält. Bezeichner zur Formulierung von Zuständen des Statusregisters wurden im Kap. 3.1.1 eingeführt. Bild 4 zeigt die Formulierung von problemangepaßten Rekonfigurationsstrategien. Als Rekonfigurationsmaßnahmen können Collections nachgeladen oder hauptspeicherresident bereitgehalten und aktiviert werden. Die Ladebedingungen (boole'sche UND-Verknüpfungen von Statusbezeichnern) werden hierzu mit dem aktuellen Systemzustand verglichen. Die Initialverteilung in Bild 4 sieht die Collections C1 und C3 auf Station 1 sowie C2 auf Station 2 vor. Außerdem wird C1 auf Station 3 resident bereitgehalten. Fällt der Prozessor von Station 1 aus, so wird - nachdem die Kommunikationsverbindungen aufgelöst sind - C1 auf Station 3 aktiviert. Verbindungsaufnahme ist nur noch zu Station 2 notwendig. Fällt zusätzlich auch Station 3 aus, wird C1 auf Station 2 nachgeladen und kann dort ggf. die Collection C2 verdrängen, die eine niedrigere Ladepriorität besitzt. In diesem Falle hat C1 keine Kommunikationsverbindung mehr.

```
LOAD  C1  TO  STA1  LDPRIO 5;  }
LOAD  C2  TO  STA2  LDPRIO 10; } Initialverteilung
LOAD  C3  TO  STA1  LDPRIO 15; }
STATE   (STA1.PR & NOT STA3.PR):
        DISCONNECT  C1.P1→C2.P1, C3.P1;
        LOAD  C1  TO  STA3  LDPRIO 5  RESIDENT;
        CONNECT  C1.P1→C2.P1;
STATE   (STA1.PR  &  STA3.PR):
        DISCONNECT  C1.P1→C2.P1
        LOAD C1  TO STA2  LDPRIO 5;
       [CONNECT  C1.P1→C2.P1];
```

Bild 4. Dynamische Rekonfiguration nach Eintreten von Statusbedingungen

In Bild 4 werden Konfliktsituationen bei der Speichervergabe durch Ladeprioritäten geregelt. In Kap. 4 wird mit der Remove-Funktion eine weitere Möglichkeit aufgezeigt.

Inkonsistenzen bei der Umverteilung können entstehen, wenn sich die Ladebedingungen nicht gegenseitig ausschließen. Würde bei der ersten Ladebedingung in Bild 4 der Zustand STA3.PR weggelassen, so würde C1 sowohl auf Station2 als auch auf Station 3 aktiv werden. Der Configuration-Teil-Übersetzer muß die Konsistenz der Ladeanweisungen abprüfen.

Auf die Durchführung der Rekonfiguration durch Systemprogramme soll hier nicht weiter eingegangen werden, hierüber wurde schon an anderer Stelle berichtet (He 79), (St 77). Eine Übersicht über ein Programmier- und Ablaufsystem findet sich in Kap. 4.

3.2.3 Wiederaufsetzen

Die Rekonfiguration, die auf ein stochastisches Ereignis hin erfolgt, bedingt ein Aktivieren und Terminieren von Collections und damit auch von Rechenprozessen (Tasks). In manchen Anwendungsfällen erfordern diese Wiederanlaufvorgänge die Einbeziehung der Ablaufhistorie, d.h. die Daten und Zustände von Prozessen vor dem Auftreten eines Fehlers (Rückwärtswiederaufsetzen).

Das Gebiet des Rückwärtswiederaufsetzens (backword error recovery) findet gegenwärtig ein starkes wissenschaftliches Interesse. Die effiziente Implementierbarkeit automatisch arbeitender Verfahren, d.h. Algorithmen, die ohne explizite Steuerung durch das Anwenderprogramm vom Betriebssystem abgewickelt werden, ist noch nicht nachgewiesen.
Auf die Einführung unterstützender Sprachmittel haben wir verzichtet, da die Anforderungen stark anwendungsabhängig sind. Aus diesem Grund wird ein vereinfachtes und in Anwendungen (He 79) bewährtes Rückwärtswiederaufsetzverfahren vorgeschlagen:

1) Wiederanlaufpunkte
   Jeder Prozeß hat genau einen Wiederanlaufpunkt, den Prozess-Start. Da das typische Aufgabenprofil von Prozessautomatisierungssystemen in kurzen Prozessen zyklischer Struktur resultiert, ist das Rücksetzen auf den Prozeß-Start auch sinnvoll und effizient.

2) Wiederanlaufdaten
   Mit dem Prozeß-Start sind automatisch auch alle lokalen Datensätze gültig. Das Bereitstellen globaler Wiederanlaufdaten ist dem Anwender überlassen. Er hat die Möglichkeit, in dem/den Ausweichteilnehmer(n) eine Datenkopie anzulegen und je nach Charakter der Daten zyklisch oder ereignisgesteuert auf den aktuellen Stand zu bringen. Die aktuellen Zustände des technisches Prozesses brauchen in der Regel nicht redundant geführt zu werden, da sie jederzeit durch Einlesen - von den Eingabebaugruppen - zu beschaffen sind.
   Mit diesen Maßnahmen kann der Anwender einerseits den sinnvollen Integritätsgrad des Wiederanlaufs, andererseits aber auch die zusätzliche Systembelastung durch Fehlertoleranzmaßnahmen selbst bestimmen.

