PEARL

ASSOCIATION
VEREIN E.V.

Vorträge zum Workshop über Realzeitsysteme

PEARL 87

am 3. und 4. Dezember 1987
in Boppard am Rhein

Tagungsleitung: Dipl.-Ing. A. Küchle

Veranstaltet vom PEARL-Verein e.V.

unter Mitwirkung von
Gesellschaft für Informatik e.V.
VDI/VDE-Gesellschaft Meß- und
Automatisierungstechnik

Herausgegeben vom PEARL-Verein e.V. - Geschäftsstelle München

PEARL 87

Workshop über Realzeitsysteme

PEARL 87

Vorträge zum Workshop über Realzeitsysteme
am 3. und 4. Dezember 1987
in Boppard am Rhein

Veranstaltet vom PEARL-Verein e.V.

unter Mitwirkung von
Gesellschaft für Informatik e.V.
VDI/VDE-Gesellschaft Meß- und
Automatisierungstechnik

Herausgegeben von Klaus Stieger,
PEARL-Verein e.V. - Geschäftsstelle München

Tagungsleitung:

Dipl.-Ing. A. Küchle
c/o Dornier System GmbH
Postfach 1360

D-7990 Friedrichshafen 1

Programmkomitee:

D. Eberitzsch	Krupp Atlas Elektronik, Bremen
Dr. R. Henn	GPP, Oberhaching
A. Küchle	Dornier System, Friedrichshafen
H. Meyerhoff	Krupp Atlas Elektronik, Bremen
Dr. K. Rebensburg	TU Berlin
Prof. Dr. H. Rzehak	UniBw München
D. Sauter	IRT, München

Tagungsorganisation:

Dipl.-Inform. Klaus Stieger
PEARL-Verein e.V.
Geschäftsstelle München
Werner-Heisenberg-Weg 39

D-8014 Neubiberg

Inhaltsverzeichnis

Vorwort

PEARL 87 - unter diesem Titel soll auch in diesem Jahr im Rahmen eines Workshops ein Forum geboten werden, in dem Praktiker aus ihren Arbeitsgebieten berichten und gewonnene Erfahrungen weitergeben.

Übersichtsvorträge und Informationen aus den Entwicklungslabors namhafter Rechnerhersteller und Softwarehäuser geben eine Orientierung über neue Trends.

Thematisch werden insbesondere die Themenkreise der Realzeitsysteme in der Fertigungsindustrie sowie die Realzeit-Datenverarbeitung unter Standardbetriebssystemen vertieft abgehandelt.
Damit soll 2 Forderungen aus der Projektpraxis Rechnung getragen werden:

- Unter dem zunehmenden internationalen Konkurrenzdruck steht die in der Bundesrepublik volkswirtschaftlich bedeutende Fertigungsindustrie unter dem Zwang, ihre Produkte effizienter und damit billiger zu fertigen, flexibel auf neue Bedürfnisse des Marktes zu reagieren und Produkte mit einem hohen Qualitätsstandard zu liefern.
  Hierbei kommt den rechnergesteuerten Fertigungsprozessen und Qualitätsüberwachungssystemen zunehmende Bedeutung zu.
  Architekturen, Normen und Standards der Realzeitverarbeitung in diesem Arbeitsgebiet bilden einen Schwerpunkt dieser Tagung.

- Infolge der aus Wirtschaftlichkeitsgründen abgeleiteten Forderung, auch in Realzeitanwendungen zunehmend sogenannte General Purpose Rechner bzw. PC's einzusetzen, ergeben sich zunehmend Überlegungen, ob Standardbetriebssysteme dieser Rechner auch den besonders hohen Anforderungen der Realzeit-Datenverabeitung Rechnung tragen.
  Hierauf sollen Vorträge speziell zu UNIX aber auch zu anderen weit verbreiteten Systemen von Herstellern Aufschluß geben.

Veranstalter und Programmkomitee wünschen sich einen regen Meinungsaustausch im Rahmen der Sitzungen - Diskussionsbeiträge sind sehr erwünscht. Nur wenn unterschiedliche Standpunkte und Meinungen vorgetragen werden, ist ein Meinungsaustausch möglich.

Lassen Sie mich abschließend einige Worte des Dankes sagen:

Zuerst den Vortragenden, die durch ihre Bereitschaft, ihr Wissen und ihre Erfahrung weiterzugeben, den entscheidenden Beitrag zum Gelingen der Tagung erbringen.

Weiterhin den Mitgliedern des Programmausschusses für die konstruktiven Beiträge und die angenehme Art der Zusammenarbeit bei Erarbeitung der Zielsetzung der Tagung und der Gestaltung des Programmes.

Friedrichshafen, den 16. November 1987

Alfred Küchle

Die Abwicklung von Realzeit-Aufträgen in MAP-Netzen

Prof. Dr. Helmut Rzehak

Universität der Bundeswehr München
Institut für Systemorientierte Informatik
Werner-Heisenberg-Weg 39, D-8014 Neubiberg
Telefon: (089) 6004-3397 oder -2542

Zusammenfassung:

Das Manufacturing Automation Protocol (MAP) ist ein mit beträchtlichen Investitionen verbundener Vorstoß von General Motors, den Bereich der Datenkommunikation in der Fertigung zu standardisieren. Der Datenaustausch zwischen Geräten und Rechnern verschiedener Hersteller, die den MAP-Definitionen genügen, soll ohne besondere Anpassungen möglich sein. MAP baut weitgehend auf Standardisierungsvorschläge der ISO für die Kommunikation in offenen Rechnernetzen auf. Die Anwendbarkeit ist auch außerhalb der Automobilindustrie in weiten Bereichen gegeben. Zum allgemeinen Verständnis wird kurz auf die Prinzipien bei der Abwicklung dieser Protokolle eingegangen. Es werden die grundsätzlichen Probleme bei der Abwicklung von Realzeitaufträgen behandelt, und die Konzepte vorgestellt, die bei der z.Zt. durchgeführten überarbeitung des MAP-Standards berücksichtigt werden! Der aktuelle Stand dieser Arbeiten wird berichtet und eine Analyse gegeben, in welcher Größenordnung Zeitbedingungen eingehalten werden können.

1. Datenkommunikation in der Fertigungsautomatisierung

1.1 Die Ausgangssituation

Der generelle Trend zur Verwendung von immer mehr und immer komplexeren Rechensystemen und Steuerungseinrichtungen mit Mikroprozessoren in der Fertigungsautomatisierung ist durch einige Besonderheiten gekennzeichnet:

- Die verwendeten Rechensysteme bzw. Automatisierungskomponenten lassen sich in der Regel nicht auf eine Systemfamilie bzw. einen Hersteller beschränken. Die Vielfalt ist tendentiell um so größer, je größer die Fertigungseinrichtung ist.

- Die Aufgabenstellungen werden immer umfassender und zwingen zur Integration von Lösungen für einzelne Teilaufgaben (Schlagwort CIM).

Zusammengefaßt bedeutet dies, daß in der Fertigungsautomatisierung zunehmend heterogene Rechnernetze verwendet werden und daß der Anschluß von Automatisierungseinrichtungen an lokale Rechnernetze immer wichtiger wird. Dies wird jedoch durch eine Vielzahl nicht übereinstimmender Schnittstellen bei Hardware und Software erheblich erschwert und kann bei immer größeren Netzen einen beträchtlichen technischen und finanziellen Aufwand bedeuten, wenn es nicht gelingt diese Schnittstellen zu vereinheitlichen.

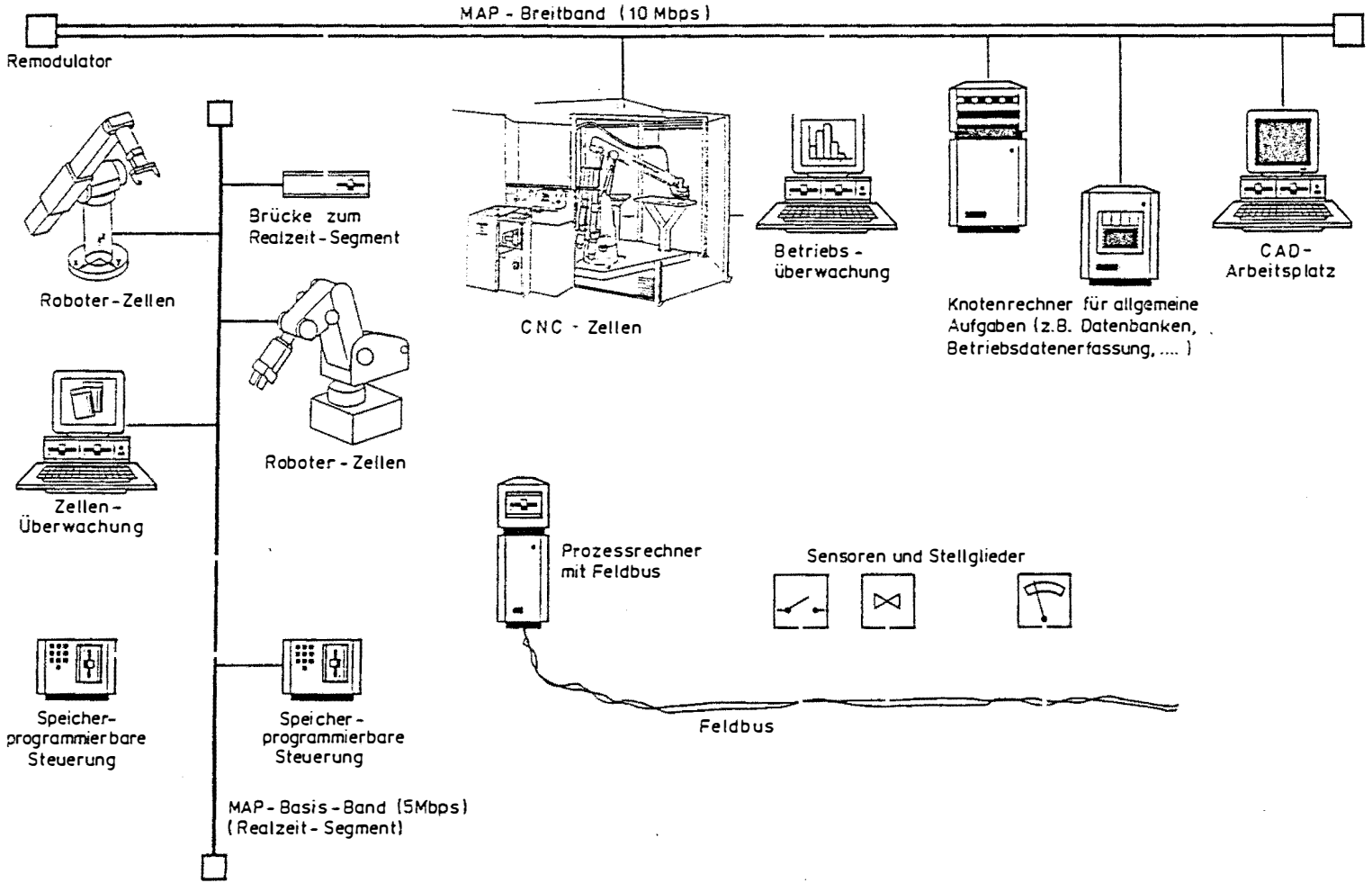
In dieser Situation wurde auf Initiative von Gerneral Motors mit beträchtlichem Aufwand versucht, durch Definition des Manufacturing Automation Protocol (MAP) einen Netzstandard für die Datenkommunikation zwischen Automatisierungseinrichtungen und Rechnern zu schaffen. MAP baut auf ISO-Normen auf, soweit diese zur Verfügung stehen. Es werden jedoch Verfahren und Parameter festgelegt, wenn auf Grund der Normen eine Auswahl oder Festlegung von Parametern oder Optio-

nen möglich oder auch erforderlich ist. Trotz großer Fortschritte der Open Systems Interconnection (OSI) Arbeitsgruppen in der ISO gibt es derzeit noch Lücken im Normungswerk, so daß für MAP Ergänzungen notwendig waren. Teilweise wurden diese Ergänzungen auch bereits in die ISO-Normen aufgenommen. Derzeit veröffentlicht ist die MAP-Version 2.1; eine neue Version 3.0 liegt als internes Arbeitspapier bereits vor und soll im Frühjahr 1988 veröffentlicht werden. In diesem Bericht sind die bekannt gewordenen wesentlichen Änderungen in MAP 3.0 bereits berücksichtigt.

1.2 Verwendungsbereich von MAP-Netzen

Eine wichtige Vorentscheidung für das zu definierende Netzprotokoll wird durch die Festlegung der verfügbaren "Knotenintelligenz" getroffen, d.h. welche Verarbeitungsleistung für die Protokoll-Abwicklung verfügbar sein soll. Die hierdurch verursachten Kosten könnnen sinnvoll nur einen Bruchteil der Kosten des ganzen Knoten darstellen. Die für MAP getroffenen Festlegungen gehen davon aus, daß einfache Sensoren als selbständige Knoten nicht in Betracht kommen. Andererseits sollen neben frei programmierbaren Prozessrechnern auch speicherprogrammierbare Steuerungen, numerisch gesteuerte Werkzeugmaschinen, Handhabungsgeräte u.ä., selbständige Knoten bilden können. Für diese Gruppe von Geräten ist es nicht ausreichend, daß MAP ein allgemeines Botschaftsformat festlegt und die Interpretation der Botschaft den Vereinbarungen zwischen den Anwenderprogrammen überläßt. Es bedarf vielmehr der Festlegung von Kommandos, die von den Geräten einheitlich interpretiert werden.

Der Trend zur Ingeration der gesamten Produktionssteuerung einschließlich Entwurf und Fertigungsvorbereitung bringt es mit sich, daß MAP-Netze aus dem Bereich der Fertigung herausführen und sich auch für die übertragung größerer Datenmengen unter weniger harten Zeitforderungen eignen müssen. Damit ergibt sich die in Bild 1.1 gezeigte typische Architektur eines MAP-Netzes.


MAP - Breitband (10 Mbps)
Remodulator
Roboter-Zellen
Brücke zum Realzeit-Segment
Roboter-Zellen
Zellen-Überwachung
CNC - Zellen
Betriebs-überwachung
Knotenrechner für allgemeine Aufgaben (z.B. Datenbanken, Betriebsdatenerfassung, ...)
CAD-Arbeitsplatz
Prozessrechner mit Feldbus
Sensoren und Stellglieder
Feldbus
Speicher-programmierbare Steuerung
Speicher-programmierbare Steuerung
MAP-Basis-Band (5Mbps) (Realzeit-Segment)

1.3 Erwartungen an die Standardisierung

Der Anwender erwartet von der Standardisierung vor allem die Interoperabilität der Systeme von verschiedenen Herstellern. Normenkonformität alleine genügt nicht, wenn nicht die Norm selbst alle wesentlichen Aspekte der Interoperabilität berücksichtgt. Die Netze müssen offene Systeme sein, d.h. es müssen während des Betriebes neue Knoten hinzufügbar und zusätzliche Programmkomponenten integrierbar sein. Die Protokolle sollen natürlich nur wenig übertragungs- und Verarbeitungsleistung absorbieren. Von großer Bedeutung ist auch die Möglichkeit vorhandene Netze schrittweise so umzugestalten, daß sie nach einigen Umrüstungen normkonform sind (Migrationswege).

2. Das ISO-Referenzmodell für die Kommunikation in Offenen Systemen

2.1 Grundlgegende Betrachtungen

Die ISO-Normen für Offene Systeme orientieren sich an einem Referenzmodell, das in der für lokale Netze gebräuchlichen Variante in Bild 2.1 dargestellt ist. Ein Kommunikationsauftrag aus einem Anwendungsprogramm (oberhalb Schicht 7) wandert durch die Schichten von oben nach unten, wobei die Schicht i-1 der nächst höheren Schicht i verschiedene Dienste zur Verfügung stellt. Der Aufruf dieser Dienste muß bestimmten Schnittstellenkonventionen genügen. Beim Zielknoten wandert der Auftrag nun durch alle Schichten von unten nach oben, bis er das Anwendungsprogramm erreicht hat, für das er bestimmt ist. Damit eine Schicht die ihr zugewiesene Aufgabe erfüllen kann, müssen bestimmte Vereinbarungen zwischen den beteiligten Knoten für diese Schicht getroffen werden (Protokoll).

Knoten A			Node B
7	Anwendung	Anwendungsbezogene Dienste	Application
6	Darstellung	Wandlung von Datenformaten	Presentation
5	Kommunikations-steuerung	Transformation von Namen in Adressen; Prüfen von Zugriffsrechten; Synchronisation	Session
4	Transport	Herstellen und Überwachen einer gesicherten End-zu-End-Verbindung	Transport
3	Netzwerk	Wegelenkung für Botschaften zwischen nicht benachbarten Knoten	Network
2	Verbindungskontrolle	Erkennen von Übertragungsfehlern	Logical Link Control
	Mediumzugriff	Abwickeln des Zugriffsverfahrens zum Übertragungsmedium	Medium Access Control
1	Phys. Signal	Aufprägen und Übertragen der Information entsprechend der gewählten physikalischen Signaldarstellung	Physical

Bild 2.1 Das ISO-Referenzmodell für lokale Netze

Kommunikation ist eine gerichtete Beziehung, d.h. es gibt einen Sender einer Nachricht und einen Empfänger. Dem entspricht das Wandern der Aufträge von oben nach unten (Sender) bzw. von unten nach oben (Empfänger) durch die einzelnen Schichten. Dabei werden jeweils Daten einer bestimmten Struktur übergeben, die nachfolgend als Protokoll-Dateneinheit (protocoll data unit, pdu) bezeichnet werden. Die Übergabe einer solchen Protokoll-Daten-Einheit von oben nach unten (Sender) wird als Anforderung (request) und die Übergabe von unten nach oben als Anzeige (indication) bezeichnet. Zur Abwicklung der Protokolle fügt jede Schicht bei einer Anforderung bestimmte Steuerinformation vorne an die Protokoll-Daten-Einheit an. Diese Steuerinformation wird von der entsprechenden Schicht beim Empfänger ausgewertet und entfernt, wenn sie die Protokoll-Daten-Einheit bei einer Anzeige erhält. Es entsteht die in Bild 2.2 gezeigte typische Anordnung von Protokoll-Daten-Einheiten. Man sieht, daß die zu übertragende Information durch die Protokolle der einzelnen Schichten beträchtlich anwachsen kann.

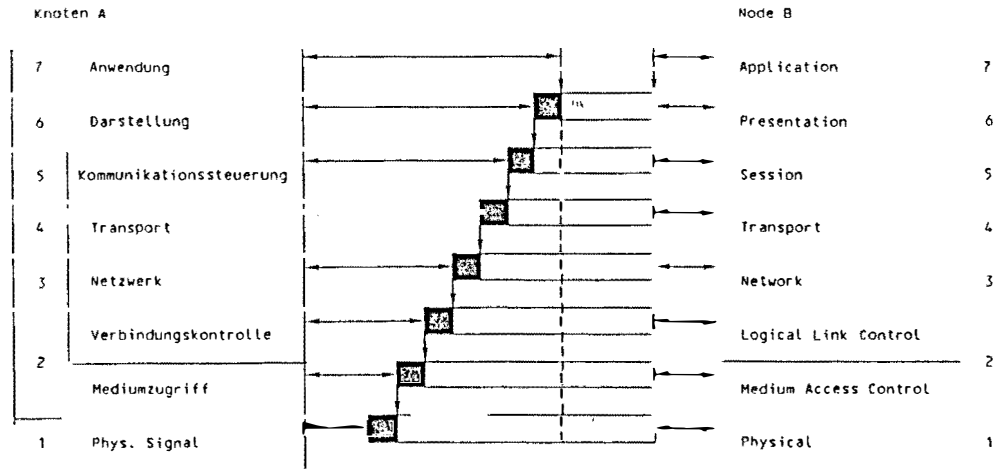


Bild 2.2 Zusammensetzung der Protokoll-Dateneinheiten

Verschiedene Anwendungs-Instanzen eines Knoten können gleichzeitig und unabhängig voneinander Kommunikationsbeziehungen unterhalten. Die Transport-Schicht stellt hierzu verschiedene Zugänge zur Verfügung. Zur Identifizierung einer Kommunikationsbeziehung wird daher neben den Knotenadressen der beteiligten Knoten noch die Nummer des zugehörigen Dienstzugangspunktes (service access point, sap) benötigt.

2.2 Verbindungslose und verbindungsorientierte Dienste

Aus historischen Gründen assoziieren die Postverwaltungen mit einer Kommunikationsbeziehung gerne eine "Verbindung" (Leitung, Kanal), die auf- und abgebaut werden muß. Dieses Denken hat auch lange Zeit die Normung für Datennetze stark geprägt, obwohl z.B. in einer Paketvermittlung kein technisches Äquivalent zu einer Verbindung zu finden ist. In den ersten Normen für offene Rechnernetze wurden daher verbindungsorientierte Dienste für alle Schichten des Referenzmodells vorgesehen. Dies hat den unbestreitbaren Vorteil, daß

nach dem erfolgten Verbindungsaufbau die Kommunikationspartner dem Vermittlungsnetz bekannt sind und die ausgetauschten Daten intern quittiert werden können. Man erreicht somit eine stabile, weitgehend gesicherte Kommunikation. In lokalen Netzen führt dies zu einem aufwendigen Protokoll für die Verbindungsaufnahme, obwohl die Nachrichten an alle angeschlossenen Stationen gelangen und dort auf Grund der mitgesendeten Adresse ausgewählt werden. Besonders störend wird dies, wenn während einer Kommunikationsbeziehung nur wenig Daten ausgetauscht werden.

Es wurden daher für die unteren Ebenen Protokolle für verbindungslose Dienste (connectionless mode) ausgearbeitet. Für eine gesicherte übertragung ist dann allerdings in der Transportschicht ein aufwendigeres Protokoll erforderlich. In MAP ist nun festgelegt, daß in den unteren Schichten bis einschließlich der Netzwerk-Schicht verbindungslose Dienste benutzt werden, während in der Transportschicht das verbindungsorientierte Protokoll für ungesicherte Transportwege (Klasse 4) verwendet wird. Dies ergbit den in Bild 2.3 dargestellten typischen Ablauf einer Verbindung zwischen zwei Instanzen der Schicht 4.

2.3 Zusammenstellung der in MAP verwendeten ISO-Normen

Anwendungs-Schicht

File Transfer, Access and Management (FTAM)
ISO DIS 8571

Manufacturing Message Service (MMS) ISO DP 9506 (entspricht RS 511); Verwendung in MAP 3.0

Darstellungs-Schicht

Die Darstellungs-Schicht ist in MAP 2.1 leer. Es ist zu erwarten, daß in MAP 3.0 eine Protokoll-Festlegung entsprechend der in ISO DIS 8822 "Connection Oriented Presentation Service Definition" oder einer Untermenge davon verwendet wird.

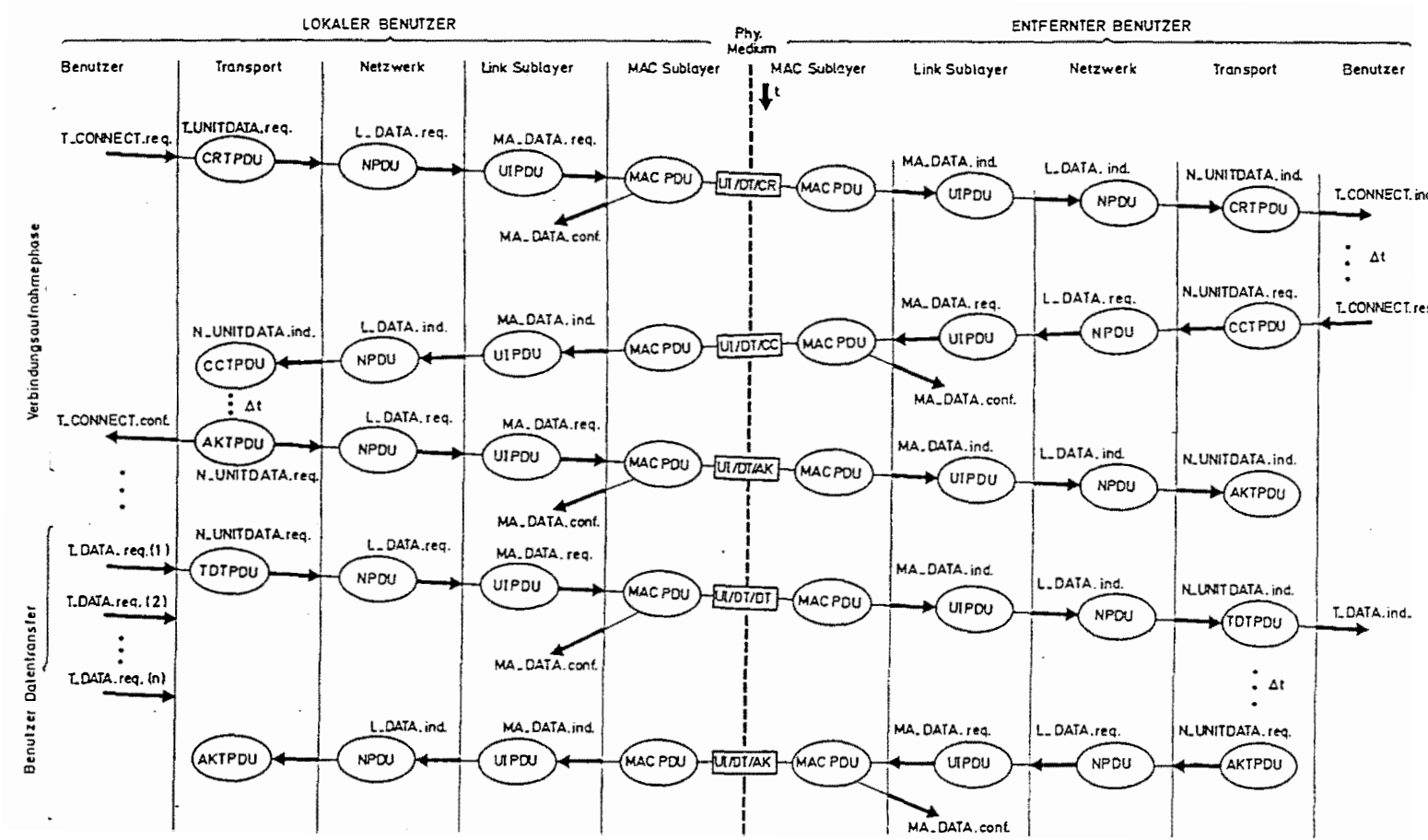


Bild 2.3 Verbindungsablauf zwischen zwei Instanzen der Transport-Schicht

Kommunikationssteuerungs-Schicht:

ISO-Session-Kernel entsprechend ISO IS 8326 "Connection-Mode Session Service Definition" (entsp. X.215)

Transport-Schicht:

Transport Service Definition ISO IS 8072 (entsp. X.214)

Verwendet wird das Klasse 4-Transport-Protokoll für ungesicherte Transportwege gemäß ISO IS 8075 "Connection oriented Transport Protocol Spezification"

Netzwerk-Schicht:

Network Service Definition ISO IS 8348 (entsp. X.213)

Verwendet wird das Connectionless Mode Protocol für Kommunikation innerhalb eines LAN mit Benutzung des Inactiv Network Layer Protocol, falls keine Auftrennung der Dateneinheit erforderlich ist und der Zielknoten mit dem dabei vereinfachten Kontrollblock adressierbar ist.

Verbindungskontroll-Schicht:

LAN-Logical Link Control ISO DIS 8802/2 (entspricht IEEE 802.2) mit LLC-Typ 1 und LLC-Typ 3 (in MAP 3.0)

Mediumzugriffs-Schicht:

LAN-Token-passing Bus Access Method ISO DIS 8802/4 (entspricht IEEE 802.4)

Phys. Signal-Schicht:

Breitband-Netz mit 10 Mbps gemäß ISO DIS 8802/4 (2-Kanal Option)

Basisband-Netz mit 5 Mbps gemäß ISO DIS 8802/4

Es kann grundsätzlich jedes Medium verwendet werden, wenn die in ISO DIS 8802/4 beschriebene Schnittstelle zur Übergabe der MAC-Symbole eingehalten wird. Normen zur Verwendung von Lichtleitern sind in Vorbereitung.

3. Zeitkritische Aufträge in lokalen Netzen

3.1 Grundsatzforderungen

Unter Realzeitbetrieb eines Rechensystems versteht man im allgemeinen eine Betriebsweise, bei der die Aufträge im Rechensystem schritthaltend mit externen Vorgängen abgearbeitet werden, d.h. daß die Verarbeitung simultan zur Datenentstehung durchgeführt wird. Dies bedeutet, daß die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sein müssen (garantierte Antwortzeiten), und daß mehrere Aufträge gleichzeitig zu bearbeiten sind (Mehrprogrammbetrieb). Die Größenordnung der Antwortzeiten ist problemabhängig. Da die einzelnen Aufträge im allgemeinen nicht völlig unabhängig voneinander bearbeitet werden können, ist eine gewisse Kooperation zwischen den beteiligten Rechenprozessen erforderlich.

Die Forderung nach garantierten Antwortzeiten impliziert, daß ein Auftrag nicht beliebig lange von der Bearbeitung ausgeschlossen werden darf. Abhängig von der Art der Kooperation kann es darüber hinaus erforderlich sein, daß Aufträge relativ zueinander einen bestimmten Bearbeitungsfortschritt machen. Es muß also ausgeschlossen werden, daß sich Rechenprozesse gegenseitig blockieren (deadlock) oder aber, daß einem einzelnen Rechenprozess Betriebsmittel auf Dauer vorenthalten werden, weil sie von anderen arbeitsfähigen Rechenprozessen nicht freigegeben werden (starvation). Da in Rechnernetzen kooperierende Prozesse auf verschiedenen Knoten ablaufen können, muß die Kommunikation so gestaltet werden, daß die oben gestellten Forderungen erfüllbar sind. Dabei stellt sich die Frage, welches die kleinstmögliche Antwortzeit ist. Da hier nur die Kommunikationsprobleme behandelt werden, sollen nur die Zeitverhältnisse bei der Kommunikation betrachtet werden. Bei diesen Betrachtungen spielt

die Zugriffsmethode auf das Kommunikationsmedium eine erhebliche Rolle.

3.2 Deterministischer oder nichtdeterministischer Zugriff zum Übertragungsmedium

Für die Regelung des Zugriffs auf das Übertragungsmedium stehen zwei Verfahren zur Diskussion. Bei dem nichtdeterministischen Verfahren "Carrier Sense Multiple Access with Collision Detection" (CSMA/CD) stellt ein sendewilliger Knoten fest, ob das Übertragungsmedium belegt ist. Ist dies nicht der Fall, so beginnt er zu senden. Insbesondere auch wegen der Kabellaufzeiten kann es vorkommen, daß mehrere Knoten zu senden beginnen. Dies führt zu Überlagerungen (Collision) und damit zu einer gegenseitigen Störung. Da die sendenden Knoten ihre eigene Sendung mitempfangen, wird diese Situation erkannt und allen Knoten übermittelt. Die sendenden Knoten müssen nach einer individuellen zufälligen Wartezeit ihre Sendung wiederholen, damit es hierbei möglichst nicht wieder zu Kollisionen kommt.

Bei dem deterministischen Verfahren mit Weitergabe der Sendeberechtigung von Knoten zu Knoten (Token-passing) wird die Sendeberechtigung in Form eines besonderen Zeichens (Token) weitergegeben. Nur wer im Besitz des Zeichens ist kann senden. Kollisionen sind ausgeschlossen. Ein Knoten darf erst wieder senden, wenn das Token einmal bei allen Knoten umgelaufen ist. Das Verfahren garantiert, daß innerhalb der Token-Umlaufzeit jede Station einmal senden konnte.

CSMA/CD ist nicht sehr komplex; allerdings treten bei zunehmendem Verkehr immer häufiger Kollisionen - auch bei der Wiederholung einer Sendung - auf, so daß eine bestimmte Maximalzeit für den Zugriff nicht garantiert werden kann. Das Token-Weitergabe-Verfahren ist komplex, da besondere Vorkehrungen für das Einschleusen neuer Knoten und gegen Verlust oder Duplizierung des Tokens bei übertragungsfehlern getroffen werden müssen.

Beide Verfahren sind in einer Reihe von Arbeiten ausführlich untersucht und verglichen worden. Eine ausführliche Darstellung findet sich z.B. in Schwartz, Telekommunication Networks. In Bild 3.1 ist das typische Verhalten der Verfahren bei unterschiedlichem Verkehr im Netz gegenübergestellt. Es ist darin die mittlere normierte Übertragungszeit - d.h. die Summe aus Zugriffs- und Transferzeit, bezogen auf die Zeit zum Senden des Paketes - in Abhängigkeit vom normierten Durchsatz - d.h. des Durchsatzes der Nutzdaten, bezogen auf die Übertragungsrate - dargestellt. Man erkennt, daß bei CSMA/CD die Mittelwerte der Zugriffszeiten immer größer werden, je größer der normierte Durchsatz ist. Die rechnerisch mögliche Übertragungskapazität (entspricht dem normierten Durchsatz 1) kann nicht erreicht werden, da wegen der immer häufiger werdenden Kollisionen ab einem bestimmten Auslastungsgrad die Nutzdatenrate sinkt. Die Zugriffszeiten können dann beliebig groß werden. Wegen dieses typischen Verhaltens wird in MAP das Token-Weitergabe-Verfahren vorgeschrieben. Die Gründe hierfür sind:

- Bei großer Auslastung des Netzes ist das Token-Weitergabe-Verfahren effizienter.
- Bei Fehlern im Fertigungsbetrieb und anderen Ausnahmesituationen wird ein höheres Kommunikationsbedürfnis erwartet. Gerade dann dürfen die Zugriffszeiten nicht steigen.
- Wenn das Netz besonderen Belastungssituationen gewachsen ist, und hierbei die Realzeitforderungen erfüllt sind, so gilt dies bei weniger kritischen Belastungssituationen erst recht. Effizienzbetrachtungen spielen dann höchstens eine untergeordnete Rolle.

Wie noch gezeigt wird, lassen sich auch beim Token-Weitergabe-Verfahren garantierte Zugriffszeiten nur unter besonderen Bedingungen angeben. Dieses Verfahren ist jedoch bei großer Netzbelastung grundsätzlich günstiger.

Parameter:

10 Mbps übertragungszeit
2 km Segmentlänge
50 Stationen
1 Bit Verzögerung pro Station
24 Bit Steuerinformation
1000 Bit mittlere Datenfeldlänge
bei exponentieller Verteilung

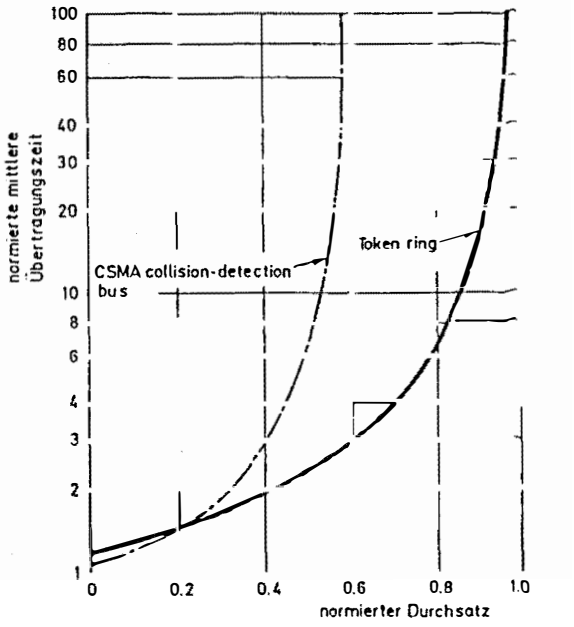


Bild 3.1 Lastverhalten der Zugriffsverfahren

3.3 Charakteristische Zeiten im übertragungsablauf

Ein typischer Kommunikationsablauf besteht darin, daß ein Knoten i eine Botschaft an einen Knoten k sendet und hierauf eine Antwort erwartet, z.B. Daten, die nur der Zielknoten k liefern kann. Hierbei ergibt sich der in Bild 3.2 dargestellte Kommunikationsablauf. Das Anwendungsprogramm richtet eine Anforderung an die Schicht 7, die entsprechend den Protokollen bearbeitet wird und nach der Zeit T_p an die Schicht 2 übergeben wird. T_p hängt von der Komplexität der Protokolle und der gewählten Implementierung ab. Nach Bearbeitung durch die LLC-Schicht (Zeit T_{LLC}) wird die Botschaft von der MAC-Schicht in einen Sendepuffer eingetragen. Wird nun der Knoten i sendeberechtigt, so überträgt er die im Sendepuffer abgelegten Botschaften (in der Reihenfolge ihres Eintreffens). Die Zeit, die dem Knoten zum Senden zur Verfügung steht, wird durch die Token Hold Time T_{HT} bestimmt. Damit

ist die Zugriffszeit T_Z, die von der MAC-Schicht zur Übergabe der Botschaft an das Medium benötigt wird, situationsabhängig. Für den Fall, daß der Sendepuffer bei jedem Token-Durchlauf vollständig geleert wird, ist der Maximalwert für T_V durch die Zeit für einen Token-Umlauf bestimmt (Token Rotation Time T_R). Zwischen T_R und $T_{HT,i}$ der einzelnen Stationen besteht ein Zusammenhang:

$$T_R = T_{Rmin} + \sum_{i=1}^{N} T_{HT,i} \qquad (3.1)$$

Dabei ist T_{Rmin} der Minimalwert für T_R, der sich bei einem Token-Umlauf ohne den Austausch von Nutzdaten ergibt. N ist die Anzahl der Stationen im Netz. Kann der Sendepuffer bei einem Token-Durchlauf nicht geleert werden, so lassen sich auch keine garantierten Zugriffszeiten angeben.

Die Zeit T_{PHY} für die Übertragung auf dem Medium ist durch die Übertragungsrate und die Länge der Botschaft bestimmt. Beim Knoten k (Empfänger) wird die Botschaft von der MAC-Schicht in einen Empfangspuffer eingetragen. Die Verweilzeit T^*_V in der MAC-Schicht des Empfangsknoten hängt von der Belastung dieses Knoten ab. Die Bearbeitungszeiten in den verschiedenen Protokollschichten sind mit T^*_{LLC} (LLC-Schicht) und T^*_p (Schicht 3 bis 7) bezeichnet. Nach Bearbeitung durch ein Anwendungsprogramm (Zeit T_B) steht das Ergebnis zur Rückübertragung bereit. Im 2. Halbzyklus wiederholt sich der beschriebene Ablauf, nur daß jetzt der Knoten k der Sendeknoten und Knoten i der Empfangsknoten ist. Es bleibt festzuhalten, daß der garantierte Zeitbedarf für beide Halbzyklen zusammen nicht kleiner als die zweifache Token-Umlaufzeit gehalten werden kann, und daß dieser Wert ohne besondere Maßnahmen auch nicht garantiert werden kann.

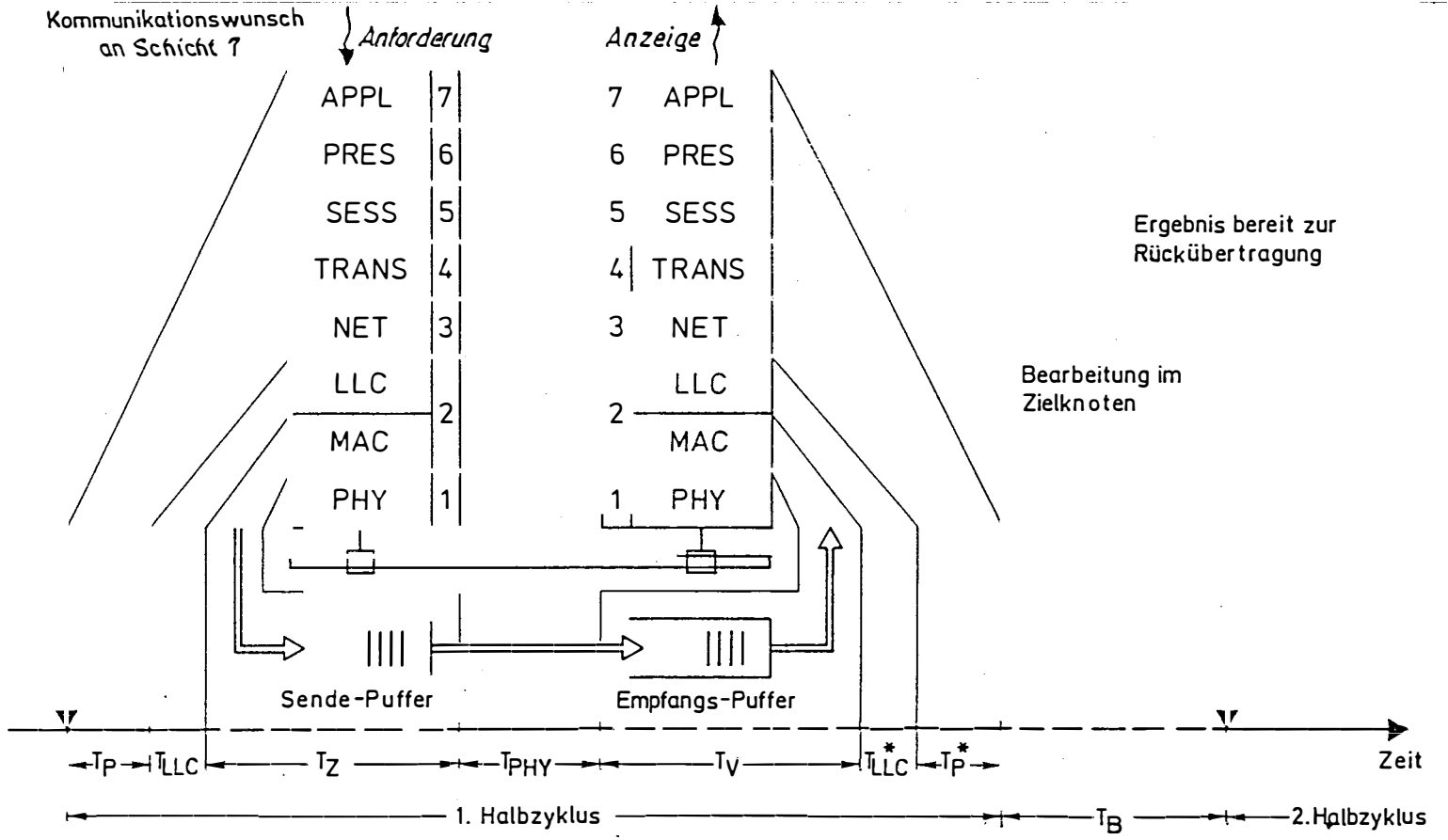


Bild 3.2 Typischer Übertragungsablauf

4. Realzeiteigenschaften der MAP-Protokolle

4.1 Realistische Erwartungen

In Abschnitt 3.3 sind bereits grundsätzliche Betrachtungen angestellt worden. Zusammenfassend gilt:

- Die garantierte Zugriffszeit T_{Vmax} kann nicht kleiner als die Token-Umlaufzeit T_R sein.

- Voraussetzung für eine garantierte Zugriffszeit ist, daß die Sendepuffer vollständig geleert werden, wenn ein Knoten sendeberechtigt wird.