3) Berücksichtigung der Interprozeßkooperation
   Die Prozeßkooperation innerhalb von Collections ist unkritisch, da alle Prozesse auf den Anfang rückgesetzt werden. Zwischen Prozessoren verschiedener Modulgruppen ist als Kooperationsmittel nur das Botschaftenprinzip erlaubt. Damit läßt sich gutes Wiederanlaufverhalten programmieren, timeout-Mechanismen bei Nichtzustandekommen von Rendezvousmanövern wirken unterstützend.

3.3 Botschaftenkonzept

3.3.1 Botschaftenmechanismen

Die Wahl des Port-Konzepts als Basis für den Botschaftenaustausch wurde bereits in Kap. 3.2 erläutert. Im folgenden werden nun die Beschreibung von Port-Objekten (Kap. 3.3.2) sowie die Beschreibung der auf diese Objekte anwendbaren Operationen (Übertragungs-Statements TRANSMIT und RECEIVE, Kap. 3.3.3) nach Syntax und Semantik definiert. Zur Auswahl der Mechanismen des Botschaftenaustausches hier zunächst einige Vorbemerkungen.
Mit dem hier zu bestimmenden Botschaftenkonzept soll ein weites Anforderungsspektrum abgedeckt werden. Der Leitgedanke für die Auswahl war deshalb die Verwendung effizienter, elementarer Mechanismen, mit denen sich unterschiedliche höhere Protokolle programmieren lassen. Andererseits zeigt die Erfahrung im Umgang mit dem Botschaftenaustausch, daß Konstrukte benötigt werden, die eine hohe Programmiersicherheit, z.B. durch Konsistenzprüfungen zur Compile-Zeit, gewährleisten.

Als Hauptunterscheidungsmerkmal kann bei Botschaftensystemen die Art der Synchronisation zwischen Sender und Empfänger aufgefaßt werden. Für heute bekannte Botschaftenkonzepte ergibt dabei die unterschiedliche Semantik der Übertragungs-Statements eine Grobeinteitung in folgende Klassen:

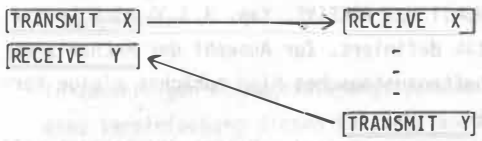
(1) Keine Synchronisation ("no-wait-send"-Mechanismen): Hierunter fallen u.a. alle Arten von Mailbox-Konzepten.

(2) Synchronisation auf den Statement-Anfang:
   Vertretern dieser Klasse liegt das Hoar'sche Rendezvous-Prinzip zugrunde (Hoa 78).

(3) Synchronisation auf das Statement-Ende:
   Hierzu können Mechanismen gerechnet werden, die "remote-procedure-calls" (Bri 78) und Botschaftenaustausch vereinigen. Ein Vertreter ist das ADA-Rendezvous-Konzept (Ada 80).

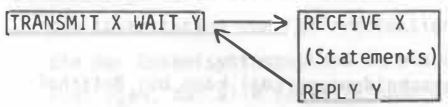
Aus den Anwendungen ergibt sich der Bedarf für einen asynchronen (1) und einen synchronen Mechanismus (2) oder (3). Eine Entscheidung nur für einen asynchronen oder synchronen Grundtyp würde die Ausprogrammierung des jeweils anderen erfordern, was zwar - für beide Fälle - in PEARL grundsätzlich möglich ist, jedoch nur auf Kosten der Programmiersicherheit und Effizienz.

Bei den synchronen Mechanismen ist (2) der elementarste. Gerade hier zeigen jedoch die Anwendungen, daß fast der gesamte Botschaftenaustausch bidirektional ist: Sender und Empfänger stehen in einer Consumer-Producer-Relation, d.h. der Consumer übergibt Aufträge an den Producer, der diese bearbeitet und ein Ergebnis zurückliefert. Bei (2) erhält man folgende Programmstruktur:



Die Verifikation dieses Protokolls durch einen Compiler ist praktisch nicht durchführbar, weil die Relation zwischen den eigenständigen Statements nicht feststellbar ist.

Wir haben uns deshalb für einen Vertreter der Klasse (3) entschieden mit der folgenden Programmstruktur:



Obwohl WAIT und REPLY syntaktisch vorgeschrieben sind, läßt sich ein niedrigeres Protokoll nach (2) ohne großen Effizienzverlust programmieren, zumal die "Statements" optional sind und auch in (2) wegen der Synchronisation ein implizites "REPLY" vom Netzwerkbetriebssystem gegeben werden muß.
Eine erhöhte Parallelarbeit von Producer und Consumer wäre dann möglich, wenn zwischen TRANSMIT und WAIT ebenfalls Statements stehen dürften. Systeme dieser Art sind jedoch kaum mit vertretbarem Software-Aufwand zu implementieren (Gil 83), der zweite Synchronisationspunkt erhöht den Verwaltungsaufwand beträchtlich.