- Für einen Kommunikationszyklus mit Rückübertragung des Ergebnisses können auch unter sonst optimalen Verhältnissen bis zu zwei Token-Umläufe vergehen.

Die Frage stellt sich also, welche obere Grenze der Token-Umlaufzeit für den Verwendungsbereich erforderlich und bei der verfügbaren Technologie auch erreichbar ist. Geht man einmal davon aus, daß in einem MAP-Segment für Realzeitaufgaben nicht mehr als 100 Knoten angeschlossen sind ($N \leq 100$) und verteilen sich die Kommunkationsbedürfnisse gleichmäßig, so kann man bei T_R = 100 ms mit einer Token Hold Time $T_{TH} \approx 1$ ms rechnen. Dies scheint technologische Randbedingungen nicht zu sprengen. In den PROWAY-Anforderungen erscheint jedoch die magische Zahl von 20 ms für die garantierte Zugriffszeit T_{Vmax}. Dies bedeutet, daß $T_R \leq 20$ ms sein muß. Für unser Beispiel ($N \leq 100$) ergibt dies eine Token Hold Time $T_{HT} \approx 0{,}2$ ms und es ist fraglich, ob das Leeren der Sendepuffer in dieser Zeit garantiert werden kann. Man kann also feststellen, daß bei der heute verfügbaren Technologie Realzeiteigenschaften, die als notwendig erachtet werden, mit dem für MAP zunächst vorgesehenen Verfahren nicht bzw. nur bei kleinen Netzen erreicht werden können. Es wurden da-

her Modifikationen ausgearbeitet, die zu einer erheblichen Verbesserung der Realzeiteigenschaften führen. Diese Modifikationen werden in MAP 3.0 als Optionen enthalten sein.

Die in MAP 3.0 vorgesehenen Verbesserungen der Realzeit-Eigenschaften haben zwei Zielrichtungen. Zum einen wird durch Modifikationen im Zugriffsverfahren erreicht, daß für gewisse Klassen von Botschaften die zeitliche Abwicklung verbessert wird. Zum anderen wird ein direkter Zugang aus dem Anwendungsprogramm zur Schicht 2 geschaffen. Hierdurch können die Verzögerungen durch die Protokollabwicklung erheblich reduziert werden. Diese Maßnahmen haben allerdings nicht nur positive Aspekte, wie im nachfolgenden dargestellt wird.

4.2 Modifikationen im Zugriffsverfahren

Die erste Modifikation betrifft das Einführen von Prioritäten beim Zugriff auf das Übertragungsmedium. Es wird dann nur für bevorzugte Botschaften im Sendepuffer garantiert, daß sie bei einem Token-Umlauf tatsächlich gesendet werden. Hierbei verwaltet die MAC-Schicht getrennte Sendepuffer für 4 Prioritäten, die in der Norm als Zugriffsklasse 6, 4, 2 und 0 bezeichnet werden. Es ist etwa an die folgende Verwendung dieser Prioritäten gedacht:

Priorität 6 : dringende Botschaften; nur für diese Priorität wird eine maximale Zugriffszeit garantiert.

Priorität 4 : normale Botschaften zur Prozesssteuerung

Priorität 2 : Betriebsdatenerfassung und Anzeige

Priorität 0 : Datei- und Programmtransfer

Die Sendepuffer für jede Priorität bilden eine selbständige Warteschlange und die Sendeberechtigung (Token) wird knoteninternen von der Priorität 6 zur nächst niedrigeren weitergereicht. Dabei arbeitet die Prioritätensteuerung etwa folgendermaßen:

Die Token Hold Time wird zunächst auf den Wert T_{HTE} (high priority token hold time) gesetzt. Es wird unterstellt, daß diese Zeit für eine Anwendung so gewählt ist, daß alle im Sendepuffer der Priorität 6 befindlichen Botschaften gesendet werden können. Für jede Priorität wird eine gewünschte Token-Umlaufzeit (T_{RT4}, T_{RT2}, T_{RT0}) definiert. Für die Prioritäten 4, 2 und 0 wird jeweils die Zeit festgehalten, die seit der letzten Weitergabe des Tokens vergangen ist (T_{RC4}, T_{RC2}, T_{RC0}). Steht nun eine Priorität $i < 6$ zur Bedienung an und ist für diese die gewünschte Token-Umlaufzeit größer als die Zeit seit der letzten Token-Weitergabe, so wird die Token-Hold-Time gleich der Differenz gesetzt:

$$T_{HT} = T_{RTi} - T_{RCi} \qquad (3.2)$$

Andernfalls wird das Token sofort weitergegeben. Damit man für die verschiedenen Prioritäten tatsächlich verschiedene Bedienqualitäten erhält, sollte folgende Regel gelten:

$$T_{RT4} \geq T_{RT2} \geq T_{RT0} \qquad (3.3)$$

Man könnte nun zu dem Schluß gelangen, daß die garantierte Zugriffszeit für die Priorität 6 durch

$$T_{V6,max} = \max (N*T_{HT6}, T_{RTi}) \qquad (3.4)$$

gegeben ist. Dies ist jedoch nicht ganz korrekt, weil die Zeitverhältnisse nur beim Beginn der Übertragung einer Botschaft geprüft werden. So können also z.B. lange Botschaften zu Abweichungen von (3.4) führen.

Eine zweite Modifikation im Zugriffsverfahren trägt dem Umstand Rechnung, daß ein Knoten erst dann eine Antwort auf eine empfangene Botschaft senden kann, wenn er sendeberechtigt wird. Will man z.B. nur sicher sein, daß eine Botschaft angekommen ist, so ist die hiermit verbundene Wartezeit lästig. Es wird daher die Möglichkeit einer unmittelbaren Antwort (immediate response, LLC-Type 3) eingeführt. Hierbei antwortet ein Knoten auf eine eingehende Botschaft mit Aufforderung zur Antwort sofort, während der ursprünglich sendende Knoten seine Sendeberechtigung behält. Der Zeitbedarf für die Antwort muß also bei der Token Hold Time berücksichtigt werden und kann zu einer Verlängerung der Token-Umlaufzeit führen.

Die zurückzusendende Antwort muß in der LLC-Schicht des angesprochenen Knotens bereits vorrätig sein. Das Verfahren eignet sich also zur Spiegelung (Echo) der gerade eingegangenen Botschaft als Empfangsquittung oder aber zur Abfrage hinterlegter Botschaften und könnte auch Commit-Verfahren für verteilte Transaktionen beschleunigen.

4.3 Der direkte Zugang zur Schicht 2

Aus der Sicht des Anwendungsprogrammes besteht der Zeitbedarf für den Botschaftsaustausch nicht nur aus der Zeit für den Zugriff auf das übertragungsmedium in der MAC-Schicht, sondern auch aus der Durchlaufzeit durch die anderen Protokollschichten (vgl. Bild 2.3). Diese Zeiten sind keineswegs vernachlässigbar klein. Es ist daher vorgeschlagen worden, daß Anwendungsprogramme für besonders zeitkritische Aufgaben einen direkten Zugang zur Schicht 2 erhalten. Dies setzt allerdings voraus, daß die beteiligten Knoten am selben Netzsegment liegen, da wegen der fehlenden Netzwerk-Schicht Knoten über die Segmentgrenze hinaus nicht adressierbar sind. Es werden nicht nur die Druchlaufzeiten durch die Schichten 3 bis 7 gespart, sondern es ergibt sich auch

eine Reduktion der zu übertragenden Daten, da keine Daten zur Steuerung der Protokollabwicklung übertragen werden müssen.

Der direkte Zugang zur Schicht 2 bringt zwei Problembereiche mit sich. Zum einen wird man wegen der fehlenden Schichten 3 bis 7 dem Anwendungsprogramm nicht die gleichen Funktionen anbieten können wie bei einem vollen Protokoll. Der Anwender wird also unterscheiden müssen, ob er auf einer 2-Schichten-Architektur oder einer 7-Schichten-Architektur arbeitet. Der andere Problembereich entsteht durch die wünschenswerte Koexistenz von beiden Architektur-Modellen auf dem gleichen Netzwerk-Segment.

In MAP 3.0 sind speziell für Realzeitaufgaben Basis-Band-Segmente vorgesehen. Ein Mini-MAP-Knoten (vgl. Bild 4.1) enthält nur die Schichten 1 und 2 sowie eine eingeschränkte Anwendungs-Schicht (Schicht 7). Daneben gibt es Knoten mit erweiterter Funktionalität (Enhanced Performance Architecture, EPA). Diese haben mehrere Zugriffspunkte zur LLC-Schicht. Genau einer davon wird für die volle Architektur (Schicht 3 bis 7) zur Verfügung gestellt. Die anderen dienen zum direkten Zugriff der Anwendungsprogramme über eine eingeschränkte Schicht 7. Ein Mini-MAP-Knoten kann nur mit einem Mini-MAP-Knoten oder einem EPA-Knoten des selben Segmentes Botschaften austauschen. Ein EPA-Knoten dagegen kann dank seiner vollen Architektur auch mit anderen Netz-Segmenten kommunizieren.

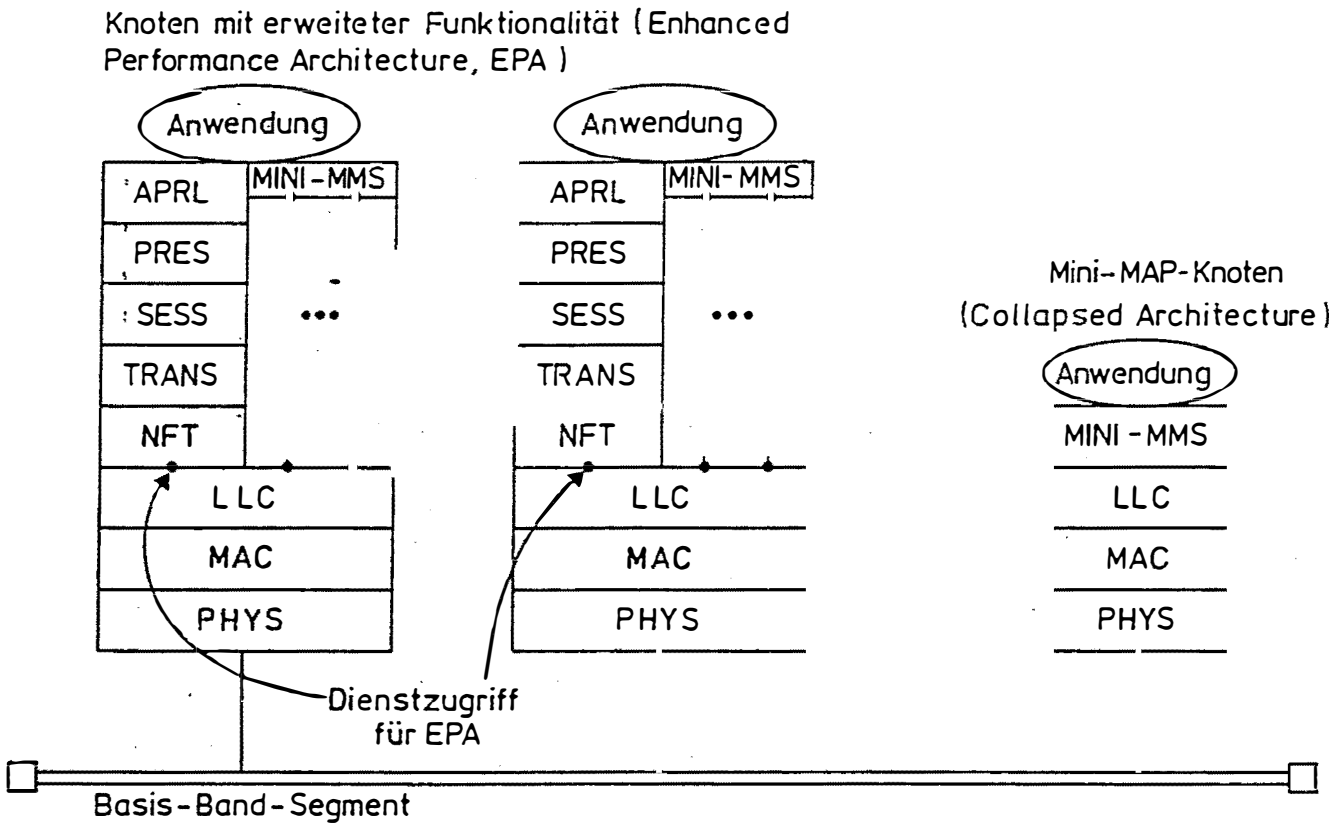


Bild 4.1 Mini-MAP und Knoten mit erweiteter Funktionalität (EPA-Knoten)

5. Die Verwendung von MAP-Komponenten

5.1 Verfügbarkeit und Anwendungsbereiche

MAP-Netze auf der Basis der seitherigen Verison 2.1 sind bisher nur als Studien- bzw. Pilotprojekte realisiert worden. Die Anwender warten im wesentlichen auf die Version 3.0, die bis zum Enterprise Network Event im Juni 1988 in Baltimore verfügbar sein wird. MAP bietet dem Anwender eine Reihe von anwendungsorientierten Funktionen, Dateizugriffsdienste und Netzwerkmanagement-Dienst an, auf die hier wegen des Schwerpunktes auf Realzeiteigenschaften nicht näher eingegangen werden kann. MAP 3.0 wird die Bedürfnisse zur Kommunikation im Fertigungsbereich entsprechend dem heutigen Stand der Technologie abdecken. Seine Druchsetzung am Markt wird neben den Kosten wesentlich von der tatsächlich erreichten Interoperabilität der Komponenten verschiedener Hersteller abhängen. Um dies zu gewährleisten werden neutrale Testzentren aufgebaut, die MAP-Komponenten auf ihre Normenkonformität prüfen. Die zur Demonstration beim Enterprise Network Event verwendeten Komponenten werden bereits gemäß MAP 3.0 getestet sein.

Die für den Anwender hauptsächlich wichtige Interoperabilität hängt jedoch nicht nur von der Güte der Konformitätsprüfungen ab sondern auch davon, ob in der Norm selbst alle wesentlichen Punkte für die Interoperabilität ausreichend geregelt sind. Man kann davon ausgehen, daß der gefundene Konsens unter den Fachleuten in den Normungsgremien und die gesammelten Erfahrungen mit früheren Versionen die Gewähr dafür gibt, daß alle essentiellen Punkte in MAP 3.0 berücksichtigt sind.

5.2 MAP und PEARL

Der im Mehrrechner-PEARL beschriebene Botschaften-Mechanismus ist in einem MAP-Netz (auch Mini-MAP) direkt implementierbar. Für das Blocking-Send-Protokoll kann z.B. der LLC-Typ 3 (vgl. Abschnitt 4.2) mit Vorteil verwendet werden. Auch läßt sich eine 1-zu-n-Kommunikation durch geeignete Gruppen-Adressierung realisieren.

MAP stellt jedoch eine Reihe von anwendungsorientierten Funktionen zur Verfügung, die über den eigentlichen Botschaftenaustausch hinausgeht. Es sollte eine einheitliche Darstellung des Aufrufs dieser Dienste mit Hilfe von Sprachelementen von PEARL (z.B. Datentypen, Prozeduraufrufe u.a.) gefunden werden (Language Binding). PEARL-Programme könnten dann auch bei unterschiedlichen Kompilierern in MAP-Netzen portabel sein.

Literatur:

Giese, E.; K. Görgen; E. Hirsch; G. Schulze; K. Truol:
Dienste und Protokolle in Kommunikationssystemen - Die Dienst- und Protokollschnittstellen der ISO-Architektur, Springer-Verlag 1985

Janetzky, D.; K.S. Watson:
Token Bus Performance in MAP and RPOWAY;
IFAC Workshop on Distributed Computer Control Systems, Mayschoss (FRG), Sept., 30 - Oct.,2,1986

Kaminski, M.:
Protocols for Communicating in the Factory;
IEEE Spectrum, April 1986

MAP 2.1:
General Motors Corporation, General Motors Technical Center, Warren, Michigan

MAP User's Group Meeting Summary:
MAP user's group meeting Sept. 10 - 11,1985, Anaheim, California; North-Holland, 1986

PROWAY C; IEC Process Data Highway:
IEC Publication 955, 1986

Schwartz, M.:
Telecommunication Networks;
Addison-Wesley, 1987

Suppan-Borowka, J.; T. Simon:
MAP-Datenkommunikation in der automatisierten Fertigung;
Datacom-Verlag, 1986

Zwoll, K.; H. Halling:
MAP-Automation Manufacturing Protocol;
Proceedings of the IFAC/IFIP-Symposium on Software for Computer Control, Graz, May, 20-23, 1986

Werkzeuge für Realzeit-Anwendungen in der Fertigungsindustrie

Autor: Herbert Mähner
Digital Equipment GmbH
Anwendungszentrum Fertigungsindustrie
Hörselbergstraße 3
8000 München 80

Zusammenfassung

Realzeit-Anwendungen in der Fertigungsindustrie kommen streng genommen nur auf der Zellenrechner- und der Prozeßebene vor. Werkzeuge für diese Fertigungsebenen unterliegen verschiedenartigen Anforderungen. Sie müssen einerseits bestehende, isolierte Echtzeitsysteme in den Datenverbund integrieren, andererseits neue Systeme mit neuster Technologie entwickeln helfen. Dabei soll die Implementierung möglichst prozeßnah und realzeitgerecht sein und hohen Benutzungskomfort gewährleisten. Dieser Beitrag beschäftigt sich mit Entwicklungswerkzeugen für dedizierte Realzeitsysteme und mit Kommunikationswerkzeugen, die den Datentransfer und die Überwachung von verschiedenartigen Prozessen in der Fertigungsindustrie unterstützen.

1.0 EINLEITUNG

Der Einsatz von Computern in der Fertigungsindustrie läßt sich in mehrere Ebenen einteilen (Bild 1), wobei die hierarchische Trennung aus technologischen Gründen nicht unbedingt in verschiedenen Computersystemen resultiert. Das gilt besonders für die unteren Ebenen, wo z.B. ein Zellenrechner durchaus mittlere Realzeit-Aufgaben der Prozeßebene übernehmen kann oder ein Zellenrechner Funktionen ausführt, die eigentlich der Fertigungsleitebene zugeordnet werden können. Durch die steigenden Anforderungen kristallisiert sich jedoch immer mehr eine klarere Trennung heraus, obwohl sicherlich die erheblich verbesserte Leistung der Mikrocomputer und anwenderspezifischen Randbedingungen diesen Trend noch verzögern können. Bestehende Rechnerstrukturen in der Fertigungsindustrie sind gemäß der spezifischen Anwenderaufgaben gewachsen, große Investitionen sind in Hardware- und Software-Erstellung eingebracht worden und sollen möglichst geschützt werden.

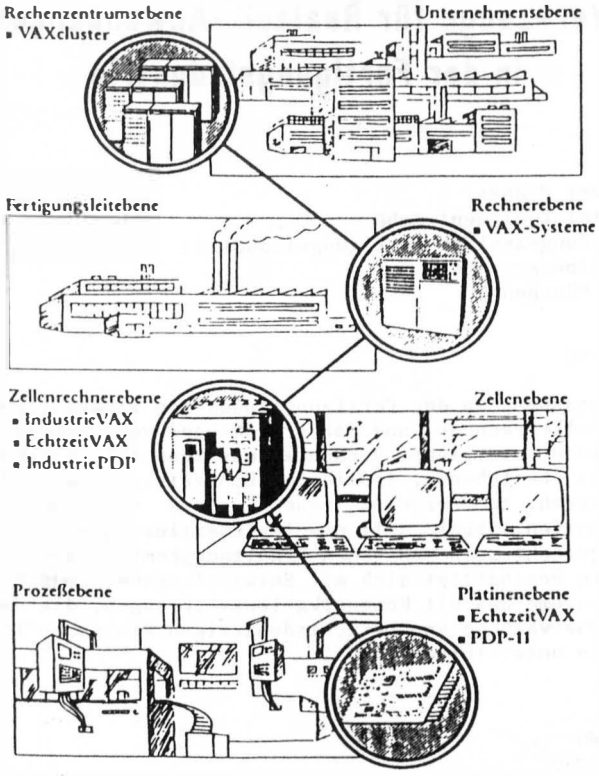


Bild 1 : Rechner-Hierarchie in der Fertigungsindustrie

Gerade Werkzeuge, die auf den unteren Ebenen der Fertigungshierarchie eingesetzt werden sollen, müssen diesem Umstand Rechnung tragen.

Der folgende Beitrag beschreibt Software-Bausteine der Firma Digital Equipment für Echtzeitanwendungen in der Fertigungsindustrie, die in den beiden unteren Ebenen der Fertigungshierarchie eingesetzt werden können. Integrationsbausteine für Produktionsplanung, CAD/CAM und (Produktions-)Datenbanken werden nicht behandelt.

2.0 Entwicklungswerkzeuge für dedizierte Realzeit-Anwendungen

Dedizierte Ablaufsysteme haben gegenüber generellen Betriebssystemen den Vorteil, daß sie speziell für eine bestimmte Anwendung konfigurierbar sind, keinen Betriebssystem-"Overhead" zeigen, eine hohe, vorhersagbare Realzeitleistung erbringen und ohne Plattenspeicher ablauffähig sind.

Die Anwendung wird in einer hohen Programmiersprache mit Realzeit-Spracherweiterungen prozeßnah implementiert. Diese Systeme kommen verstärkt für neue Hardwareprodukte und Anwendungen zum Einsatz, wie z.B. Netzwerk-Serverprodukte, intelligente Substationen, generische Zellenrechner, Kommunikations- und Firmware-Software. Die jeweiligen Entwicklungswerkzeuge für PDP- und VAX-Rechner werden im folgenden beschrieben.

2.1 MicroPower/Pascal-Entwicklungswerkzeug

MicroPower/Pascal ist ein Softwareentwicklungswerkzeug für PDP-Zielsysteme. Auf der Basis einer erweiterten Implementierung der Programmiersprache PASCAL und einem integralen Ablaufbetriebssystem werden dedizierte Anwendungen realisiert. Durch die Integration des Entwicklungswerkzeuges in die VAX/VMS-Umgebung einerseits und die optimalen Ablaufeigenschaften des Zielsystems andererseits, wird eine optimale Synthese von Entwicklungsumgebung und Ablaufeigenschaften erreicht. Für Entwickler in einer PDP-11-Umgebung ist das Werkzeug auch für MicroRSX oder RSX und RT-11 erhältlich.

Merkmale

* Integration in die VAX/VMS Entwicklungsumgebung
* Unterstützung des PASCAL-ISO-7185-Standards und Spracherweiterungen durch einen leistungsfähigen Pascal-Compiler
* Integrales Ablaufsystem optimiert für Echtzeitbedingungen
* Unterstützung von dynamischer, paraller Programmausführung mit dynamischen Prozessen und Prioritäten, kein Zeitscheibenbetrieb
* Dynamische Systemstrukturen zur Synchronisation und Kommunikation auch für netzwerkweiten Einsatz (LAN)

* direkte Programmierung von Hardware-Schnittstellen, Bitmanipulation und Anbindung von Interrupt-Service-Routinen in PASCAL

* Modulare, konfigurierbare Zielsysteme für sämtliche Q-Bus PDP-11 Systeme und PDP-11 Ein-Platinen-Computer (KXT11-BA,-CA,KXJ11-CA)

* Optionale Ein-/Ausgabe-Dienstleistungsprogramme, Dateiverwaltung und Netzwerkservice (DECnet kompatibel)

* Symbolische Testhilfen durch das Hilfsprogramm PASDBG

* Anwendungsgenerierung durch MPBUILD-Kommandoprozeduren

Die erweiterte PASCAL-Implementierungssprache basiert auf dem ISO-7185-Standard und hat zusätzliche Datenstrukturen und Betriebssystemfunktionen implementiert, die einen dynamischen Mehrprogrammablauf (multiprogramming, multitasking) erlauben. Die Vorteile von PASCAL als strukturierte Programmiersprache und die damit verbundenen Erleichterungen in der Kodier-, Test- und Wartungsphase einerseits, sowie die Möglichkeiten der PASCAL-Spracherweiterungen haben MicroPower zu einem effizienten Softwarewerkzeug werden lassen.
Die Spracherweiterungen und integrierten Systemfunktionen machen mehr als den doppelten Sprachumfang des Standard-PASCAL. Der PASCAL-Compiler erlaubt eine Vielzahl von Optionen, z.B. zur Anzeige der Spracherweiterungen, für verschiedene Laufzeit- und Instruktionsbedingungen sowie Debug- und Statistik-Informationen.

MicroPower/Pascal ist so konzipiert, daß eine Gesamtanwendung in Module zerlegt und einzeln compiliert werden kann. Andererseits können die verschiedenen Aufgaben der Anwendung in einzelne Programme und dynamische Prozesse gegliedert werden, die dann über die Systemfunktionen miteinander synchronisiert werden und/oder kommunizieren.

Die verschiedenen Objektmodule der einzelnen Programme werden durch die MPBUILD-Kommandoroutinen zusammen mit dem Ablaufbetriebssystem zu einem ablauffähigen Gesamtsystem gebunden, das für den Echtzeitbetrieb optimiert ist. Durch den speicherresidenten Betrieb sind die Reaktionszeiten vorhersagbar, auch für eine Vielzahl von Programmen. Das Gesamtsystem kann aus dem EPROM gestartet oder über Datenträger (Platte, Band),

sowie Übertragungsmedium (serielle Leitung, Ethernet) geladen und ausgeführt werden. Das Ablaufbetriebssystem ist modular für sämtliche Q-Bus-PDP-11 Systeme, Single-Board-Computer und Ein-/Ausgabe-Computer konfigurierbar. Damit werden auch Multiprozessorsysteme mit einer Master-CPU und mehreren IOP-Prozessoren unterstützt (Bild 3).
Eine Vielzahl von peripheren Schnittstellen- und Dienstprogrammen können in das Gesamtsystem bei Bedarf integriert werden und so die Größe des Systems ganz auf die Erfordernisse der Anwendung abgestimmt werden. Der optionale Dateiservice ermöglicht z.B. Zugriff auf die gängingen Plattenspeicher, während der Netzwerkservice, z.B. über Ethernet, für eine Kommunikation zu VAX/VMS-,VAXELN- oder MicroRSX/RSX-Systemen sorgt (DECnet kompatibel).

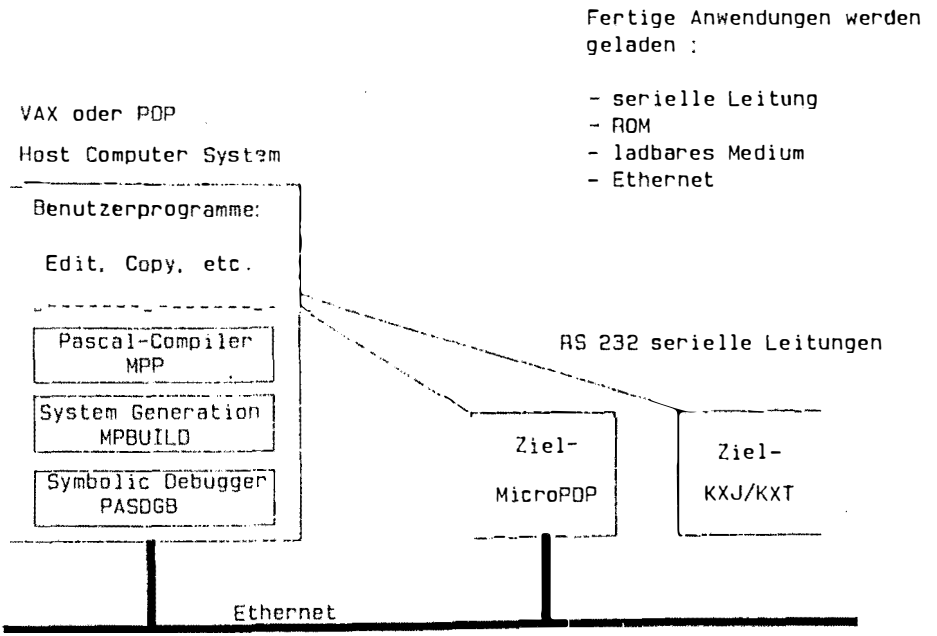


Bild 2: MicroPower/Pascal Entwicklungsumgebung

Während der Entwicklungsphase gilt das sog. Host-Target-Entwicklungsprinzip, wobei die Entwicklung z.B. auf einem VAX/VMS-System geschieht (Editieren, Compilieren und Systembinden), während das Testen über eine serielle Leitung mittels symbolischem Debugger PASDBG im angeschlossen

Zielsystem selbst abläuft (Bild 2). Damit hat man eine optimale Entwicklungs- und Ablaufumgebung miteinander kombiniert. PASDBG erlaubt direktes symbolisches Testen des Systemes, z.B. mit der Anzeige der derzeitigen Prozesszustände, Variablen und Systemstrukturen sowie den Verweis auf den derzeitigen PASCAL-Quellenkode.

2.2 VAXELN-Toolkit-Entwicklungswerkzeug

VAXELN ist ein Softwareentwicklungswerkzeug für MicroVAX- und VAX-Zielsysteme. Auf der Basis einer erweiterten Implementierung der Programmiersprache EPascal und einem integralen Ablaufbetriebssystem werden dedizierte Anwendungen realisiert.

Merkmale

* volle Integration in die VAX/VMS-Umgebung
* Unterstützung des PASCAL-ISO-7185-Standards und Spracherweiterungen durch den leistungsfähigen EPascal Compiler
* Unterstützung von VAX C durch die C-Laufzeitbibliothek und von FORTRAN durch eine Untermenge der VMS-Fortran-Laufzeitbibliothek
* Integrales Ablaufsystem optimiert für Echtzeitbedingungen
* Unterstützung von dynamischer, paraller Programmausführung mit dynamischen Prozessen und Prioritäten, kein Zeitscheibenbetrieb
* Nachladen von vollständigen Programmen über Netzwerk oder Massenspeicher
* Dynamische Systemstrukturen zur Synchronisation und Kommunikation für transparenten, netzwerkweiten Einsatz (LAN)
* direkte Programmierung von Hardware-Schnittstellen, Bitmanipulation und Anbindung von Interrupt-Service-Routinen in EPascal
* Modulare, konfigurierbare Zielsysteme für die VAX-11/725/730/750, MicroVAX (I und II), IndustrieVAX, VAX-8500/8550/8700 , RT-VAX 1000 und Echtzeit-VAX CPU-Modul KA620
* Optionale Ein-/Ausgabe Dienstleistungsprogramme, Dateiverwaltung und Netzwerkservice (DECnet)
* Symbolische Testhilfen durch das Hilfsprogramm EDEBUG
* Anwendungsgenerierung durch das Hilfsprogramm EBUILD

Die erweiterte EPascal-Implementierungssprache basiert auf dem ISO-7185-Standard und hat zusätzliche Datenstrukturen und Betriebssystemfunktionen implementiert, die einen dynamischen Mehrprogrammablauf (multiprogramming, multitasking) erlauben. Der EPascal-Compiler erlaubt eine Vielzahl von Optionen und erlaubt ein effektives Einbinden von Benutzerbibliotheken und Modulen mit z.B. den IMPORT-, EXPORT-Anweisungen.

VAXELN ist so konzipiert, daß eine Gesamtanwendung in Module zerlegt und einzeln kompiliert werden kann, ohne Beeinträchtigung der Konsistenz der Software (sog. Type-checking über Modulgrenzen). Andererseits können die verschiedenen Aufgaben der Anwendung in einzelne Programme und dynamische Prozesse gegliedert werden, die dann über die Systemfunktionen miteinander synchronisiert werden und/oder kommunizieren. Dadurch kann nach

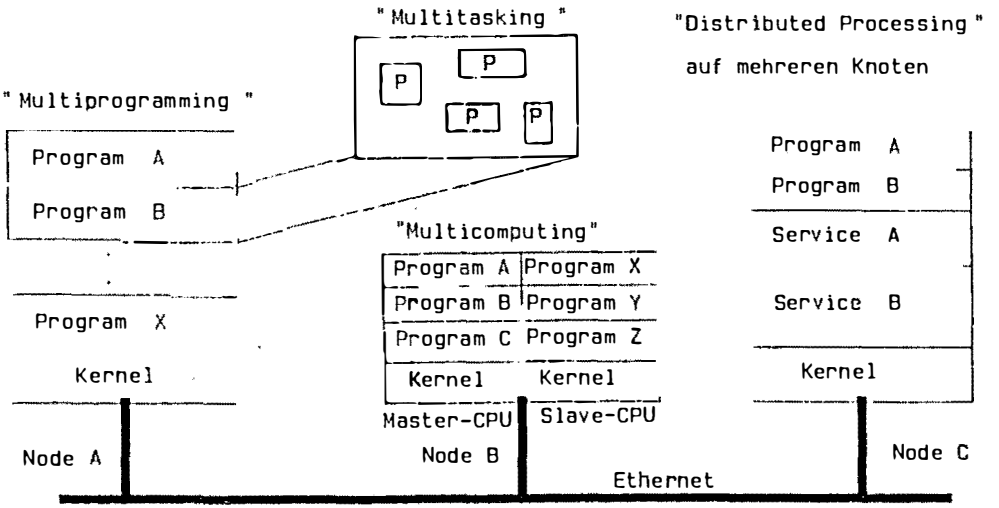


Bild 3: Aufbau von VAXELN oder MicroPower Systemen

den heutigen Gesichtspunkten der Softwareentwicklung kodiert, getestet und Softwarewartung betrieben werden. Dabei wird auch die Aufteilung einer Anwendung über mehrere Knoten vom System transparent unterstützt (Bild 3).

EPASCAL ist die Implementierungssprache für die Service- und Schnitt-

stellenprogramme, die in Quellenform zum Selbststudium Teil des Toolkits sind und damit z.B. die Möglichkeiten der hardwarenahen Programmierung aufzeigen. Die verschiedenen Objektmodule der einzelnen Programme werden mit dem VMS-Linker und den entsprechenden Laufzeitbibliotheken zu Programmen zusammengebunden, die dann durch das EBUILD Hilfsprogramm zusammen mit dem Ablaufbetriebssystem zu einem ablauffähigen Gesamtsystem gebunden werden. Das ablauffähige System ist für den Echtzeitbetrieb optimiert. Durch den speicherresidenten Betrieb sind die Reaktionszeiten voraussagbar und gleichbleibend auch für ein System mit einer Vielzahl von Programmen. Das Gesamtsystem kann aus dem EPROM gestartet oder über Datenträger (Platte, Band) sowie Übertragungsmedium (Ethernet) geladen und ausgeführt werden.

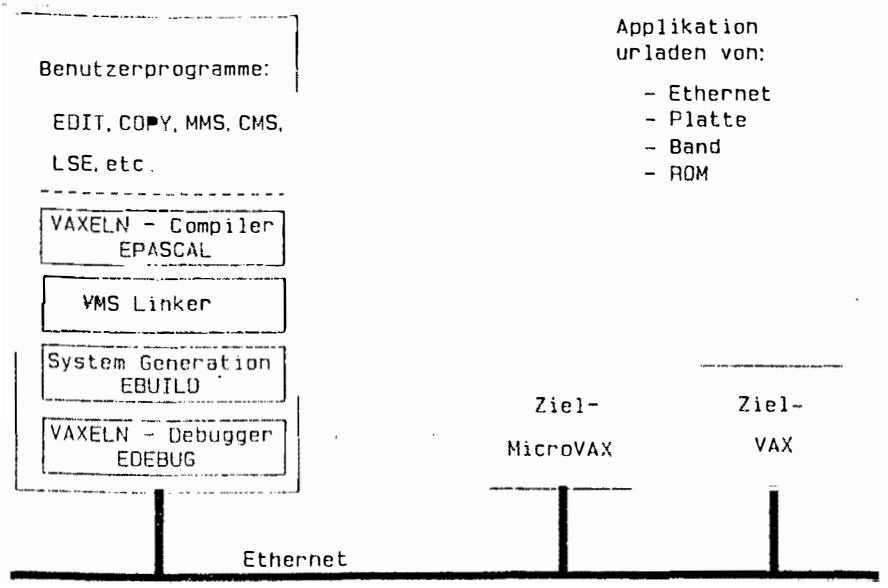


Bild 4: VAXELN-Entwicklungsumgebung

Eine Vielzahl von peripheren Schnittstellen- und Dienstprogrammen können in das Gesamtsystem bei Bedarf integriert werden und so die Größe des Systems ganz auf die Erfordernisse der Anwendung abgestimmt werden. Der optionale Dateiservice ermöglicht den Zugriff auf Plattenspeicher, während der Netzwerk-Service über Ethernet für eine Kommunikation zu VAX/VMS-, MicroRSX/RSX- oder MicroPower-Systemen sorgt (DECnet kompatibel). Weiterhin erlaubt VAXELN eine transparente Verteilung der An-

wendung auf mehrere Knotensysteme im LAN. Die Filestruktur ist VAX/VMS kompatibel, wobei die Datei auf einem lokalen Plattenspeicher liegt oder netzwerkweit Über Ethernet auf einem VAXELN- oder VAX/VMS-System zugreifbar ist.

Während der Entwicklungsphase gilt das sog. Host-Target-Entwicklungsprinzip, wobei die Entwicklung auf einem VAX/VMS-System geschieht (Editieren, Compilieren und Systembinden), während das Testen Über das Ethernet mittels symbolischem Debugger EDEBUG im angeschlossen Zielsystem selbst abläuft (Bild 4), wobei mehrere Systeme gleichzeitig getestet werden können. Damit hat man eine optimale Entwicklungs- und Ablaufumgebung miteinander kombiniert. EDEBUG erlaubt direktes symbolisches Testen des Systemes mit z.B. Verweis auf den derzeitigen PASCAL-Quellenkode, Anzeige der Prozesszustände, Variablen und Systemstrukturen.

3.0 Realzeit-Kommunikationswerkzeuge

Ein wichtiges Element bei komplexen Realzeit-Anwendungen in der Fertigungsindustrie ist die Vernetzung der Computersysteme untereinander, die man grob in die Ebenen Administration, Zellebene und Feldbusebene einteilen kann. Für Realzeit-Anwendungen sind vorwiegend die Zell- und Feldbusebenen von Interesse. In der Zellebene hat sich MAP als Kommunikationsstandard noch nicht etablieren können, da erst MAP Version 3.0 als allgemeiner Standard für die Fertigungsindustrie interessant ist. Das liegt an den erheblichen Verzögerungen bei der Verabschiedung der Norm einerseits und andererseits an der Erkenntnis, daß das Standard-MAP-Protokoll für die Anbindung von Realzeit-Anwendungen kaum geeignet ist. Parallel dazu hat sich gezeigt, daß die Basisband-Technologie (Ethernet) auch in der Fertigung erfolgreich eingesetzt wird, z.B. die Anbindung von SIEMENS S5-Steuerungen (SINEC H1).

Von entscheidender Bedeutung für Realzeit-Anwendungen ist die Feldbusebene, die die Zellenrechnerebene mit der Prozeßebene verbindet. In der Prozeßebene gibt es eine Vielzahl von verschiedenster Peripherie, die

über serielle Leitungen, Bus-Systeme oder direkt mit dem Zellenrechner verbunden ist (Bild 5), wobei die Anzahl der Übertragungsprotokolle für für die einzelnen Verbindungen ebenso vielfältig ist. Der folgende Abschnitt behandelt die Anbindung von Prozeßperipherie, den BITBUS und SINEC-H1.

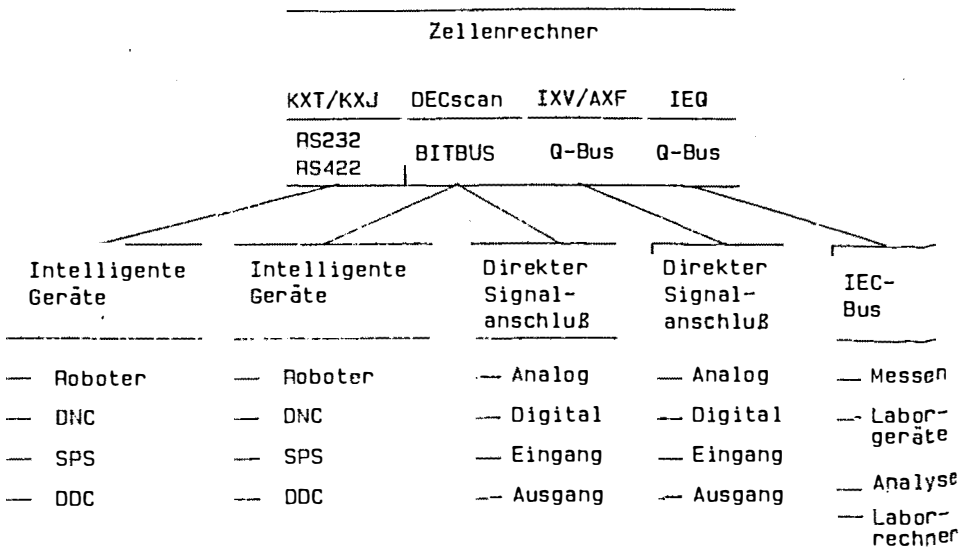


Bild 5: Mögliche Prozeßanschlüsse mit Zellenrechnern

3.1 KXT/KXJ Ein-/Ausgabecomputer

Der KXT11-CA und KXJ11-CA Ein-/Ausgabecomputer sind selbständige Ein-Platinen-Computer, die asynchron zum Hauptrechner bestimmte Kommunikationsaufgaben, parallele Ein-/Ausgabe oder Datenvorverarbeitung übernehmen. Sie haben eine eigene CPU, RAM/ROM Speicher, Peripherieanschlüsse (parallele, serielle E/A, Timer/Counter) und können über den Q-Bus Daten austauschen bzw. das gesamte System urgeladen. Es können mehrere dieser Geräte auf einem Q-Bus betrieben werden, wobei die Anwendung mit Hilfe von MicroPower/Pascal programmiert wird. Für die Fertigungsindustrie wurden diese Ein-/Ausgabecomputer in mehreren Projekten zur Ankopplung von intelligenten Substationen oder speziellen Prozeßrechnern

benutzt. Als VAX-Dust-S5 wird ein Hardware/Software-Produkt zur Anbindung von SIEMENS S5-Steuerungen (DUST 3964/3964R) an einen MicroVAX Zellenrechner angeboten (Bild 6).

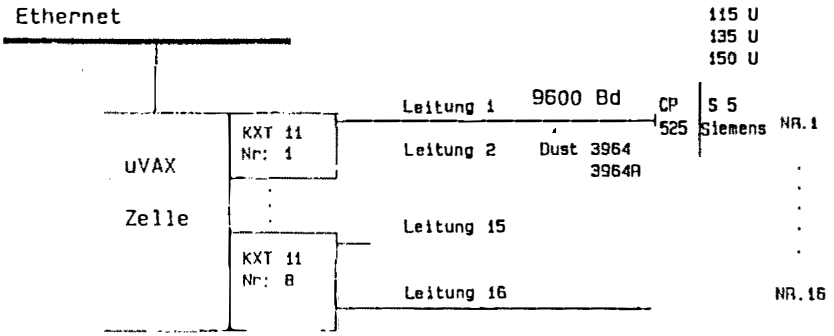


Bild 6: Ankopplung von SIEMENS S5-Steuerungen über KXT E/A-Computer

3.2 DECscan

Neben den verschiedenen seriellen Punkt-zu-Punkt-Verbindungen bieten serielle Bus-Systeme eine Möglichkeit, Prozeßperipherie von einem Zellenrechner aus bedienen zu können. Der BITBUS der Firma INTEL ist ein solcher sog. Feldbus, der in verschiednen nationalen und internationalen Gremien zur Normung ansteht. Der BITBUS ist eine serielle Verbindung (RS485) zu maximal ca. 250 Substationen mit einer maximalen Länge von 12km. Durch integrierte Interface-Chips können Anwender extrem kostengünstig eigene Peripherie an den Bus anbinden. Unter dem Namen DECscan wird von Digital ein Hardware/Software-Produkt angeboten, das die Ankopplung von BITBUS-Geräten an die MicroVAX ermöglicht. Es enthält ein intelligentes Q-Bus/BITBUS Schnittstellen-Modul, die Schnittstellen-Software und ein industrielles Software-Unterstützungswerkzeug (Bild 7).

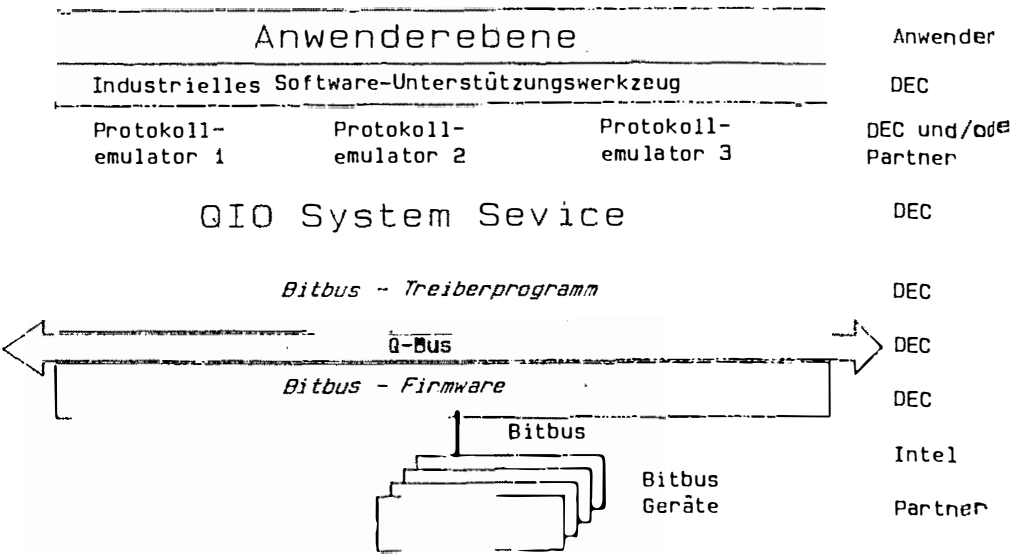


Bild 7: Struktur des DECscan Software-Werkzeugs

In Zusammenarbeit mit Partnerfirmen wird hier für bestimmte Peripheriegeräte eine leicht bedienbare Benutzeroberfläche geschaffen, die mit Hilfe von Bibliotheksfunktionen verschiedene Ein-/Ausgabefunktionen, Regelalgorithmen o.ä. realisieren.

3.3 Baseway

Baseway ist ein komplettes Lösungspaket zur Ankopplung von SPSen (speicherprogrammierbaren Steuerungen) an einen VAX Zellenrechner. Das Design von Baseway zielt auf eine fabrikweite Integration der Steuerungen ab und unterstützt daher den Anschluß von sehr vielen, auch räumlich verteilten Steuerungen. Der sinnvolle Einsatzbereich liegt bei ca. 20 bis 1000 Steuerungen von verschiedenen Herstellern (Bild 8).
Die Baseway Hardware besteht im wesentlichen aus einem VAX Rechner und den dazugehörigen Servern zur Ankopplung der unterschiedlichen SPSen.

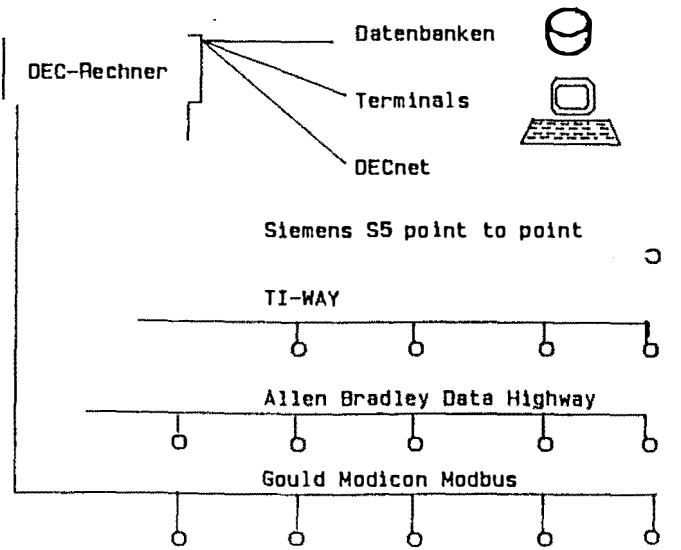


Bild 8: Ankopplung von speicherprogrammierbaren Steuerungen

Die Baseway Software bietet folgende Merkmale :

* Anschluß von unterschiedlichen SPSen, wie Siemens-S5, Allen-Bradley-Data-Highway, Texas Instruments TI-Way, Gould Modicon Modbus
* der Operator spricht alle unterschiedlichen Steuerungen über dieselben, komfortablen Bildschirmmasken an
* alle SPS-Programme werden zentral mit entsprechendem Zugriffsschutz verwaltet und auf Wunsch geladen
* zu allen Funktionen gibt es standardisierte Unterprogrammaufrufe

3.4 SINEC-H1 Ankopplung über Ethernet

Die SIEMENS S5 Steuerung erlaubt mit dem Kommunikationsbaustein CP 535 eine Ankopplung an das Ethernet (SINEC-H1). Die Kopplungssoftware nutzt das OSI-Protokoll (die unteren 4 Ebenen). Im Rahmen einer Projektrealisierung wurde von DEC die Kopplung mehrerer S5-Steuerungen an eine uVAX hergestellt. Die im Bild 9 dargestellte Trennung des SIEMENS-LANs vom DEC-LAN ist für den Wartungs- oder Diagnosefall eine oft gewünschte Lösung.

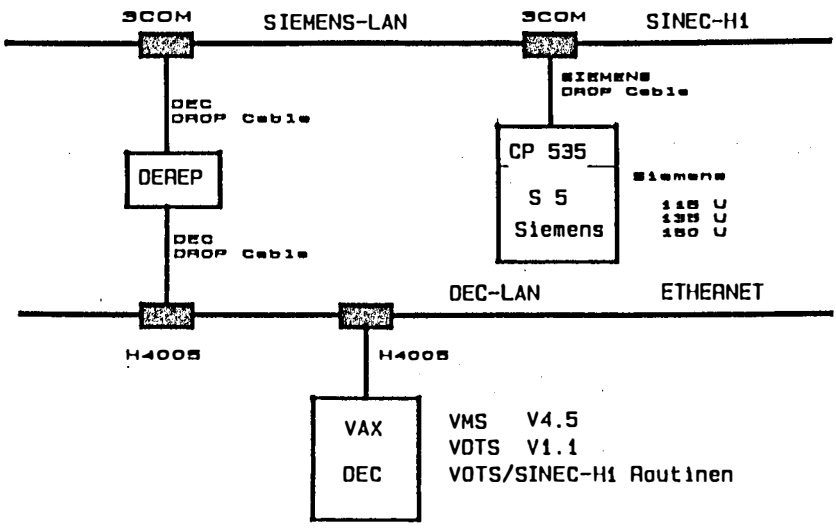


Bild 9: Ankopplung von SIEMENS S5-Steuerungen über Ethernet

Zusammenfassung

Durch die Vielfältigkeit in der Fertigungswelt kann man nicht jede Prozeßperipherie durch fertige Schnittstellensoftware unterstützen. Es bleibt in der Regel eine mehr oder minder große Zusatzleistung an Anpassungssoftware oder eine Komplett-Lösung. Man kann nur hoffen, daß sich in Zukunft viele Anbieter auf gemeinsame Standard-Protokolle einigen, um den Entwicklungsaufwand für neue Peripherie begrenzen zu helfen. Um weltweit die gewonnenen Erfahrungen austauschen zu können, hat DEC zur Unterstützung von Projekten eine Daten- und Informationsdatenbank eingerichtet.

Literatur:

[1] Heinen,Roger, VAX Executive develops realtime applications, Computer Design, p. 79, March 1984
[2] Mähner, Herbert, Einsatz und Leistungsfähigkeit von Echtzeit-Pascal für Mikrocomputer, 11. internationaler Kongress Mikroelektronik, Tagungsbericht Vortrag Nr.481, München 13.-15. 10. 1984
[3] VAXELN Technical Summary, Digital Equipment, 1987
[4] VAX Realtime User's Guide, Digital Equipment, November 1986, EK-VAXRT-UG001
[5] Mähner, Herbert F., MicroPower/Pascal: Programmiersprache und Entwicklungssystem fuer Echtzeitaufgaben, Elektronik, Heft 21, Okt.1984

Ein lose gekoppeltes Rechnersystem für Echtzeitverarbeitung in einem autonomen mobilen Roboter*

Ralf Hinkel, Thomas Knierlemen, Ewald von Puttkamer

Fachbereich Informatik, Universität Kaiserslautern
Erwin Schroedinger Straße ,D-6750 Kaiserslautern, West Germany
Telefon: 0631/205 -2656 oder -2624

Schlüsselwörter:

autonome mobile Roboter, Echtzeitverarbeitung, Multiprozessor-Architektur, verteiltes Rechnersystem, autonomes Kontrollsystem

Zusammenfassung:

In einer allgemeinen Einführung wird das Gebiet der autonomen mobilen Roboter (AMR) beschrieben und eine Abgrenzung gegenüber anderen mobilen Systemen vorgenommen. Die praktischen Einsatzmöglichkeiten, speziell im Industrie- und Dienstleistungsbereich, mit den jeweiligen Problemen und Fragestellungen werden diskutiert und mit existierenden fahrerlosen Transportsystemen (FTS) verglichen. Für das Projekt Mobot-III werden die Zielsetzung, die Einsatzumgebung und die Systemkonzeption vorgestellt. Dabei werden speziell die Entscheidungskriterien für die gewählte Rechnerarchitektur und die Kommunikationsstruktur erläutert. Autonome mobile Roboter stellen wie andere komplexe Echtzeitsysteme hohe Anforderungen an das Rechnersystem. Dabei müssen Randbedingungen wie Echtzeitfähigkeit, geringe Leistungsaufnahme, gute Testmöglichkeiten, Modularität und Erweiterbarkeit berücksichtigt werden. Für das MOBOT-III- Projekt wurde ein zweistufiges verteiltes System entworfen und gebaut.

* Die hier vorgestellten Ergebnisse wurden teilweise durch das Ministerium für Wirtschaft und Verkehr, Rheinland-Pfalz gefördert.

1. Einführung

Autonome mobile Roboter sind in den letzten Jahren ein zentrales Forschungsgebiet der Robotik geworden. Gründe dafür sind nicht nur wissenschaftliche Fragestellungen und die Erprobung von Methoden der künstlichen Intelligenz sondern auch vielseitige praktische Einsatzmöglichkeiten.

Unter einem autonomen mobilen Roboter (AMR) versteht man eine Maschine, die sich selbständig und aus eigener Kraft innerhalb einer von ihr erkannten Umgebung frei bewegen kann, um gestellte Aufgaben auszuführen /1/. D.h. der Roboter muß je nach Einsatzumgebung über genügend Sensorik und Rechenleistung verfügen, damit er den erhaltenen Auftrag mit eigener Entscheidungsfreiheit durchführen kann. Dazu gehört insbesondere die Wahl des Weges und das selbständige Erkennen und Umfahren von Hindernissen.

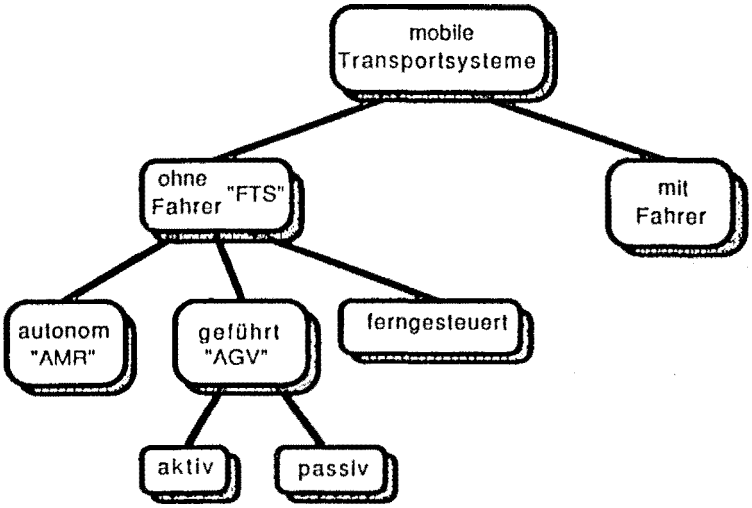


Bild 1: Kontrollmöglichkeiten für mobile Transportsysteme

Die wesentlichen Bestandteile eines AMR's sind die Antriebs- und Steuerungskomponente, das Sensorsytem, das Kontrollsystem und die Kommunikationseinheit. Entsprechend der Anwendung ist der AMR mit applikationsspezifischen Komponenten wie Montagerahmen, Hub- oder Drehtisch, Reinigungsaggregaten oder Manipulatoren ausgestattet, die jedoch den prinzipiellen Aufbau des AMR's nicht beeinflussen. Das charakteristische Merkmal eines AMR's ist die selbständige Planung und Ausführung der Aufgabenstellung. Dieser hohe Grad an Autonomie wird durch die Klassifkation der Kontrollmöglichkeiten für mobile Transportsysteme in Bild 1 belegt. Dabei ist insbesondere die Abgrenzung gegenüber den zur Zeit in der Industrie eingesetzten AGV's (automated guided vehicle) hervorzuheben. Der Unterschied liegt darin, daß der AMR für die Navigation keinen

Leitdraht benötigt, sondern sich in seiner Umgebung orientiert, indem er charakteristische Punkte als Landmarken speichert und wiedererkennt.

Bei der Realisierung eines AMR's ist die geplante Einsatzumgebung (Bild 2) ein entscheidender Faktor, der die Auswahl und den Aufbau der einzelnen Systemkomponenten bestimmt. Beispielsweise wird ein Fahrzeug, das außerhalb von Gebäuden ("outdoor") arbeitet, mit Kameras und einem aufwendigen System zur Bildauswertung ausgestattet sein, auf die beim Einsatz innerhalb von Gebäuden ("indoor") unter Umständen verzichtet werden kann. Der Indoor-Bereich läßt sich nach der Umgebungsstruktur weiter unterteilen in Büro-, Grossraum- und Fertigungsumgebung. Nimmt man die Korridore und Verbindungshallen als primäres Einsatzgebiet der Büroumgebung so bewegt man sich in relativ klaren Strukturen, die durch Wände und Türen begrenzt sind. Ähnliches gilt für Grossraumumgebungen wie Turnhallen oder Metrostationen, nur daß hier die zusammenhängende Fläche wesentlich größer ist. Die Fertigungsumgebung mit all ihren Einrichtungen stellt hingegen eine äußerst unstrukturierte Umgebung dar.

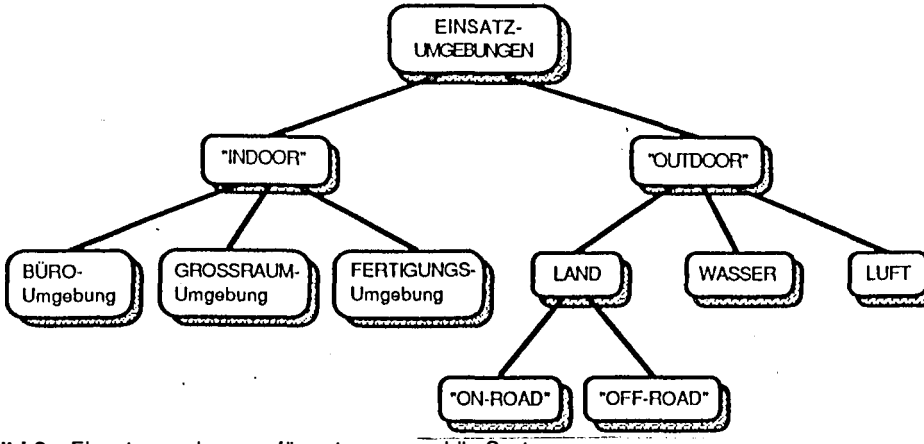


Bild 2: Einsatzumgebungen für autonome mobile Systeme

Die Komplexität der Aufgabenstellung wird weiterhin dadurch beeinflußt, ob sich der AMR in einer statischen oder dynamischen Umwelt zurechtfinden muß und wieweit mobile Hindernisse auftreten können. Wird eine statische Umgebung gefordert, so führen kleinste Veränderungen, wie ein umgestellter Stuhl oder ein versetzter Papierkorb zum Stillstand bzw. Ausfall des Systems.

Eine Frage, die zu Beginn gelöst werden muß, ist die Art und Weise wie der AMR seine Arbeitsumgebung erlernt. Dazu gibt es prinzipiell zwei Möglichkeiten, erstens die offline Grundrißgenerierung mittels entsprechendem CAD-System, d.h. die Umgebung muß explizit einprogrammiert werden, und zweitens die selbständige sensorgestützte Konstruktion des internen

Weltmodells. Letztere kann durch ein interaktives "Teach-In"-Verfahren vereinfacht werden, bei dem ein Operator das System zu den markanten Umgebungspunkten dirigiert

2. Einsatzmöglichkeiten

Grundsätzlich kann man zwei Einsatzgebiete für AMR im "indoor"-Bereich unterscheiden: den Einsatz im Dienstleistungsbereich und den Einsatz in der Fertigungsumgebung. Beide sollen im folgenden näher betrachtet werden.

2.1. Fertigungsumgebung

Das langfristige Ziel der CIM-Bestrebungen, die Fertigung nach Kundenauftrag, kann nur durch sehr kurze Entwicklungs- und Fertigungsdurchlaufzeiten erreicht werden. Daß dafür höchste Flexibilität in allen Teilbereichen der Produktion gefordert ist, steht außer Frage. Die flexible Automatisierung der Transportaufgaben zwischen Lager und Fertigung, sowie innerhalb des Fertigungsprozeß ist dabei ein wichtiges Teilziel. In Kombination mit automatisierten Lagersystemen kann die innerbetriebliche Logistik verbessert und somit die Gesamtdurchlaufzeit wesentlich reduziert werden.

Die flexible Automatisierung des Materialtransports unter möglichst geringen Lagerzeiten ist ein angestrebtes Ziel im Fertigungsbereich. Stand der Technik sind fahrerlose Transportsysteme, die entweder mit Hilfe eines passiven Metallbandes /2/ oder induktiv mit einem stromdurchflossenen Leiter /3/ geführt werden und somit einen eingeschränkten Grad an Flexibilität zulassen. Der nächste Schritt ist das spurfreie Fahren ohne Kontakt mit einem Leitdraht und entspricht somit der sensorgeführten Steuerung von AMR. Damit kann die gesamte Ablaufsteuerung wesentlich flexibler gestaltet werden und die Probleme wie Gegenverkehr oder blockierte Fahrbahnen können gelöst werden.

Für das spurfreie Fahren gibt es zwei Möglichkeiten, einmal die Navigation mit künstlichen Markierungen und zum anderen die Orientierung mittels der Umgebungsstruktur. Die Navigation mit Hilfe eines Echtzeit-Bildverarbeitungssystems, das sich an vorhandenen oder zusätzlich aufgebrachten hellen Farbmarkierungen orientiert /4/, ist ein Beispiel für den ersten Ansatz und wurde in diesem Jahr vorgestellt.

Aufgrund der komplexen Umgebungsstruktur einer Fertigungshalle stellt die Navigation nach der Umgebungsstruktur ein großes Problem dar. Im allgemeinen sind nur kleine Wandsegmente sichtbar, sodaß zur Orientierung zusätzliche Referenzflächen benutzt werden müssen. Eine denkbare Möglichkeit wäre die Verwendung von stationären Werkzeugmaschinen oder größeren

Einrichtungsgegenständen, falls diese eine relativ homogene Seitenstruktur besitzen /5/. Die Anzahl der verwertbaren Referenzflächen ist jedoch erfahrungsgemäß äußerst niedrig und das explizite Einprogrammieren dieser Punkte ist mit einigem Aufwand verbunden.

Eine andere Möglichkeit ist die Verwendung von flexibel positionierbaren Stellwänden zur Markierung von Kreuzungen und Abzweigungen, mit denen die Knoten der Fahrbahn festgelegt werden. Der AMR benutzt diese Knoten zur Positionskorrektur und Neuorientierung, um zwischen zwei Knoten eine Geradeausfahrt mit Hilfe der internen Sensorik durchführen zu können. Durch diese Struktur ist es dem AMR möglich, ähnlich wie in der Büroumgebung, seine Umwelt selbst aufzunehmen und Umgebungsveränderungen automatisch in seine interne Darstellung einzutragen. Ein weiterer Vorteil dieses Ansatzes liegt darin, daß innerhalb der Fahrbahn die Fahrspur frei gewählt werden kann und somit die Probleme wie Gegenverkehr und lokale Hindernisse im Rahmen der Fahrbahn gelöst werden können. Bild 3 demonstriert den Einsatz dieser Orientierungshilfen, deren Höhe relativ variabel gehalten werden kann (20 bis 100 cm) und vergleicht sie mit der Leitdrahtführung.

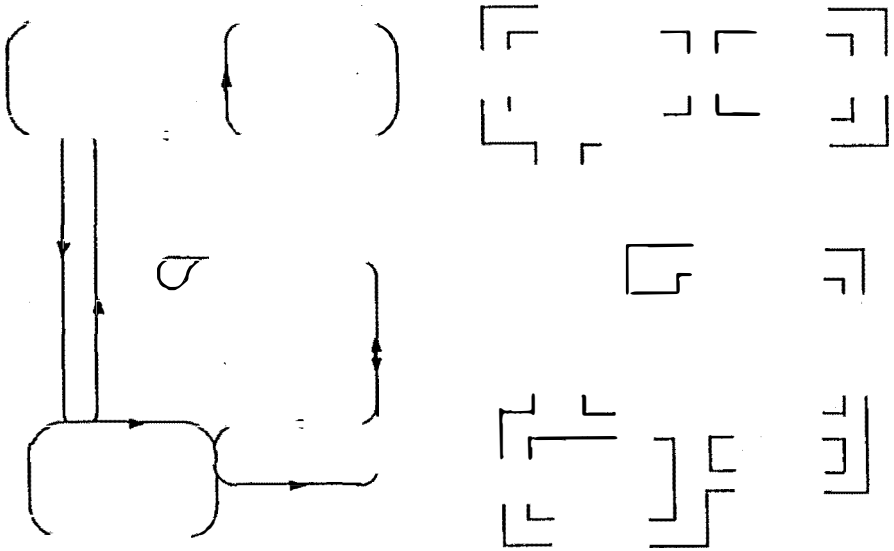


Bild 3: Leitdrahtführung versus Umgebungsnavigation mittels Stellwänden

Dieser Ansatz ist mit minimalem Aufwand zu realisieren und beeinträchtigt weder die Produktionsumgebung noch die Flexibilität des AMR's. Im Gegensatz zur Führung mit Leitdrähten oder Markierungen ist eine Änderung des Fahrkurses unproblematisch und mit geringem Aufwand durchführbar.

Sicherlich spielt auch der Sicherheits- und Kostenaspekt bei den Transportfahrzeugen eine große

Rolle. Ein Fahrzeug, das durch eine passive Leitspur geführt wird, ist sicherheitstechnisch einfacher zu handhaben, als ein im Korridor freifahrendes Fahrzeug. Auch ist es durch die einfachere Sensorik sicher kostengünstiger als ein AMR. Deshalb wird man zumindest in naher Zukunft die Systeme mit Spurbindung beibehalten, aber die Steuerung solcher Fahrzeuge "autonom" machen. D. h. die Fahrzeuge werden u. a. in der Lage sein das Spurnetz selbst zu erlernen, bei Blockaden selbständig eine neue Strecke suchen und Informationen mit anderen Fahrzeugen austauschen. Am Zielpunkt angekommen werden sie, wenn nötig, ihre Spur kurzfristig verlassen um flexible Montageaufgaben ausführen zu können. Somit wäre eine Kombination zwischen Bahnbindung im Transportbereich und lokaler Bahnfreiheit im Montagebereich eine denkbare Lösung für den Einsatz von AMR in der Fertigungsumgebung.

Neben der Automatisierung des Materialtransports kommt dem FTS auch im Montagebereich eine wachsende Bedeutung zu. Speziell die Abkehr von der Fließbandproduktion und damit vom starren Fördersystem hin zur flexiblen Montage erfordert ein neues Fördersystem als Verkettungselement zwischen den einzelnen Montageschritten. Prinzipiell können drei Gestaltungsformen unterschieden werden /6/: der Taxi-Betrieb, die mobile Werkband und das Mitfahrsystem.

Beim Taxi-Betrieb dient das FTS als reines Überbringersystem zwischen den einzelnen Montageplätzen /7/, es liefert das Montagegut an und holt es nach der Bearbeitung wieder ab. Diese Verkettung wird vorwiegend dann eingesetzt, wenn an den einzelnen Arbeitsplätzen längere Bearbeitungszeiten entstehen. Bei geringeren Bearbeitungszeiten ist eine zeitweise Trennung zwischen Montageobjekt und Transportfahrzeug nicht mehr sinnvoll und führt zum System der mobilen Werkbank. Das Mitfahrsystem ist dadurch gekennzeichnet, das der Werker seine Montageaufgabe von einer am FTS angebrachten Standplattform aus durchführt, während sich das Fahrzeug mit langsamer Geschwindigkeit fortbewegt.

Die Integration dieser Montagesysteme in komplexe Produktionsprozesse stellt neben dem flexiblen Transport ein wichtiges Einsatzgebiet für fahrerlose Transportsysteme dar. CIM-Konzeptionen der Zukunft werden entscheident durch die Möglichkeiten der FTS beeinflußt.

2.2. Dienstleistungsbereich

Die typischen Aufgaben für einen AMR im Dienstleistungsbereich sind Transportaufträge jeder Art und Reinigungsarbeiten für Fußböden /8/. Als Einsatzumgebungen kommen beispielsweise große Verwaltungs- und Bürogebäude, Krankenhäuser, Schulen- und Sporthallen in Betracht. Allen gemeinsam ist die relativ einfache Struktur der internen Verbindungswege, d.h. breite Korridore mit wenigen Einrichtungselementen, großräumige Eingangshallen, meist automatische Verbindungstüren und Fahrstühle.

Diese Umgebung bildet eine gute Grundlage für den Einsatz von AMR mit Radantrieb. Lediglich die Aufzüge müssen um eine automatische Kommunikationseinheit ergänzt werden, mit der sie die Steuerbefehle des AMR's aufnehmen können.

Unter diesen Voraussetzungen können sowohl routinemäßige Transportaufgaben der Ver- und Entsorgung, für die es bestimmte Fahrpläne gibt, als auch Spezialaufträge mit Zielvorgabe erledigt werden. Die Problematik liegt dabei weniger in der Umgebungsstruktur, die im allgemeinen nicht sehr komplex ist, sondern in der Behandlung von beweglichen Hindernissen.

Im Gegensatz zu Transportaufträgen, die normalerweise während der Dienstzeit bearbeitet werden müssen, können Reinigungsaufgaben außerhalb der Arbeitszeit durchgeführt werden. Dies bringt den Vorteil, daß die Anzahl der Personen, die sich in unmittelbarer Nähe des AMR's aufhalten, stark reduziert ist. Die Schwierigkeit bei Aufträgen im Reinigungssektor, wie Polieren von Hartböden oder Kehrsaugen von Teppichböden liegt bei dem flächendeckenden Abfahren der Umgebung. Die dabei zu fahrenden parallelen Bahnen erfordern ein präziseres Sensorsystem und genauere Navigationsalgorithmen als bei Transportaufgaben.

3. MOBOT-III - Ein autonomer mobiler Roboter

3.1. Einsatzumgebung und Systembeschreibung

Mobot-III ist ein autonomer mobiler Roboter, der für Aufgaben aus dem Dienstleistungsbereich (innerhalb von Gebäuden) konzipiert ist. Eine unbekannte Umgebung, die gewissen Randbedingungen genügt, soll automatisch mit der Navigationssensorik (Bild 4) aufgenommen und in mehrschichtiger Darstellungsform gespeichert werden. Damit ist es dem Roboter möglich sich in dieser Umwelt zurechtzufinden und zielgerichtete Aktionen durchzuführen.

Als erste Einsatzumgebungen sind großräumige Büros, kleine Hallen und Korridore geplant. Dabei wird keine statische Umwelt gefordert, sondern neben zeitlichen Veränderungen, wie verstellte Papierkörbe oder Stühle und geschlossenen bzw. geöffneten Türen sind auch ständig bewegte Objekte (Personen) erlaubt. Weiterhin soll die Aufnahme und Verarbeitung der Umgebungsdaten in Realzeit durchgeführt werden, d.h. daß der AMR ohne Stillstandphasen seine Aufgaben ausführt und dabei Kollisionen mit Hindernissen vermeidet.

MOBOT-III ist ein Dreirad, bei dem das vordere Rad angetrieben und gelenkt wird (Bild 4). Somit können die Hinterräder frei mitlaufen und der bei der Navigation störende Schlupf der Antriebsräder entfällt. Die Hinterräder sind mit inkrementellen Positionsgebern ausgerüstet und besitzen eine Auflösung von 2000 Schritten pro Umdrehung. Die Abmessungen des Fahrzeugs liegen

bei 70 * 52 * 65 Zentimetern (L * B * H), wobei das reine Fahrzeuggewicht etwa 55 Kg beträgt. Die Stromversorgung des Fahrzeugs besteht aus zwei 12 Volt Bleibatterien mit zusammen etwa 2 kWh Kapazität. Das System ist für eine Maximalgeschwindigkeit von 2 m/sec und eine durchschnittliche Arbeitsgeschwindigkeit von 1 m/sec ausgelegt.

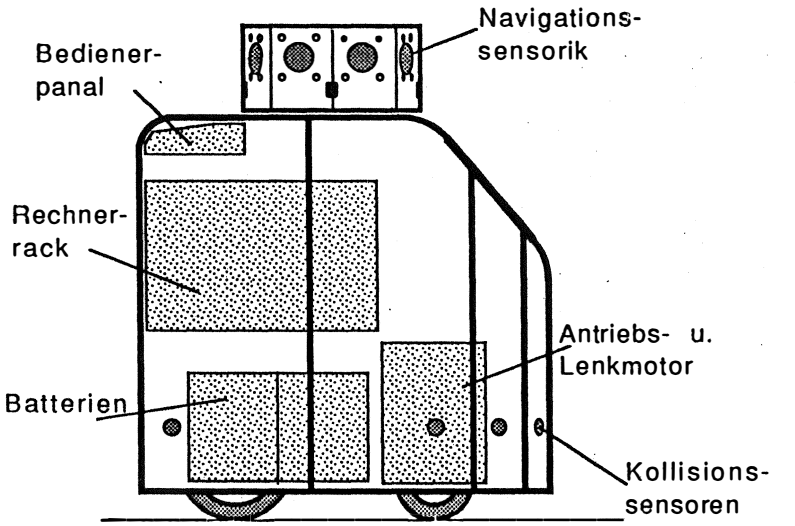


Bild 4: MOBOT-III - Aufbau

Ein 4-stufiges Sensorsystem übernimmt die Aufgaben der externen Datenaufnahme. Dabei wird ein umlaufender Sensorkopf mit 8 Ultraschall- und 4 Laserabstandssensoren für die Navigation und detaillierte Hinderniserkennung eingesetzt. Mit 13 ringförmig um das Fahrzeug angeordneten Ultraschallsensoren wird die grobe Hinderniserkennung durchgeführt und zur Kollisionsvermeidung im Nahbreich bis ca. 40 cm werden Infrarotabstandssensoren eingesetzt. Sollte diese Schutzzone trotzdem durchbrochen werden, so stoppt ein taktiler Sicherheitsring die Fahrzeugbewegung.

3.2 Entwurfsentscheidungen bezüglich des Rechnersystems

Betrachtet man sich ein AMR mit seinen komplexen Hardware-Funktionen wie Antrieb, Lenkung, sowie den unterschiedlichen Sensoren und Kommunikationseinrichtungen, so erkennt man schnell, daß ein zentraler Rechner eine komplexe Software erfordert. Betrachtet man nur die Bewegungsfunktionen, so werden ausgehend von der Zielpunktvorgabe über die Wege- und Bahnplanung bis zur Motorkoordination und Reglung komplexe Entscheidungsvorgänge durchlaufen. Gleiches gilt auch für die Sensordatenverarbeitung, die von der Steuerung der Sensorik, der Vorverarbeitung und Datenreduktion über die Fusion der Sensordaten und Generierung der

Umgebungslandkarte bis hin zur Erkennung von Objekten einen großen Rechenaufwand erfordert.

Bei der Konstruktion eines AMRs kommt daher, neben dem Sensorsystem, dem Kontrollsystem in seinem hardware- und softwaremäßigen Aufbau große Bedeutung zu. Für die Konfiguration des Rechnersystems, das die Hardwarerealisierung des Kontrollsystems darstellt, sind Randbedingungen wie OnBoard-Rechnersystem, Echtzeitfähigkeit, geringe Stromaufnahme, kleine Baugröße und Sicherheit zu beachten. Soll das System nach oben hin offen sein, spielt sicher auch die Erweiterbarkeit und Modularität der Hardware eine große Rolle.

Weiterhin müssen effiziente Fehler- und Selbstdiagnosefunktionen implementiert werden, die ein leistungsfähiges Testsystem unterstützen. Dazu müssen einfache Hardware-Funktionen, wie Ladung der Akkus, Spannungswandlung oder Temperaturüberwachung durch kleinere Rechner überwacht und gesteuert werden, damit Fehlfunktionen, wie sie durch das Absinken der Versorgungsspannung oder zu hoher Stromentnahme verursacht werden, frühzeitig durch das Rechnersystem selbst erkannt werden. Basis für ein gutes Testsystem ist sicherlich auch eine komfortable Kommunikationseinrichtung. Hierzu gehören neben den Standardschnittstellen, wie RS232, eine telemetrische Datenübertragungseinrichtung mittels Infrarot- oder Funkstrecken, und direkte Ein- und Ausgabemedien, wie graphikfähigem LCD-Schirm und Sprache. Bei der Programmentwicklung ist es sicherlich von Vorteil, wenn neue Programme direkt, unter Umgehung des EPROM-Weges, in jedes beliebige Modul geladen und dort in einem laufenden System getestet werden können.

Bezüglich der Konstruktion des Rechnersystems eines AMRs kann man dieses grob in zwei Ebenen einteilen (Bild 5). Die untere Ebene (Lowlevel) umfasst dabei die gesamten Hardware-Schnittstellenfunktionen mit der Motorsteuerung, der Systemüberwachung und den Kommunikationseinrichtungen. Auch die einfache Sensorik, wie Schalter, Infrarottaster, Ultraschallsystem und Positionssensorik ist Bestandteil dieser Ebene. Im Gegensatz dazu ist die Highlevel-Ebene praktisch unabhängig von der gewählten Hardwarestruktur und beherbergt mit ihren Rechnern und Datenspeichern das Kontrollsystem, d.h. sie stellt die eigentliche "Intelligenz" des Fahrzeugs dar.

Der Aufbau dieser oberen Ebene ist zur Zeit noch Gegenstand der Forschung /9/, sodaß der Entwurf des Rechnersystems dieser Ebene sehr flexibel gehalten werden muß. Die Lowlevel-Ebene dagegen ist mit ihren einzelnen Modulen überschaubar, sodaß hier keine Änderungen innerhalb der Module zu erwarten sind. Diese Ebene muß lediglich so flexibel sein, daß neue Module einfach ohne Hardware-Änderungen hinzugefügt werden können.

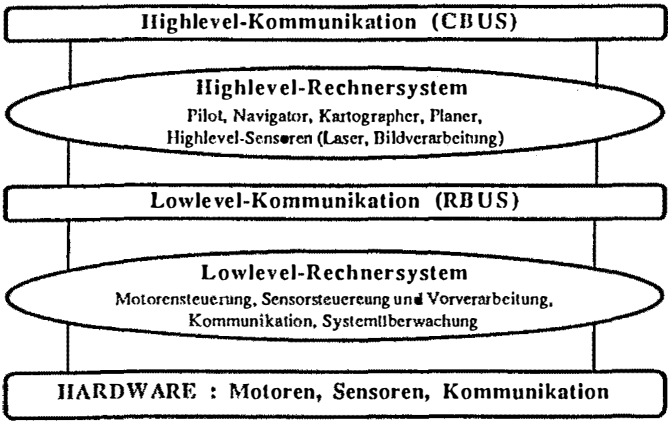


Bild 5: : Zwei-Ebenen-Struktur des Rechnersystems

Nicht zuletzt spielt auch der Sicherheitsaspekt eine große Rolle. Ein mobiles System, das bis zu 2 m/sec Geschwindigkeit erreichen soll, ist sicherlich für die in der Nähe befindlichen Personen und Gegenständen eine Gefahrenquelle. Deshalb muß bei der kleinsten Unregelmäßigkeit das System die Motoren stoppen. Wie in anderen Sicherheitssystemen müssen auch hier mehrere unabhängige Instanzen das System überwachen und das Fahrzeug anhalten können.

Aufgrund all dieser Überlegungen fiel die Entscheidung zugunsten eines zweistufigen dezentralen, mittels LANs gekoppelten Rechnersystems (Bild 6). Im folgenden werden nun das verwendete Bussystem und die Rechnermodule vorgestellt

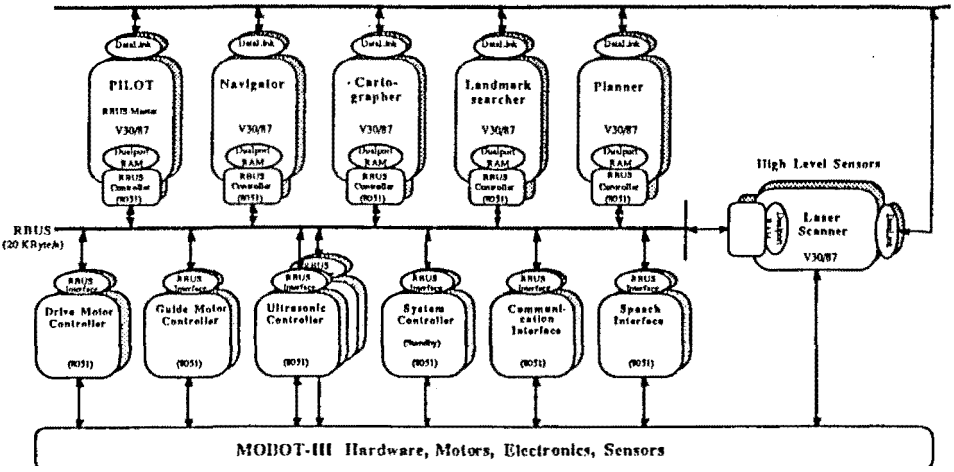


Bild 6: MOBOT-III Rechnersystem

3.3 Der Lowlevel-Bus (RBUS)

Nachdem die Entscheidung nun zugunsten eines dezentralen Systems gefallen war, stellte sich die Frage nach dem Aufbau der gemeinsammen Modulschnittstelle. Sicherlich kann man sich theoretisch sehr komfortable und funktionale Schnittstellen vorstellen. Jedoch erzwingen die Randbedingungen des AMRs und die Verfügbarkeit der benötigten Hardware bestimmte Kompromisse.

Eine Vorgabe an die Lowlevel-Ebene war die Bedingung, auch sehr einfache Module, die nur aus einem einfachen Mikrokontroller bestehen, einsetzen zu können. Deshalb war es nicht möglich ein LAN wie Ethernet einzusetzen, da der benötigte Hardwareaufwand zur Implementierung der Schnittstelle aufwendiger als die Steuerungshardware selbst geworden wäre. Solche Systeme erfordern neben dem eigentlichen Kontroller meist noch die zusätzliche Möglichkeit des DMA, damit die Daten von der Schnittstelle schnell in den Speicher transportiert werden können.

Die Entscheidung fiel letztlich zugunsten eines 187.5 KBaud schnellen seriellen Bussystems, wie es von der INTEL 8051 und 8096 Familie hardwaremäßig unterstützt wird. Dies bedeutet zwar die Festlegung auf eine bestimmte Rechnerfamilie, bereitet aber keine Probleme, da es neben mehreren Zweitherstellern auch eine große Anzahl aufwärtskompatibler Prozessoren gibt. Auf Basis dieser Mikrokontroller sind schon unterschiedliche Master-Slave-Bussysteme entwickelt und vorgestellt worden. Ein auf den ersten Blick Interessantes System, wie der INTEL 8044 Kontroller mit der BITBUS-Schnittstelle /10/, hätte zwar die Schnittstellen-Software etwas vereinfacht, wurde aber wegen der fehlenden CMOS-Version nicht eingesetzt. Außerdem ist er der einzige Kontroller mit der BITBUS-Schnittstelle, sodaß keine Auswahl bezüglich des Prozessors mehr möglich gewesen wäre. Auch sind heute serielle Kontrollerbaustelne, wie der INTEL 82510, erhältlich, die ebenfalls die seriellen Möglichkeiten der 8051 Familie besitzen und einen Anschluß an jeden Prozessortyp ermöglichen. Der Name "RBUS" steht als Abkürzung für Remote-BUS, was die Aufgabe dieses Busses treffend beschreibt : Hier wird Hardware ferngesteuert.

Die asynchrone Übertragung erfolgt bitserieil mittels Start- und Stopp-, aber ohne Paritybit. Die wichtigste Eigenschaft dieser Schnittstelle ist die Möglichkeit ein neuntes Datenbit zu übertragen und bei Empfang eines solchen Bits im Slave einen Interrupt zu erzeugen (Wake-Up). Mit diesem Mechanismus lassen sich jetzt einfach mehrere Kontroller über die Schnittstelle koppeln. Wird vor jeden Datenblock eine Zieladresse mit aktivem 9-ten Datenbit gesendet, erhalten alle angeschlossenen Kontroller einen Interrupt Ihrer seriellen Schnittstelle. In der entsprechenden Interruptprozedur müssen sie nun nachsehen, ob das empfangene Adressbyte mit der ihnen zugeordneten Adresse identisch ist. Sind sie angesprochen, dann steuern sie gemeinsam mit dem Sender den weiteren Busablauf. Im anderen Fall löschen sie das Empfangsregister und führen das zuvor unter-

brochene Programm weiter. Dieser grundlegende Mechanismus bestimmt weitgehend den Aufbau des RBUS-Systems und seiner Protokolle.