3.3.2 Port-Spezifikation

Der Gültigkeitsbereich von PORT-Objekten erstreckt sich wegen der Vereinbarung im Configuration-Teil auf eine Collection. Der Zugriff auf ein Port-Objekt im Problem-Teil eines Moduls dieser Collection verlangt eine Port-Spezifikation. Die Syntax einer Port-Spezifikation zeigt Bild 5. Bei allen folgenden Syntaxdiagrammen werden u.a. syntaktische Variable verwendet, deren Definition in der Sprachbeschreibung für Full-PEARL (DIN 66253) zu finden ist. Ebenfalls sind die Regeln der verwendeten Syntax-Notation dort entnommen.

```
port_spc:
  'SPC'  port_id 'PORT' port_attr [port-signals] .
port_id:
   id                                              .
port_attr:
   in_port_attr   /   out_port_attr                .
in_port_attr:
   'IN' single_trf_item_type    synch_mechanism    .
synch_mechanism:
    no_wait_send  /  replied_send                  .
no_wait_send:
    'BUFFER' '('FIXED_const_denot')'  /
                                                   .
replied_send:
   'REPLY' single_trf_item_type                    .
out_port_attr:
   'OUT' single_trf_item_type [waiting_send]       .
waiting-send:
   'WAIT' single_trf_item_type                     .
port-signals:
   'SIGNAL' (signal_id // ',')                     .
```

Bild 5. Syntax einer Port-Spezifikation

Ein Port wird beschrieben durch den Botschaftentyp (single_trf_item_type), den er verarbeiten kann. Als Botschaften können die Inhalte aller definierten Datenobjekte verwendet werden, d.h. die Datenobjekte werden nicht explizit als Botschaften deklariert. Wie in Kap. 3.2 erläutert, müssen Ports ein Richtungsattribut besitzen. An einem "OUT"-Port kann nur gesendet, von einem "IN"-Port nur empfangen werden: die Übertragung erfolgt demnach von einem OUT-Port zu den IN-Ports, mit denen dieser verbunden ist. Zur Behandlung von Übertragungsfehlern können einem Port SIGNALS zugeordnet sein. Die Angabe des Synchronisationsmechanismus beim IN-Port steht mit der Semantik der Übertragungsstatements in Zusammenhang: fehlt die Angabe ganz, so kann über diesen Port nur das "no-wait-send"-Protokoll abgewickelt werden, d.h. der Sender wartet nur bis die Botschaft gesendet ist. Die Botschaft wird dann auf Empfangsseite vom Netzbetriebssystem gepuffert, bis sie vom Empfänger abgeholt wird. Optional kann der IN-Port selbst einen eigenen

Pufferbereich verwalten, dessen Größe durch das BUFFER-Attribut angegeben wird. Bei Angabe von "REPLY" beim IN-Port muß beim OUT-Port entsprechend ein 'WAIT' angegeben werden. In diesem Falle wartet der Sender auf Annahme und ggf. Verarbeitung der Botschaft sowie das Rückübertragen des Ergebnisses. Diese Rückübertragung erfolgt über dieselbe Verbindung, also vom IN-Port zum OUT-Port. Sende- und Empfangs-Statements müssen dann ebenfalls eine reply- bzw. wait-option (vgl. Kap. 3.3.3) aufweisen. Auf diese Weise können dann sowohl Port-Verbindungen, als auch die Zuordnung Statement-Port von einem Compiler auf Konsistenz überprüft werden. Beispiele für Port-Spezifikationen sind im folgenden Kapitel zu finden.

3.3.3 Übertragungs-Statements

In Kapitel 3.3.1 wurden die beiden Grundtypen von Übertraguns-Statements eingeführt, der no-wait-send- bzw. der send-reply-Mechanismus. Bild 6 zeigt die Syntax.

```
transmit_stmt:
  'TRANSMIT' transmit_params .
transmit-params:
  trf_expr  'TO' port_id [wait_option] .
wait_option:
  'WAIT' trf_expr [timeout_option] .
timeout_option:
  'TIMEOUT' duration_expr['REACT'unlabelled_stmt].
receive_stmt:
  simple_receive / selected_receive .
simple_receive:
  'RECEIVE' receive_params [end_receive_option] .
receive_params:
  'trf_expr' 'FROM' port_id [reply-option] .
end_receive_option:
  otherwise_part / timeout_option .
otherwise_part:
  'OTHERWISE' unlabelled_stmt .
reply-option:
  [unlabelled_stmt] 'REPLY' trf_expr .
selected_receive:
  'RECEIVE'
      'SELECT'   receive-params+
      ('OR'      receive-params+)+
      [end_receive_option]
.
```

Bild 6. Syntax der Übertragungs-Statements TRANSMIT und RECEIVE

Die Verwendung der beiden Protokolle wird durch Angeben bzw. Weglassen der wait-option (Sender) und reply-option (Empfänger) unterschieden. Konsequent wird der Aufruf einer TIMEOUT-Überwachung an all den Stellen gestattet, an denen auf den Kommunikationspartner gewartet werden muß. Auf Empfangsseite kann wahlweise auch ein otherwise-part angegeben werden, der der Abarbeitung von alternativen Anweisungen dient, wenn keine Aufträge vorliegen. Die Wirkung der Übertragungs-Statements sei anhand einiger Programmbeispiele erläutert.