Bei der Busvergabe wurde zugunsten eines Master-Slave-Systems mit Round-Robin entschieden, bei dem die jeweiligen Highlevel-Module nach einem festgelegten Schema die Masterfunktion erhalten. Damit konnte die relativ geringe Busrate von 20 KByte/sec optimal ausgenutzt werden. Da die Lowlevel-Einheiten (die meist nur aus einem Mikrokontroller bestehen) keine Masterfunktion übernehmen, werden die von ihnen zur Verfügung gestellten Informationen vom jeweiligen Busmaster abgerufen und nach dem Broadcasting-Prinzip von allen anderen Highlevel-Moduln ebenfalls aufgenommen.

Damit der Bus durch die Verarbeitungszeiten der Slaves nicht zulange belastet wird, wurde der Datenverkehr strikt in drei Transferrichtungen eingeteilt. Die ersten beiden Modi realisieren dabei eine feste Punkt zu Punkt Verbindung, wobei beim Master-Slave-Transfer (MST) der jeweilige Master Daten oder Kommandos an die Applikation des Slaves sendet und im Slave-Master-Transfer (SMT) Daten vom Slave abholt (Bild 7). Der dritte Bustransfer, der wie ein SM-Transfer abläuft, holt ein Datenpaket von einer Einheit ab und stellt es dabei allen angeschlossenen Highlevel-Einheiten zur Verfügung. Ein Beispiel dafür ist die Positionsmeldung des Bahnrechners, die in allen anderen Moduln benötigt wird.

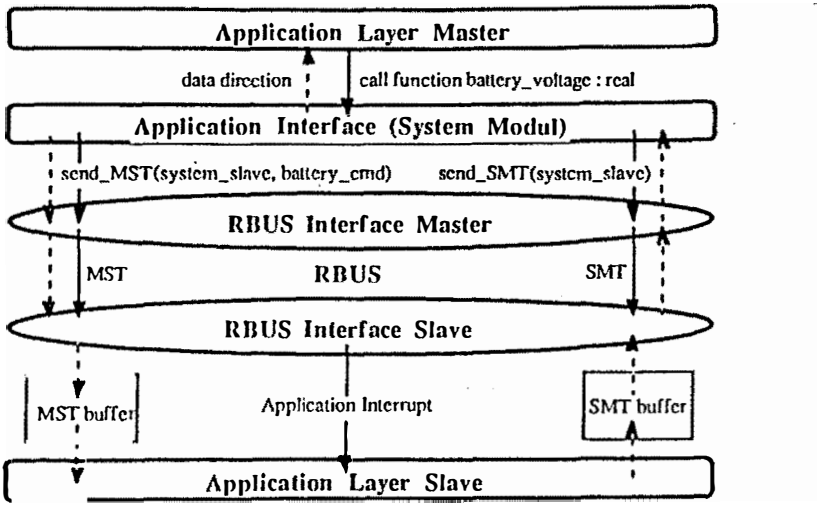


Bild 7 : Transferverlauf zwischen Master und Slave (-----> data, ⟶ command)

An einem Beispiel wird dieses Vorgehen verständlich. Betrachtet man die Ultraschallsysteme, so vergeht zwischen dem Kommando eine Messung auszuführen und dem Zeitpunkt, ab dem das

Ergebnis vorliegt, eine Zeitspanne von einigen Zehnmillisekunden. Würde der Master hier auf das Ergebnis warten, wäre während dieser Zeit der Bus blockiert. Auch ist es hier sinnvoller einmal ein MST- Kommando zur Festlegung der Betriebsparameter zu übergeben, um dann nur noch über folgende SM- Transfers das Ergebniss abzuholen. Die Trennung in MST und SMT ist auch deshalb nötig, da der im Slave befindliche RBUS-Treiber bei einem eingehenden MST-Befehl diesen nicht interpretieren und auf Daten der Anwendungsebene zugreifen kann.

Da es also keinen direkt funktionalen Zugriff des Masters zum Slave (außer bei Test- und Diagnosebefehlen) gibt, muß ein funktionaler Zugriff auf der Anwendungsebene in zwei Teile aufgespalten werden. Zuerst wird das Kommando zum Slave mit einem MS-Transfer übertragen und dann nach einer vorgegebenen Zeit, bzw. asynchron durch den Slave angefordert, das Ergebnis mit einem SM-Transfer abgeholt.

Auf der Basis der beiden Transfer-Modi wurde ein Paketformat implementiert, das im Gegensatz zu anderen Formaten keinen geschlossenen Aufbau besitzt. Es enthält zwar auch Adress-, Kontrol-, Status-, Datenlängen-, CRC- Feld und natürlich das Datenfeld selbst, wird aber nicht in einem zusammenhängenden Block übertragen. Stattdessen wird an bestimmten Stellen auf die Quittierung des Slaves gewartet. Damit wird neben der Synchronisation von Master und Slave erreicht, daß der Bus nur dann belastet wird, falls der Slave auch bereit ist das Paket zu übernehemen.

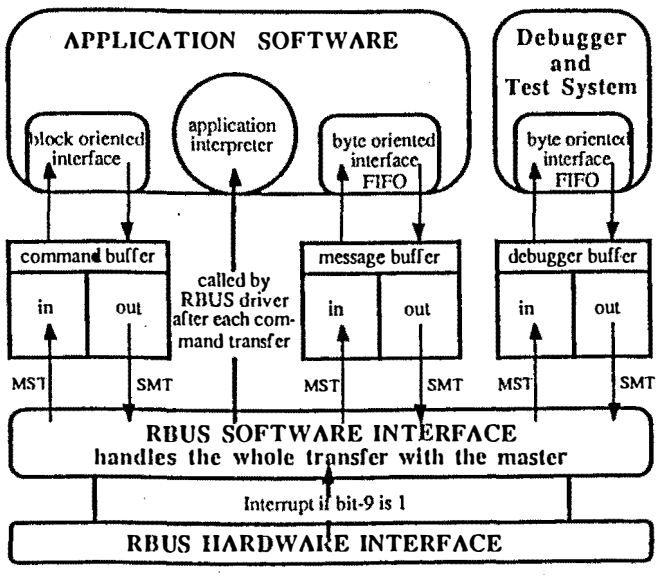


Bild 8 : RBUS Softwarestruktur auf den Interfacerechnern

Das RBUS-System übernimmt auf den 8051-Einheiten außer der reinen Transportfunktion der Nachrichten auch Betriebssystemfunktionen (Bild 8). Zur Aufnahme der Nachrichten von der

Anwendungsebene des Masters, bzw. zur Übergabe von Nachrichten an diese, steht der Command-Puffer zur Verfügung. Er kann für jede Transferrichtung jeweils genau ein Paket aufnehmen. In diesem Puffer wird über die Transportbefehle "Application-Master-Slave" (AMS) und "Application-Slave-Master" (ASM) eine Nachricht eingeschrieben bez. abgeholt. Nachdem ein Transfer erfolgreich abgeschlossen wurde, wird eine zuvor mit der Anwendungsebene vereinbarte Prozedur, der sogenannte Application-Interpreter, gestartet. Dies ermöglicht die rasche Behandlung eines solchen Ereignisses durch die Anwendungsebene.

Neben dem Command-Puffer wurden auf der Netzwerkebene noch zwei weitere Puffer mit jeweils einer MS- und SM-Datenrichtung implementiert. Während im Kommandopuffer nur jeweils eine Nachricht der kommunizierenden Anwendungsebenen eingetragen werden kann, fungieren die beiden anderen Puffer als zeichenorientierte FIFOs. Dieser zeichenorientierte Transfer dient hauptsächlich der Diagnose und dem Test der Module. Dazu werden die von einer Anwendugsebene auf einem Modul erzeugten Statusmeldungen vom Master zu einer I/O-einheit transportiert.

Aufgebaut wird der Übertragungskanal durch entsprechende Kommandos der Kommunikations-moduln. Der sogenannte Debugger-Puffer, der wie der Message-Puffer funktioniert, dient der Kommunikation mit dem Testsystem des jeweiligen Rechnermoduls. Über ihn können Speicherplätze angezeigt, modifiziert und auch Programme geladen und gestartet werden. Damit ist es möglich das gesamte Rechnersystem während des Betriebs zu testen und von außen mit neuer Software zu versorgen. Im Fehlerfall können alle Einheiten so direkt angesprochen und auf ihren Zustand hin untersucht werden. Auch ist es möglich zu bestimmten Zeitpunkten Speicherdumps durchzuführen, sodaß der Modulstatus festgehalten werden kann.

3.5 Rechnermodule

Bei der Konzeption der Rechnermodule spielte die Frage nach dem eingesetzten Prozessortyp, zumindest in der Highlevel-Ebene, eine untergeordnete Rolle, da die RBUS-Schnittstelle davon unabhängig ist. Die wichtigsten Vorgaben für das Rechnersystem waren :

- Die ausschließliche Verwendung von CMOS-Bauteilen um den Stromverbrauch so gering wie möglich zu halten. Damit ist die Stromversorgung einfacher, es werden keine Lüfter benötigt und die Rechner können vor Umwelteinflüssen sicher geschützt werden.
- Die Möglichkeit eine leistungsfähige Hochsprache einsetzen zu können, damit die Teamarbeit unterstützt und das Softwaresystem überschaubar bleibt.
- Der Rechner sollte universell einsetzbar sein und die Möglichkeit einer leistungsfähigen Realarithmetik besitzen, damit die komplexen Bahn- und Koordinatenberechnungen schnell und bei hoher Genauigkeit durchgeführt werden können.
- Der Sicherheit wegen soll jedes Rechnermodul mit einem Watchdog-Timer ausgerüstet sein,

damit Programmfehler nicht zum totalen Absturz des Systems führen.
Der Rechner soll über einen kleinen nichtflüchtigen Speicher verfügen, damit Systemparameter, auch über ein Ausschalten hinaus, abgelegt werden können.

Entschieden wurde in der oberen Ebene vorerst zugunsten eines INTEL 8086/87 Systems, allerdings wurde als Rechner, der etwas schnellere V30 von NEC eingesetzt. Zwei Gründe haben dabei hauptsächlich die Wahl entschieden. Erstens gibt es ein großes Angebot an Entwicklungssoftware, die kostengünstig auf IBM PC/XT/AT Systemen lauffähig ist. Zweitens läßt sich die einmal entwickelte Software sehr einfach auf die leistungsfähigeren Prozessoren iAPX286 und iAPX 386 portieren, sodaß bei Bedarf die Rechenleistung erhöht werden kann.

Eine offene Frage bezüglich der Rechnermodule betraf den Anschluß der Module an den Lowlevel-Bus. Dazu stehen grundsätzlich die folgenden beiden Möglichkeiten offen. Erstens die direkte Ankopplung mit einem seriellen Kontrollerbausteln und zweitens die indirekte Kopplung über einen 8051 Mikrokontroller. Um den Durchsatz des Hauptprozessors durch den Datentransfer nicht unnötig zu verringern, wurde zugunsten des Konzepts mit dem Mikrokontroller entschieden. Dies ermöglicht auch eine größere Flexibilität im Nachrichtenformat, da zum Handling der Schnittstelle ein eigener Rechner zur Verfügung steht. Offen bleibt nun noch, wie der Hauptprozessor mit dem Interfaceprozessor kommuniziert. Eine Portverbindung wurde wegen der hohen Rechnerbelastung und der Zugriff über DMA wegen der unterschiedlichen Busstruktur ausgeschlossen.

Die Entscheidung fiel aus zwei Gründen zugunsten einer Dualport-Struktur (Bild 9). Erstens kann der Interfaceprozessor, ohne daß der Master dazu Rechenzeit zur Verfügung stellen muß, die empfangenen Daten im gemeinsamen Speicher zur Verfügung stellen, bez. von dort zum Senden abholen. Zweitens kann der Interfacerechner die Daten aufbereiten und sich um die Probleme der Datensicherung und des Datentransports kümmern.

Softwaremäßig sieht es für den Hauptrechner so aus, als ob die im Dualport befindlichen Daten direkt von der jeweiligen Hardware dort eingetragen worden wären. Führt der Interfaceprozessor in regelmäßigen Zeiten die Datentransfers zu den einzelnen Modulen durch, so braucht der Hauptrechner die aktuellen Daten nur bei Bedarf aus dem Dualport-Ram zu lesen. Wann die Datentransfers zu den Slaves durchgeführt werden müssen, wird dem Interfacekontroller vom Hauptrechner beim Systemstart mitgeteilt. Beispielswelse wird zur Abfrage der Systemparameter, wie Akku-Kapazität,-Spannung und Temperatur, ein Auftrag definiert, der alle 500 msec diese Daten vom Systemkontroller in den Dualport-Speicher des Masters transferiert. Benötigt der Master nun die aktuellen Werte, so sieht er einfach im Dualport-Speicher nach.

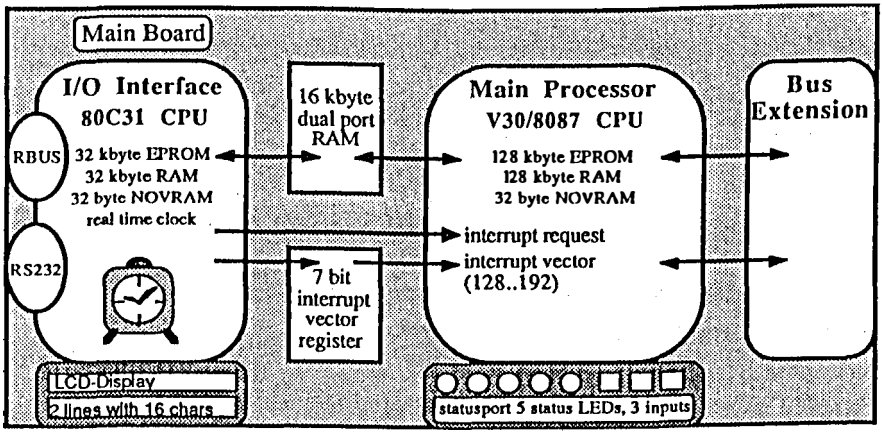


Bild 9 : MOBOT-III-Rechnermodul

Damit der Master nicht selbst dauernd diese Parameter überwachen muß, kann der Interfacerechner im Gefahrenfall beim Hauptrechner einen Interrupt erzeugen kann. Da der Interfacerechner aber die Semantik der Daten nicht kennt, muß der die Nachricht sendende Slave diese mit einer speziellen Markierung versehen, damit dann beim Master ein Interrupt erzeugt wird. Durch dieses Verfahren wird der Master entlastet und der Slave, der seine Hardware ja zu genüge kennt, trifft die Entscheidung, ob eine besondere Situation das Eingreifen des Masters erfordert.

4. Schlussbetrachtung

Das vorgestellte 2-stufige, verteilte Rechnersystem bildet die Hardwarebasis für das Kontrollsystem des autonomen mobilen Roboters MOBOT-III. Seine Installierung ist im wesentlichen abgeschlossen, sodaß sich der Schwerpunkt der Arbeiten auf den Entwurf und die Implementierung der Applikationssoftware konzentriert. Diese kann aufgrund des gewählten Kommunikationssystems sowie der Dual-Port-Struktur einfach und über funktionale Schnittstellen mit den dezentralen Sensor- und Aktuator-Moduln kommunizieren. Somit konnte zur Implementierung die Programmiersprache MODULA-2 gewählt werden, die u. a. den Vorteil bietet, daß sie durch Lehrveranstaltungen eingeführt wird und die meisten Studenten mit ihr vertraut sind.

5. Literaturverzeichnis

/1/ Jerkov, G., Knieriemen, T., "Autonome Mobile Roboter - Einführung und Überblick" Interner Bericht 171/87 im Fachbereich Informatik der Universität Kaiserslautern

/2/ ..., Transcar Produktbeschreibung der Firma Telelift GmbH & Co, 1987

/3/ Braun, P., "Fahrerlos in die Zukunft: automatische Transportsysleme", TÜ Bd. 26 Nr.7/8, 1985

/4/ ..., "Identifikationssysteme in der Produktionslogistik", Logistik im Unternehmen, VDI-Verlag, September4/87

/5/ Drunk, G., "Mobiler autonomer Roboter fährt ohne Draht", VDI-Nachrichten Nr. 13, März 1987

/6/ Schmidt, M., "Montageverkettung mit FTS", ZwF 82 Carl Hanser Verlag, Sept. 1987

/7/ ..., "Neues hochautomatislertes Montagesystem", Sonderdruck aus Daimler-Benz intern, März 1986

/8/ ..., Produktbeschreibung MIDI-ROBOTS, Ramonville-Saint-Agne (F), 1986

/9/ Harmon, S.Y., "Practical Implementation of Autonomous Systems: Problems and Solutions", Preprints of the International Conf. of Intelligent Autonomous Systems, Amsterdam, Dez. 1986

/10/ ..., INTEL "Distibuted Control Modules Databook (BITBUS Specification)", Santa Clara, 84

PC-MEDOS,
ein Echtzeit-Multitasking-Betriebssystem
für Personal Computer

Verfasser:

Dr. H.-G. Scheurer

MikroTec Mikrocomputersysteme-
Automatisierungstechnik GmbH
Silberstr. 1
7570 Baden-Baden
Tel.: 07221 - 61015

Zusammenfassung

Bedingt durch die fehlenden Multitasking- und Echtzeiteigenschaften von PC/MS-MEDOS war es bisher praktisch nicht möglich, Personal Computer auch für Aufgaben der Industrieautomatisierung einzusetzen. Deshalb wurde eine Betriebssystemerweiterung für MS-DOS entwickelt (PC-MEDOS), das einerseits die Funktionen von MS-DOS beibehält und gleichzeitig zusätzliche Echtzeit- und Multitasking-Funktionen zur Verfügung stellt. So werden z.B. durch PC-MEDOS folgende Aufgaben ermöglicht

- Ablaufsteuerung eines Mehrprogramm-Betriebes (multitasking- und interruptgesteuerte Taskumschaltung)

- Unterstützung der Kommunikation zwischen individuellen Tasks über Messages und Mailboxes

- Unterstützung der Synchronisation zwischen individuellen Tasks und der Betriebsmittelverwaltungen über Semaphoren

- Unterstützung der Reaktionen auf externe Ereignisse (Interrupts)

Mit diesem Betriebssystem können Personal Computer problemlos für Standard-Automatisierungsaufgaben eingesetzt werden.

Einleitung

Fast in allen Bereichen der Büroautomatisierung, sei es in Textverarbeitung, Finanzbuchhaltung oder aber auch im Handel, beim Handwerk und selbst im technischen Büro, z.B. mit CAD, hat der Personal Computer seinen Einzug gehalten. Ausschlaggebend für die große Akzeptanz des Personal Computers waren folgende wesentliche Eigenschaften:

- Die in den letzten Jahren wirklich beachtliche Leistungsfähigkeit der Hardware verbunden mit einem äußerst günstigen Preis

- Die große Anzahl standardmäßig auf dem Markt vorhandener Software zu ebenfalls sehr günstigen Preisen

- Die relativ leichte Bedienbarkeit der Geräte bedingt durch die einfache Programmierbarkeit und auch bedingt durch das als Standard sich durchgesetzte Betriebssystem MS/PC-DOS

- Die große Anzahl von zusätzlich auf dem Markt vorhandenen Hardwaremodule wie z.B. Zusatzsteckkarten, Farbmonitore usw.

Deshalb war es auch nicht verwunderlich, daß verschiedene Anwender und Hersteller seit ungefähr 2 - 3 Jahren versuchen, den Personal Computer auch für die Aufgaben der Industrieautomatisierung einzusetzen. Zunächst wurde sich darauf beschränkt, die Hardware so umzuarbeiten bzw. zu ergänzen, daß diese Personal Computer auch in der Industrieumgebung einsetzbar sind, z.B. dadurch, daß sie als 19"-Einschubgeräte zur Verfügung stehen und den rauhen Umgebungsbedingungen der Industrie gerecht werden.

Durch diese Maßnahmen waren die Geräte zwar hardwareseitig dazu geeignet, Aufgaben der Prozeßdatenverarbeitung zu übernehmen, nicht jedoch vom Standard-Betriebssystem MS-DOS aus. Dieses Betriebssystem ist nicht echtzeit- und multitaskingfähig und eignet sich deshalb nicht zum Einsatz für Industrieautomatisierungsaufgaben. Zwar gibt es andererseits eine Reihe von echtzeit-/multitaskingfähigen Betriebssystemen, die auf der Hardware der Personal Computer lauffähig sind z.B. RMX86, diese Betriebssysteme sind aber nicht MS-DOS-kompatibel, d.h. für diese Betriebssysteme stehen auch nicht die große Anzahl von Standard-Programmen zur Verfügung.

Deshalb wurde mit PC-MEDOS ein Betriebssystem entwickelt, das sowohl echtzeit- und multitasking-fähig ist und gleichzeitig MS-DOS-kompatibel. Damit wird es dem Anwender dieses neuen Betriebssystems möglich, die unbestrittenen Stärken eines Realzeit-betriebssystems bei Multitasking-Echtzeitaufgaben, wie z.B. Meßdatenerfassung, Prozeßsteuerung usw. zu nutzen und gleichzeitig eine große Anzahl von Standard-Programmen der MS-DOS-Welt einzusetzen. Dies führt zu bisher nicht möglichen und vollkommen neuen Einsatzmöglichkeiten von Personal Computern bzw. Industrie-Personal Computern,z.B.bei der Maschinen-steuerung, in der Prozeßdatenverarbeitung und der Prozeßleittechnik.

Aufbau und Eigenschaften von PC-MEDOS

PC-MEDOS ist ein Echtzeit-Multitasking-Betriebssystem für Intel-Prozessoren 8088, 8086/80286. Zusammen mit MS-DOS/PC-DOS (ab Version 3.1) läßt es sich auf allen gängigen Personal Computern einsetzen. Da bei der Entwicklung von PC-MEDOS äußerster Wert darauf gelegt wurde, daß dem Anwender alle normalerweise von MS/PC-DOS unterstützten Systeme nach wie vor und ohne Einschränkung zur Verfügung stehen, ist PC-MEDOS nicht ohne MS-DOS lauffähig und ist deshalb streng

genommen als MS-DOS-Erweiterung zu betrachten. Bei einem Anwendungssystem wird MS-DOS als Task von PC-MEDOS gestartet und steht anschließend zur weiteren Nutzung zur Verfügung.

Zusätzlich zu den von MS/PC-DOS bekannten Systemaufrufen stellt PC-MEDOS noch folgende Funktionen zur Verfügung, die für einen Echtzeit-Multitasking-Betrieb unbedingt erforderlich sind

- Ablaufsteuerung eines Mehrprogrammbetriebes (Multitasking) über zeit- und interruptgesteuerte Taskumschaltung

- Unterstützung der Kommunikation zwischen individuellen Tasks über Messages und Mailboxes

- Unterstützung der Synchronisation zwischen individuellen Tasks und der Betriebsmittelverwaltung über Semaphoren

- Unterstützung der Reaktion auf externe Ereignisse (Interrupts)

Die Eigenschaften dieser verschiedenen, zusätzlichen Möglichkeiten werden nachfolgend kurz beschrieben, wobei vielleicht als Einleitung zum besseren Verständnis das Zusammenspiel dieser verschiedenen PC-MEDOS-Einrichtungen wie folgt dargestellt werden kann:

- Tasks sind die aktiven Objekte, d.h. z.B. die Anwender-Programm-Module

- Zur Kommunikation mit anderen Tasks (Austausch von Messages) wenden sie sich an eine Mailbox

- Zur Synchronisation mit anderen Tasks wenden sie sich an ein Semaphor

- Beim Warten auf ein externes Ereignis (Interrupt) wenden sie sich an eine Interrupt-Schnittstelle.

Da in einem realzeitfähigen Betriebssystem, im Gegensatz z.B. zu zeitgesteuerten Betriebssystemen, es nicht vorhersehbar ist, wann irgendwelche Ereignisse auftreten, ist zur Koordination der verschiedenen Kommunikationsmöglichkeiten eine entsprechend umfangreiche Software erforderlich, die nachfolgend im einzelnen beschrieben wird:

Task-Management

Um die Aufgaben einer Asynchron-Überwachung und -/Steuerung zeitlich paralleler Teilprozesse lösen zu können, werden diese Teilprozesse mit verschiedenen Tasks realisiert, wobei die gegenseitige Abhängigkeit der Tasks das Prozeßgeschehen wiedergeben.

Darüber hinaus bedeutet die Aufteilung eines Anwendungsproblemes in Tasks einfacheren logischen Entwurf, Modularität, bessere Test- und Wartungsbedingungen und eine effektivere Ausführung durch höheren Durchsatz.

Eine Task hat generell zwei Ziele:

- Ihr erstes Ziel ist, eine spezifische Aufgabe zu erfüllen

- Ihr zweites Ziel ist die exklusive Kontrolle über den Prozessor zu erhalten, um in der Ausführung dieser speziellen Aufgabe voranzuschreiten

Eine der Hauptaufgaben des Betriebssystems PC-MEDOS ist, die Konkurrenz zu steuern, welche sich ergibt, wenn mehrere Tasks die exklusive Kontrolle über den Prozeß wünschen. Zur Ablaufsteuerung dieser Konkurrenz führt das Betriebssystem PC-MEDOS für jede Task einen Prozeßzustand und eine Prozeßpriorität.

So befindet sich eine Task z.B. immer in einem der Prozeßzustände 'bereit', 'aktiv', 'suspendiert' oder

'wartend'. Die Priorität einer Task ist ein Wert zwischen 1 und 253, wobei 1 die höchste Priorität ist.

Nur maximal eine Task im System kann sich im Zustand 'aktiv' befinden, wobei jeweils die 'bereite' Task mit der höchsten Priorität in diesen Zustand versetzt wird. Der Übergang von Zustand 'bereit' in den Zustand 'aktiv' bedeutet für die Task die Erlangung der Kontrolle über den Prozessor. Die Task behält die Kontrolle über den Prozessor, bis eines der folgenden Ereignisse auftritt:

- Die Task gibt von sich aus den Zustand 'aktiv' auf (z.B. sie suspendiert sich, sie wartet auf Zeit, sie wartet auf eine Message, sie wartet auf einen Interrupt)

- Eine andere Task erreicht den Zustand 'bereit', welche eine höhere Priorität als die Aktiv-Task besitzt.

Kommunikations-Schnittstellen (Exchanges)

Die in PC-MEDOS vorhandenen Kommunikations-Schnittstellen wurden dazu eingerichtet, um die Intertask-Kommunikation zu ermöglichen, die Task-Synchronisation zu erreichen und einen gegenseitigen Ausschluß von Betriebsmitteln zu ermöglichen. PC-MEDOS übernimmt selbständig die Verwaltung der nachfolgend beschriebenen Kommunikations-Schnittstellen und führt gleichzeitig bei Bedarf automatisch die erforderliche Task-Umschaltung bzw. andere Reaktionen durch.

Folgende Kommunikationsmittel stehen zur Verfügung:

Mailboxes

Eine Mailbox wird ausschließlich zur Intertask-Kommunikation verwendet. Möchte eine Task A eine Message an eine Task B senden, so sendet sie diese Message an eine Mailbox und Task B muß diese Mailbox aufsuchen, wo sie, falls noch keine Message eingetroffen ist, die Möglichkeit hat, für die gwünschte Zeitdauer zu warten.
Jede Mailbox hat zwei Warteschlagen:

- Eine Task-Warteschlange (prioritätsgesteuert)
  Hier befindet sich die Task, welche in dieser Mailbox auf das Eintreffen einer Message wartet

- Eine Message-Warteschlange (FIFO)
  Hier befindet sich die Message, welche an diese Mailbox gesandt und noch von keiner Task abgeholt wurde.

Zu jedem Zeitpunkt ist maximal eine der beiden Warteschlangen nicht leer, d.h. warten eine oder mehrere Tasks in der Task-Warteschlange auf das Eintreffen von Messages, dann können sich in der Message-Warteschlange keine auf ihre Abholung wartenden Messages befinden. Warten umgekehrt Messages in einer Message-Warteschlange auf ihre Abholung, dann kann in der Task-Warteschlange keine Task auf das
Eintreffen einer Message warten. Diese ist nämlich schon eingetroffen.

Die Mailboxes werden von MEDOS zur Verfügung gestellt. Die Anzahl der vorhandenen Mailboxes ist im Konfigurations-Modul 'CONFIG' einstellbar.

Messages

Der Datenaustausch an Mailboxes erfolgt über Messages. Eine Message wird dabei nicht physikalisch befördert, statt dessen wird von der sendenden Task nur die Adresse der Message an die Mailbox gesendet und von der Empfänger-Task empfangen.

Eine Message muß einen Header fester Struktur besitzen, hinter dem ein Datenpuffer beliebiger Länge und Struktur stehen kann. Die Messages werden vom Anwender selbst definiert.

Semaphoren

Die Semaphoren dienen zur Task-Synchronisation und ermöglichen einen sychronisierten Zugriff auf exklusive Betriebsmittel oder gegenseitigen Ausschluß aus kritischen Bearbeitungsabschnitten (z.B. Zugriff auf gemeinsame Daten). Ein Semaphor ist ein Zähler von abstrakten Einheiten. Es gibt Einheiten an anfordernde Tasks ab und nimmt Einheiten von abgebenden Taks entgegen. Ein Semaphor hat nur eine Warteschlange, die sogenannten Task-Warteschlange (prioritätsgesteuert).
Kann die Anforderung einer Task an Einheiten aus dem Semaphor momentan nicht befriedigt werden, dann wird die Task in die Warteschlange eingereiht und wartet auf die Befriedigung ihrer Anforderung. Gibt eine andere Task Einheiten an dieses Semaphor ab und befinden sich eine oder mehrere Tasks in der Task-Warteschlange, dann wird immer versucht, die Anforderung der ersten Task in der Warteschlange, der höchstprioren, zu befriedigen.

Die Semaphoren werden von MEDOS zur Verfügung gestellt. Die Anzahl vorhandener Semaphore ist im Konfigurations-Modul 'CONFIG' einstellbar.

Interrupt-Schnittstellen

Interrupts und die Interrupt-Bearbeitung sind der Kern eines Echtzeitbetriebssystems. Externe Ereignisse treten asynchron auf und müssen vom Betriebssystem erkannt und vom Anwendungssystem mit möglichst kurzer Reaktionszeit bearbeitet werden.

Prinzipiell unterscheidet man zwischen Hardware-Interrupts und Software-Interrupts.

Hardware-Interrupts

Wenn ein unter MEDOS angemeldeter Hardware-Interrupt das Eintreffen eines externen Ereignisses signalisiert, löst dies ein implizites 'CALL' eines von MEDOS installierten zentralen Interrupt Handlers aus. Jedes Eintreffen eines Hardware-Interruptes ist einem Senden einer Message des Betriebssystems MEDOS an die Interrupt-Schnittstelle gleichzusetzen.

Nicht bei MEDOS angemeldete Hardware-Interrupts werden weiterhin wie vor der Installation von MEDOS durch DOS bearbeitet, wobei die Standard-DOS-Interrupt-Bearbeitung nach wie vor zur Verfügung steht.

Für diejenigen Hardware-Interrupts, über welche MEDOS die Kontrolle übernehmen soll - hierzu gehört in jedem Fall der Zeitinterrupt - muß eine Anmeldung bei MEDOS erfolgen.

Zur Bearbeitung der Hardware-Interrupts stehen unter MEDOS zwei Möglichkeiten zur Verfügung

- Bearbeitung durch eine Interrupt-Service-Routine

  In diesem Fall wird der Interrupt vollständig von einer Interrupt-Service-Routine bearbeitet. Für die Dauer der Laufzeit der Routine
  sind alle Interrupts-Level gesperrt.

- Bearbeitung durch eine Interrupt-Task

  In diesem Fall signalisiert eine Interrupt-Service-Routine des Betriebssystems den Interrupt an eine Interrupt-Schnittstelle und eine an dieser Interrupt-Schnittstelle wartende Task.

Software-Interrupts

PC-MEDOS unterscheidet zwei Software-Interrupts

- DOS-Interrupt-Funktionen, die DOS, DOS-Anwendungen und die verschiedenen MEDOS-Tasks gemeinsam nutzen

- Anwender-Interrupt-Funktionen, die vom Anwender definiert werden und durch die verschiedenen MEDOS-Tasks gemeinsam verwendet werden.

DOS-Software-Interrupts

Einige Funktionen des Betriebssystems DOS sind nicht wiedereintrittsfähig. Trotzdem können Tasks der MEDOS-Anwendung und DOS-Anwendungen uneingeschränkt DOS-Funktionen verwenden, da MEDOS dafür Sorge trägt, daß sich nur jeweils ein einziger Prozess in einer DOS-Funktion aufhalten kann. Hierzu muß der Interrupt, über welchen die DOS-Funktion zu erreichen ist, bei MEDOS angemeldet werden.

Anwender-Software-Interrupts

Für Anwender-Interrupt-Funktionen, die vom Anwender definiert wurden und durch die verschiedenen MEDOS-Tasks gemeinsam verwendet werden, kann eine Anmeldung erfolgen. Dieses Verfahren bietet den Vorteil, daß der Anwender den Interruptvektor für diesen Interrupt nicht selbst zu installieren braucht und seine Interrupt-Service-Routine nicht wiedereintrittsfähig gestalten muß, wenn die Funktion von mehreren MEDOS-Tasks benutzt werden soll.

Zur Realisierung dieser soeben beschriebenen Intertask-Kommunikation stehen für PC-MEDOS folgende Systemaufrufe zur Verfügung:

- SVC007, Message senden

- SVC008, Auf Message oder Zeit warten
- SVC009, Auf Interrupt oder Zeit warten
- SVC010, Auf Zeitstart warten
- SVC029, Semaphor initialisieren
- SVC030, Semaphor belegen
- SVC031, Semaphor freigeben
- SVC040, Kreieren einer Task
- SVC041, Löschen einer Task
- SVC045, Interrupt-Überwachung eröffnen
- SVC046, Interrupt-Überwachung beenden.

Wegen der Kürze der zur Verfügung stehenden Zeit wird nicht auf nähere Einzelheiten dieser Systemaufrufe eingegangen. Vielmehr soll nun noch etwas näher auf die praktische Anwendung von PC-MEDOS eingegangen werden:

Erstellen und Installieren einer MEDOS-Anwendung

Das Echtzeit-Multitasking-Betriebssystem MEDOS ist auf allen IBM-PC-XT- oder IBM-PC-AT-kompatiblen Rechnern lauffähig.
Ausser evtl. Prozeß-Peripherie wird keine zusätzliche Hardware benötigt.

Eine MEDOS-Anwendung entsteht durch das Zusammenbinden eines MEDOS-Datenmoduls (Konfigurierung) und des MEDOS-Betriebssystemkerns mit den verschiedenen vom Anwender erstellten Tasks zu einer '.EXE.'-Datei.

Vom Anwender erstellte Tasks können dabei in Assembler, PLM86, C, Pascal oder in anderen Sprachen, deren Compiler relokatible Intel-Objekt-Formate erzeugen, auf dem Personal Computer entwickelt, mit dem entsprechenden Compiler übersetzt und mit dem MS-Linker zu einer lauffähigen MEDOS-Anwendung gebunden werden.

Zur Installation wird diese '.EXE.'-Datei wie ein DOS-Programm gestartet, wobei sich MEDOS selbst

installiert und DOS als Task niederster Priorität übernimmt. Damit können unter DOS weitere DOS-Anwendungen gestartet werden, welche die durch die ereignisgesteuerte MEDOS-Anwendung ungenutzte Prozessorzeit nutzen.

Soll dagegen der Personal Computer ausschließlich durch die MEDOS-Anwendung genutzt werden, läßt sich die oben erwähnte MEDOS-'.EXE'-Datei mit dem Parameter /P starten.

Reinstallieren einer MEDOS-Anwendung

Durch Eingabe von 'EXIT'auf der Kommando-Ebene der MEDOS-Task DOS werden MEDOS und die MEDOS-Anwendung beendet und aus dem Speicher entfernt. Voraussetzung hierfür ist natürlich, daß DOS als MEDOS-Task installiert wurde. Ist dies nicht der Fall, muß eine der Anwender-Tasksden MEDOS-Aufruf 'M_EXIT'ausführen.

Benutzung von DOS-Funktionen

Unter MEDOS gibt es zwei verschiedene Typen von Prozessen:

- Tasks der MEDOS-Anwendung
- der DOS-Kommando-Prozessor und DOS-Anwendungen (wobei es sich bei den DOS-Anwendungen um Tochter-Prozesse des DOS-Kommando-Prozessors handelt)

Obwohl die Mehrzahl der Funktionen des Betriebssystems DOS nicht wieder eintrittsfähig ist, können beide Typen von Prozessen beinahe uneingeschränkt DOS-Funktionen verwenden, da MEDOS dafür Sorge trägt, daß sich nur jeweils ein einziger Prozess in einer DOS-Funktion aufhalten kann. Dies gilt auch für DOS-Funktionen, die ihrerseits wieder andere DOS-Funktionen aufrufen.

Um Medos zur Koordination mehrerer Prozesse zu veranlassen, welche die gleiche DOS-Funktionen verwenden, wird durch einen Eintrag im MEDOS-Datenmodul (Konfigurierung) oder zur Laufzeit durch einen MEDOS Systemaufruf die entsprechende Interrupt-Nummer zur Überwachung angemeldet.

Folgende Restriktionen sind dabei zu beachten:

Allgemeine Restriktionen

Eine Task oder eine DOS-Anwendung darf nicht während der Ausführung einer DOS-Funktion blockiert werden, da dadurch alle weiteren Prozesse, welche diese DOS-Funktion ebenfalls benutzen, auch blockiert werden können. Eine solche Blockade kann eintreten durch das Anhalten einer Ausgabe mit Ctrl-Numlock oder Ctrl-S (X-off) oder durch eine Ausgabe auf ein nicht bereites Gerät.

Restriktionen für Tasks der MEDOS-Anwendung

Für Tasks der MEDOS-Anwendung bestehen folgende Restriktionen bezüglich der Verwendung von DOS-Funktionen:

- DOS-Funktionen mit implizitem Warten (z.B. gepufferte Eingabe) müssen auch dann, wenn sie nicht quasisimultan von mehreren Prozessen benutzt sind, durch MEDOS überwacht werden (siehe oben), damit sie durch entsprechende MEDOS-Routinen zu ersetzten sind

- Eine Task darf nicht durch Verwendung der Funktion 'Exec' (Int 21H, 4BH) einen Tochter-Prozess erzeugen

- Eine Task darf sich nicht beenden durch Verwendung einer der DOS-Funktionen (Int 21H, 0H), (Int 21H, 31H), (Int 21H, 4CH), Int 20H oder Int 27H.

Restriktionen für DOS-Anwendungen

Für DOS-Anwendungen bestehen folgende Restriktionen bezüglich der Verwendung von DOS-Funktionen:

- DOS-Funktionen mit impliziertem Warten (z.B. gepufferte Eingabe) müssen auch dann, wenn sie nicht quasisimultan von mehreren Prozessen benutzt sind, durch MEDOS überwacht werden (siehe oben), damit sie durch entsprechende MEDOS-Routinen zu ersetzen sind

- Ein gering dimensionierter Stack in verschiedenen DOS-Anwendungen (z.B. dBASE III, dBASE III plus) läßt das Starten eines Tochter-Prozesses innerhalb dieser Anwendung nicht zu.

Zusatzsoftware für PC-MEDOS

Abschließend sei noch darauf hingewiesen, daß z.B. bei Verwendung eines AT-kompatiblen Personal Computers (10 MHz mit Wait-States)
die Task-Umschaltzeit 56 Mikrosekunden und die Zeit für das Senden und das Empfangen einer Message zusammen 706 Mikrosekunden in Anspruch nimmt. Diese Zeiten belegen, das mit PC-MEDOS sehr schnell auf realzeitbedingte Ereignisse reagiert werden kann und deshalb PC-MEDOS problemlos für solche Fälle einsetzbar ist.

Mit PC-MEDOS wurden inzwischen schon mehrere Automatisierungsprojekte realisiert, sodaß für dieses Betriebssystem inzwischen eine Reihe von zusätzlichen Standard-Programmen zur Verfügung steht. Diese reichen von Schnittstellen-Treibern für die serielle Kopplung mit verschiedenen Prozeßdaten-Interface-Systemen (z.B. PC-PROPA von MikroTec GmbH, SIEMENS S5), Device-Treibern für Prozeßein-/Ausgabeplatinen, die direkt in den Personal Computer eingesteckt werden können (z.B. von SIEMENS) sowie Standard-

Programme zur Meßdatenerfassung und Verarbeitung wie z.B.

PC-MARS, Standard-Programm zur Meßwerterfassung, Aufbereitung, Regelung und Steuerung

PC-DIAGRAMM, Balken- und Kurvendarstellung von Analogsignalen

PC-PROLEIT, Farbgrafiksystem für Prozeßleittechnik auf Personal Computern

Gerade die Realisierung von PC-PROLEIT, d.h. die Darstellung von Prozeßbildern in Echtzeit auf dem Personal Computer unter gleichzeitiger Erfassung und Verarbeitung der Signale sowie Abspeicherung dieser Signale auf Platte wäre ohne das Echtzeit-Multitasking-Betriebssystem PC-MEDOS nicht möglich gewesen.

Dieses Beispiel mag nochmals den eingangs erwähnten Hinweis bekräftigen, daß die z.Zt. auf dem Markt befindlichen Personal Computer mit der entsprechend ausgerüsteten Hardware und zusammen mit dem Echtzeit-Multitasking-Betriebssystem PC-MEDOS durchaus für komplexe Automatisierungsaufgaben eingesetzt werden können.

Kommunikation im heterogenen Rechnerverbund

Dipl.-Ing. Werner Gärtner
Badenwerk AG, Abt. EF
Postfach 16 80
7500 Karlsruhe
Tel. 0721/692/2662

Zusammenfassung

Der Einsatz dezentraler Rechner zur Durchführung dedizierter Aufgaben verlangt leistungsfähige Datenaustauschverfahren, um die in den unterschiedlichen Rechnern erzielten Ergebnisse weiteren Bearbeitungsfunktionen unter akzeptablen Zeitbedingungen zugänglich zu machen. An die Datenübertragung und die Datenstruktur auf der Anwenderebene werden i.a. aufgabenspezifische Anforderungen gestellt. Dieser Bereich der Anwendungsprotokolle ist somit nicht vollständig für alle denkbaren Aufgaben durchgehend normierbar. In Zukunft jedoch können sich die speziellen Anwendungen in zunehmendem Maße auf die innerhalb dieses Bereiches entstehenden normierten Serviceelemente stützen. Vorteile dagegen bringt heute schon die Nutzung der inzwischen im Rahmen des ISO- Referenzmodells für offene Systeme genormten Transportprotokolle (Schichten 1 bis 4). Kommunikationssysteme enthalten somit i.a. normierte unterlagerte Funktionseinheiten und anwendungsbezogene Systemteile. Hier soll ein in PEARL programmiertes Anwenderkommunikationssystem für einen heterogenen WAN-/ LAN- Rechnerverbund im EVU- Bereich vorgestellt werden.

1 Ziel des Vortrages

Der Vortrag soll einen Ueberblick ueber das bei der Badenwerk AG Karlsruhe realisierte sogenannte Koppelrechner- Projekt geben, dessen Software zum grossen Teil in PEARL geschrieben ist.

2 Übersicht

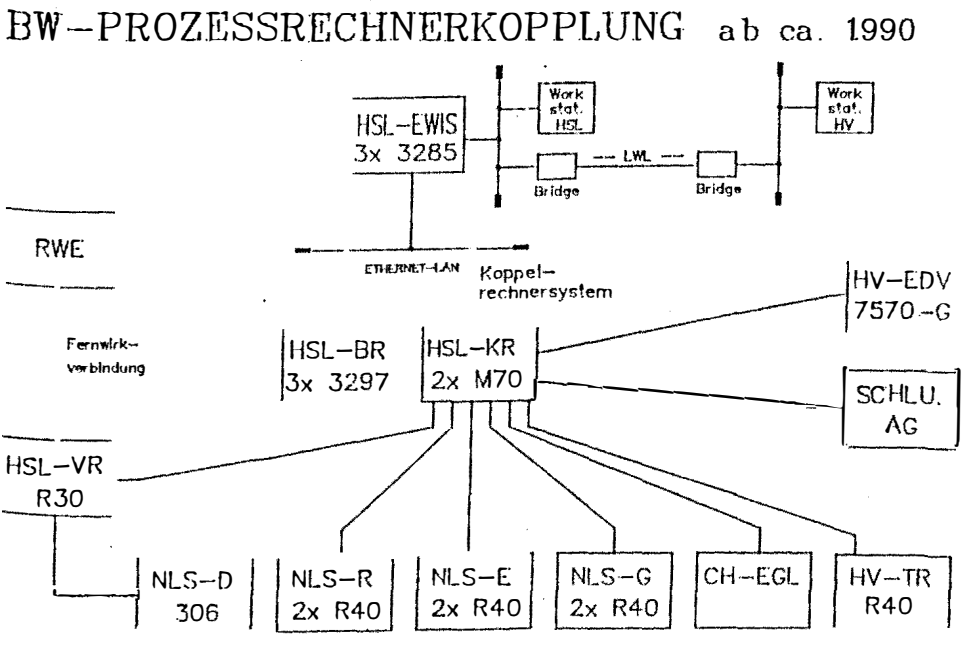
Die Badenwerk AG hat im Rahmen ihres Koppelrechner- Projektes, dessen aufgabenbezogene Software im Auftrag des Badenwerkes und in Zusammenarbeit mit dem Badenwerk von der Fraunhofer- Gesellschaft erstellt wurde, die Grundlage für einen entstehenden heterogenen Rechnerverbund geschaffen. Der Koppelrechner verbindet über ein sternförmiges Netz Rechner verschiedener Fabrikate mit unterschiedlichen Aufgaben miteinander. Das zur Erfüllung der Anforderungen erforderliche Anwender- Kommunikationssystem ist in PEARL-M (Fa. Siemens) programmiert. PEARL-M ist Basis- PEARL mit einigen Ergänzungen. Bei der Auswahl von PEARL als höhere Programmiersprache erhoffte man sich positive Auswirkungen auf das Projekt durch die Realzeiteigenschaften der Sprache sowie durch die Portier- und Wartbarkeit der Softwaremodule.

3 Zweck des Koppelrechner- Projektes

Das Koppelrechner- Projekt wurde erforderlich, um den derzeitigen und zukünftigen netzbezogenen Kommunikationsaufgaben des Badenwerkes eine zukunftssichere Basis zu geben. Der Datenaustausch dient der Bearbeitung verschiedener Aufgaben. Dazu zählen Netzsicherheitsrechnungen für Netze unterschiedlicher Spannungsebenen unter Einbeziehung der Nahbereichsinformationen aus Nachbarnetzen, Störungsanalysen, die Informationsbeschaffung zur Planungs- und Wartungsunterstützung sowie die zentrale Archivierung von wichtigen Meß- und Zählwerten.

Bild 1 zeigt den bis ca.1990 geplanten Ausbau des Netzes mit

dem Koppelrechner- Doppelrechnersystem (HSL-KR; Rechner: 2 x M70, Siemens) als zentralem Kommunikationsknoten. Im derzeitigen Ausbau sind drei 110-kV- Netzleitstellen (NLS), ein Koppel- Vorrechner (HSL-VR; Rechner: R30, Siemens) sowie ein Testrechner (HV-TR)) angeschlossen. Der Vorrechner macht dem Koppelrechner die Daten zweier 306- Rechner (Siemens) zugänglich, darunter auch der derzeit noch tätige Rechner der Hauptschaltleitung Daxlanden. Außerdem adaptiert er eine Ferwirknahtstelle. Das Bild 1 läßt auch die unterschiedlichen Schnittstellen im heterogenen Rechnerverbund erkennen. Es sind sowohl WAN- als auch LAN- Verbindungen zu integrieren. Der LAN- Verbund mit den 3 Rechnern des Betriebsrechnersystems (HSL-BR), den 3 Rechnern des Energiewirtschaftlichen Rechnersystems (HSL-EWIS) und dem KR- Doppelrechnersystem (HSL-KR) wird in den Jahren 1988/89 aufgebaut. An den Koppelrechner sind weitere Rechner anschließbar.



Litengesteuerte Datenübertragung im Rahmen der TNA:
auf Abruf, zyklisch, spontan, einmalig sofort

Bild 1

4 Realisierungsziele in Stichworten

Hinter dem Koppelrechner- Projekt stehen folgende Realisierungsziele:

- Implementierung eines Durchleitbetriebes zur Vermittlung angeschlossener Teilnehmer über unterschiedliche Schnittstellen und Kommunikationsebenen (Gatewayfunktion)
- Implementierung eines Datenbasisbetriebes zur zentralen Vorhaltung aktueller netzbezogener Daten
- erweiterbare zentrale Datenbasis
- listengesteuerter Datenverkehr
- generierbare Steuerlisten
- flexibler Datenzugriff über Anforderungslisten
- Datenschutz in Leserichtung über Zulässigkeitslisten
- automatischer Austausch der Steuerlisten nach Änderung des Listenstatus
- Anschaltung mehrerer Teilnehmer mit einem jeweils teilnehmerspezifischen Steuerlistensatz
- Systemintegration in ein WAN/ LAN- Netz
- erhöhte Systemverfügbarkeit durch ein Doppelrechnersystem mit gleichem Stand der Steuerlisten im prozeßführenden Rechner und im Standby-Rechner
- Nutzung vorhandener Normen im Rahmen des ISO- Referenzmodells
- modulare Struktur der Funktionseinheiten mit der damit verbundenen Portier- und Wartbarkeit sowie der Ausbaufähigkeit des Systems

5 Realisierung der gestellten Anforderungen

5.1 Kommunikationselemente zur Kommunikationssteuerung

Dem Datenaustausch liegen Steuerlisten zugrunde. Die Steuerlisten werden zwischen den Kommunikationspartnern im Rahmen eines Listenabgleichs ausgetauscht. Welche Listen auszutauschen sind, erfahren die Partner über Listenstatusinformationen. Zur Realisierung des Listenabgleichs und des Datenverkehrs wurden geeignete Protokollelemente, die sog. Telegrammidentifikationen, festgelegt. Die Telegrammidentifikationen steuern die Kommunikation auf der Anwenderebene. Das Bild 2 faßt die vereinbarten Identifikationen zusammen. Diese Elemente treten paarweise in entgegengesetzter Übertragungsrichtung auf (Initiierung, Reaktion). Die Elemente geben u.a. Auskunft über den Status der den Datenverkehr steuernden Listen, kennzeichnen Steuerlisten und liefern den Rahmen für zu übertragende Daten. Die Flußkontrolle bei Segmentierung der Datentelegramme wird über zusätzliche ACK- und NACK- Elemente gesteuert, die durch Modifikation der IND- Telegrammidentifikationen entstehen.

Technologische Daten des Netzes (Meldungen, Meßwerte, Zählwerte) werden über numerische technologische Adressen mit 4 bzw. 5 Adreßkomponenten der Hierarchie Station, Spannungsebene, Abzweig, Element/ Meßwertart/ Zählwertart, Adreßattribut angesprochen. Hinter den numerischen Komponenten, die somit einen 4- bzw. 5- Adreßtupel bilden, verbergen sich die alphanumerischen Komponenten des technologischen Namens. Jede numerische Adreßkomponente referiert damit eine Liste mit alphanumerischen Namen, eine sog. Textliste. Die Textlisten für alle Adreßebenen sind beim Listenabgleich auszutauschen.

Ebenfalls auszutauschende Zulässigkeitslisten dienen dem Zugriffsschutz in Leserichtung. Die Zulässigkeitslisten beinhalten Datentypen zur Kennzeichnung der Datenart (z.B. Meldungen) und die zu den Datentypen gehörenden technologischen Adreßtupel der zum Lesen freigegebenen Daten. Die Adreßtupel

beziehen sich auf die Textlisten. Don't care- Adreßkomponenten geben eine gesamte Adreßhierarchie für den Zugriff frei, z.B. alle Meldungen eines Abzweiges.

Teleid.-Bezeichng.	Inhalt	Typ-Nr.	Länge (Byte)	I/R	m/o	s
OPEN-REQ	Listenstati	1	48	I	o	nl
OPEN-CONF	"	2	48	R	o	nl
TAB- REQ	Anforderung Text- oder Zulässigkeits-liste	3	38	I	o	nl
TAB- IND	Text- oder Zulässigkeits-liste	4	38	R	m	jl
ANF- SEND	Anforderungs-liste mit Datentyp und Adressen	5	38	I	m	nl
ANF- IND	Quittung,evtl. mit Daten	6	38	R	o/m	jl
LOE- REQ	Lösch-instruktion für Anf.liste	7	24	I	o	nl
LOE- IND	Quittung	8	24	R	o	nl
ABR- REQ	Datenabruf auf referierte Anford.liste	9	24	I	o/m	nl
ABR- IND	Adressen und Daten	10	24	R	m	jl
DATA-SEND	Adressen und Daten entspr. Anford.liste	11	24	R	m	jl
DATA-IND	Quittung	12	24	R	o	nl
FILE-REQ	spezieller Auftrag	13	24	I	m	nl
FILE-IND	Daten (Files, Protokolle)	14	24	R	m	jl

Bild 2 Telegrammidentifikationen

Abkürzungen:
Typ-Nr. : Nummer zur Kennzeichnung der Telegrammidentifikation
I/R: Initiierung/Reaktion
m/o: mit/ohne nachfolgenden Datentyp- Datenanhang
s: Telegramm kommt segmentiert vor (ja/nein)

Der Datenverkehr selbst stützt sich auf Anforderungslisten. Diese Listen beinhalten wie die Zulässigkeitslisten Datentypen zur Kennzeichnung der gewünschten Datenart und die Adreßtupel der angeforderten Daten. Die Anforderungslisten müssen sich unter Angabe einer Nummer und eines abseits der automatischen Kommunikation auszutauschenden Paßwortes auf eine Zulässigkeitsliste beziehen. Damit überprüft der Empfänger der Anforderungsliste die Zulässigkeit der Datenanforderung. Der anfordernde Teilnehmer verwendet die Anforderungsliste zur Schreibschutz- und Vollständigkeitskontrolle.

Das Protokoll kennt vier Arten von Anforderungslisten:
spontane Listen initiieren eine ereignisgesteuerte Übertragung, zyklische Listen bewirken eine zeitgesteuerte zyklische Übertragung, abrufgesteuerte Listen verursachen jeweils nach Abruf einen Sendevorgang, Listen der Art "einmalig sofort" haben unmittelbar nach ihrem Empfang eine einmalige Datenübertragung zur Folge.

Der Listenabgleich vollzieht sich damit in der Austauschfolge Listenstatus, Textlisten, Zulässigkeitslisten, Anforderungslisten

Text- und Zulässigkeitslisten werden vom Teilnehmer nach Vergleich der ausgetauschten Listenstati angefordert. Anforderungslisten, die auf der Basis der erhaltenen Text- und Zulässigkeitslisten zu erstellen sind, werden an den Teilnehmer gesendet. Damit ist die Grundlage für den nachfolgenden Datenverkehr geschaffen.

Kann ein Teilnehmer aus Leistungsgründen am Listenabgleich nicht teilnehmen, so erlauben lokal erstellte sog. Pseudo-Steuerlisten die Umgehung des Listenaustausches.

Die Listensätze werden teilnehmerspezifisch angelegt. Je Teilnehmer können bis zu 16 Anforderungslisten einer Anforderungslistenart bereitgestellt werden.

Das Bild 3 zeigt die prinzipiellen Zusammenhänge des Listenabgleichs. Dabei tritt der Teilnehmer i als Schreibteilnehmer auf, d.h. er liest Daten vom Teilnehmer k und schreibt sie in seine Datenbasis.

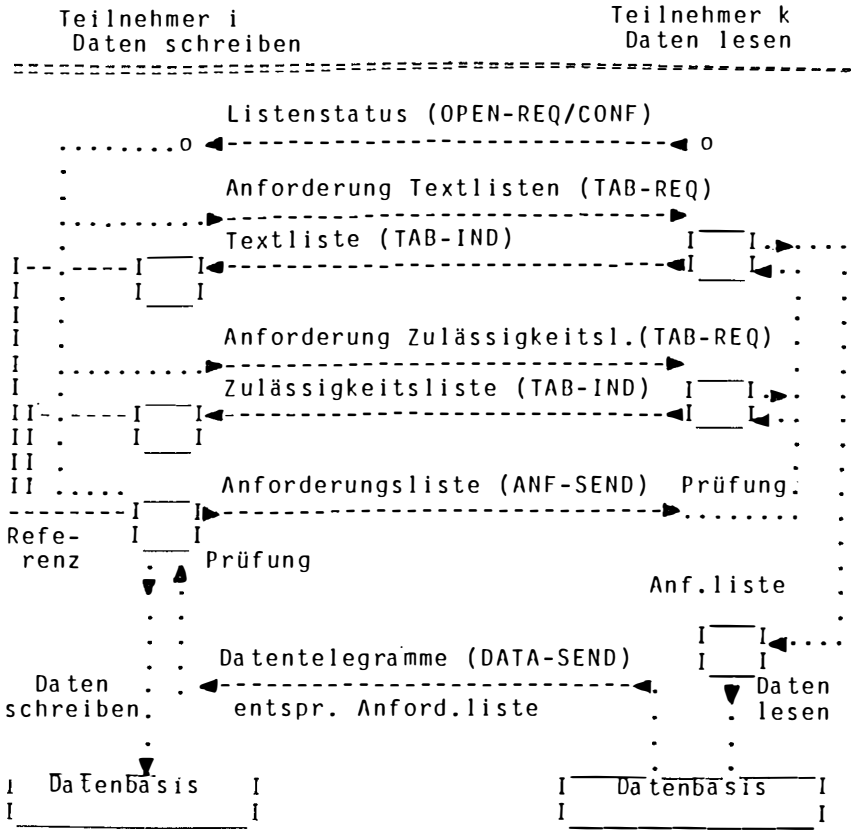


Bild 3 Listenfolge beim Listenabgleich zwischen den Kommunikationspartnern Teilnehmer i und Teilnehmer k; zugehörige Telegrammidentifikationen

5.2 Telegrammstruktur

Dem Datenaustausch liegt eine vereinbarte Telegrammstruktur zugrunde. Anhand des nun zu erläuternden Anwendertelegrammes mit seinen Untereinheiten werden einige analog zum Telegrammaufbau erforderliche hierarchisch angeordnete Bearbeitungsfunktionen erkennbar, auf die noch eingegangen wird.

5.2.1 Prinzipieller Aufbau eines Telegrammes

Das Bild 4 zeigt den Telegrammaufbau. Die Untereinheiten des Telegrammes sind mit Großbuchstaben gekennzeichnet. Das eigentliche Anwendertelegramm reicht von B bis F.

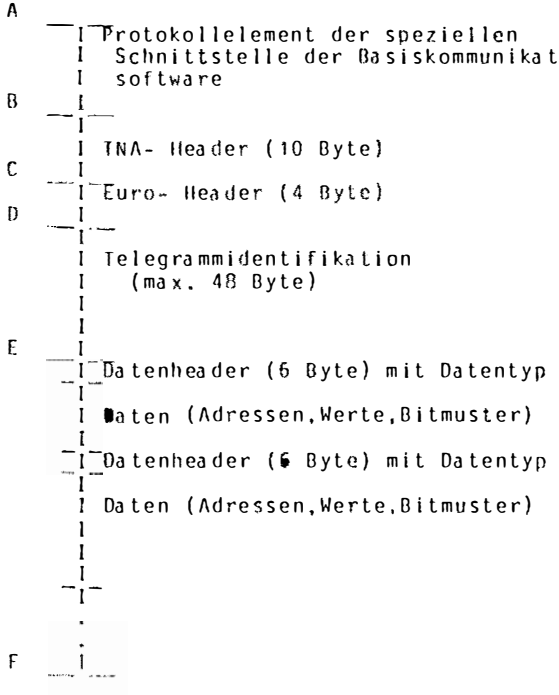


Bild 4 Telegrammaufbau
Laenge A-B : schnittstellenspezifisch
Laenge B-F : =<2048 Byte (Anwendertelegramm)
Laenge D-F : =<2034 Byte

Die Längenangaben beziehen sich auf die vereinbarte Länge eines Anwendertelegrammes bzw. eines Anwendertelegrammsegmentes von max. 2048 Byte. Ist ein Datentelegramm länger als 2048 Byte, so sorgt ein Segmentierungs- und Assemblierungsverfahren auf der Anwenderebene dafür, daß Telegrammsegmente mit einer maximalen Länge von 2048 Byte (Anwenderteil) übertragen werden. Zusammen mit einer begleitenden Flußkontrolle ist damit gewährleistet, daß im heterogenen Netz auch Rechner mit beschränkten Pufferressourcen lange Telegramme verarbeiten können.

5.2.2 Untereinheiten eines Telegrammes

5.2.2.1 Schnittstellenspezifischer Vorspann A- B

Dieser Telegrammteil berücksichtigt die Anforderungen der Schnittstellen der Basiskommunikationssoftware. Im Falle der Transportebene nach ISO 8072/73 treten hier die in der Norm genannten Header auf (/1/).

5.2.2.2 TNA-Header B- C

Die Abkürzung TNA steht für "Telegramme neuer Art". Diese Bezeichnung kennzeichnet die im Rahmen des Projektes spezifizierte Kategorie von Telegrammen. Der TNA- Header des Telegrammes enthält Informationen, die von übergeordneten Bearbeitungsstufen verwendet werden. Dazu zählen u. a. die Zielrechner- und Zielprozeßfindung (Routing, Aktivierung von Gatewayfunktionen) sowie die Identifikation von Telegrammsegmenten. Die Ziel- und Quelladresse besteht jeweils aus drei hierarchisch gegliederten Komponenten (jeweils 1 Byte), nämlich der EVU- Kennung, der Teilnehmerkennung und der Prozeßkennung.

5.2.2.3 Euro-Header C- D

Dieser sog. Europäische Telegrammheader wurde dem UCPTE-Vorschlag für Anwenderdaten entnommen (/2/). Er beinhaltet u.a. die Länge der Information in Byte.

5.2.2.4 Telegrammidentifikation D- E

Die Telegrammidentifikationen steuern die Kommunikation auf der Anwenderebene. Mit ihrer Hilfe führen die Partner den automatischen Listenabgleich als Vorstufe des Datenverkehrs sowie den Datenverkehr selbst durch.

5.2.2.5 Datenbereich E- F

Dieser Bereich des Telegrammes nimmt die Datenheader und die ihnen zugeordneten Daten (Adressen, Werte, Bitmuster) auf. Der Datentyp im 6 Byte langen Datenheader kennzeichnet die unter ihm genannten Daten, z.B. aktuelle Meßwerte. Die Daten bestehen aus einer Folge von Datenelementen mit fester Elementlänge. So werden beispielsweise unter der Telegrammidentifikation ANF-SEND (Anforderungsliste senden) Datentypen zur Datenanforderung aufgeführt. Deren Datenelemente bestehen aus technologischen Adressen und Angaben über das Format des Wertes in der Rückantwort. Die Telegrammidentifikation DATA- SEND beantwortet die Anforderungen in ANF- SEND. Die zu den Anforderungsdatentypen korrespondierenden Datentypen für die Rückantwort spezifizieren die Antwortdatenelemente. Ein solches Element enthält u.a. die technologische Adresse und das angeforderte Datum. Bisher sind 21 von 255 möglichen Datentypen definiert.

5.3 Funktionsbereiche des Softwaresystems

Das System gliedert sich grob gesehen in zwei Funktionsbereiche. Der erste Bereich umfaßt die Funktionen, die nicht nach außen in Erscheinung treten, also keiner Absprache bedürfen. Der zweite Bereich beinhaltet die Funktionen, die das Datenaustauschprotokoll behandeln. Die konkrete Implementierung der Funktionen beider Bereiche ist Sache des Anwenders. Der zweite Bereich muß jedoch die statischen und dynamischen Eigenschaften des vereinbarten Protokolles gewährleisten. Dieses Protokoll war festzulegen, da keine Norm zur Verfügung steht.

Die Implementierung nutzt die Funktionen des vorhandenen Basiskommunikationssystems (SINEC, Siemens), das dem Betriebssystem überlagert ist und Kommunikationsschnittstellen für den homogenen und heterogenen Rechnerverbund zur Verfügung stellt. Die Dienste des Basiskommunikationssystems realisieren zusammen mit Controllerfunktionen Firmenstandards und normierte Standards der Ebenen 1- 4 des ISO- Referenzmodells.

5.4 Funktionen analog zum Telegrammaufbau

Das Bild 5 zeigt in einer prinzipiellen Darstellung die den Telegrammteilen entsprechenden Bearbeitungsstufen innerhalb der implementierten Software. Die real vorhandene Softwarestruktur weicht von dieser am Telegrammaufbau orientierten Darstellung etwas ab. Sie beinhaltet weitere Funktionen, u.a. die Doppelrechnerkomponenten.

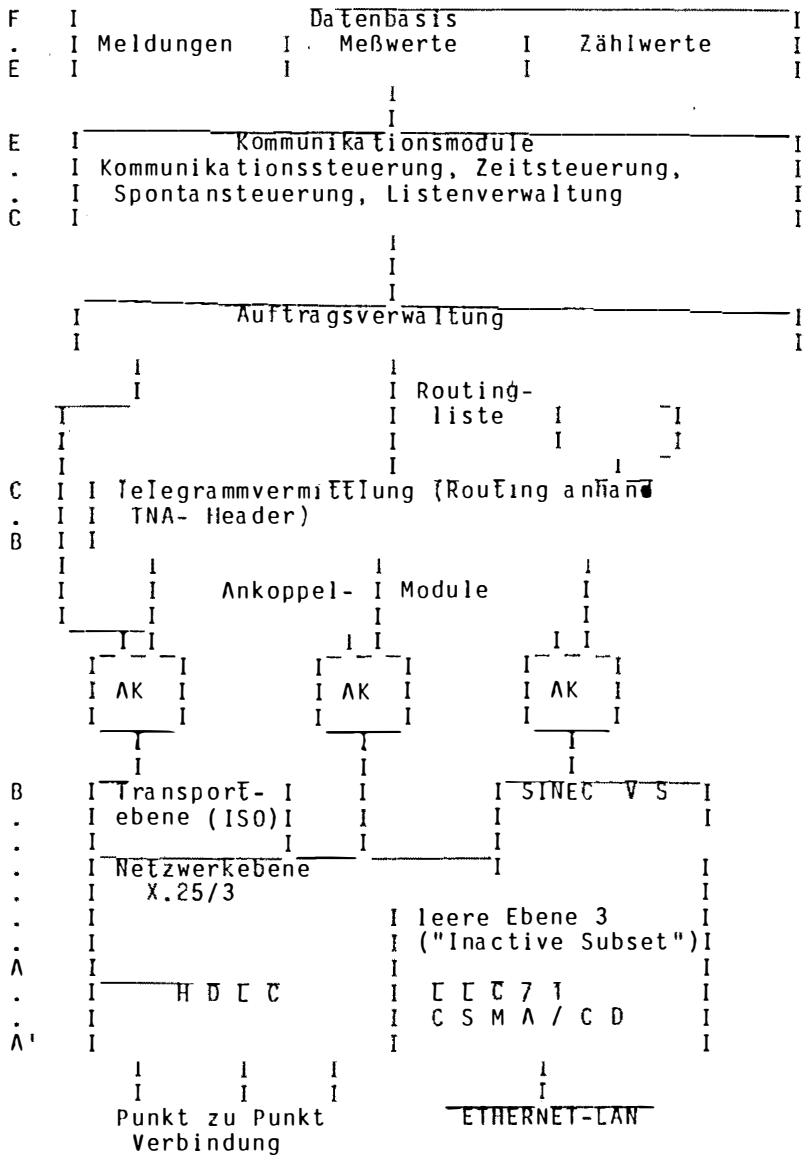


Bild 5 Telegrammbearbeitung

Am linken Rand des Bildes 5 sind die zugehörigen Telegrammabschnitte nach Bild 4 aufgetragen. A- A' kennzeichnet den Protokollvorspann der unteren Ebene. Die Basiskommunikationssoftware und die controllerresidenten Dienste der ISO- Transportebene verwenden den Abschnitt A - B. Die mit AK bezeichneten Ankoppelmodule stellen die Verbindung zwischen der Basiskommunikationssoftware und der Anwenderkommunikationssoftware her. Sie entlasten das Anwendersystem von den Anforderungen der speziellen Schnittstellen, so daß deren Funktionsoberfläche auf die Abwicklung von Sende- und Empfangsaufträgen reduziert wird.

Die Routingfunktion verwendet den Abschnitt B- C des Telegrammes. Das Routing kann mit einer Schnittstellenanpassung verbunden sein. Zu sendende Telegramme werden über ein Auftragssystem den Ankoppelmodulen übergeben. Beim Senden wird eine Sendepriorität berücksichtigt, die der Quellrechner aus der Dringlichkeit der Übertragung ableitet und im TNA- Header vermerkt.

Im Rahmen des Durchleitbetriebes mit eventueller Schnittstellenanpassung verkehren die Teilnehmer im Bereich der Ebenen 1 bis 4 direkt mit dem Koppelrechner. Die Anwenderebene stützt sich dagegen auf ein Ende- zu Ende- Protokoll.

Empfangene Telegramme, die den Datenbasisbetrieb des Koppelrechners adressieren, werden über die Telegrammvermittlung und das Auftragssystem den Kommunikationsfunktionen des Koppelrechners übergeben. Zu sendende Telegramme nehmen den umgekehrten Weg.

Der Koppelrechner bearbeitet in seiner Eigenschaft als Kommunikationsteilnehmer mit Datenbasisbetrieb den Telegrammabschnitt C- F.

Die Kommunikationsfunktionen, die den Listenabgleich und Datenaustausch zwischen den Kommunikationspartnern abwickeln, legen

ihren Aktionen den Telegrammabschnitt D- E (Telegrammidentifikation) zugrunde.

Die Schreib- und Lesefunktionen auf der Datenbasis verwenden den Telegrammabschnitt E- F mit Datentypen und Daten.

Die Telegramme und Steuerlisten werden in einem vom Basiskommunikationssystem zur Verfügung gestellten Puffersystem gehalten und durch Aufträge über das Auftragssystem weitervermittelt.

Spezielle Listengeneratoren erlauben die teilnehmerspezifische Erstellung der Steuerlisten. Die Listen liegen auf Platte und werden im Rahmen ihrer Aktivierung im Puffersystem hinterlegt und der zentralen Listenverwaltung bekanntgegeben.

5.5 Doppelrechnerbetrieb

Im Doppelrechnerbetrieb, das ist in diesem Fall ein Betrieb mit einem prozeßführenden Rechner und einem Standby- Rechner, aktualisiert der Standby- Rechner seinen Steuerlistenstand. Bei Ausfall des prozeßführenden Rechners aufgrund von Programm- oder Gerätefehlern übernimmt der Standby- Rechner die Prozeßführung. Anhand der aktuellen Steuerlisten ist er über die ihm nun zugeschalteten Verbindungen in der Lage, seine Datenbasis aufzufrischen.

6 Ausblick

Mit dem Koppelrechner konnten im Rahmen des kürzlich abgeschlossenen Probebetriebes die ersten Betriebserfahrungen gesammelt werden. Dabei waren 5 Rechner an den Koppelrechner angeschlossen. Vier dieser Rechner sendeten zur Belastung des Koppelrechnersystems u.a. auch kurzzyklisch Daten im 10 sec-Takt. Der fünfte Rechner sendete und empfing Daten. Zur Implementierung der Kommunikationsfunktionen auf diesen Rechnern wurden Softwaremodule des Koppelrechners unter Berücksichtigung

der speziellen Schnittstellen portiert.

In Zukunft werden weitere Teilnehmer an den Koppelrechner angeschlossen. Der vorgesehene LAN- Anschluß erfordert Funktionen, die sich auf eine um die Dienste nach ISO 8072/73 erweiterte Basiskommunikationssoftware stützen. Bei dieser Weiterentwicklung wird sich die modulare Struktur des Softwaresystems sowie der Einsatz von PEARL wie schon bei den durchgeführten Portierarbeiten positiv auswirken.

7 Literatur

/1/ Definition der Dienste der Transportschicht DIN ISO 8072, Spezifikation der verbindungsorientierten Protokolle der Transportschicht DIN ISO 8073

/2/ UCPTE ad-hoc-Gruppe Datenübertragung, Dokument Nr.3, Anwenderverfahren für den Echtzeit- Datenaustausch zwischen Lastverteiler- Rechnern

Zeitmessungen an PEARL-Systemen

von L. Frevert, Fachhochschule Bielefeld

Zusammenfassung

Es werden die Methoden kritisch betrachtet, mit denen die Ausführungszeiten von Anweisungsteilen ermittelt werden können. In der verwendeten Methode wurde gezählt, wie oft Schleifen während fest vorgegebener Zeiten durchlaufen werden. Das führt bei kleinen Ausführungszeiten zu größeren Genauigkeiten, als wenn man die Anzahl der Schleifendurchläufe vorgibt und die Zeiten abliest. Auf der Basis von Mustermoduln sind Module für Messungen an einer große Anzahl verschiedener Elementaroperationen relativ rasch erzeugbar; die Meßprogramme benötigen keine Parametrisierung. Bisher wurden je etwa 130 verschiedene Messungen auf zwei Rechnertypen durchgeführt. Die Meßergebnisse ermöglichen es, Programmierern Hinweise auf Optimierungsmöglichkeiten zu geben, Compilerbauer auf bisher unerkannte Schwachstellen hinzuweisen und detaillierte Leistungsvergleiche zwischen verschiedenen PEARL-Systemen aufzustellen.

1. Einleitung

Gute Assemblerprogrammierer kennen normalerweise die Ausführungszeiten der Maschinenbefehle aufgrund von Angaben des jeweiligen Prozessor-Herstellers und können deshalb zeitkritische Programmteile optimieren. Einigermaßen genaue Leistungsvergleiche verschiedener Prozessortypen allein anhand der Befehlsausführungszeiten sind jedoch schwierig, weil unterschiedliche Befehlssätze unterschiedliche Optimierungsstrategien erfordern. Deshalb werden solche Leistungsvergleiche üblicherweise anhand von sogenannten Benchmark-Programmen vorgenommen, deren Laufzeit mit der Stoppuhr gemessen wird. Jedes Benchmark-Programm entspricht jedoch einem ganz bestimmten Benutzerprofil und ist nur dann von praktischem Wert, wenn dieses Benutzerprofil und die geplante Anwendung sich

nicht zu sehr unterscheiden.

Einem Anwender einer höheren Programmiersprache ist es in der Regel nicht möglich, Programme gezielt zu optimieren, weil er nicht weiß, wieviel Rechenzeiten bestimmte Sprachkonstrukte erfordern, da hier zusätzlich die Güte des verwendeten Compilers eingeht und weil bei konventionellen Sprachen detaillierte Messungen derartiger Zeiten relativ aufwendig sind. Üblicherweise macht man auch hier nur pauschale Leistungsvergleiche mit Benchmark-Programmen. Die Ergebnisse sind dabei oft überraschend: Ein kurzes Programm mit FLOAT-Rechnungen und Benutzung trigonometrischer Funktionen benötigte auf demselben Rechner (einem ATARI 1040 ST+) bei Programmierung in PEARL (RTOS-UH-Compiler) 25 Sekunden Laufzeit, bei Verwendung von PASCAL (MCC-PASCAL) 77 Sekunden, bei C (Megamax C bzw. Lattice C) 85 bzw. 318 Sekunden. (Alle Rechnungen mit einfacher Genauigkeit; bei doppelter Genauigkeit benötigte PEARL 55 Sekunden.)

Bei Computern in Rechenzentren, auf denen Programme mit vielen verschiedenen Anwendungsprofilen laufen, mögen pauschale Leistungsvergleiche mit Benchmark-Programmen ausreichend sein. Bei Prozeßrechnern, die für eine bestimmte Aufgabe eingesetzt werden sollen, können solche Messungen jedoch nur ungefähren Anhalt für deren Eignung geben. Deshalb gibt DIN 19242 Meßvorschriften und Beispiele für "synthetische Benchmarks", mit denen gezieltere Leistungsvergleiche auf Rechnern mit Echtzeitbetriebssystemen möglich sind. Insbesondere beim Einsatz für zeitkritische Aufgaben sollte der Programmierer jedoch auch im einzelnen wissen, welche Ausführungszeiten zum Beispiel Prozeduraufrufe kosten und wie sich Rechnungen mit einfacher oder doppelter Genauigkeit zueinander verhalten.

Rechenzeitmessungen an Echtzeitsystemen sind einerseits leicht automatisierbar, weil derartige Systeme per Programm ablesbare Uhren haben, andererseits aber auch schwieriger, weil man hier nicht nur die Ausführungszeit arithmetischer Anweisungen, sondern auch diejenige für Betriebssystem-Funktionen wie Taskwechsel und

Reaktion auf Interrupts kennen möchte. Bei Ein/Ausgabe-Operationen interessiert hier nicht nur die gesamte Ausführungszeit, sondern auch die Zeit, die der Rechnerkern dabei benötigt, der bei modernen Systemen aus mehreren Prozessoren bestehen kann.

Der PEARL-Anwender-Ausschuß beschäftigt sich seit einiger Zeit mit dem Leistungsvergleich von PEARL-Systemen. Auf seine Anregung sind an der Fachhochschule Bielefeld über hundert PEARL-Module entstanden, von denen jeder die Prozessorzeit einer einzelnen Anweisung oder eines Anweisungsteiles mißt. Die Meßresultate sollen dem Anwender dabei helfen, die Eignung eines PEARL-Systems für seine spezielle Prozeßsteuerungs-Aufgabe zu beurteilen und ihm bei der Programmentwicklung helfen, zeitaufwendige Anweisungsfolgen durch bessere zu ersetzen. Um ein Beispiel zu nennen: beim RTOS-UH auf dem ATARI 520 ST+ dauert "IF BITGESETZT=='1'B THEN...." mehr als zwanzigmal länger als das gleichwertige "IF BITGESETZT THEN...." Für den Compilerbauer resultiert aus solchen Meßresultaten der Anreiz, sein Produkt zu überarbeiten.

2. Zeitmessung bei vorgewählter Wiederhol-Häufigkeit

Bei Echtzeitsprachen wie PEARL scheint es relativ einfach zu sein, die Rechenzeiten einzelner Anweisungen oder Anweisungsteile zu messen, weil die interne Uhr des Rechners von Meßprogrammen abgefragt werden kann. Um einigermaßen genaue Ergebnisse zu bekommen, muß man sich jedoch die möglichen Fehlerquellen überlegen.

Die einfachste und naheliegendste Meßmethode besteht daraus, die zu untersuchende Operation (z. B. "C:=A*B;") in einer Schleife n-mal zu wiederholen und die dafür benötigte Zeit T_a zu messen, indem die Uhrzeiten vor und nach der Schleifenausführung per Programm abgelesen werden; eine zweite Messung mit einer Vergleichsanweisung "C:=A;" im "Testrahmen" ergibt dann die Vergleichszeit T_v. Die Ausführungszeit t_a einer einzelnen Operation (hier der Multiplikation) ist dann $t_a=(T_a-T_v)/n$.

3. Betrachtungen über Meßgenauigkeiten

Die Genauigkeit einer derartigen Messung wird in erster Linie von der Genauigkeit bestimmt, mit der die Uhrzeiten abgelesen werden können. Die interne Uhr des Rechners wird mit einem Taktabstand T_{takt} weitergesetzt. (Er kann durch mehrmaliges Ablesen der Uhr in sehr kurzen Abständen bestimmt werden.) Bild 1 zeigt, daß bei einer gemessenen Zeitdifferenz von 3 Takten die wirkliche Zeitdifferenz knapp über 2 Takten oder knapp unter 4 Takten liegen kann. Deshalb sind Messungen von Uhrzeitdifferenzen mit einer Ungenauigkeit von T_{takt} behaftet.

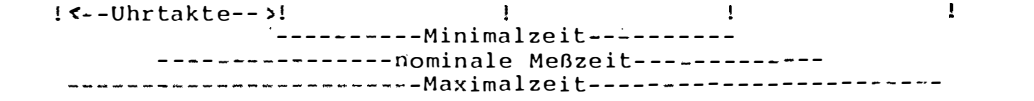


Bild 1: Taktabstand der Uhrtakte und daraus resultierende Meßzeitfehler.

Der Meßfehler des Termes $T_a - T_v$ ist deshalb höchstens $2 * T_{takt}$. Für den relativen Fehler der Ausführungszeit ergibt sich daraus nach Umformung von $(T_a - T_v)$

$$\frac{\Delta t_a}{t_a} = \frac{2 * T_{takt}}{T_a} * \frac{1}{1 - \frac{T_v}{T_a}}$$

Die Genauigkeit, mit der t_a bestimmt werden kann, hängt deshalb empfindlich vom Verhältnis von T_v zu T_a ab. Liegt dieses nahe bei Eins, kann ausreichende Genauigkeit nur durch Vergrößerung von T_a gewonnen werden.

Eine weitere Fehlerquelle kann darin liegen, daß ein PEARL-Betriebssystem bei jedem Interrupt der Rechneruhr prüfen muß, ob eventuell eine eingeplante Task gestartet oder fortgesetzt werden muß. Diese Interrupt-Bearbeitungszeit geht von der Prozessorzeit

ab, die für Benutzerprogramme (hier die Zeitmeßprogramme) zur Verfügung steht und kann nicht mit Hilfe von Benutzerprogrammen gemessen werden. Bei Mehrbenutzer-Systemen sucht das Betriebssystem außerdem in festen Zeitabständen nach weiteren Benutzern; auch diese Rechenzeit geht von der Zeit für Benutzerprogramme ab. (Selbstverständlich dürfen Zeitmessungen nur gemacht werden, wenn keine weiteren Benutzer auf dem System sind.)

Auf den ersten Blick könnte man annehmen, daß diese Rechenzeiten für Betriebssystem-Prozesse den Rechner für Benutzerprogramme nur scheinbar langsamer machen. Man darf jedoch nicht davon ausgehen, daß die Rechenzeiten für diese Routinen konstant sind. Messungen auf einem Rechner mit Mehrbenutzer-Betriebssystem deuten darauf hin, daß dort die Rechenzeiten für die Bearbeitung der Uhr-Interrupts innerhalb von 10 Sekunden insgesamt um etwa 3 Millisekunden schwanken.

Einen weiteren, wenn auch geringen Beitrag zum Meßfehler kann auch die Ablesung und Aufbereitung des Standes der Rechneruhr und ihre Speicherung in Variablen des Meßprogrammes liefern. Die dazu benötigten Rechenzeiten heben sich zwar wegen der Differenzbildung $T_a - T_v$ in erster Näherung heraus; bei genauer Betrachtung darf man jedoch nicht voraussetzen, daß die zur Aufbereitung ausgeführten arithmetischen Operationen konstante Rechenzeiten haben.

4. Das Verfahren nach DIN 19242

Bei der bisher beschriebenen Meßmethode wird die gesamte Ausführungszeit einer Anweisung (in DIN 19242 "Verweilzeit" genannt) gemessen. Bei modernen Rechnern laufen E/A-Anweisungen größtenteils parallel zur Arbeit des Rechenprozessors; letzterer wird nur dazu benutzt, deren Ausführung anzustoßen. Um nur die dafür benötigte Zeit (in DIN 19242: "Belegungszeit des Zentralprozessors") zu messen, werden in DIN 19242 zusätzliche Messungen vorgeschrieben: während der Messung der Verweilzeit läuft im Hin-

tergrund eine Task (DIN 19242: "Hintergrundprogramm") mit geringerer Priorität. Wie Bild 2 zeigt, verlängert sich auf einer Einprozessormaschine die Ausführungszeit dieser Hintergrundtask, weil ihr zugunsten der Verweilzeitmessung Rechenzeit entzogen wird. Falls die Meßtask den Rechnerkern zwischenzeitlich freigibt, kann die Hintergrundtask derweil weiter arbeiten. Bei E/A-Anweisungen wird die Ausführungszeit der Hintergrundtask deshalb nur um die Zeit verlängert, die das Betriebssystem auf dem Zentralprozessor für den Anstoß der E/A und für die Taskwechsel benötigt.

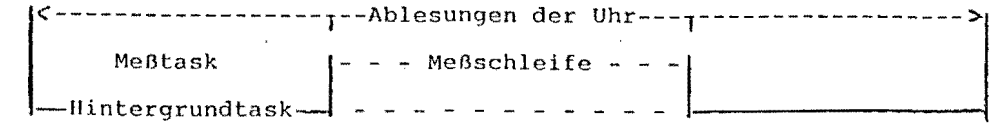


Bild 2: Zeitmessungen nach DIN 19242: Die Zeit zwischen Beginn und Ende einer Hintergrundtask verlängert sich dadurch, daß die Meßtask ihr den Prozessor entzieht.

Die Belegungszeit des Zentralprozessors durch die Meßschleife kann deshalb bestimmt werden, indem man die Zeiten miteinander vergleicht, um die sich die Hintergrundtask verlängert, wenn man sie mit und ohne die Meßschleife laufen läßt. Bei Mehrprozessormaschinen ist das Verfahren in dieser einfachen Form allerdings nicht verwendbar, sondern muß so erweitert werden, daß so viele Hintergrundtasks benutzt werden, wie Prozessoren vorhanden sind.

Das Verfahren von DIN 19242 ist entwickelt worden, um die Verweil- und Belegungszeiten von "synthetischen Benchmarks" zu bestimmen. Sie bestehen aus Anweisungsfolgen. Beim Vergleich ist der Testrahmen leer. Da zusätzlich vorgeschrieben wird, daß T_a im Minutenbereich liegen soll, ist bei Messungen nach DIN 19242 sowohl T_{takt} klein gegen T_a und die anderen erwähnten Fehlerzeiten, als auch das Verhältnis von T_v zu T_a klein gegen Eins, sodaß die dadurch verursachten Fehler vernachlässigt werden können.

Ziel der hier beschriebenen Messungen war jedoch, die Ausführungszeiten von Teilen einzelner Anweisungen genau zu messen. Wenn dabei kleine Unterschiede zwischen Meß-Operation und Vergleichsoperation bestehen, z.B. bei der Messung der für eine Parameterübergabe in eine Prozedur erforderliche Zeit, sind bei der Verwendung der DIN-Methode Meßzeiten von mehreren Minuten erforderlich. Man muß außerdem bei jedem Einzeltest anhand vorläufiger Ergebnisse die notwendige Laufzeit ermitteln und im Testprogramm die notwendige Anzahl der Schleifendurchläufe entsprechend festlegen. Diese Parameterwahl läßt sich zwar automatisieren, erfordert dann aber relativ aufwendige Programme.