Ein Beispiel für die Verwendung des no-wait-send-Protokolls zeigt Bild 7. Im Konfigurationsteil wird eine 1→n Kommunikationsstruktur vereinbart. Botschaften, die dem OUT-Port P1 der Collection TALKER übergeben werden, werden vom Netzbetriebssystem automatisch an die angeschlossenen IN-Ports LISTENER_1.P1,..., LISTENER_N.P1 gesendet und dort ggf. gepuffert. Bei 1→n Strukturen wird ausschließlich das not-wait-send-Protokoll zugelassen (keine Angabe von REPLY bzw. WAIT), da multiple Antworten nur in Spezialfällen benötigt werden und eine synchronisierte Form der 1→n-Kommunikation leicht ausprogrammiert werden kann. Die Pufferverwaltung im no-wait-send-Protokoll muß Maßnahmen bei folgenden Ausnahmefällen vorsehen:

- Puffer leer beim Empfangen:
  Die Empfangstask wird blockiert und wartet auf das Eintreffen weiterer Botschaften

- Puffer voll beim Senden:
  Sender und Empfänger können durch Signale informiert werden und entsprechende Vorkehrungen treffen, d.h. der Sender wartet in keinem Falle auf die Abnahme einer Botschaft. Es werden keine gepufferten Botschaften überschrieben.

Bild 8 zeigt ein Beispiel für die Verwendung des send-reply-Protokolls. Mit Hilfe einer n→1-Kommunikationsstruktur kann ein Produzent die Aufträge von n Konsumenten bearbeiten. Ein Auftrag wird - in Analogie zu einem Prozeduraufruf - durch eine Botschaft als "aktueller Parameter" übergeben und ausgeführt. Da die Verarbeitungs-Statements wieder Kommunikationsaufrufe enthalten können, ist eine Schachtelung der Auftragsbearbeitung möglich. Das Resultat wird - ähnlich dem Prozedur-RETURNS - als Antwort zurückgegeben. Um weitere Aufträge ausführen zu können, muß der Produzent im Programmablauf wieder zu dem RECEIVE zurückkehren. Während bei der 1→n-Struktur auf multiple

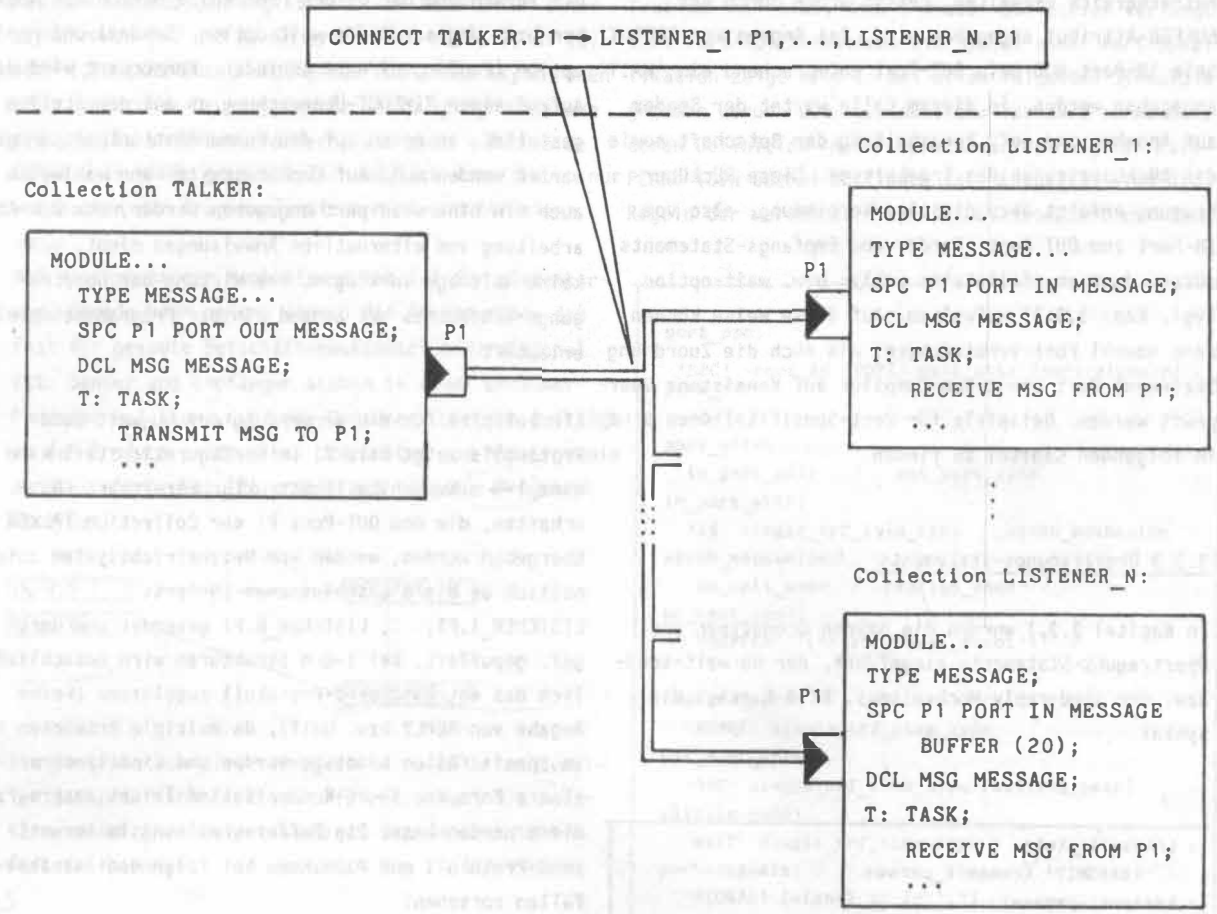


Bild 7. Programmstruktur bei einer 1→N-Kommunikation mit "no-wait-send"-Protokoll

Antworten verzichtet wurde, werden bei n→1 Strukturen multiple Aufträge untersagt, d.h. der Produzent bedient zu einem Zeitpunkt nur einen der angeschlossenen Consumenten. Mit der n→1-Struktur wird jedoch klar festgelegt, wer die Dienste des Produzenten in Anspruch nehmen darf.