5. Zählung der Wiederholungen innerhalb vorgewählter Meßzeit

Bei Verwendung von PEARL ist eine Verwendung kleiner, immer gleicher Meßzeiten dadurch möglich, daß man nicht die Laufzeiten für eine feste Anzahl von Schleifendurchläufen mißt, sondern bei fest vorgegebener Laufzeit T die Anzahl N_a der in dieser Zeit ausgeführten Schleifendurchläufe bestimmt und mit der Anzahl N_v der in derselben Zeit ausgeführten Vergleichsschleifen vergleicht. Die Laufzeit wird dabei durch eine mit höherer Priorität laufende Steuertask bestimmt, die sich mit der Rechneruhr synchronisiert (Bild 3).

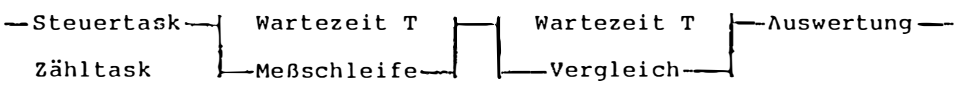


Bild 3: Die zur Zeitmessung benutzt Zähltask kann die Meßschleife nur ausführen, während die Steuertask wartet. Ausgewertet wird die Zahl der Schleifenwiederholungen während dieser Wartezeiten.

Die Zähltask besteht dabei aus einer Schleife, in der die Meßoperation und die (eventuell leere) Vergleichsoperation in bedingt durchlaufenen Programmteilen stehen (Bild 4). Diese Schleifenorganisation hat den Vorteil, daß sie auch bei optimierenden Compilern benutzt werden kann.

```
ZAEHLTASK: TASK PRIO 5;
   WHILE SCHLEIFE REPEAT
     IF MESSEN THEN
        /* Na hochzählen */
        /* zu messende Anweisung */
     FIN;
     IF VERGLEICHEN THEN
        /* Nv hochzählen    */
        /* Vergleichsanweisung */
     FIN;
   END;
END;
```

Bild 4: Wesentlicher Teil der Meßtask, welche die Schleifendurchläufe für Meß- und Vergleichsoperation zählt.

Die Steuertask startet die Zähltask, synchronisiert sich mit der Rechneruhr, setzt die Steuervariablen und wertet die Messungen aus (Bild 5).

```
STEUERTASK:TASK PRIO 1;
   SCHLEIFE='1'B; MESSEN:='0'B; VERGLEICHEN:='0'B;
   /* Zählvariable nullsetzen */
   ACTIVATE ZAEHLTASK;
   AFTER SYNCZEIT RESUME;  /* Synchronisierung mit Uhr */
   VERGLEICHEN:='1'B;     /* zuerst Vergleichsmessung */
   AFTER T RESUME;        /* Messzeit abwarten */
   VERGLEICHEN:='0'B;
   AFTER SYNCZEIT RESUME;  /* Synchronisierung mit Uhr */
   MESSEN:='1'B;
   AFTER T RESUME;  /* gleiche Zeit für eigentliche Messung */
   MESSEN:='0'B;
   SCHLEIFE:='0'B;       /* Zähltask beendet sich später */
   /* Zählungen auswerten */
END;
```

Bild 5: Die Steuertask synchronisiert sich mit der Rechneruhr, setzt Steuerbits, die durch die Meßtask abgefragt werden, und wartet während der Meßzeit.

Das Verfahren mißt in der bisher beschriebenen Form auch bei E/A-Anweisungen die gesamte Ausführungszeit; man kann jedoch auch die Rechnerkern-Belegungszeit leicht bestimmen, indem man noch eine weitere (bei Mehrprozessormaschinen mehrere) Hintergrund-Task mit geringerer Priorität als die Zähltask laufen läßt, die immer dann zählt, wenn die Zähltask den Rechnerkern freigibt (Bild 6). Eine

dritte Vergleichsmessung bei terminierter Zähltask und zählender Hintergrundtask ermöglicht dann, die Auslastung des Rechnerkerns durch die E/A-Operation zu bestimmen.

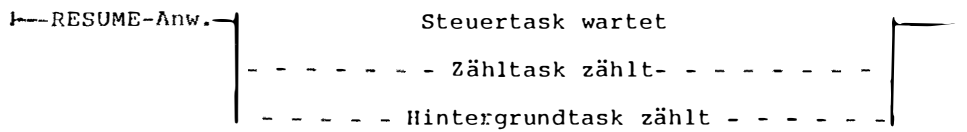


Bild 6: Messung der Prozessorzeit bei E/A-Anweisungen: während die Zähltasks den Prozessor nicht benötigt, kann die Hintergrundtask Schleifendurchläufe zählen.

Seien

- T die Meßzeiten (AFTER T RESUME)
- t_a die Ausführungszeit der zu messenden Operation,
- t_k die Rechnerkernzeit der zu messenden Operation,
- t_v die Rechnerkernzeit der Vergleichsoperation,
- t_s die Rechnerkernzeit für Schleifenorganisation und Zählen der Schleifendurchläufe,
- N_a die Zahl der Schleifendurchläufe, in denen die zu messenden Operation und die Vergleichsoperation beide von der Meßtask aus geführt werden,
- N_v die Zahl der Schleifendurchläufe, in denen nur die Vergleichs-Operation ausgeführt wird,
- N_h die Zahl der Schleifendurchläufe der Hintergrund-Task während der Messung von N_a,
- N_l die Zahl der Schleifendurchläufe der Hintergrund-Task, während die Meßtask nicht läuft

so gelten folgende Gleichungen, wobei sich die ersten beiden auf dieselbe Messung beziehen:

$$T=N_a*(t_a+t_v+t_s)$$
$$T=N_a*(t_k+t_v+t_s)+N_h*t_s$$
$$T=N_v*(t_v+t_s)$$
$$T=N_l*t_s$$

aus denen sich

$$t_a = \frac{T}{N_a} * \left(1 - \frac{N_a}{N_v}\right)$$

und

$$t_k = \frac{T}{N_a} * \left(1 - \frac{N_a}{N_v} - \frac{N_h}{N_l}\right)$$

ergeben.

6. Fehlerbetrachtung

Eine der Fehlerquellen des zuletzt beschriebenen Verfahrens besteht daraus, daß die Zähltask am Ende der Messung während der Ausführung der Meßschleife durch die Steuertask unterbrochen wird. Dadurch ist es ungewiß, ob die gezählte Operation schon durchgeführt ist: die Zählwerte N sind mit einer Unsicherheit von Eins behaftet. Aus diesem Grunde ist das Verfahren (bei vergleichbaren Meßzeiten) ungenauer als DIN 19242, wenn die Ausführungszeit der gezählten Operation größer ist als T_{takt}, während es im umgekehrten Falle genauer ist.

Auf den ersten Blick könnte man annehmen, daß die Werte für N_a und N_v mit dieser Unsicherheit von höchstens Eins gezählt werden können. Bei wiederholten Messungen mit Anweisungen von sehr kurzer Ausführungszeit ergeben sich jedoch Schwankungen von N_a oder N_v, die größer als Eins sind. Sie werden vermutlich dadurch verursacht, daß die Zeiten für diejenigen Betriebssystem-Prozesse (Bearbeitung der Uhr-Interrupts), die das Meßprogramm unterbrechen, nicht konstant sind. Beim ATARI 510 ST+ betragen diese Schwankungen bei reinen Zählschleifen höchstens 4 in den Zählungen der Schleifendurchläufe (insgesamt ca. 500000) bei einer Meßzeit von insgesamt 10 Sekunden, beim einem Prozeßrechner mit Multiuser-Betriebssystem etwa 70 (bei insgesamt etwa 250000). Man kann diese Zahlen umrechnen in einen relativen Fehler $\Delta T/T$ von rund 0.001% bzw. 0.03%.

Ein weiterer Fehler kann bei den verwendeten Meßzeiten T von 10 Sekunden vernachlässigt werden: die wirkliche Meßzeit ist kleiner als die nominelle Meßzeit T, weil T ab dem letzten Uhrtakt gerechnet wird, der die Fortsetzung der Steuertask verursacht hat, die Zähltask aber während des Neusetzens der Steuerbits und der Einplanungs-Anweisung "AFTER T RESUME;" noch nicht laufen kann. Für Einplanungen werden bei den untersuchten PEARL-Systemen höchstens 3 Millisekunden benötigt: der ATARI 520 ST+ bzw. der Vergleichsrechner benötigen für eine RESUME-Einplanung inklusive Wiederstart der Task 0.5 bzw. 2.5 Millisekunden.

Der relative Fehler des Meßergebnisses t_a ergibt sich als Summe der relativen Fehler von T, N und dem Klammerterm. Beim Klammerterm ist der relative Fehler

$$\left(\frac{1}{N_a} + \frac{1}{N_v} + 2 * \frac{\Delta T}{T}\right) * \frac{\frac{N_a}{N_v}}{1 - \frac{N_a}{N_v}}$$

und wird stark von der Größe des Quotienten N_a/N_v beeinflußt. Wenn N_a und N_v ungefähr gleich sind, geht er gegen $2/(N_v-N_a)$, kann also untragbar groß werden.

Die für die Messungen entwickelten Programme berechnen auch die Meßunsicherheiten. Sie lagen bei allen Messungen unter 10%, bei den meisten sogar unter 1%.

7. Messung von Tasking-Anweisungen

Auch die Messung von Tasking-Anweisungen ist mit dem beschriebenen Verfahren möglich. Um z. B. zu bestimmen, wieviel Zeit die Ausführung einer ACTIVATE-Anweisung, Start und Beendigung einer Task erfordern, wurde in der Zähltask eine leere Task aktiviert,

deren Priorität unterhalb der Steuertask und über der Zähltask lag (Bild 7).

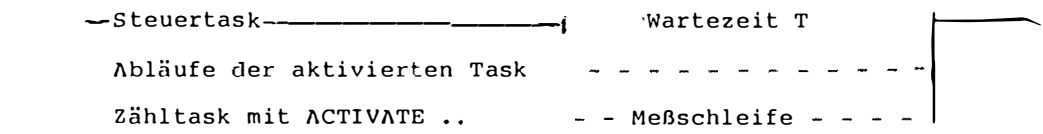


Bild 7: Messung der Zeit für Taskaktivierung und -start: eine Task höherer Priorität wird in der Meßschleife immer wieder aktiviert und läuft entsprechend oft.

Etwas schwieriger ist die Messung der Zeiten für SUSPEND - CONTINUE. Auf den ersten Blick scheint die Sache einfach: in die zusätzliche Task A wird "IF MESSUNG THEN SUSPEND; FIN;" geschrieben, in die Meßschleife "IF MESSUNG THEN .... ACTIVATE A; CONTINUE A; FIN;". Um die Zeit für das "ACTIVATE A;" zu eliminieren, liegt es nahe, beim Vergleichen eine leere Task B mit "ACTIVATE B;" zu aktivieren. Das kann jedoch zu Meßfehlern führen, weil die Dauer von Tasking-Anweisungen von der Länge der Tasklisten des Betriebssystems und dem Eintragungsort abhängen kann (beim RTOS-UH beispielsweise auch von der Speicherbelegung). Es darf deshalb nur eine zusätzliche Task für diese Messung geben; sie muß "IF MESSUNG THEN SUSPEND; FIN; IF VERGLEICHEN THEN ; FIN;" enthalten, so daß sie die SUSPEND-Anweisung nur ausführt, wenn das Bit MESSUNG von der Steuertask gesetzt ist. Das wieder führt zu der Komplikation, daß Meßschleife und A die Steuerbits nacheinander abfragen und diese zwischenzeitlich von der Steuertask geändert werden können, sodaß SUSPEND und CONTINUE nicht mehr paarig ausgeführt werden. Steuertask, Meßtask und Task A mußten deshalb über Semaphore sauber koordiniert werden. Ohne diese Koordination kann es insbesondere bei der Messung von Semaphoranweisungen zu Verklemmungen kommen.

8. Anwendungserfahrungen

Es wurde zunächst versucht, bei dem Verfahren nach DIN 19242 die Festlegung der Meßzeit dadurch zu automatisieren, daß das Programm zuerst die Ausführungszeiten der zu testenden Operationen grob bestimmte und daraus die Anzahl der notwendigen Wiederholungen berechnete. Wegen der sich ergebenden großen Laufzeiten wurde dieser Weg dann nicht weiter verfolgt, sondern die zuletzt beschriebene Methode entwickelt. Einige Anweisungen wurden mit beiden Methoden untersucht; die Ergebnisse stimmten erwartungsgemäß gut überein.

Mit der neuen Methode wurde dann eine große Anzahl Messungen für verschiedene Anweisungen und Anweisungsteile auf einem Prozeßrechner und an einem RTOS-UH-PEARL-System auf einem ATARI 520 ST+ durchgeführt. Die Meßzeiten T betrugen dabei durchweg 10 Sekunden. Auf beiden Rechnern liefen selbstverständlich während der Messungen keine weiteren Programme. Für einige Anweisungen wurden mehrere Messungen durchgeführt, um ihre Reproduzierbarkeit untersuchen zu können. Diese war bei arithmetischen Anweisungen sehr gut. Wenn das Betriebssystem involviert war, ergaben sich in einigen Fällen größere Schwankungen, was wohl darauf zurückzuführen ist, daß die fraglichen Betriebssystem-Routinen je nach Speicherbelegung und Vorhandensein ruhender Tasks unterschiedliche Laufzeiten haben. Beim ACTIVATE auf dem ATARI wurden z. B. Werte zwischen 0.9 und 2.25 Millisekunden gemessen,

Praktisch wurde so vorgegangen, daß alle rechnerspezifischen Programmteile, insbesondere der Systemteil, in einem Modul INIT zusammengefaßt wurden, der auch die Unterprogramme für Initialisierung, Auswertung und Protokollierung der Messungen enthält. Außerdem wurde ein Mustermodul entwickelt, in den die zu testenden Anweisungen und die Texte für die Protokollierung, sowie Task- und Prozedurnamen eingesetzt wurden, um die Testmodule zu gewinnen.

Je neun der so entstandenen Testmodule wurden mit dem schon er-

wähnten Modul INIT und einem Modul für die Starttask zu einem PEARL-Programm gebunden. Mit Hilfe einiger Unterstützungsprogramme dauerte die Erzeugung inklusive Probelauf eines derartigen PEARL-Programmes für neun Einzelmessungen etwa eine Stunde.

9. Meßergebnisse

Insgesamt sind bisher etwa 130 verschiedene Operationen auf einem ATARI 520 ST+ (mit RTOS-UH Version 2.0) und auf einem Prozeßrechner untersucht worden. Dabei wurden einige "Ausreißer" entdeckt: EXOR dauert auf dem ATARI etwa zwanzigmal länger als AND oder OR; eine Anweisungsfolge REQUEST S; RELEASE S; (ohne Warten) benötigt auf dem Prozeßrechner etwa etwa zehnmal länger als auf dem ATARI, der nur 32 Mikrosekunden braucht. Überraschend war auch, daß die Zeiten auf dem Prozeßrechner etwa doppelt so hoch waren, wenn statt auf tasklokale auf modulglobale FIXED-Variable zugewiesen wurde.

Bei dem Prozeßrechner handelte es sich um einen KRUPP ATLAS ELEKTRONIK EPR 1300, der 1979 ausgeliefert worden ist. Auch das PEARL-System ist alt (Stand von 1980), weil sich eine Fachhochschule den Kauf neuerer Versionen nicht leisten kann. Beim RTOS-UH ist die CHAR-Verarbeitung gegenüber der ersten Version bedeutend schneller geworden; es wäre interessant, auch auf einem neuen EPR 1300 Messungen vorzunehmen.

Pauschal kann man über einen Vergleich beider Rechner folgendes aussagen: bei Operationen zur CHAR-Verarbeitung ist der ATARI mindestens zehnmal schneller als der andere Rechner (beim RTOS-UH kann man jedoch nicht auf Teilketten, sondern nur auf Einzelzeichen aus CHAR-Ketten zugreifen), bei Zuweisungen, Schleifenorganisation und einfachen IF-Verzweigungen etwa doppelt so schnell. Dafür sind beim Prozeßrechner Bitshift-Operationen und FLOAT(23)-Rechnungen etwa fünfmal schneller. Aktivierungen mit Taskstart führt der Prozeßrechner schneller aus (0.77 ms), dafür braucht er für Taskstarts per Einplanung (1.23 ms) und für RESUME (2.5 ms)

etwa drei- bzw. viermal so lange wie der ATARI.

Programmstrukturierung durch Verwendung von Prozeduren kostet Zeit: Aufrufe von RESIDENT REENT-Prozeduren verbrauchen beim Prozeßrechner 1.2 Millisekunden, beim ATARI dagegen nur 0.25; RESIDENT-Prozeduren hingegen werden vom Prozeßrechner in nur 0.11 Millisekunden aufgerufen und wieder verlassen (RTOS-UH kennt nur RESIDENT REENT). Die Übernahme eines FIXED-Parameters erledigt der Prozeßrechner in 6 Mikrosekunden, der ATARI hingegen benötigt 43, weil beim RTOS-UH-PEARL die Parameter-Verträglichkeit erst zur Laufzeit geprüft wird.

Die Meßergebnisse zeigen, daß Resultate aus einem Vergleich mit Benchmark-Programmen mit großer Vorsicht zu bewerten sind, wenn die Eignung eines Rechners für eine bestimmte Anwendung zur Debatte steht. Der ATARI dürfte bei Textverarbeitung in der Regel merklich schneller sein als der Prozeßrechner. Wenn jedoch in einem Programm Teilketten häufig umgespeichert oder auf Gleichheit geprüft werden sollen, muß man beim ATARI eigens für diesen Zweck geschriebene Prozeduren aufrufen, wodurch sich dessen Vorsprung womöglich ins Gegenteil verkehrt. Der Prozeßrechner wird seine größere Schnelligkeit bei FLOAT-Rechnungen nur in speziellen Programmen ganz ausspielen können: jede Zuweisung, jede Schleife vermindert seinen Vorsprung.

Die Ergebnisse sind auch lehrreich für Fälle, bei denen die Rechenleistung bei Verwendung verschiedener Programmiersprachen verglichen werden soll: es muß dann beispielsweise darauf geachtet werden, daß sich die Programme auch in Hinblick darauf entsprechen, daß Variable prozedur- bzw. tasklokal oder modulglobal (bei FORTRAN im COMMON) deklariert werden.

Den Mitgliedern des PEARL-Anwender-Ausschusses gebührt Dank für kritische Diskussionen über die Meßmethoden und deren Auswertungen; die Meßergebnisse für C- und PASCAL-Programme stammen von meinen Kollegen Dr. Ehrich und Dr. Eisenack.

Neuere Trends in der Entwicklung von Echtzeit-Expertensystemen für die Leitwartentechnik

W. Fedderwitz
KRUPP ATLAS ELEKTRONIK GmbH
Bremen

1. Einleitung

Echtzeitexpertensysteme für die Leitwartentechnik unterstützen das Bedienpersonal auf der Warte von Verteilnetzen regionaler Gas-, Wasser- oder Elektroenergieversorgungsunternehmen bei der Handhabung und Führung ihres Netzes. Das Expertensystem ist in diesem Einsatzgebiet kein Ersatz der bereits bestehenden Prozeßautomatisierung, sondern eine Ergänzung. Es wendet sich solchen Aufgaben zu, die bisher vollkommen vom menschlichen Bediener erbracht werden mußten.

Erste Implementationen dieser Expertensysteme sind im Rahmen von Forschungsvorhaben auf Rechnern entstanden, die speziell für Programme der künstlichen Intelligenz vorgesehen sind (spezielle Lispmaschinen) /2,3/.

Die Zukunft gehört aber zweifellos einer Integration der Expertensysteme (XPS) in die übrige EDV der Leitwarte, und zwar aus mehreren Gründen. Auch von der Hardware des Expertensystems wird dieselbe Ausfallsicherheit verlangt wie von der übrigen EDV, stellt doch das Expertensystem einen wesentlichen Bestandteil der gesamten Automatisierung dar. So entsteht die Anforderung, Laufzeitversionen des XPS auf Standardhardware in "Prozeßrechnerqualität" zu implementieren.

Die Softwareengineering-Aufgabe "Expertensystem" führt neben neuen Anforderungen auf Teilprobleme, die in der konventionellen DV bereits bearbeitet werden. Eine Integration der Expertensysteme in die bestehende DV bietet entscheidende

Vorteile bei der Übernahme solcher Teillösungen.

Insgesamt muß sich die Leitwarte dem Benutzer einheitlich darbieten. Sie stellt genau die Intelligenz zur Verfügung, die zur Bearbeitung der jeweiligen Anfrage erforderlich ist. Eine Meßwertabfrage etwa wird von der zuständigen Software durch einfache Präsentation dieses Wertes erledigt. Die Frage nach der Planung zukünftiger Schalthandlungen wird mit Unterstützung des XPS gelöst, ohne daß dem Benutzer die unterschiedliche Herkunft der Antworten gegenwärtig werden muß.

Die grundsätzliche Architektur eines Realzeitexpertensystems wird in /5/ vorgestellt. Aus diesen Forschungsgebieten heraus wird auf den vorstehenden Themenkreis an Hand zweier herausgegriffener Beispiele eingegangen, ohne damit den Anspruch auf vollständige Behandlung des Themenkreises stellen zu können.

2 Parallelarchitekturen

Unter Parallelität wollen wir hier den Einsatz "echter" Parallelverarbeitungsmaschinen verstehen, in denen also mehr als eine CPU vorhanden ist.

2.1 Problemunabhängige Parallelisierung

Parallelrechner zum Einsatz in wissensbasierten Systemen werden zur Zeit weithin diskutiert, z.B. /4/. Parallelität bedeutet in diesen Diskussionen eine Parallelität in niedriger und problemunabhängiger Ebene.

Die beiden Hauptvertreter dieser Art von Parallelität, AND und OR-Parallelität, lassen sich am einfachsten durch Beispiele erklären.

2.1.1 AND-Parallelität

Eine Regel mit zusammengesetzter Prämisse

IF subprämisse-1 AND subprämisse-2 THEN action

wird bei Vorwärtsauswertung genau dann gefeuert, wenn beide Subprämissen wahr sind. Jede Subprämisse läßt sich einem Prozessor zur Bearbeitung übergeben. Die Entscheidung wahr - unwahr kann für jede Prämisse getrennt und zeitlich parallel zur anderen fallen. Danach ist der Kontrollfluß wieder zu vereinigen, um zu entscheiden, ob beide Subprämissen wahr sind.

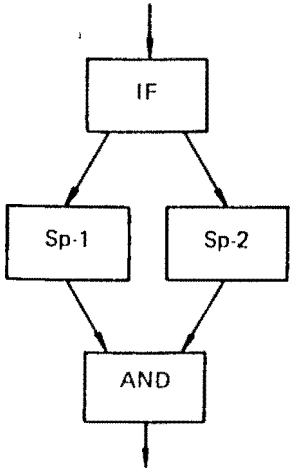


Bild 1 AND-Parallelität

2.1.2 OR-Parallelität

Dieselbe Überlegung läßt sich für Oder-Verknüpfungen anstellen:

If sp-1 OR sp-2 THEN action

wird diese Regel z.B. rückwärts ausgeführt, d.h. versucht, die Bedingungen zu finden, die gelten müssen, damit action

ausgeführt werden kann, so lassen sich die beiden Alternativen parallel untersuchen. Die Zuordnung vorwärts zu AND, rückwärts zu OR ist hier beispielhaft. Auch mit den jeweils anderen Zuordnungen ergibt sich ein sinnvoller Ansatz.

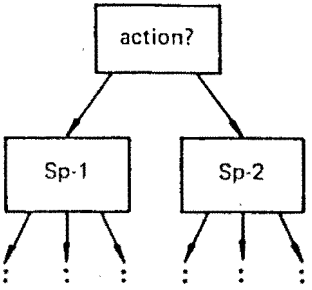


Bild 2 ●R-Parallelität

2.1.3 Probleme der AND/OR-Parallelität

Die beiden geschilderten Ansätze erscheinen zunächst sehr attraktiv. Da sich die Möglichkeit der Parallelisierung aus der Sprache ergibt, läßt sie sich formal und ohne Problemverständnis durchführen. Die Parallelisierung ist automatisierbar und belastet nicht den Ersteller des Regelwerkes. Dieselbe Art der Parallelisierung läßt sich auch auf die Klauseln eines Prolog-Programms anwenden, ein weiterer Punkt für diese Art der Programmabarbeitung.

In Schwierigkeiten geraten diese Parallelisierungen dann, wenn sich die Randbedingungen nicht an sehr eng gezogene Voraussetzungen halten.

2.1.4 Seiteneffekte

Betrachten wir dazu die AND-Parallelität. Wenn die Subprämisse-1 einen Seiteneffekt hat (also eine global zugängliche Datenstruktur verändert), kann das Ergebnis der Auswertung von Subprämisse-2 davon abhängen, ob sie vor oder nach der Subprämisse-1 ausgewertet wird.

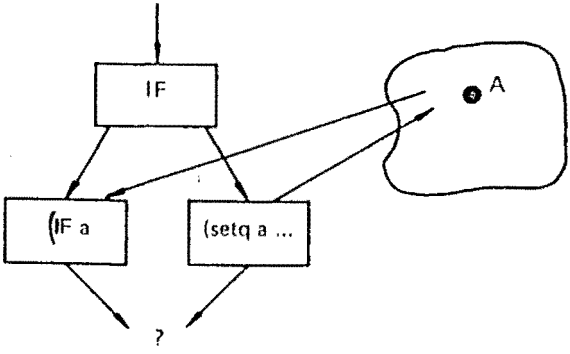


Bild 3 Seiteneffekte durch Modifikation globaler Daten

Praktische Erfahrung zeigt nun, daß solche Abhängigkeiten in Regelwerken leider nicht immer vermieden werden und oftmals trickreich untergebracht sind. Ursache ist weniger böser Wille des Erstellers als vielmehr die Tatsache, daß sich bestimmte Sachverhalte mit sauberen, seiteneffektfreien Regeln nur wesentlich umständlicher formulieren lassen. Zu demselben Ergebnis kommt man auch bei Prologprogrammen.

Aus dieser Erkenntnis lassen sich verschiedenste Forderungen ableiten. Man kann verlangen, daß Regeln im "reinen deklarativen Sinn" zu verwenden sind und daß solche Seiteneffekte dem Ziel expliziter Wissensdarstellung krass wiedersprechen. Dadurch wird allerdings in der Praxis zunächst nichts geändert.

2.1.5 Aufwand

Auch wenn seiteneffektfreie Regeln vorliegen, haben viele Probleme die Eigenschaft, bei "blinder" Parallelisierung durch AND/OR den Gesamtaufwand zur Lösung gegenüber einer gut kontrollierten sequentiellen Abarbeitung so weit zu erhöhen, daß kaum Zeitersparnis eintritt.
Dieses Phänomen rührt daher, daß die Entscheidung darüber, ob ein Pfad noch verfolgt werden muß oder nicht, bei Parallelisierung erst später möglich ist. Betrachten wir dazu die AND-Parallelität. Scheitert die erste Subprämisse, kann die Analyse der weiteren Subprämissen beendet werden, da die Gesamtprämisse sicher nicht mehr wahr wird. Im Gegensatz dazu führt die "blinde" Parallelisierung die Analyse sämtlicher Teilprämissen durch.

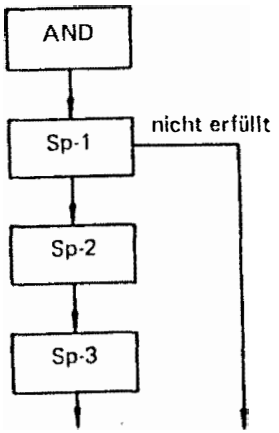


Bild 4 Aufwand serieller gegen paralleler Abarbeitung

Wir wollen an dieser Stelle nur den Schluß ziehen, daß sich diese Parallelität (noch) nicht für den täglichen automatischen Einsatz eignet. Forschungsarbeiten werden auf diesem Gebiet aber sicher weiterlaufen, um mit den skizzierten Schwierigkeiten fertigzuwerden.

Inferenz auf nicht mehr zutreffenden Eingangsdaten beruht.

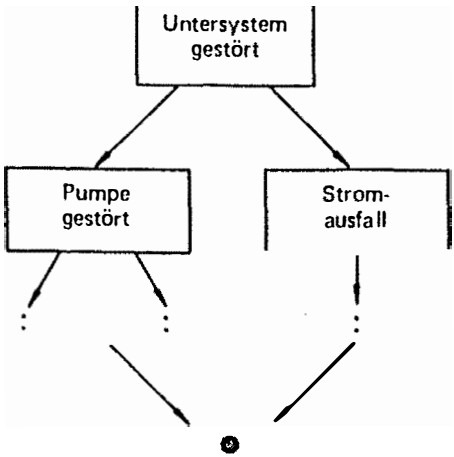


Bild 5 Parallelität aus der Struktur des Problems

2.2.2 Globale Daten

Jeder Expertensystemkern, der parallel zu anderen inferiert, wird neben lokalen Daten auf globale Datenbestände zugreifen müssen.
Während jedoch bei einem "klassischen Programm", z.B. einem Regelalgorithmus, relativ einfach angebbar ist, welche globalen Daten überhaupt verwendet werden und wie sie ggf. modifiziert werden, ist der genaue Ablauf des Inferenzprozesses und damit die Arten des Datenzugriffs auf globale Bestände weniger einfach vorhersehbar. Der gesamte Inferenzvorgang muß zur Vermeidung inkonsistenter Zustände in eine Folge einzelner Transaktionen aufgegliedert werden.

Solche Transaktionen lassen sich aber nur in Trivialfällen einfach abgeben. Im allgemeinen erfordert die korrekte Definition der Transaktionen einen Detaillierungsgrad der Inferenzablaufanalyse, den man unter dem Stichwort "datengetrieben" oder, etwas unseriöser formuliert "Es ist nicht vorher-

Der Nachteil dieser Lösung, um damit zu beginnen, besteht darin, daß die Parallelisierung nicht vor dem Ersteller verborgen bleibt. Der Ersteller des Systems hat die Aufgabe, sich über eine zweckmäßige Aufteilung des Gesamtproblems in parallelisierbare Teile Gedanken zu machen und dies mit den zur Verfügung gestellten Hilfsmitteln auszudrücken.

Diese Situation unterscheidet sich grundsätzlich nicht von der Situation in der "klassischen" Echtzeitprogrammierung. Prozeßrechner zur Verwaltung und Führung von technischen Prozessen werden häufig in Form paralleler Tasks programmiert. In der klassischen DV wird dieses Konzept z.B. von PEARL und ADA unterstützt.

Der Vorteil dieses Ansatzes ist darin zu sehen, bereits jetzt in Prototypen einsetzbar zu sein.

2.2.1 Besonderheiten bei der Anwendung in XPS

Situationen (siehe /5/), die sich um parallel vorliegende verschiedene Aspekte des Prozeßgeschehens kümmern, liefern den natürlichen Einstieg in eine Parallelisierung aus dem Problem heraus.
Einzelne Kerne, die von verschiedenen Situationen beauftragt worden sind, arbeiten parallel zueinander, ebenso die Situationen selbst in der Abarbeitung ihrer Nachrichten.

Der erste Schritt zu konsistenten Verarbeitungsergebnissen besteht darin, den einzelnen Inferenztasks die unkontrollierte Ergebnisausgabe durch irgendwelche Seiteneffekte zu verbieten. Das Inferenzergebnis wird nach Abschluß der Vorganges an eine zentrale Stelle, den Situationsintendanten, zurückgegeben. Dadurch wird der Zeitraum, in dem exclusive Operationen stattfinden müssen, um Inkonsistenz zu vermeiden, deutlich kleiner. Insbesondere lassen sich Inferenzergebnisse von der Verwendung ausschließen, wenn inzwischen von anderen Ergebnissen her bekannt ist, daß diese spezielle

sehbar, wie, in welcher Reihenfolge, ein Expertensystem im einzelnen die Regeln auswertet, wenn es einen speziellen Fall vorgelegt bekommt" gerade vermeiden wollte.

2.3 Weiterentwicklung von Expertensystemshells

Marktgängige Shells unterstützen die Parallelität zur Zeit nicht. Aus den von uns verfolgten Forschungsvorhaben heraus sind jedoch zwei Shellentwicklungen durchgeführt worden, die Parallelität unterstützen.

Solche Shells lassen sich mehrfach instanziieren, um parallel mehrere Inferenzvorgänge ablaufen lassen zu können. Das Transaktionsprinzip wird dabei syntaktisch vorgesehen, d.h. wenn eine Transaktion identifiziert ist, läßt sie sich im System formulieren. Die weitaus schwierigere Definition der einzelnen Transaktionen ist weiterhin Aufgabe des XPS-Erstellers.

3 Datenbanken

Wir wollen hier nicht in das aktuelle Thema einsteigen, Datenbanken mittels KI-Methoden mit intelligenten Abfrage- oder Verwaltungsmechanismen auszurüsten, sondern uns mit dem Thema beschäftigen, ob Datenbanken sinnvoll und vorteilhaft in Echtzeitexpertensystemen eingesetzt werden können, um die vom System benötigten Daten zu halten.

3.1 Probleme konventioneller Lispwelten mit der Datenhaltung

Die Lispwelt ist inkrementell, d.h. jede einmal geschaffene (Daten-)struktur geht nicht verloren, auch nicht, wenn das erzeugende Programm abgebrochen wird, andere Programme gestartet werden usf. Es ist also z. B. möglich, in einem Programm eine Datenstruktur aufzubauen und sich dann am nächsten Tag zu überlegen, ein weiteres Programm zu schreiben, das diese Struktur wieder verwendet - sie ist noch da.

(Der Garbagecollektor entfernt nur solche Strukturen, deren Erreichbarkeit explizit aufgegeben wurde. Die Einzelheiten lassen sich in einer Sprachbeschreibung für Lisp nachlesen, z.B. /1/.)

Im Gegensatz dazu wird bei üblichen Programmiersprachen eine Datenstruktur (z.B. ein Array), wenn man sie nicht explizit rettet, verloren gehen, wenn das jeweilige Programm verlassen wird.

In der Lispmaschine lagern also sämtliche Daten, alle Symbole, Listen, Arrays und höhere Strukturen, wie Instanzen von Frames, Fenster, Prozesse usf. ständig im (virtuellen) Speicher. Die explizite Auslagerung von Daten (etwas explizit auf Platte schreiben und wieder einlesen) ist untypisch für eine Lispmaschine bzw. ein Lispprogramm, da die inkrementelle Eigenschaft der Lispwelt verwendet wird.

Die Lispwelt hat mit der wirklichen Welt gemeinsam, daß sie sich nur vorwärts entwickelt. Es ist ein besonderes Problem, einen alten Zustand wieder herzustellen. Hierbei hat eine Lispwelt allerdings ein paar Vorteile. Das Abschalten der Maschine und anschließender Neustart lädt eine Startform der Welt von der Platte. Dieser "Boot" ist allerdings in einem Echtzeitexpertensystem zur Prozeßführung höchst unzweckmäßig. Während die Maschine wieder anläuft, sollte tunlichst nichts im Prozeß passieren, da dann die Unterstützung des XPS fehlt. Wenn die Maschine wieder läuft, ist sie nicht "auf dem laufenden". All ihr Wissen über den Prozeß war ja im Speicher, der durch diesen Vorgang verloren gegangen ist.

Da es unrealistisch ist, anzunehmen, das eine Maschine nie "abstürzt", - während der Entwicklung eines Expertensystems wird es wohl eher die Regel sein - muß es möglich werden, die Maschine gezielt in einen wählbaren Zustand vor dem Crash zu bringen.
Die Wiederholung des gesamten Experimentes am Prozeß ist normalerweise unmöglich oder mit zu hohen Kosten bzw. Zeitaufwand verbunden. Die sonst unkritischeren langsamen Prozesse sind in dieser Hinsicht besonders schwierig. Expe-

rimente, die eine Woche oder länger brauchen, macht man nicht bei jedem Maschinenhalt einfach nochmal.

Oftmals kann man zur Erprobung auch nicht auf kostengünstigere Simulationen des Prozesses ausweichen - das Expertensystem wird genau in solchen Bereichen eingesetzt, wo die zur Simulation nötigen Kenntnisse das Prozesses gar nicht vorhanden sind.

Zur Lösung dieser Probleme ist es also erforderlich, einen Teil der Daten nach einem Boot der Maschine so zur Verfügung zu haben, daß das gezielte Wiederherstellen eines alten Zustandes möglich wird.
Unabdingbar scheint dazu, die übliche Sichtweise auf eine Lispwelt als in diesem Sinne nicht strukturiertes Ganzes aufzugeben. Selbst wenn es möglich wäre, in hinreichend feinen Abständen den gesamten Speicher als Dump auf die Platte zu schreiben, löst das "wiedereinsetzen in diesen Stand" oft das aufgetretene Problem des Absturzes nicht. Es müssen problemangepaßt einige Daten übernommen werden, während andere modifiziert werden, damit die Maschine auf einen neuen Weg kommt.

Die zu übernehmenden Daten sind im wesentlichen die Instanzen der Signal- und Situationswelt sowie Zustände und Ergebnisse der einzelnen Expertensystemkerne. Wir werden uns deshalb mit diesem Aspekt weiter beschäftigen.

Weitere Daten, die während des Ablaufes entstehen, wollen wir über eine Störung nicht hinwegretten. Dies führt zu einer entsprechenden Anforderung an das Design des Programmes.

3.2 Hardwarevorkehrungen

Sicherheit gegen einen Teil von Widrigkeiten wie Stromausfall und bestimmte Störungen der Hardware bietet eine Rechnerarchitektur, die zunächst die Daten in gewohnter Weise im Speicher hält, sie aber im Fehlerfalle z.B. mittels Energie aus einem Akku komplett auf die Platte rettet. Aus der

Prozeßrechnertechnik sind diese Lösungen bekannt und bewährt. Eine Lispmaschine für den industriellen Einsatz sollte auf diese Maßnahme nicht verzichten.
Allerdings werden dadurch die insbesondere bei der Entwicklung auftretenden Softwarefehler nur unvollständig abgedeckt, sodaß diese Maßnahme andere nicht vollständig ersetzen kann.

3.3 Eine Datenbank für die Signalwelt

Signale lassen sich recht einfach in einer konventionellen Datenbank halten. Da die Signale Abbilder der Meßwerte aus der konventionellen Prozeßautomatisierung sind, verwenden sie keine besonderen Eigenschaften von Lispdatenstrukturen. Die Speicherung von z.B. numerischen Einzelwerten oder festen, nicht zur Laufzeit modifizierten Strukturen aus diesen Werten bereitet einer konventionellen Datenbank keine Probleme.

Es bietet sich an, in einem das Expertensystem integrierenden Konzept die Signalwelt mit Hilfe des Datenhaltesystems der konventionellen Automatisierung zu realisieren. Solche Datenbanken erfüllen bereits heute die Anforderung, Störungen von Hard- und Software ohne Datenverlust zu überleben. Außerdem entfällt die anfangs geschilderte doppelte Datenhaltung mit ihren Problemen. Um gegenüber der speicherresidenten Lösung Nachteile bei der Zugriffsgeschwindigkeit möglichst klein zu halten, muß solch ein Konzept über entsprechende Caches verfügen.

3.4 Datenbanken für Situationen und Kerne

Arbeitsergebnisse auf dieser Ebene sind "irgendwelche" Lispdatenstrukturen. Es gibt somit zwei Wege, an dieser Stelle über einen Maschinenneustart hinweg zu kommen.

3.4.1 Verzicht auf Datenbanken

Die Situationswelt und die XPS-Kerne bestehen aus einem großen Anteil statischer Daten, im wesentlichen den Programmen (Regeln, Framedefinitionen, Relationen), die sich zur Laufzeit nicht ändern und eine externe Repräsentation (Quelltextfile) haben.

Variable Daten lassen sich im Prinzip wieder gewinnen, indem der Ablauf des Systems an Hand der geretteten Daten der Signalwelt (im "Schnellgang", d.h. rascher als in Echtzeit, so die Maschine dies hergibt) bis zum gewünschten Zeitpunkt wiederholt wird. Benutzereingaben müssen dazu ebenfalls gerettet worden sein.

Der Verzicht auf Datenbanken in diesem Bereich ist die zur Zeit für Prototypen ins Auge gefaßte Lösung. Sie wurde trotz der offensichtlichen Nachteile favorisiert, weil die Machbarkeit einer Alternative ohne intensive Untersuchungen nicht zu beurteilen ist.

3.4.2 Datenbanken für beliebige Lispstrukturen

Eine konventionelle Datenbank besitzt (fast) sämtliche Merkmale, die der Speicher einer Lispmaschine haben muß. Die Speicherung beliebiger Lispdatenstrukturen geschieht durch Aufbau von "Verzeigerungen" zwischen elementaren Bestandteilen und entsprechenden, in die Sprache Lisp integrierten, Abfragemechanismen. Diese Form der Datenhaltung ist mit einer Datenbank ebenfalls möglich. Nun kann man sicher nicht den Speicher einfach durch ein Datenbanksystem ersetzen, ohne einen dramatischen Verlust an Geschwindigkeit zu haben. Die Datenbank muß also "im Hintergrund" zu einem normalen Speicher in Form eines "Writethru"-Konzeptes versuchen, Änderungen des Speichers mitzubekommen.

Intensive eigene Untersuchungen zu diesem Thema sind bisher nicht durchgeführt worden. Auch eine, allerdings nicht erschöpfende Literatursuche, erbrachte keine Hinweise auf

ausgeführte Prototypen zu diesen Überlegungen. Die Tragfähigkeit solcher Überlegungen bedarf zunächst intensiver Untersuchungen, bevor weitere Aussagen gemacht werden können.

4 Schluß

Die beiden diskutierten Themen wurden während der Prototypentwicklung im Rahmen von Forschungsvorhaben als Problembereiche für den Alltagseinsatz identifiziert.
Neben den "Üblichen" Forschungsrichtungen und Problemen kommen bei Einzug der Expertensysteme in den industriellen Alltag weitere, bisher in der KI wenig beachtete Probleme auf. Austausch von Konzepten und Lösungen zwischen der KI-Welt einerseits und den Datebankwelten sowie der Echtzeitprogrammierung andererseits können befruchtend auf die Weiterentwicklung wirken.

LITERATUR

/1/ Patrick Henry Winston, Berthold Klaus Paul Horn : LISP, Addison-Wesley Publishing Company, 1981

/2/ Wittig, T.: The KRITIC Project: Domain 2: Controlling Complex Power-Distribution Networks, in : Proceedings of the BIRA/ACM Conference "Expert Systems and Artificial Intelligence in Industry", Antwerpen, 1986

/3/ Fedderwitz, W.: Erste Erfahrungen aus dem Arbeitskreis Expertensysteme, in : Symposium Leittechnik 1987, Krupp Atlas Datensysteme, Bremen, 1987 (BL2210A001)

/4/The American Association of Artificial Intelligence: Proceedings of the Sixth National Conference on Artifical Intelligence, Seattle, Washington, July 1987

/5/ B. Eiben, J. Eisermann, W. Fedderwitz: Ein Expertensystem für die Leitwartentechnik, in : H. Balzert, G. Hezer, R. Lutze (Hrsg.) Expertensysteme 87, Tagung des German Chapter of the ACM, B.G. Teubner Verlag, Stuttgart.