Die n→1-Kommunikationsstruktur ermöglicht die Bearbeitung von Aufträgen, deren Reihenfolge und Zeitpunkt des Eintreffens nicht-deterministisch sind. Diese Programmstruktur ist charakteristisch für Aufgaben der Prozeßdatenverarbeitung. Häufig tritt sogar der Fall auf, daß die eintreffenden Botschaften unterschiedliches Format haben. Um solche Aufträge zu bewältigen, müßten die Ports mehrere Botschaftsformate verarbeiten können. Da hier jedoch sehr leicht unübersichtliche Port-Verbindungsstrukturen entstehen können, haben wir uns für eine Erweiterung des RECEIVE-Statements zu einem Selected-RECEIVE entschieden.

Ein Beispiel hierzu zeigt Bild 9. Die Botschaften unterschiedlichen Formats werden von verschiedenen Ports empfangen und den durch das Schlüsselwort "OR" getrennten SELECT-Alternativen zugeführt. Wie bei der n→1-Kommunikationsstuktur kann zu einem Zeitpunkt nur ein Auftrag bearbeitet werden. Stehen beim Eintritt in das RECEIVE-Statement mehrere Aufträge an, so wird - nach einem nichtdeterministischen Verfahren - einer davon ausgewählt und die anderen zu einem späteren Zeitpunkt bearbeitet (erneuter Statement-Durchlauf). Steht kein Auftrag an, so wird auf das Eintreffen eines Auftrags gewartet, wenn kein OTHERWISE-Part angegeben ist.

Die hier gezeigten Beispiele können keine exakte Beschreibung der Semantik der Übertragungs-Statements und der PORT-Vereinbarungen geben, dazu ist hier auch nicht der Platz. Die Erstellung eines Reference-Manuals ist in Bearbeitung.

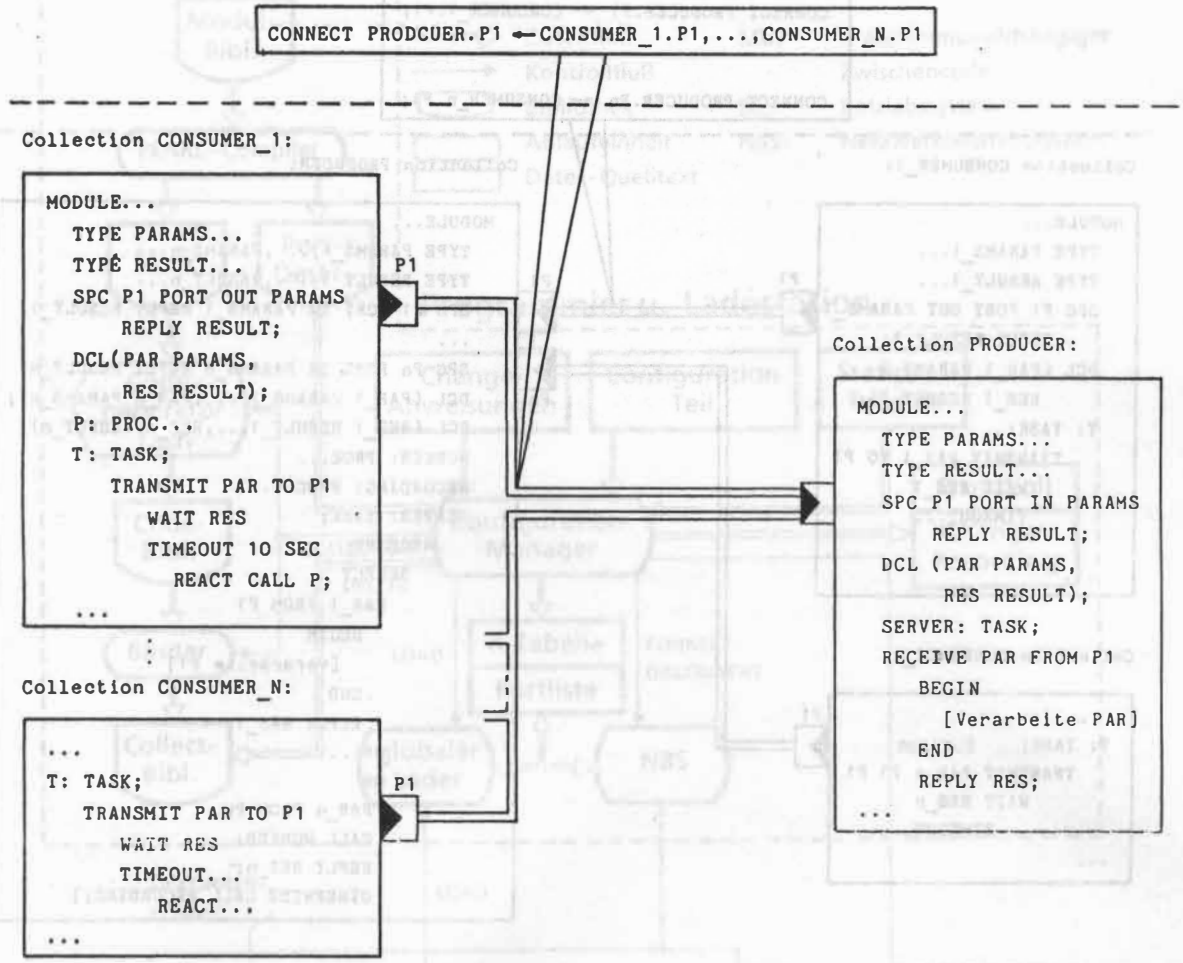


Bild 8. N→1-Kommunikation bei gleichem Botschaftenformat

4. Programmier- und Ablaufumgebung

Die Anweisungen des Configuration-Teils sind für die direkte Ansteuerung von Systemprogrammen prädestiniert und können deshalb auch als Kommandosprache aufgefaßt werden. Bisher wurden folgende Kommandos definiert:

COLLECTION: Erstellen einer neuen Collection
Einordnen in eine Collection-Bibliothek

LOAD: Laden einer Collection

CONNECT: Verbinden der Ports von Collections

Mit dem gewählten Konfigurationskonzept ist der Ausbau eines Automatisierungssystems zu einem on-line-modifizierbaren System realisierbar.

Die "gewollte" Veränderung der Konfiguration bei weiterlaufendem System kann nicht durch Editieren und Neuübersetzen des Configuration-Teils erfolgen, sondern muß durch gezielte Eingabe von Änderungs-Anweisungen geschehen. Es müssen auch die inversen Konfigurationsfunktionen definiert sein:

DELETE: Entfernen einer Collection aus der Collection-Bibliothek. Die DELETE-Operation darf erst angewendet werden, wenn die Collection nicht mehr geladen ist.

REMOVE: Entfernen einer Collection von einer Station. Die REMOVE-Operation darf erst durchgeführt werden, wenn die Collection keine Port-Verbindungen zu anderen Collections mehr besitzt.

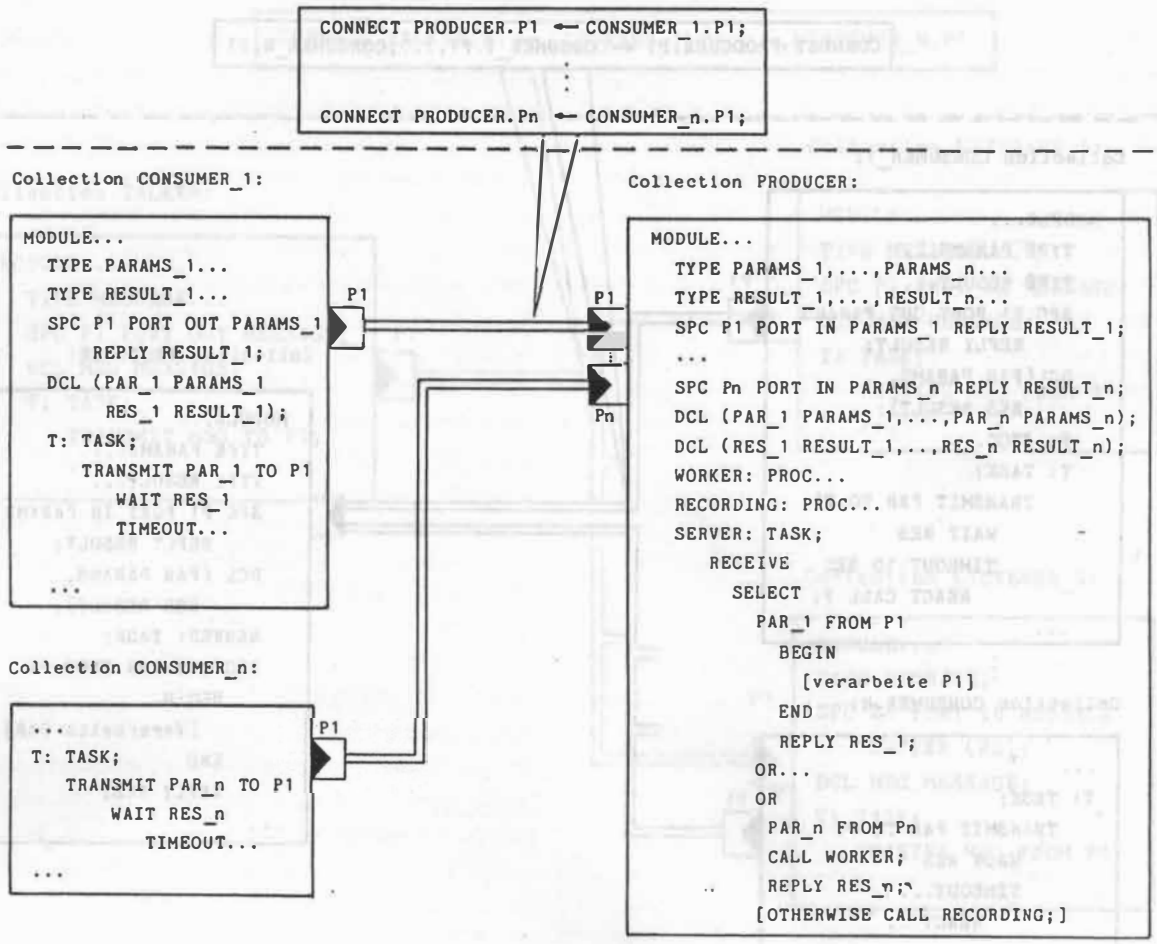


Bild 9. N →1-Kommunikation bei ungleichem Botschaftenformat

DISCONNECT: Auflösen von Port-Verbindungen. Botschaften, die an Ports gesendet werden, die keine Verbindungen mehr besitzen, haben keine Wirkung und werden vernichtet. Dem Sender kann dies durch Induzieren eines SIGNALs bekanntgemacht werden.