Zur Verwendung von PEARL bei der Programmierung von Knotenrechnern in dedizierten verteilten Systemen

Dr. W. Röhrl

G P P Gesellschaft für
Prozessrechnerprogrammierung mbH
Kolpingring 18a
8024 Oberhaching
Tel.: 089/ 61 30 4-1

Zusammenfassung:

Ein PEARL-System, mit dem Mikroprozessor-Systeme in spezialisierten lokalen Rechnernetzen (dedizierte verteilte Systeme) programmiert werden sollen, muß aufgrund der speziellen Eigenschaften der Zielrechner bestimmte Anforderungen genügen; dies trifft sowohl auf die Basis-Software - Betriebssystem und PEARL-Laufzeitsystem - als auch auf das Übersetzungssystem - Compiler-Oberteil und Code-Generator - zu. Vor allem folgende Problemkreise sind hierbei zu berücksichtigen: Laden des Zielrechners, RAM-ROM-Aufteilung, Echtzeit-Verhalten, Konfigurierbarkeit der Basis-Software.

1. Dedizierte verteilte Systeme

Durch das Aufkommen billiger Mikroprozessor-Systeme werden seit einiger Zeit verteilte Daten-Verarbeitungs-Systeme (VS) realisiert; dies sind Systeme, bei denen Komponenten eines Programmsystems auf mehrere, zu einem Netz zusammengeschaltete Rechner verteilt sind. Die Vorteile eines derartigen Systems wurden bereits vielerorts diskutiert und sollen nur noch summarisch genannt werden:

- erhöhte Rechenleistung durch parallel operierende Knoten-Rechner und die Möglichkeit der Lastverteilung
- erhöhte Ausfall-Sicherheit durch elektrische Trennung und redundante Auslegung der einzelnen Rechner
- Unterstützung eines modularen Hardware- und Software-Konzepts, so daß einzelne Komponenten möglichst rückwirkungsfrei modifiziert werden können.

Im allgemeinen besteht ein VS also aus einer Reihe von Knoten-Rechnern (KR), die über irgendein Kommunikations-Medium (Lichtleiter, Draht-Verbindung, Funkstrecke usf.) zu einem Netz verbunden sind. Im folgenden soll nun eine Untergruppe von VS betrachtet werden, die als dediziertes VS bezeichnet werden soll und gegenüber dem allgemeinen VS in etwa so abgegrenzt werden kann:

- Das Rechner-Netz besitzt eine nur lokale Ausdehnung, die in der Größen-Ordnung von wenigen Metern bis hin zu einigen Kilometern liegen kann. Typische Beispiele wären eine Fabrikhalle, in der das VS zur Fertigungs-Steuerung verwendet wird, oder spezielle rechnergestützte Nahbereichs-Kommunikations-Systeme. Weiterhin soll das Netz eines dedizierten VS eine relativ einfache Topologie besitzen: dies ist dann gegeben, wenn je zwei KR unmittelbar Nachrichten austauschen können und dies nicht über dazwischenliegende KR, die nur als transiente Knoten fungieren, tun müssen (Ring-Netze, Punkt-Punkt-Verbindungen, Stern-Netze etc.). Aufgrund der Lokalität der Netze müssen die einzelnen KR nicht über globale öffentliche Kommunikations-Kanäle verbunden werden und sind damit auch nicht gezwungen,

aufwendige Protokolle wie das ISO/OSI-Schichtenmodell zur Inter-KR-Kommunikation zu realisieren.

- Das Rechner-Netz dient zur Bearbeitung einer fest umrissenen Aufgabe. Dies bedeutet negativ, daß die einzelnen KR keine Vielzweck-Rechner mit umfangreicher Software-Ausstattung sind. Vielmehr kann ein KR auf eine bestimmte Aufgabe derart spezialisiert sein, daß er nur im Verbund mit anderen KR, d.h. nur als Komponente eines VS sinnvoll arbeiten kann (eingebettetes System). Als Beispiel seien Meßwert-Abnehmer, die eine Vorverarbeitung und Reduktion der Meßdaten durchführen, primär als 'Rechenknechte' ausgelegte KR, oder KR, die eine komfortable Mensch-Maschine-Schnittstelle anbieten, genannt. Derartige spezialisierte KR werden meist als Mikroprozessor-Systeme realisiert. Die Spezialisierung der KR bezieht sich aber nicht nur auf die Software, sondern oft auch auf die eingesetzte Hardware. So kann ebenso Spezial-Peripherie wie auch spezielle Prozessoren (EA-Prozessoren, Arithmetik-Prozessoren, Signal-Prozessoren) zum Einsatz kommen. Hier zeigt sich auch ein Vorteil von VS: die Änderung der Hardware-Ausstattung eines KR kann für die anderen KR völlig transparent bleiben - ebenso auch ein Wechsel von Basis- oder Anwender-Software im KR. Schließlich sei noch darauf hingewiesen, daß KR, die zur Verarbeitung von externen Ereignissen dienen, i.d.R. als Realzeitsystem betrieben werden müssen.

2. Die Programmierung von KR in einem dedizierten VS

Es soll nun betrachtet werden, welche Anforderungen an ein PEARL-System zu stellen sind, mit dem KR in einem dedizierten VS im eben umrissenen Sinn programmiert werden sollen. Es sei betont, daß es bei diesen Überlegungen nicht darum geht, Gründe für die Wahl von PEARL anzuführen (hierfür gibt es projektspezifisch jeweils eine Reihe von Gründen sowohl technischer als auch nicht-technischer Art); vielmehr soll lediglich dargestellt werden,

welchen Anforderungen ein PEARL-System genügen sollte/müßte, wenn mit dessen Hilfe KR i.o.S. zu programmieren sind. Grundsätzlich gilt aber, daß eine prozess-orientierte Sprache wie PEARL eine für die Programmierung von KR gut geeignete Sprache ist, wenn KR im hier besprochenen Sinne für die Bearbeitung von Realzeit-Aufgaben eingesetzt werden.
Die zu programmierenden KR sind durch folgende Eigenschaften charakterisiert:

- Spezialisierung des KR auf eine Aufgabe. Dies bedeutet, daß die KR meist nur als Ziel- oder Ausführungs-Rechner und nicht als Entwicklungs-Rechner anzusehen sind. Das Programmsystem für einen KR wird also zunächst auf einem Gast-Rechner erstellt; anschließend wird es in ablauffähiger Form in den KR gebracht und in diesem ausgeführt. Die Trennung von Gast- und Ziel-Rechner hat den positiven Nebeneffekt, daß das System, auf dem die Programme entwickelt werden, nicht auch die Eigenschaften des Zielsystems haben muß: der Entwicklungs-Rechner muß z.B. nicht unbedingt ein Echtzeit-Betriebssystem besitzen. Umgekehrt kann das Zielsystem klein und dediziert bleiben, da es nicht Hard- und Software für eine Entwicklungs-Umgebung bereitstellen muß.

- Spezial-Peripherie. Dedizierte KR müssen oft auf spezielle Umwelt-Ereignisse reagieren und können daher besondere Ein- und Ausgabe-Mechanismen besitzen. Diese externen Schnittstellen sind zumindest z.T. nicht durch Standard-Treiber bedienbar, sondern benötigen spezielle Treiber-Programme zu ihrem Betrieb.

- Eingeschränkte Hardware-Ausstattung. Dedizierte KR besitzen nicht unbedingt die komplette Standard-Peripherie wie Externspeicher, Terminal oder Standard-Hardware wie Arithmetik-Prozessor etc. Die Gründe für diese Einschränkungen mögen etwa in besonders rauhen Umweltbedingungen oder dem spezialisierten Aufgabenbereich des Rechners liegen. Die eingeschränkte Hardware-Ausstattung des KR führt zu dem Wunsch, auch nur eingeschränkte Basis-Soft-

ware einzusetzen. Es macht etwa wenig Sinn, im Betriebssystem ein Filesystem, einen Lader etc. vorzusehen, wenn der KR keinen Extern-Speicher besitzt und das gesamte Programmsystem des KR im ROM untergebracht ist. Außerdem führt die Minimierung von Hard- und Software-Einsatz meist zu einer höheren Betriebssicherheit des KR.

- Der KR muß oft unter Realzeitbedingungen operieren. KR werden etwa zur Steuerung technischer Prozesse, zur Messwert-Erfassung usw. eingesetzt. In diesen Einsatzgebieten müssen die KR oft Echtzeit-Verhalten zeigen, da die Antwortzeit des Systems einen bestimmten Maximalwert nicht überschreiten darf und das System eine bestimmte Rate von Ereignissen in einer vorgegebenen Zeit bearbeiten können muß.

2.1 Die Überführung von Programmen in das Zielsystem

Im folgenden seien einige Hinweise zur Basis-Konfiguration Entwicklungs-/Zielsystem gegeben, wenn die zu programmierenden KR kein Entwicklungs-Betriebssystem besitzen. In diesem Zusammenhang stellt sich vor allem die Frage, wie das auf dem Gast-Rechner entwickelte Programm in den Ziel-Rechner überführt werden soll. Hierbei sind mindestens drei Varianten denkbar:

1) Der KR besitzt keinen ROM-Bereich mit Basis-Software. In diesem Fall ist es notwendig, das entwickelte Programmsystem mithilfe eines weiteren Systems in den Speicher des KR zu überführen und dort zur Ausführung zu bringen. In der Testphase kann das z.B. eine Anordnung sein wie der In-Circuit-Emulator (ICE) der Fa. Intel.
   Anmerkung: Bei der eben geschilderten Variante muß es möglich sein, die gesamte Basis-Software (einschließlich Betriebssystem, Treibern, Selbsttest-Programm usw.) mit dem PEARL-Programm zusammen zu binden.
2) Der KR besitzt einen ROM-Bereich mit einer Lade-Funktion. Durch diese Lade-Funktion wird man vom Vorhandensein zusätzlicher Systeme wie eines ICE unabhängig. Es wird

jedoch nun eine externe Schnittstelle vorausgesetzt, über die das Programmsystem in den KR geladen werden kann. Es sei darauf hingewiesen, daß die Lade-Funktion unter bestimmten Voraussetzungen sehr einfach sein kann (und damit wenig Raum im Ziel-Rechner einnimmt):

Sämtliche Adreß-Bezüge im zu ladenden Programmsystem sind bereits aufgelöst und verabsolutiert (beim Intel-8086-System etwa nach dem Lauf der Programme LINK86 und LOC86).

Das zu ladende Objekt-Programm ist in einer einfachen internen Form dargestellt; außerdem wird es nicht in binärer, sondern in Alpha-Form mit Prüfsumme übertragen (bei Intel etwa nach dem Lauf des OH86). Die einfache interne Form garantiert eine wenig aufwendige Interpretation durch die Lade-Funktion im KR. Die Darstellung in Alpha-Zeichen erlaubt es, das Objekt-Programm über eine Standard-Schnittstelle des Gast-Rechners (z.B. Terminal-Schnittstelle) in den Ziel-Rechner zu 'kopieren'.

Die Schnittstelle, über die die Programme geladen werden, sollte einfach zu bedienen sein, wie dies etwa bei einer seriellen Leitung der Fall ist.

Anmerkung: Auch hier gilt wieder, daß die Basis-Software mit dem Anwender-Programm gebunden und geladen werden muß.

3) Der KR besitzt einen ROM-Bereich mit einer Lade-Funktion und der gesamten Basis-Software. Der Vorteil dieser Variante gegenüber der vorherigen besteht darin, daß weniger Code und Daten zu laden sind. Die Schnittstelle zwischen Anwender- und Basis-Software muß nunmehr so ausgelegt sein, daß die Anwender-Software ohne die Basis-Software gebunden werden kann. Die Anwender-Software selbst ist wieder über eine Lade-Funktion oder über ein System wie den ICE zu laden.

Für alle Varianten gilt, daß der KR während der Testphase im wesentlichen nur RAM-Speicher enthält. Nach der Testphase werden die invarianten Teile des Programms in einem ROM-Speicher festgehalten und dieser in den KR gebracht.

2.2 Anforderungen an die Basis-Software im Zielsystem

Aus den bisherigen Überlegungen läßt sich ein Forderungs-Katalog für die im KR eingesetzte Basis-Software ableiten.

- Die Basis-Software muß die Aufteilung in RAM/ROM-Bereiche unterstützen. Dies bedeutet, daß das System mit beliebigen RAM-Inhalten gestartet werden kann (Code und invariante Daten sind im ROM abgelegt). Werden bestimmte RAM-Daten vorausgesetzt, so sind diese RAM-Bereiche explizit zu initialisieren.

- Die Basis-Software sollte Realzeit-Verhalten zeigen. Ein PEARL-Programm kann nur dann als Echtzeit-System arbeiten, wenn dies auch durch die Basis-Software (und ggfs. Hardware) unterstützt wird. Grundsätzlich gilt, daß auf einer Echtzeit-Basis-Software sowohl Echtzeit- als auch Nicht-Echtzeit-Programme ablaufen können.

- Ein wichtiges Problem in einem VS besteht in der Gleichheit der Uhrzeit in allen KR. Aus Gründen der Flexibilität ist es durchaus wünschenswert, die Anwender-Software am Verteilen der Uhrzeit teilhaben zu lassen bzw. die aktuelle System-Zeit über die Anwender-Software im System zu verteilen. Die Basis-Software muß daher Schnittstellen zum Lesen und Setzen von Uhrzeit (und Datum) bereitstellen. Hierbei ist es wichtig, daß das Verändern der aktuellen Uhrzeit einen genau definierten Einfluß auf aktuell vorliegende PEARL-Schedules besitzt (etwa kein Einfluß auf relative Schedules wie "AFTER 10 SEC RESUME"). Weiterhin dürfte es bei der Bearbeitung von harten Echtzeit-Aufgaben oft hilfreich sein, wenn der Anwender die Zeit-Auflösung in der Basis-Software vorgeben kann. Dies beinhaltet sowohl die Konfigurierung der Zeitauflösung im Betriebssystem-Kern (Interruptrate der Software-Uhr), als auch die Festlegung der dem PEARL-System zugrundeliegenden Zeit-Einheit.

- Die Basis-Software sollte in ihrer Größe konfigurierbar sein. Wird etwa in einem PEARL-Programm kein Bezug auf eine Alphic-Dation genommen, so kann z.B. das Laufzeit-Paket der Formatierten Ein/Ausgabe entfallen. Diese Forderung läßt sich oft schon über den Binder erfüllen (es werden nur diejenigen Teile des Systems gebunden, die auch über die PEARL-Quelle(n) referenziert werden). Ebenso sollte auch das Betriebssystem konfigurierbar sein: ist in dem KR etwa keine Druckerschnittstelle vorhanden, so muß kein Drucker-Treiber im System vorhanden sein. Die Forderung nach Konfigurierbarkeit trägt der Tatsache Rechnung, daß KR für die Erfüllung einer bestimmten Spezial-Aufgabe programmiert werden müssen, der KR jedoch nur mit wenig Speicher versehen ist.

- Wenn ein KR spezielle Peripherie bedienen soll, so kann es notwendig werden, in das System eigene Treiber einzubringen. Die Basis-Software sollte eine genau definierte Schnittstelle besitzen, über die Treiber in das System eingebunden werden können.

- Eine Bemerkung zur Portabilität der Basis-Software: Ein VS besitzt u.a. den Vorteil, daß es möglich ist, die Hardware eines KR zu ändern, ohne daß dies Auswirkungen auf die restlichen KR besitzt. Dieser Vorteil kann dann wirklich ausgenutzt werden, wenn die im KR eingesetzte Basis-Software (möglichst) portabel ist, da sie dann bei (weitgehender) Konservierung ihrer alten Eigenschaften auf die neue Hardware übertragen werden kann.

2.3 Anforderungen an das PEARL-Übersetzungssystem

Zu den eben angeführten Forderungen an die Basis-Software gibt es auch entsprechende Forderungen an das PEARL-Übersetzungssystem (Compiler-Oberteil und Code-Generator).

- Der Code-Generator des Übersetzungssystems muß die RAM-ROM-Aufteilung unterstützen. Außerdem sollte der PEARL-Anwendungs-Programmierer die Möglichkeit besitzen, die Plazierung seiner Daten in das RAM bzw. ROM steuern zu können; andernfalls müßten sämtliche Daten per PEARL-Code initialisiert werden - also auch während des Programmlaufs invariante Daten.

- Zur Unterstützung der Echtzeit-Programmierung sollte die Einbindung von Assembler-Programmen genau spezifiziert sein (Prozedur-Schnittstelle mit Beschreibung der Ablage der einzelnen PEARL-Objekte). Damit können besonders zeitkritische Programm-Teile in Assembler geschrieben werden. Weiterhin besteht die Möglichkeit, über ein Assembler-Interface Programme, die in einer dritten Sprache geschrieben sind, anzuschliessen.

- Der generierte Code sollte ggfs. durch Optionen beeinflußbar sein. Derartige Optionen sind jedoch extrem maschinenabhängig, so daß hier keine allgemeine Aussagen gemacht werden können.

- Im Rahmen des Übersetzungssystems ist zwischen zwei Arten von Portabilität zu unterscheiden:
  - .. Das Übersetzungssystem kann selbst auf einem anderen Rechner zum Ablauf gebracht werden. Dies kann Vorteile bei der Programm-Entwicklung bringen, da die Entwicklungs-Umgebung nicht an einen Maschinen-Typ fixiert ist.
  - .. Das Übersetzungssystem kann Code für eine verschiedene Ziel-Maschinen generieren. Portabilität in diesem Sinn meint, daß das Übersetzungssystem (insbesondere der Code-Generator) mit möglichst geringem Aufwand an eine

neue Ziel-Maschine angepaßt werden kann. Die bereits erwähnte Portabilität der Basis-Software kann nur dann genutzt werden, wenn auch das Übersetzungssystem Code für die neue Ziel-Maschine generieren kann.

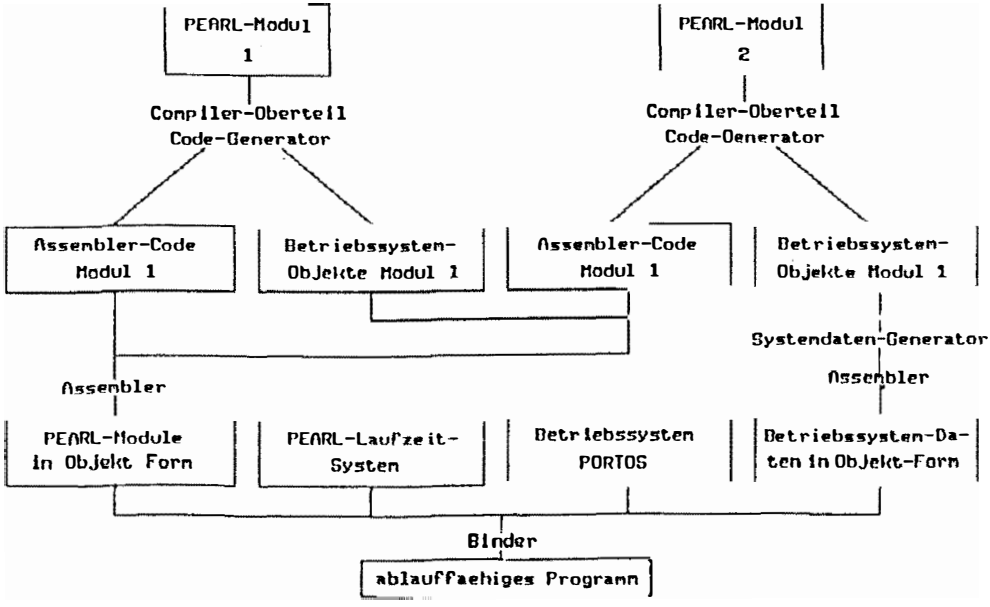
2.4 Zur Kommunikation zwischen den KR

Ein VS setzt Kommunikation zwischen den einzelnen KR unabdingbar voraus. Es ist zu beachten, daß es sich bei dedizierten VS im angegebenen Sinn nur um lokale Netze mit einer einfachen Topologie handelt. Daher ist es nicht notwendig, in einem KR etwa das gesamte ISO/OSI-Schichtenmodell zu realisieren. Vielmehr dürfte sich folgende Vorgehensweise anbieten:
Durch einen in die Basis-Software eingebrachten Treiber werden Datenblöcke mit einer bestimmten maximalen Länge (Pakete) an einen gewünschten KR gesandt. Zum Treiber gehört ein Fehler-Management, durch das das mit Fehlern behaftete reale Übertragungs-Medium virtuell fehlerfrei wird. Im Treiber werden damit die Ebenen 1 und 2 des ISO/OSI-Modells realisiert (z.B. HDLC-Prozedur). Statt eines Treibers ist etwa z. B. an den Anschluß eines ETHERNET-Moduls zu denken; diesem werden die Daten-Pakete mitsamt der Empfänger-Identifikation übergeben. Aufgrund der einfachen Netz-Topologie werden Daten-Pakete direkt beim Empfänger abgeliefert; damit entfällt die Netzwerks-Ebene (Routing) des ISO/OSI-Modells.
Besonders bei kleinen KR kann es aus Speicherplatz-Gründen ratsam sein, nicht mehr als die Übertragungs- und Leitungs-Ebene im Bereich der Basis-Software zu realisieren. Die Kommunikations-Struktur zwischen den einzelnen KR sollte so ausgelegt sein, daß eine zu übertragende Nachricht nicht größer als das zwischen den Rechnern übertragbare Daten-Paket ist, da damit auch die Transport-Ebene entfallen kann. In größeren KR-Systemen bzw. bei höheren Anforderungen an die Kommunikation sollte im Bereich der Basis-Software ein Botschaften-System vorliegen, das eine höhere Art der Inter-KR-Kommunikation erlaubt. Es ist aber auch durchaus möeglich, im Bereich der Anwender-Software auf der Basis der Übertragung einzelner Pakete ein komfortableres und auf die

jeweiligen Bedürfnisse abgestelltes Kommunikations-System zu realisieren. Dafür ist übrigens PEARL als prozess-orientierte Sprache sehr gut geeignet (Implementierung des Kommunikations-Prozesses im KR).
Schließlich sei noch bemerkt, daß eine weitere Vereinfachung und Vereinheitlichung der Kommunikation zwischen KR in einem "Mehrrechner-PEARL"-System erreicht werden könnte.

3. Überblick über das GPP-PEARL-System für die Intel-8086-Familie

Das GPP-PEARL-System für die Intel-8086-Familie ist für die Programmierung von Mikroprozessor-Systemen konzipiert, die höchstens mit einer Lade-Funktion ausgestattet sind. Derartige Mikroprozessor-Systeme können als KR in einem dedizierten VS verwendet werden. Das GPP-PEARL-System besitzt folgende Struktur:



Jeder PEARL-Modul wird durch den Compiler-Oberteil in eine maschinen-unabhängige Zwischensprache übersetzt, die wiederum durch den Code-Generator in einen Modul mit Assembler-Code der entsprechenden Ziel-Maschine (Intel-8086) überführt wird. Gleichzeitig erzeugt der Code-Generator zu jedem PEARL-Modul einen Hilfs-Modul, in dem die in dem PEARL-Modul deklarierten Betriebssystem-Objekte aufgelistet sind (Tasks, Semaphore, Dations etc.). Sind nun alle PEARL-Quellen übersetzt, so wird durch den Systemdaten-Generator auf der Basis der in den Hilfs-Moduln abgelegten Informationen ein Assembler-Modul mit den zu dem PEARL-Programm gehörigen Betriebssystem-Objekten erzeugt. Anschließend sind sämtliche Assembler-Moduln zu übersetzen und mit dem Betriebssystem PORTOS und dem PEARL-Laufzeitsystem zu einem ablauffähigen Programm zu binden. Der Mechanismus der Systemdaten-Generierung gestattet es, aus den PEARL-Quellen völlig automatisch ein ablauffähiges Programm zu erzeugen. Eine Änderung der Zahl der Betriebssystem-Objekte ist damit lediglich auf eine Änderung des PEARL-Quell-Programms beschränkt.

Abschließend seien einige Eigenschaften des GPP-PEARL-Systems in Bezug auf die in 2.2. und 2.3. aufgeführten Kriterien genannt:

- Das PEARL-Übersetzungssystem unterstützt die RAM/ROM-Aufteilung, wobei der PEARL-Anwendungs-Programmierer die Ablage von Daten in RAM- oder ROM-Bereich über das INV-Attribut in der PEARL-Quelle steuern kann. Ebenso ist auch die Basis-Software auf die RAM/ROM-Aufteilung hin ausgelegt.

- Das PEARL-Übersetzungssystem ist einerseits portabel in dem Sinn, das es auf verschiedenen Rechnern lauffähig ist. Andererseits muß wegen der Übersetzung in eine maschinenunabhängige Zwischen-Sprache bei einer Änderung der Ziel-Maschine lediglich der Code-Generator modifiziert werden. Zur Portabilität des Systems trägt auch bei, daß die Basis-Software portabel ist, da diese zu einem großen Teil ebenso wie das Übersetzungssystem in einer portablen höheren Sprache geschrieben ist.

- Der Code-Generator kann durch Optionen gesteuert werden, über die z.B. der 8086-Adressierungs-Mechanismus festgelegt wird.

- Das Betriebssystem PORTOS ist ein Echtzeit-Betriebssystem, das kurze Reaktionszeiten auf externe Ereignisse garantiert (maximale Interrupt-Latenzzeit des Betriebssystem-Kerns beim 8086-Prozessor ca. 50 µsec). Außerdem besitzt der Betriebssystem-Kern eine genau definierte Treiber-Schnittstelle, mithilfe derer neue Geräte-Treiber in das Basis-Software eingebracht werden können.

- Die durch das Betriebssystem angebotenen Dienste sowie andere Betriebssystem-Parameter (etwa die Zeitauflösung) können über einen Betriebssystem-Konfigurator bequem modifiziert werden.

- Die Basis-Software bietet Schnittstellen zum Lesen und Setzen der Uhrzeit. Beim Setzen der Uhrzeit werden außerdem laufende PEARL-Schedules derart berichtigt, daß relative Schedules (wie "AFTER 10 SEC RESUME") vom Setzen der Uhr unberührt bleiben, während absolute Schedules (wie "AT 24:00:00 ACTIVATE TASK0) hinsichtlich der neuen Uhrzeit berichtigt werden.

UNIX als Entwicklungsumgebung für Realzeit-Anwendungen - Ein Überblick -

Hans Windauer, Werum GmbH
Glogauer Straße 2 A, 2120 Lüneburg

Zusammenfassung

Anläßlich der PEARL-Tagung 1986 präsentierte G. Färber technische Alternativen für die Auflösung des Widerspruchs "Einsatz von UNIX bei Realzeit-Anwendungen" /3/. Der vorliegende Beitrag befaßt sich mit diesem Thema aus marktorientierter Sicht unter dem Aspekt, ob UNIX als *Entwicklungsumgebung* für Realzeit-Anwendungen geeignet ist und *auf lange Zeit standardisiert* verfügbar sein wird. Die wesentlichen Aussagen sind:

- UNIX verbreitet sich zunehmend auch außerhalb von Entwicklungsstätten.
- Die Standardisierungsbemühungen um UNIX konvergieren.
- Der angestrebte Standard wird langfristig verfügbar sein.
- Unter UNIX steht eine Vielzahl von Software-Werkzeugen für die Entwicklung von Realzeit-Systemen zur Verfügung – bis hin zu vollständigen, integrierten Produktionsumgebungen.
- Diese Möglichkeiten können bereits heute durch PEARL-Anwender genutzt werden.

1. Verbreitung von UNIX

Ursprünglich – vor etwa 20 Jahren – wurde UNIX von Software-Entwicklern der Bell Laboratorien als komfortable Entwicklungsumgebung für Software-Entwickler entworfen und realisiert. Mittlerweile wird UNIX längst nicht mehr nur zur Entwicklung von Software eingesetzt, sondern auch als Betriebssystem für Anwendungen in den Bereichen CAD, Verwaltung, Textverarbeitung, Büro-Automatisierung, Kommunikation, Datenbanken u.a. Weltweit waren 1986 etwa 593.000, in Westeuropa etwa 50.000 UNIX-Systeme im Einsatz (Quelle: Yates). Laut Diebold wird 1990 jeder vierte Mehrplatzcomputer mit einer UNIX-Version als Betriebssystem ausgeliefert sein.

Fast jeder Hersteller bietet heute bereits eine UNIX-Version an (vgl. /2/):

Mainframe	Minicomputer	Microcomputer	Workstation
– Amdahl	– AT&T/Olivetti	– Altos	– Apollo
– CDC	– Celerity	– AT&T/Olivetti	– ATM
– DEC	– Computer Automation	– Charles River	– Bull
– IBM	– Data General	– Corvus	– Cad Tech
– Unisys	– DEC	– Cromemco	– Convergent Technology
⋮	– GEC (U.K.)	– DEC	– DEC
	– Gould/SEL	– Hewlett-Packard	– Hewlett-Packard
	– Hewlett-Packard	– Intel	– IBM
	– IBM	– Kontron	– ICL
	– Nixdorf	– Motorola	– Masscomp
	– Paradyne	– NCR	– Siemens
	– Perkin-Elmer	– Onyx	– Sun
	– Prime	– PCS	– Tektronix
	– Pyramid	– Pixel	– Texas Instruments
	– Ridge	– Plexus	⋮
	– Siemens	– Siemens	PC
	– Unisys	– Televideo	– Apple
	⋮	– Thomson (Micromega)	– AT&T
		– Unisys	– DG
		– Wicat	– IBM
		– Zilog	– NEC
		⋮	– Siemens
			– Tandy
			⋮

2. Standardisierung von UNIX

Die günstigen Wachstumsprognosen gründen im wesentlichen auf den Standardisierungsbemühungen von IEEE und X/OPEN, die die Vielfalt der UNIX-Derivate unter Berücksichtigung des "De-facto-Standards Systems V" von AT&T vereinheitlichen wollen.

In dem IEEE-Komitee P 1003 erarbeiten mehr als 60 Firmen und Organisationen (darunter IBM, DEC, AT&T, HP, X/OPEN) eine herstellerunabhängige Standard-Betriebssystem-Schnittstelle auf der Basis der System V Interface Definition (SVID) von AT&T. Die Arbeit erfolgt in mehreren Subkomitees (vgl. /7/):

- IEEE P 1003.1 : Operating system kernel POSIX
  (Portable Operating System based on UNIX)
- IEEE P 1003.2 : Shell and tool specifications
- IEEE P 1003.3 : Verification test procedures
- IEEE P 1003.4 : Reallime facilities

In P 1003.4 arbeitet insbesondere die Firma Hewlett-Packard mit, die mit HP-UX bereits langjährige Erfahrungen hinsichtlich der Erweiterung von UNIX um Realzeit-Eigenschaften besitzt.

Parallel dazu arbeitet die X/OPEN-Gruppe seit 1984 an einer einheitlichen Systemumgebung (nicht nur Betriebssystem) auf Basis der SVID. Zur X/OPEN-Gruppe gehören folgende Firmen:

Seit 1984 :	Bull
	ICL
	Nixdorf
	Olivetti
	Siemens
Seit 1985 :	Philips
	Ericsson
Seit 1986 :	DEC
	Unisys
	Hewlett-Packard
Seit 1987 :	AT&T

Die Arbeitsergebnisse sind in Form des X/OPEN Portability Guide veröffentlicht, der für die Mitglieder u.a. die Implementierung der X/OPEN System V Specification (XVS) vorschreibt. Wichtig ist, daß XVS wie POSIX eine *hersteller-unabhängige UNIX-Schnittstelle* definiert und kein *Betriebssystem* (wie AT&T System V) ist, so daß XVS durchaus auf Basis anderer Betriebssysteme als AT&T System V.2 implementiert werden kann – z.B. auch auf Basis von VMS!

In diesem Sinne scheinen die Standardisierungsbemühungen von AT&T, IEEE und X/OPEN gemäß folgendem Bild zu konvergieren (vgl. /7/), zumal die Unterschiede zwischen POSIX, SVID nach AT&T und SVID nach X/OPEN (XVS) nur noch sehr gering sind.

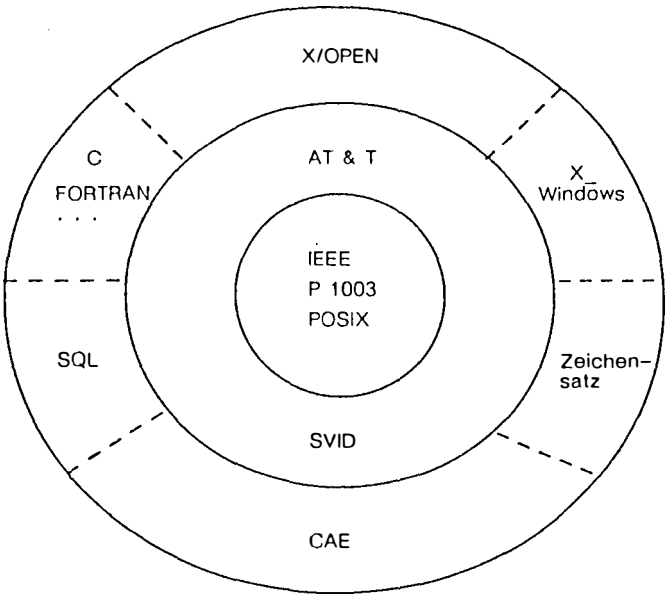


Bild 1: Konvergenz der Standards

3. Verfügbarkeit des angestrebten Standards

Ein Blick auf das derzeitige UNIX-Angebot einiger Hersteller zeigt, daß der angestrebte Standard rasch verfügbar sein wird:

Hersteller	UNIX-System	UNIX-Version
Apollo	UNIX	SVID (AT&T)
AT&T	UNIX	AT&T System V
DEC	ULTRIX32	SVID (X/OPEN) mit Erweiterungen
Hewlett-Packard	HP-UX	SVID (X/OPEN) mit Realzeit-Erweiterungen
IBM (Microsoft)	XENIX 5.0	SVID (AT&T)
IBM	AIX	SVID (AT&T) mit Erweiterungen
Intel	UNIX	AT&T System V
Nixdorf	UNIX	AT&T System V
Modcomp (ATM)	UNIX	AT&T System V

PCS	MUNIX	AT&T System V mit Erweiterungen
Siemens	SINIX	SVID (X/OPEN) mit Erweiterungen

Wichtige Hersteller bieten für erfolgreiche Computer-Familien gar kein anderes Betriebssystem als UNIX an, so Hewlett-Packard für die HP 9000-Familie, Nixdorf für die TARGON-Familie und Siemens für die MX500-Familie.

4. Entwicklung unter UNIX, Ausführung in Realzeit

4.1 UNIX und Realzeit-Aufgaben

In /3/ werden technische Möglichkeiten zur Lösung von Realzeit-Aufgaben mittels UNIX dargestellt:

a) Kleinere Modifikation des UNIX-Kerns zur Verbesserung des Realzeitverhaltens.

b) Neustrukturierung des UNIX-Kerns bei Beibehaltung der Systemcall-Schnittstelle.
(In Zukunft werden hierbei die Ergebnisse des Realtime-Subkomitees P 1003.4 von IEEE Bedeutung gewinnen.)

c) Heterogene Multiprozessor-Konfigurationen mit Aufgabenteilung zwischen Standard-UNIX-Systemen und dedizierten Realzeit-Systemen.

Die Festlegungen von IEEE (POSIX) und X/OPEN lassen nun noch eine weitere Möglichkeit zu, ein UNIX-System mit Realzeit-Eigenschaften zu realisieren:

d) Implementierung der SVID nach X/OPEN oder POSIX auf Basis eines vorhandenen Betriebssystems, das bereits Realzeit-Eigenschaften besitzt.

Verfügbare Beispiele für b) und c) sind (ohne Anspruch auf Vollständigkeit):

Neustrukturierung des UNIX-Kerns

- AIX auf IBM-PC-RT (IBM 6150) (vgl. /3/)
- PPX auf TARGON/32 von Nixdorf, basierend auf AT&T System V (vgl. /3/)
- RTU auf Masscomp, basierend auf AT&T System V2 (vgl. /4/)
- UTX/32RT auf Gould CONCEPT 32, basierend auf AT&T System V u. BSD 4.2 (vgl. /5/)
- HP-UX auf HP 9000-Familie

Multiprozessor-Konfiguration

- Intel 320
basierend auf Multibus II mit den Betriebssystemen XENIX und RMX286 auf verschiedenen Prozessoren Intel 80386.

PCS PEARL Engine 68000
bestehend aus zwei (oder mehr) Systemen PCS Cadmus 9.xxx, die über Ethernet vernetzt sind; ein System wird mit MUNIX, die anderen Systeme werden mit dem Realzeit-Betriebssystemkern BAPAS-K von Werum betrieben (vgl. /9/).

Multiprozessorsystem von Thomson
basierend auf VMEbus mit den Betriebssystemen UNIX System V (AT&T) und pSOS auf verschiedenen Prozessoren MC 68020 (vgl. /1/).

Ein Beispiel für den Fall d) ist dem Autor nicht bekannt. Laut /8/ wird jedoch das X/OPEN-Mitglied DEC "die herstellerneutralen Standards in seinen Betriebssystemen VAX/VMS und ULTRIX ... zügig implementieren".

4.2 Entwicklung von Realzeit-Software unter UNIX

Für alle Phasen des Software-Lebenszyklus stehen unter UNIX verschiedene Software-Werkzeuge zur Verfügung. Beispiele für portable Produkte sind:

Analyse-, Spezifikations-, Entwurfswerkzeuge
EPOS, PROMOD (in Vorbereitung für UNIX), Teamwork

Compiler, Debugger
Ada, C, Pascal, PEARL

Versions-, Konfigurationskontrolle
sccs, make, Teamwork, VICO

Benutzeroberflächen
X_WINDOWS

Dokumentationssysteme (Text und Grafik gemischt)
ALIS, HIT, OFFICE, WPS

Wichtig ist, daß die meisten zukunftsorientierten Projekte zur Schaffung von integrierten Produktionsumgebungen auf UNIX basieren. So werden auch die vier deutschen vom BMFT geförderten Verbundvorhaben POINTE, PROSYT, RASOP und UNIBASE und das EG-Projekt PCTE (Portable Common Tool Environment) auf Basis von UNIX durchgeführt.

Allein in die vier deutschen Vorhaben investieren der BMFT und die beteiligten Firmen mehr als 200 Millionen DM.

Über Software-Werkzeuge hinaus bietet die UNIX-Welt starke Unterstützung in den wichtigen Bereichen

- vernetzte Systeme (NFS, RFS, TCP/IP, ...) und

- Datenbanksysteme (ORACLE, INGRES, INFORMIX, BAPAS-DB).

5. Möglichkeiten für PEARL-Anwender

Die sich aus der Verbreitung und Standardisierung von UNIX ergebenden Vorteile können von PEARL-Anwendern ungehindert genutzt werden. Bereits heute stehen PEARL-Compiler und -Ablaufsysteme für folgende Rechenanlagen unter UNIX zur Verfügung:

- Apollo Domain DN 3000
- HP 9000 - Familie
- PCS Cadmus 9000 - Familie
- PCS PEARL Engine 68000
- Sun Workstation.

Auf Apollo, Cadmus und Sun werden die E/A-Anweisungen von PEARL-Tasks synchron ausgeführt; die Realzeit-Erweiterungen von HP-UX und die heterogene Struktur der PEARL Engine 68000 erlauben eine asynchrone Ausführung der E/A-Anweisungen.

Außerdem steht PEARL in PROSYT, der Produktionsumgebung für verteilte Systeme in der Technik, als Bestandteil einer durchgängigen Werkzeuglinie auf PCS Cadmus unter UNIX zur Verfügung.

Literatur

/1/ Adelving, G.: Ein Echtzeitkern für UNIX mit VMEbus-Steuerung. Design & Elektronik Nr. 11, 26.05.87, S. 116 - 117.

/2/ Domann, P.: UNIX-Markt, Einsatz, Trends. In: Tagungsband "UNIX in der industriellen und kommerziellen Praxis", Düsseldorf, 01. - 03.06.87.

/3/ Färber, G.: Unix- und Realzeit-Anwendungen. In: Tagungsband "PEARL 86, Boppard, 4. und 5.12.86", S. 9 - 25. Hrsg.: PEARL-Verein e.V., Geschäftsstelle München.

/4/ Graefe zu Baringdorf, W.: RTU - ein echtzeitfähiges UNIX-Betriebssystem. Tagungsunterlagen "UNIX in Deutschland, GUUG-Jahrestagung '87, Karlsruhe, 22. - 24.09.87", Abschnitt 3.1.

/5/ Hovermann, H.: UNIX und Realtime - Ein Widerspruch? In: Tagungsunterlagen "UNIX in Deutschland, GUUG-Jahrestagung '87, Karlsruhe, 22. - 24.09.87", Abschnitt 3.2.

/6/ Isaak, J.: Standards For The Unix System. UNIX/WORLD, October 1986, S. 34 - 38.

/7/ Lambert, M.: X/OPEN: Past, Present & Future. A Review of The Activities of The X/OPEN Group. Erhältlich bei X/OPEN Group.

/8/ Pelda, U.: Warum UNIX? unix/mail 5 (1987) 2, S. 11.

/9/ Schindler, F.; Windauer, H.: PEARL Engine 68000. In: Tagungsband "PEARL-Tagung '84", Düsseldorf, 06. – 07.11.84, S. 99 – 102.

Adding Real Time Capabilities to the UNIX* Operating System

Suzanne M. Daughty

Sol F. Kavy

Steven R. Kusmer

Douglas V. Larson

David C. Lennert

Frank-Peter Schmidt-Lademann

Hewlett-Packard Company

ABSTRACT:

Adapting the Unix operating systems to real-time markets is a lucrative challenge. By adding a little new functionality and a lot of performance tuning, Unix systems can support more demanding real-time applications such as those found on the factory floor, tapping into a multi-billion dollar market demanding a portable software environment such as System V. Most of the needed real-time functionality is already found in System V and 4.2BSD. Performance tuning is needed in the area of response time, especially process dispatch latency, which on typical Unix systems is measured in seconds rather than milliseconds. This paper presents what functionality is needed to adapt Unix systems to real-time markets, how they may acquire the needed performance, and how this combination satisfies real customer needs.

HPUX/RT is a Unix implementation with a BSD4.2 Kernel that implements the SVID and is enhanced with BSD4.2 functionality and HP proprietary services. HPUX/RT was especially tuned to provide deterministic realtime response and enhanced with functionality needed for realtime applications. We will discuss the methods used to achieve realtime performance in the HPUX/RT operating system.

---

* Unix is a registered trademark of AT&T in the United States and other countries.

1. Requirements for Real-Time Systems

The UNIX operating system is found in many marketplaces. It is the operating system for scientific supercomputers and PCs. However, one of the final frontiers for the acceptance of UNIX systems is in the real-time marketplace, and for a good reason: the real-time customer is the most demanding customer there is. The real-time customer's demands fall within three categories:

Performance

Real-time applications are primarily measured by their performance. Therefore, real-time customers will expect to squeeze the last ounce of performance out of a real-time system to meet their needs, and they will sometimes take measurements, their computer vendor never expected. The performance characteristics they measure are typically in terms of response time or throughput. An example of a response time measure is "How long after the receipt of an interrupt from my parallel I/O card can the system run my process which was waiting for that interrupt?" An example of a real-time throughput measure is "How long will it take for me to push my two gigabytes of data from my device to the filesystem?". The real challenge is that both questions will often be asked by the same customer!