Die sichere Substititon einer Collection, ohne das System zu stoppen, kann also in der Reihenfolge DISCONNECT, REMOVE, DELETE, INSERT, LOAD, CONNECT durchgeführt werden. Hinzufügen kann durch INSERT, LOAD, CONNECT erfolgen.

Bild 10 zeigt die grobe Struktur einer Beispielimplementation. Darin wird davon ausgegangen, daß sich die Programmentwicklungsstation als Teilrechner im Netz befindet. Soll die Programmentwicklung auf einem externen Host-Rechner erfolgen, ist eine andere Struktur der Programmentwicklungshilfen zu wählen.

Den Systemprogrammen in Bild 10 kommen folgende Aufgabenbereiche zu:

- Der PEARL-Compiler übersetzt die PEARL-Moduln, die er einer Modulbibliothek entnimmt. Er erzeugt zwei Arten von Ausgabeelementen:
  . den maschinenunabhängigen Zwischencode (falls ein homogenes System zugrunde liegt, kann auch sofort Maschinencode erzeugt werden)
  . die Namen, der Typ und die Adressen der Ports des Modules werden in einem namensgleichen Bibliothekselement einer zweiten Bibliothek abgelegt (Port-Deskriptoren).

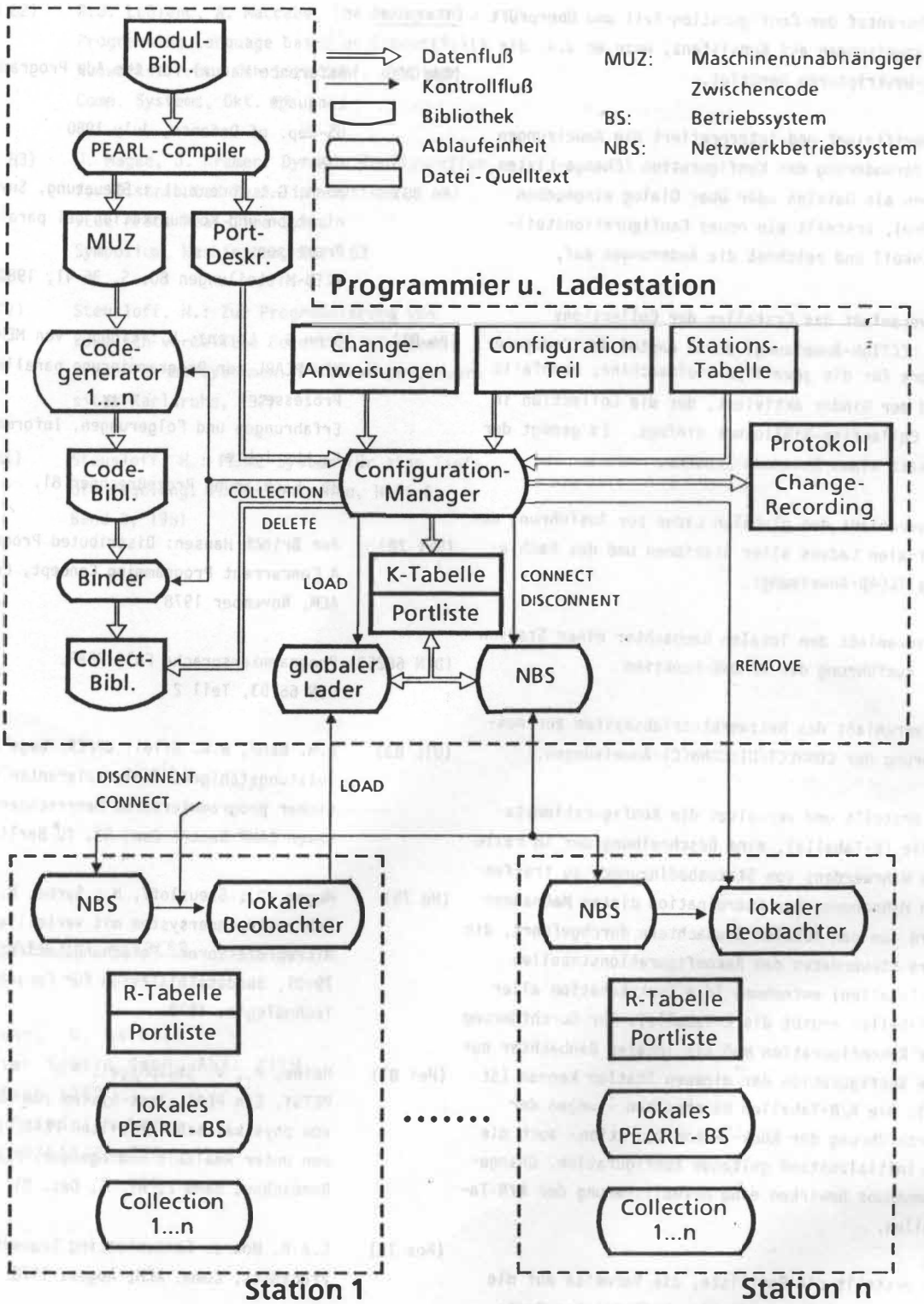


Bild 10. Programmier- und Ablaufsystem

- Der Stationsteil-Übersetzer transformiert den Stationsteil in eine für die Interpretation geeignetere Form (Stationstabelle).

- Der Configuration-Manager verarbeitet die Anweisungen des Configuration-Teils und der Change-Anweisungen:

. er übersetzt den Configuration-Teil und überprüft die Anweisungen auf Konsistenz, wozu er u.a. die Port-Deskriptoren benötigt,

. er verifiziert und interpretiert die Anweisungen zur Veränderung der Konfiguration (Change-Listen können als Dateien oder über Dialog eingegeben werden), erstellt ein neues Configurationsteil-Protokoll und zeichnet die Änderungen auf,

. er veranlaßt das Erstellen der Collections (COLLECTION-Anweisung) durch Anstoß des Codegenerators für die jeweilige Zielmaschine; ebenfalls wird der Binder aktiviert, der die Collection in die Collection-Bibliothek einfügt. Es genügt der Einsatz eines Standard-Binders,

. er veranlaßt den globalen Lader zur Ausführung des initialen Ladens aller Stationen und des Nachladens (LOAD-Anweisung),

. er veranlaßt den lokalen Beobachter einer Station zur Ausführung der REMOVE-Funktion

. er veranlaßt das Netzwerkbetriebssystem zur Ausführung der CONNECT/DISCONNECT-Anweisungen,

. er erstellt und verwaltet die Konfigurationstabelle (K-Tabelle), eine Beschreibung der im Falle des Wahrwerdens von Statusbedingungen zu treffenden Maßnahmen. Die Koordination dieser Maßnahmen wird von den lokalen Beobachtern durchgeführt, die ihre Steuerdaten den Rekonfigurationstabellen (R-Tabellen) entnehmen (die Konkatenation aller R-Tabellen ergibt die K-Tabelle). Zur Durchführung der Rekonfiguration muß ein lokaler Beobachter nur die Konfiguration der eigenen Station kennen (St 77). Die K/R-Tabellen beschreiben - wegen der Durchführung der Rück- Rekonfiguration- auch die im Initialzustand geltende Konfiguration. Change-Kommandos bewirken eine Aktualisierung der K/R-Tabellen.

. er erstellt die Portliste, die Verweise auf die sich aktuell in der Station befindlichen Ports enthält. Sie wird im Rekonfigurationsfalle von den lokalen Beobachtern aktualisiert.

- Nicht gezeigt in Bild 10 ist ein PEARL-Testsystem für Mehrrechnersysteme (Hei 81), das an die neuen Programmstrukturelemente angepaßt werden muß.

Literatur

(Ada 80) Reference Manual for the Ada Programming Language
US-Dep. of Defense, July 1980

(Bo 80) Bonn, G.; Lorenz, L.: Steuerung, Snychronisation und Kommunikation bei parallelen Prozessen
IITB-Mitteilungen 80, S. 36-41, 1980

(Bo 81) Bonn G.; Lorenz, L.: Eignung von MEHRRECHNER-PEARL zur Programmierung paralleler Prozesse;
Erfahrungen und Folgerungen, Informatik-Fachberichte
39. Fachtagung Prozeßrechner 81.

(Bri 78) Per Brinch Hansen: Distributed Process: A Concurrent Programming Concept, Comm. ACM, November 1978

(DIN 66253) Programmiersprache Full-PEARL
DIN 66253, Teil 2

(GIL 83) P.M. Behr, W.K. Giloi: UPPER: Wege zu leistungsfähigen, fehlertoleranten und sicher programmierbaren Mehrrechnersystemen CAMP Report Juni 83, TU Berlin

(He 79) Heger, D.; Steusloff, H.; Syrbe, M.: Echtzeitrechnersystem mit verteilten Mikroprozessoren. Forschungsbericht DV 79-01, Bundesministerium für Forschung und Technologie, 1979.

(Hei 81) Heine, P., v. Stülpnagel, F.:
PETSY, Ein PEARL-Test-System zum Austesten von physikalisch verteilten PEARL-Programmen unter Realzeit-Bedingungen, PEARL-Rundschau, Band 2, Nr. 6, Dez. 81

(Hoa 78) C.A.R. Hoare: Communicating Sequential Processes. Comm. ACM, August 1978

(Kra 83) J. Kramer et. al.: CONIC: An Integrated Approach to distributed computer control systems IEE Proc., Vol, 130, Pt. E, No. 1, January 1983

(LeB 82) R.J. LeBlanc, A. Maccabe: The Design of a Programming Language based on Connectivity Networks. Proc. 4th Int. Conf. on Distr. Comp. Systems, Okt. 82

(Mag 83) J. Magee, J. Kramer: Dynamic Configuration for Distributed Real-Time-Systems IEEE Proc. Real-Time-Syst. Symposium, Washington, Dec. 83

(St 77) Steusloff, H.: Zur Programmierung von räumlich verteilten dezentralisierten Prozeßrechnersystemen. Disseration Universität Karlsruhe, 1977

(St 81) Steusloff, H.: PEARL-System für eine Tiefofenregelung. PEARL-Rundschau, Heft 6, Band 2, 1981

Adressen der Autoren

E. Fahr, G. Barbian
Dornier System GmbH, Abt. ZITW
Postfach 1360
7990 Friedrichshafen
Tel.: 07545/85-363
-468

U. Bügel, G. Bonn
Fraunhofer-Institut für Informations- und Datenverarbeitung (IITB)
Sebastian-Kneipp-Str. 12-14
7500 Karlsruhe
Tel.: 0721/6091-457
-301