Determinism

Customers expect that a real-time system will react in a deterministic manner. For example, it is not enough to have a good response most of the time -- you must provide good response all of the time. Real-time customers often build a computer into a system that has unforgiving constraints, which is usually because the system is controlling or monitoring other devices or machinary. As an example, a real-time computer built into a steel mill whose steel travels at 30 mph will be expected to respond quickly to alarm conditions. If the computer unexpectedly becomes busy for a whole second, the steel in the steel mill will have treveled 44 feet, and could possibly be strewn over the steel mill floor.

Flexibility

In the end, it is the real-time customers who truly

know best how e real-time computer can solve their applications needs. Customers must be provided with tools for writing their own drivers and for measuring system performance. They must be provided with source code, because they choose to understand in-depth how a system performs and they might want to tune it for their application. On the other hand, vendors of real-time computers must be humble, because real-time customers are glad to tell then how to build their systems!

The remainder of this paper presents e definition of a real-time system and then explains the real-time features implemented on the HP 9000 Series 800 computers. The HP 9000 is HP's computer family for engineering and manufacturing, and it runs HP-UX, a superset of AT&T SVID Issue 1. The Series 800 is HP's Precision Architectures computer line under the HP-UX operating system.

2. What is e Real-Time System?

A real-time system is a system that can respond in a deterministic and timely manner to events in the real world. Events in the real world could mean either large amounts of data that must be processed fast enough to prevent losing data (data throughput), or discrete events that must be recognized and responded to within certain time constraints (response time).

The table 1 presents and categorizes some real-time applications. It is important to note, that this list is by no means comprehensive; its purpose is to show the variety and pervasiveness of real-time applications.

3. Adding Real-Time Capabilities to UNIX

Given a definition of real-time and some sample real-time applications, the next question is "How can the UNIX operating system be augmented to meet the requirements of real-time applications?". While using System V as a base, HP-UX answers these questions in two parts: 1) by incorporating functionality from 4.2 BSD and adding new functionality from HP, and 2) by doing performance tuning on the kernel and filesystem. To better understand this approach, it is helpfull to be familiar wit HP's goals for adding real-time capability to the UNIX operating system:

Table 1: General Real-Time Applications and Some Examples

General Real-Time Applications	Examples
process monitoring and control	petroleum refinery paper mill chocolate factory
data aquisition	pipe-line sampling data inputs from a chemical reaction
communications	controlling satellites telephone switching systems
transaction-oriented processing	airlines reservation systems on-line banking stock quotations system
flight simulation and control	autopilot shuttle mission simulator
factory automation	material tracking parts production electronic assembly machine or instrument control
transportation	traffic light system air traffic control
detection systems	radar systems burglar alarm systems
interactive graphics	image processing video games

- Any real-time features implemented must not prevent SVID compatibility.
- Whereever possible, real-time features should be adopted from either System V or 4.2BSD. Only where needed real-time feature does not exist should HP add a new feature.
- Real-time features must be portable.
- Performance tuning must be transparent to user process.
- Real-time response must be comparable to real-time response on the HP 1000 A900 (HP's top-of-the-line real-time A-Series computer.

HP-UX on the Series 800 has met these goals. In addition, HP

is lobbying through standards-setting bodies to encourage their adoption of HP-UX's real-time features as part of an existing or evolving standard such as SVID or IEEE P1003.

3.1 Real-Time Features in HP-UX

This section introduces the real-time features of HP-UX on the Series 800 computers, explains their origin (either System V, 4,2BSD or HP) and also explains how each feature addresses certain concerns about real-time capabilities of UNIX systems.

The following features provide real-time capability to HP-UX:

Added Functionality

- Priority-based preemptive scheduling
- Process memory locking
- Privilege mechanism to control access to real-time priorities and memory locking
- Fine timer resolution and time-scheduling capabilities
- Interprocess communication and synchronisation
- Reliable signals
- Shared memory for high bandwidth communication
- Asynchronous I/O for increased throughput
- Synchronous I/O for increased reliability
- Preallocation of disk space
- Powerfail recovery for increased reliability

Performance Tuning

- Kernel preemption for fast, detrministic response time
- Fast file system I/O
- Miscallaneous performance improvements

3.2 Added Functionality

3.2.1 Priority-Based Preemptive Scheduling Priority-based preemptive scheduling lets the most important process execute first, so that it can respond to events as soon as possible. The most important process executes until it sleeps voluntary or finishes executing, or until a more important process preempts it. Priority based means that a more important process can be assigned a priority higher than other processes, so that the important (high priority) process will execute before other processes. Preemptive means that the high priority process can interrupt or preempt the execution of a lower priority process, instead of waiting for it to be preempted by the opereting system when its time slice is completed or it needs to block.

The scheduling policy of traditional UNIX systems strives for fairness to all users and acceptable response time for terminal users. The kernel dynamicelly adjusts process priorities, favoring interactive processes with light CPU usage at the expense of those using the CPU heavily. Users are given some control of priorities with the nice(2) system call, but the nice value is only one factor in the scheduling formula. As a result, it is difficult or impossible to guarantee that one process has a priority greater than another process. Therefore, each process in a traditional UNIX system effectively has to wait its turn, no matter how important it might be to the real-time application.

HP-UX presents a solution to this problem by adding a new range of priorities, called real-time priorities. Priorities in the real-time range do not fluctuate like priorities in the normal range, and any process with a priority in the real-time range is favored over any process with a priority in the normal range, including those making system calls and even system processes. Important as real-time processes are, interrupt processing is given priority over them. If several real-time processes have the same priority, they are timesliced.

Processes with real-time priorities are favored not only for receiving CPU time, but also are favored for swapping and for file system access. Real-time processes are the last to be swapped out (except for locked processes), and the first to be swapped in. File system requests for real-time processes go to the head of the disk request queue. All of this preferential treatment gives real-time processes very good response, but at the expense of the rest of the system.

Real-time priorities are set by the user either programmaticelly with HP's new rtprio(2) system call, or interactively with the rtprio(1) command. By default, processes are time-shared and continue to be executed according to the normal scheduling policy. Aside from setting a process to a real-time priority, the rtprio(2) system call and rtprio(1) command can be used to read the priority of a real-time process and change the priority of a real-time process to be timeshared.

3.2.2 Process Memory Locking A second important feature in a real-time system is the ability to lock a process in memory so that it can execute without waiting to be paged in or swapped in from disk. In the UNIX and HP-UX operating systems, processes are not normally locked in memory; they are swapped and/or paged in from disk as needed.

HP-UX has adopted a solution to this problem from System V. The plock(2) system call allows a process to lock its executable code and/or its data in memory to avoid unexpected swapping and paging. Also, a process can lock additional data or stack space with the datalock(3C) subroutine, and lock shared memory segments as needed with the shmctl(2) system call.

3.2.3 Controling Access to Real-Time Capabilities Because the priority scheduling and mamory locking featuras of HP-UX are quite powerfull, it is desirable to allow only certain users to access them. If, for example, all users had acces to these capabilities, they could potentially set all of their processes to a high real-time priority and try locking them in memory, which would defeat the purpose of the real-time system. Or a novice user could assign real-time priority of 0 to a process that happens to execute an infinite loop, thus locking up the entire system.

To prevent scenarios such as these. HP-UX created a feature called privilege groups. Privilege groups enable certain users (other than just the superuser) to access the powerfull real-time priority and memory locking features of HP-UX. A privilege group is a group to which the superuser assigns privileges. Existing privileges are real-time priority assignment (RTPRIO), memory locking (MLOCK) and a third not real-time related privilege (CHOWN). The superuser assigns one or more of these privileges to one or more groups with HP's setprivgrp(2) systemcall or setprivgrp(1) command, and assigns certain users to become members of these groups with the 4.2BSD-based setgroups(2) systemcall.

3.2.4 Fine Timer Resolution and Time-Scheduling Another important feature in real-time operating systems is fine timer resolution and time-scheduling capabilities. For high resolution clock based applications, both repetitive and nonrepetitive, it is important to be able to execute a process or subroutine at a precise time. For example, a real-time application might require various sensor readings at 20 millisecond intervals.

Standard features in System V that deal with time, such as alarm(2) which has a resolution of one second, and cron(1) and at(1) which have a resolution of a minute, are not precise enough for many real-tima applications. Therefore, HP-UX has adopted a solution from 4.2BSD, known as interval timers. Each process can enable its own interval timer to interrupt itself once or at repeated intervals, with whatever precision the underlying hardware and operating system can support. The interval is defined in units of

seconds and microseconds to keep the timer interface portable despite the system dependant resolution. For HP-UX on the Series 800, the system clock resolution for scheduling alarms is 10 milliseconds for measuring time 1 microsecond.

3.2.5 Interprocess communication and Synchronization A real-time operating system must provide interprocess communication and synchronization facilities. Interprocess communication and synchronization is important because real-time applications often involve several asynchronous processes that need to exchange information.

Pipes and signals are common interprocess communication facilities in the UNIX operating system. A pipe is essentially an I/O channel through which data is passed with the read(2) and write(2) system calls. An advantage of using a pipe is that it provides synchronization by blocking reader processes when the channel is empty end blocking writer processes when the channel is full. The disedvantage of using pipes are 1) they require the communicating processes to have a common ancestor process that sets up the channel, and 2) they are often slow because the kernel has to copy the data from the writer process to the system buffer cache and then back again to the address space of the reader process. Many UNIX systems including HP-UX support named pipes, which overcome the first problem, but still have the performance penalty of copying data.

A signal is essentially a software interrupt sent to a process by the kernel or by a user process. A process can install a handler for almost any signal, and the handler will execute when the signal is received. Signals can be a good event or alarm machanism because one process can send a signal to inform another process that an event occured, and then the other process can immediately enter its handler to respond to the event. The disadvantages of using signals are 1) they pass little or no data (not even who the sender process is), and 2) they are traditionally unreliable when sent repeatedly or when a process tries to wait for a signal. HP-UX has therefore adopted e reliable signal interface from 4.2BSD, in addition to the System V signal interface. The 4.2BSD signal interface solves the reliability problems of the system V interface, but is more complicated to use. The 4.2BSD signal interface introduces blocking of signals in similar manner as interrupts can be masked on the hardware level.

HP-UX has adopted three IPC (Inter Process Communication) facilities from System V: shared memory, semaphores and messages. These facilities

allow communication and synchronization between arbitrary unrelated processes. An elaborate semaphore facility allows solutions to both simple and complex synchronization problems. A message passing facility allows transfer of arbitrary data structures, along with the ability to prioritize messages. The most important IPC facility for real-time applications is the shared memory facility. Two or more processes can attach the same segment of memory to their data space and read from and write to it. Shared memory can be locked into physical memory.

3.2.6 Asynchronous I/O Asynchronous I/O is I/O that overlaps with process execution or other I/O, typically resulting in increased throughput. Both the UNIX and HP-UX operating systems implement system asynchronous I/O to certain drivers, but HP-UX allows you to communicate with some drivers that do system asynchronous I/O, so you can take advantage of their asynchronous ability.

HP-UX implements system asynchronous facilities for terminals, pipes, named pipes and sockets. The asynchronous I/O facilities that HP-UX provides for terminals are:

The nonblocking I/O facility: Before launching an I/O request, a user process can set a flag to inform the driver that the driver should cause the I/O request to return immediately if the request can not be satisfied without blocking the user process. The request may return partially satisfied.

The SIGIO facility: Before launching an I/O request, a user process can set a flag to enable the driver to send the SIGIO signal to the process when the data has arrived in the drivers input buffer.

The select(2) facility: A user process can call select(2) to check if an I/O request should be issued to one or more devices. The driver sets a bit in a user supplied bit mask for each file descriptor that the user asked about and on which I/O can be performed.

The FIONREAD facility: Before launching a read(2) request, a user can ask the driver to tell it how many characters in the drivers input buffer are available for reading.

These facilities can be used individually or together. For example, suppose you want to read from several terminals and you are not sure, which terminal will send you data or when

to expect this data, if any. You do not want to launch a series of read(2) requests to each terminal, because you might end up missing data from one or more terminals as you try to read from some terminal that will never send you data. Instead, you could enable the SIGIO facility for each terminal so that each can inform you when data has arrived in its input buffer. When SIGIO is sent, you could call select(2) to find out which terminal(s) are ready for reading.

3.2.7 Synchronous I/O A real-time application sometimes prefers to do synchronous I/O operations to make sure that its I/O request actually completed. In synchronous I/O, a process initiates an I/O request and then suspends until the I/O request completes. The file system normally does asynchronous writes, which means that a write(2) returns when the data has been only written to the buffer cache, not to the disk. The data is written from the buffer cache to the disk later, while the process continues to execute. Although this asynchronous disk write increases your process's throughput, the disadvantage is that you cannot be sure that your data has actually been written to the disk. Therefore, HP-UX provides a flag called O_SYNCIO that lets you perform a synchronous disk write. This ensures, that your data actually was written to disk.

3.2.8 Summary of Real-Time Functionality Added to HP-UX Table 2 summarizes the functionality that was added to HP-UX. It presents the system call associated with the particular feature and the origin of the system call (either System V, 4.2BSD or HP).

3.3 Performance Tuning

The performance tuning that HP has implemented in HP-UX on the Series 800 computers is as important as the added real-time functionality. The following features, kernel preemtion, fast file system I/O, and miscellaneous performance improvements comprise the main part of HP's performance tuning efforts.

3.3.1 Kernel Preemption for Faster Response Time An important requirement for real-time systems is a quick and deterministic response time. One of the main concerns about the real-time capability of UNIX systems is that a process can execute in kernel mode for long periods of time (more than 1 second) without allowing a higher priority process to preempt it. Instead, the process keeps executing in kernel mode until it blocks or finishes, while the high priority process must wait. A process executing in user mode gets preempted much more quickly.

Table 2: Real-Time Functionality in HP-UX

Real-Time Function	Associated system call	Origin
Priority based preemptive scheduling	rtprio(2)	HP
Memory locking	plock(2)	System V
Privilege groups	getprivgrp(2) setprivgrp(2)	HP and 4.2BSD
Fine timer and time scheduling	setitimer(2) gettimeofday(2)	4.2BSD
Reliable signals	sigvector(2) sigblock(2) sigpause(2)	4.2BSD
Shared Memory	shmget(2) shmctl(2) shmop(2)	System V
Other IPC facilities	pipe(2) msg{getlctllop} sem{getlctllop}	System V
Asynchronous I/O	ioctl(2) flags select(2)	HP and 4.2BSD
Synchronous I/O	fcntl(2) with O_SYNCIO	HP
Preallocating disk space	prealloc(2)	HP
Powerfail recovery	signal(2) with SIGPWR	HP and System V

HP-UX on the Series 800 solves this problem by implementing a preemptable kernel. At certain safe places in the kernel called preemption points or preemption regions, HP-UX keeps kernel structures at a consistent state, so that a higher priority real-time process can get control of the CPU at that point.

Implementation of Kernel Preemption

Two types of preemption have been implemented: there is the synchronous method which allows preemption at a specific point during kernel execution, and a asynchronous method

which allows preemption anywhere during a region of kernel execution. When a higher priority real-time process becomes runnable, kernel preemption is requested by setting the reqkpreempt flag and generating a hardware supported interrupt. The now pending preemption request is serviced immediately if the running process is in user mode or in a preemptable kernel region, or at the first point when either:

a. the KPREEMPTPOINT() macro is executed which tests the reqkpreempt flag to allow a synchronous preemption,
b. the spl level drops to splpreemptok() which allows the pending preemption interrupt and thereby a asynchronous preemption,
c. user mode is entered (one case where the spl level drops to splpreemptok()), or
d. swtch() is called which always transfers to the highest priority runnable process.

The reqkpreempt flag and the preemption interrupt are both cleared by swtch() whenever it switches to a new (highest priority) process. In total, approximately 180 synchronous preemption points and 20 asynchronous preemption regions were added to the HP-UX kernel.

Limitations

There is one overriding limitation on what kernel preemption can accomplish: Kernel preemption can only preempt an operation which is being executed within a process context. It cannot preempt interrupt processing code and allow a process to execute because UNIX does not support this type of operation. Therefore, all interrupt processing is implicitly considered to be of higher priority than any (realtime) process. Interrupt requests mainly come from the I/O system but additionally the UNIX system allows non I/O code to be executed in an interrupt processing context. This facility, called the callout queue, causes a kernel procedure to be executed at a specific time offset. The procedure is invoked from a interrupt processing context during clock interrupt processing. To minimize callout queue execution overhead, in HP-UX a seperate system process was created to provide a preemptable process context in which to execute some lengthy callout queue code like gathering statistics and recalculating timesharing priorities.

Measured Improvement

In order to tell how long the kernel executes without blocking or preempting, and where in the kernel these long execution paths are, the kernel was instrumented to collect

timing measurements. Timing measurements are taken by sampling the time at kernel entry and exit and also whenever the kernel changed between preemptable and non preempable state. The time intervals during which the kernel executes in non preemptable state are logged together with the pc stack traces of the start and the end of the interval. The time spent in the interrupt context is also logged.

To determine the improvement made in real-time dispatch time, two sets of measurements were taken. One set with kernel preemption enabled and one set with it disabled under the same workload. The workload consisted of tests which validate the correct working of all kernel functions.

The raw results consist of a stream of times. Each time represents an interval when the kernel was non preemptable. As one way of summarizing the results, a distribution was computed which represents the percentage of total kernel execution time that was spent in non preemptable codepaths of less than x milliseconds. Figure 1 shows this distribution for both preemption on and preemption off.

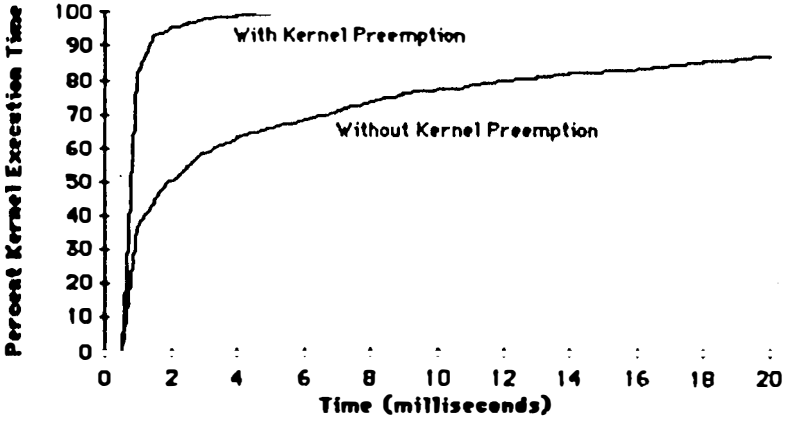


Figure 1

Table 3 shows that HP-UX kernel preemption has provided significant improvements in real-time process dispatch time. In the worst case observed, the improvement was well over 50 fold. In the case where preemption is off the longest non-preemptable code paths are typically large data copies during process creation (fork), process overlay (exec), or user I/O operations. In the cases where preemption is on, the current longest code paths are now in the terminal driver.

Table 3: Non-preemptable Kernel Time			
	Preemption Off	Preemption On	Improvement
90% kernel	40 ms	1.4 ms	x28
99% kernel	129 ms	3.4 ms	x37
max kernel	1127 ms	14.6 ms	x77

These results represent the status of HP's preemption tuning activities as of this writing; work is currently underway to further reduce these times. Future work will entail further improvements to real-time performance via increased kernel semaphoring in order to address more stringent application needs.

3.3.2 Fast File System I/O The traditional UNIX filesystem does not meet the performance and reliability requirements of real-time systems for the following reasons:

- Data blocks are often scattered randomly throughout the disk, resulting in large seek times for sequential reeds.
- The Data blocksize of 512 or 1024 can be inefficient for large read and write requests.
- There is only one copy of the superblock containing all vital data about the file system structure on the disk. If it gets corrupted all data on the filesystem is lost.

The HP-UX file system has adopted its solution from the McKusick or 4.2BSD file system. Two important features in the HP-UX file system are the implementation of cylinder groups, which reduce file seek time and add raliability, and the addition of two block sizes for increased speed without wasting space. Cylinder groups together with allocation algorithems provide locality for related data and spread unrelated data resulting in short seek times. Eech cylinder group has a copy of the vital superblock information for reliability. The HP-UX file system uses a hybrid block size to deal with the time and space tradeoff of big versus small size blocks. There is a blocksize of 4K or 8K and a fragment size of 1/8, 1/4, 1/2 or the same size as the block size. Large file I/O requests are allocated and accessed a block at a time, while smaller requests are alloceted and accesed a fragment at a time.

3.3.3 Miscellaneous Performance Improvements HP-UX on series 800 is tuned for both real-time response and throughput. Benchmarks representing various workloads (for real-time and other enviroments) were run to track and improve the performance of specific paths in the operating

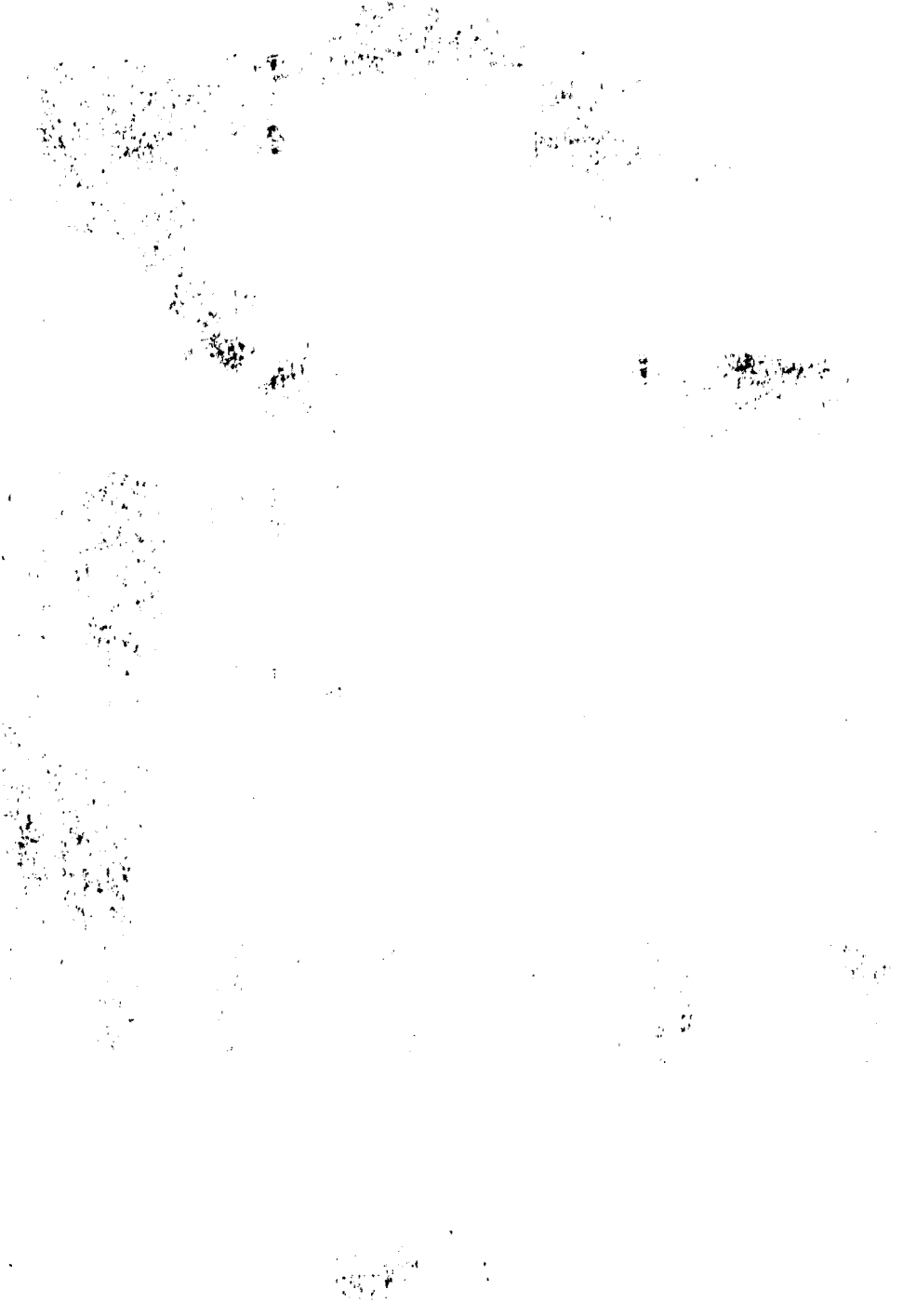
system. Other measures such as time from interrupt to driver were measured with a logic analyzer interface to the hardware. This systematic aproach to performance tuning led to very significant results, with many performance measures improving by a factor of two or more during product development. Also this approach led to justifiable returns, including support for two-hand clock replacement algorithm and conversion of various kernel data structures from linear lists to hashed lists.

4. Conclusion

The functionality additions and performance improvements described in this paper form the foundation by which HP is enabling its version of the UNIX operating system to successfully enter the real-time marketplace. The features decribed are rather simple to implement, and in fact, most of them are already in System V or 4.2BSD. HP is working with the IEEE P1003 committee and the real-time subcommittee to help form a common standard by which any vendor can gain the needed functionality.

References

1. David C. Lennert; "Decreasing Realtime Process Dispatch Latency Through Kernel Preemption"; USENIX Conference Proceedings, Summer 1986, pages 405-413
2. Suzanne M Doughty, Sol F. Kavy, Steve R. Kusmer, Douglas V. Larson; "Unix for Real Time"; UniForum Conference Proceedings, Winter 1987, page 219-230



Einflußfaktoren auf die Echtzeit-Leistung beim Einsatz von VAX-Systemen

Dipl Ing. Christoph Schmidt
Digital Equipment GmbH
Freischützstraße 91
8000 München 81

Bus-Struktur, CPU-Architektur und -Geschwindigkeit, Eigenschaften des Betriebssystems, Programmiermöglichkeiten des Ein/Ausgabesystems,Eigenschaften und Geschwindigkeit der Ein/Ausgabesteuerung, Charakteristika des Anwender-programmes dies alles sind Faktoren die für Anwendungs-programmierer und Benutzer von großer Wichtigkeit sind, da sich durch geeignete Kombination der Parameter eine maximierung der erreichbaren Leistung bewirken läßt.

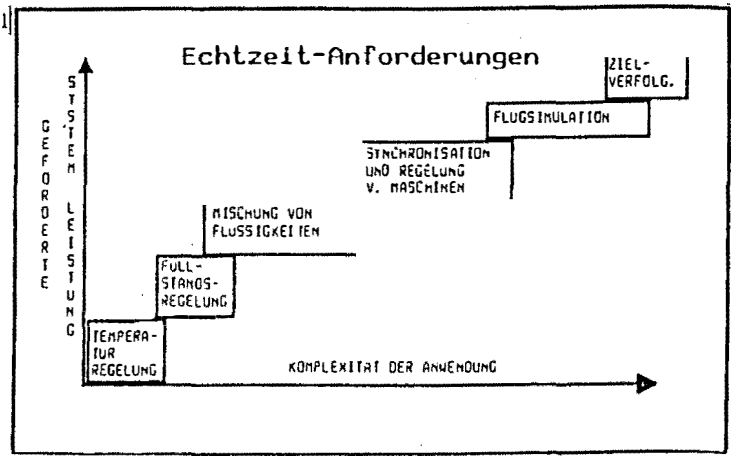


Bild 1. Zusammenhang von Systemleistung und Komplexität der Anforderungen

16-Bit Rechner sind für Echtzeitanwendungen weit verbreitet und sehr beliebt. Die Entwicklung auf dem Echtzeitsektor geht jedoch heute zu immer höherer Komplexität der Anwendung, was immer höhere Systemleistungen bezüglich Rechengeschwindigkeit und Datendurchsatzrate nach sich zieht.
Hier sind die 16-Bit Rechner am Ende ihrer Leistungsfähigkeit angelangt und es wird die Forderung nach 32 Bit Adressraum gestellt. Um solche 32-Bit Rechner jedoch effektiv programmieren zu können muß das Betriebssystem multi-tasking-fähig sein, da sonst komplexe Systemprogrammierung anfällt. Multitasking bedeutet jedoch, dass Ressourcen für die Verwaltungsaufgaben verbraucht werden und so längere Reaktionszeiten auf Interrupts anfallen.
Sowohl das universale Betriebssystem VAX/VMS als auch die Echtzeit-Exekutive VAX-ELN sind multitaskingfähig und erlauben optimale Kontrolle über die leistungsbestimmenden Faktoren.

Es gibt viele Faktoren, die die Leistung eines Rechnersystems unter Echtzeitbedingungen beeinflussen. Dies sind vor allem

die Busstruktur des Gesamtsystems, die Architektur und Geschwindigkeit der CPU, ferner bestimmte Eigenschaften des Betriebssystems und die unterschiedlichen Arten der Steuerung der Ein-/Ausgabe. Wichtig bei allen Multitasking-Systemen sind auch die Möglichkeiten der Intertask-Synchronisation und -Kommunikation und ihre optimale Nutzung.

Ein großer Einflußfaktor ist letztlich das Anwenderprogramm, das jedoch hier wegen der Verschiedenartigkeit der Anwendungen nicht behandelt werden kann.

Die meisten Rechner haben eine Struktur mit Systembus, der CPU und Hauptspeicher bedient. Dieser Bus ist bei kleineren Rechnern der einzige, und auf ihm laufen auch alle E-/A-Operationen.

Leistungsfähigere Systeme haben separate E-/A-Busse, die so konfiguriert werden können, daß sie die Ein-/Ausgabe-operationen kanalisieren, anstatt externe Geräte direkt mit dem Systembus zu verbinden. Der Systembus bestimmt den maximalen Datendurchsatz des Rechners (Ein-/Ausgabeband-breite).Die E/A-Bandbreite ist deshalb eines der Hauptkriterien für die Auswahl des Rechnersystems bei Echtzeitanwendungen. Wenn die Applikation eine höhere Datenrate erfordert als das System liefern kann, dann ist offensichtlich ein solches System für die Applikation nicht geeignet.

In der Praxis ist es ohne spezielle Erfahrungen und Fähigkeiten der Systemprogrammierung nicht möglich, mehr als 50-75% der Nenndatenrate für Echtzeitanwendungen zu erreichen, auch wenn das Rechnersystem vollständig dediziert ist.

Nach der Wahl eines Systems mit genügend Bandbreite, können separate E-/A-Busse zur Entzerrung und Optimierung des Datenflusses zum und vom System gewählt werden.

So lassen sich zum Beispiel Massenspeicher auf einen und schnelle E-/A-Geräte auf einen anderen E-/A-Bus gruppieren.

Die folgende Tabelle gibt eine Übersicht über die maximale Ein-/Ausgabe-Bandbreite heutiger Systeme. Dieses sind jedoch HW-Spezifikationen, die den Verwaltungsaufwand des Betriebssystems nicht berücksichtigen. Die erreichbare Durchsatzrate hängt natürlich in erster Linie von der CPU-Last und von der eingesetzten Schnittstelle ab.

E/A-Bus	Computer	Adress-Ltg.	Daten-Ltg.	Max. Datenrate	Anzahl Bus-Adaptoren
UNIBUS	VAX 8650	18	16	1.5 MB/s	6
	VAX 8600	"	"	"	6
	VAX 785	"	"	"	4
Q-Bus	MicroVAX	22	16 gemultiplexed	3.3 MB/s	./.
VAXBI	VAX 8250/ 8350	32	32	13.3 MB/s	1
	VAX 8550	"	"	"	2
	VAX 8700/ 8800	"	"	"	4

Bild 2. Nenndurchsatzraten heutiger VAX-Rechnersysteme

Geschwindigkeit und Design der CPU sind kritische Faktoren wenn die Applikation programmgesteuerte Ein-/Ausgabe oder Signalverarbeitung in Echtzeit erfordert.

Die Frage ist hier, ob die CPU so aufgebaut ist, daß sie eine Steuerung über vektorisierte Interrupts erlaubt, oder ob nur rein sequentielle Verarbeitung unterstützt wird.

Beide Arten des CPU-Designs haben spezifische Vor- und Nachteile, ihr Einsatz hängt von den Erfordernissen der Applikationen ab. Da bei der VAX-Architektur beide

Steuerungsarten möglich sind, hat der Benutzer die Möglichkeit, die für ihn am besten geeignete Art der Steuerung (Polling oder Interrupt) zu wählen.

Unter Polling versteht man das durchlaufen einer Abfrageschleife, bis eine bestimmte Bedingung eintritt (z.B. Eingabe abgeschlossen). Da in diesem Fall die CPU keine anderen Aufgaben erfüllen kann bis die Bedingung erfüllt ist, kann der Einsatz dieser Arbeitsweise nur dann gerechtfertigt sein, wenn das Ergebnis der Operation zur weiteren Verarbeitung benötigt wird.
Die andere Art der Ein-/Ausgabesteuerung durch Interrupt ist wesentlich effektiver in der Nutzung der CPU. Bei Auftreten des Interrupts muß jedoch mit Verzögerungen (Context-Switch Latenzzeiten) gerechnet werden. Diese sind nun in hohem Maße von der Belastung des Gesamtsystems durch mehrere Benutzer oder Prozesse abhängig.

Wenn die Möglichkeit besteht, eine Schnittstelle mit DMA-Steuerung einzusetzen, spielt natürlich die CPU-Geschwindigkeit eine erheblich geringere Rolle, da dann der Datentransfer in den Hauptspeicher ohne Mitwirkung der CPU geschieht, diese dann nur für die Verwaltung des Systembusses sorgt und in der Zwischenzeit andere Aufgaben erledigen kann.

Bei DMA-Steuerung lassen sich, identische Rechnerkonfigurationen vorausgesetzt, Steigerungen der Ein-/Ausgabeleistung um mehr als eine Größenordnung gegenüber programmgesteuerter Ein-/Ausgabe erzielen.
Weiterhin ist zu berücksichtigen ob gleichzeitig mit der Datenein-/Ausgabe eine Verarbeitung der Daten stattfinden muß wie z.B. Schwellwerttest oder Fouriertransformation. Während ein vergleichen mit einer bestimmten Größe durchaus noch zusätzlich zur Ein-/Ausgabe stattfinden kann ist eine FFT meistens nur nachfolgend durchführbar. Hier können evtl. spezielle Signalprozessoren erhebliche Leistungsverbes-

serungen erbringen.

Bei der Planung für eine neue Applikation steht man häufig vor der Frage: Ist das System leistungsfähig genug? Dies läßt sich entweder durch Berechnung der für alle Operationen verbrauchten Zeit oder durch ein Test-Programm ermitteln. Die Berechnung ist in den meisten Fällen nur möglich, wenn das Programm sehr klein und mit einfachen Operationen verwirklicht ist, bei größeren Programmen und speziell solchen, die in einer Hochsprache geschrieben sind, empfiehlt sich das Testen mit einem Benchmark-Programm.

Großen Einfluß auf die Echtzeitleistung haben auch die Eigenschaften des Betriebssystems, das hier immer als multitaskingfähig vorausgesetzt wird. Natürlich haben single user - single tasking - Systeme den entscheidenden Vorteil des geringen Verwaltungsaufwandes, sind aber auch nur beschränkt einsetzbar; ihre Programmierung kann jedoch bei komplexen Aufgabenstellungen zu unüberwindlichen Schwierigkeiten führen.

Für den Entwickler sehr hilfreich sind System-Services, da sie ihm viele Aufgaben abnehmen. Da diese Dienste wie Task-Scheduling, Software Prioritäten und Ein-/Ausgabe-steuerung oft jedoch eine Funktionalität haben, die auf alle erdenklichen Gegebenheiten Rücksicht nimmt, sind sie mit einem bestimmten Verwaltungsaufwand behaftet. Dieser Aufwand drückt sich in längerer Bearbeitungszeit für einen bestimmten Dienst aus und kann bei dedizierten Anwendungen meistens umgangen werden.

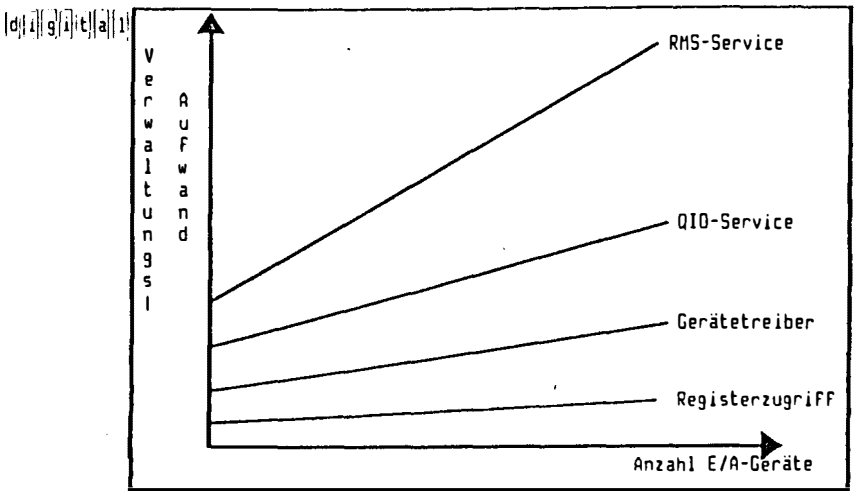


Bild 3. Zusammenhang von Verwaltungsaufwand und angeschlossenen Geräten

Dies ist durch systemnahe Programmierung möglich, die zum Beispiel mit einem speziellen Gerätetreiber, den der Benutzer selbst erstellt, realisiert wird.
Eine Anleitung zum Schreiben solcher Treiber ist Teil der normalen VMS-Dokumentation.

In einem prioritätsgesteuerten, multitaskingfähigen Betriebssystem wie VMS ist es prinzipiell möglich Echtzeitanforderungen und die Anforderungen mehrerer menschlicher Benutzer gleichzeitig abzuhandeln, wobei jedoch im Einzelfall ein Echtzeittask mit höherer Priorität bis zur vollständigen Ausführung die anderen Benutzer blockieren kann. Dies kann unter Umständen zur Frustration der menschlichen Teilnehmer führen, die zeitkritischen Operationen können jedoch beendet werden. Eine Abhilfe können hier nur separate Rechnersysteme schaffen, was meistens teurer und auch Aufwendiger wird.

Eine weitere Möglichkeit der Optimierung der Echtzeitleistung besteht in der Wahl verschiedener Stufen des Komforts der

Ein-/Ausgabesteuerung. Es muß hier immer ein Kompromiß zwischen Programmieraufwand und System-overhead gefunden werden.

Am komfortabelsten für den Entwickler ist natürlich der RMS-Dienst, der völlig geräteunabhängig die Verwaltung der Massenspeicher und sonstigen Ein-/Ausgabe Schnittstellen übernimmt. Die nächste Stufe ist der QIO-Systemservice, der schon speziellere Kenntnisse erfordert, gefolgt vom Gerätetreiber bei dem meistens eine Programmierung in Assembler-Sprache erforderlich ist und schließlich der direkte Registerzugriff, der den geringsten "Overhead" benötigt, aber bei fehlerhafter Programmierung auch zu katastrophalen Ergebnissen führen kann.

Bei Multitasking-Systemen sind Mittel zur Kommunikation und Synchronisation mehrerer Tasks nötig zur Steigerung der Effizienz und Geschwindigkeit des Gesamtprogramms.

Es gibt davon in VMS vier Arten:

ASTs (Asynchronous System Traps) zur schnellsten Reaktion auf externe Ereignisse. ASTs reagieren auf ein externes Ereignis, das einen Interrupt erzeugte, durch Ausführung einer vom Benutzer geschriebenen Interrupt Service Routine.

Event Flags, die zur Meldung synchroner oder asynchroner Ereignisse dienen können. Sie werden mit Standarddiensten des Betriebssystems bearbeitet und haben deshalb längere Antwortzeiten als ASTs.

Nachricht-Systeme, sog. Mailboxen, können vom Betriebssystem oder vom Benutzer verwaltet werden und sind wiederum langsamer als Event Flags.

Schließlich gibt es noch die gemeinsamen Bereiche in

Haupt- und Massenspeicher, die von mehreren Tasks beschrieben und gelesen werden können. Die Zugriffsteuerung muß durch Systemdienste wie Event Flags und Lock Management geschehen.

Die Optimierung einer Echtzeitanwendung ist letztlich eine Aufgabe des Endbenutzers. Hier spielt es nun eine große Rolle, welche Wahlmöglichkeiten der Benutzer hat um die maximale Leistung seines Systems zu erreichen. Das Betriebssystem VMS gibt hier dem Anwender maximale Flexibilität.
Zum Beispiel können Systemdienste in bestimmten Bereichen einer Anwendung gewählt werden, wo universalität gefordert is und in anderen Bereichen kann durch spezielle dedizierte Routinen Verwaltungsaufwand eingespart werden. Außerdem erlaubt VMS das festhalten bestimmter Tasks im Hauptspeicher oder sogar das Ausschalten des "paging" um spezielle Vorteile dieser Arbeitsweise auszunutzen und den Overhead zu minimieren.

Nachdem man eine Auswahl der CPU und eines geeigneten Bussystems getroffen hat, muß man sich Gedanken über die Eigenschaften der Ein-/Ausgabeschnittstelle machen, da diese hohen Einfluß auf die erreichbare Datenrate hat.

So ist zu erwägen ob programmgesteuerte Ein-/Ausgabe oder DMA-fähigkeit gefordert sind. DMA-fähige Schnittstellen können, wie bereits erwähnt, Geschwindigkeitssteigerungen um mehr als eine Größenordnung erreichen. Während z.B. bei Programmgesteuerter Ein-/Ausgabe die erreichbaren Datenraten mit A/D-Wandler zwischen 1 und 5kHz liegen, kann ein DMA-fähiger Wandler zwischen 50 und mehreren hundert kHz erreichen.

Der Einsatz einer DMA-fähigen Schnittstelle ist jedoch nur dann sinnvoll wenn die Applikation die Übertragung von relativ großen Datenblöcken ermöglicht. In Fällen wo nur ein

paar Bytes pro Operation transferiert werden, ist die DMA-fähigkeit von geringem oder gar keinem Vorteil, ja sie kann sogar unwirtschaftlich sein, da die Hardware natürlich teurer ist.

Die Art der von einer Schnittstelle unterstützten Ein-/Ausgabeoperation kann die maximale Geschwindigkeit unabhängig vom eingesetzten Rechnersystem bestimmen. So lassen sich zum Beispiel A/D-Wandlungen mit hoher Auflösung oder weitem Dynamikbereich nicht mit derselben Geschwindigkeit wie niedriger auflösende Konversionen erreichen, ohne einen entsprechenden Preis für die Schnittstelle.

Weiterhin ist von Bedeutung ob eine Karte für Ein- und Ausgabeoperationen verwendbar ist und ob sie beide Arten gleichzeitig unterstützt. Beim gleichzeitigen Ausführen von Ein- und Ausgabe ist der Datendurchsatz fast immer geringer als beim Transfer in nur einer Richtung. Der Grund hierfür ist die Software-Verwaltung.

Einen starken Einfluß hat die Größe des Pufferspeichers der verwendeten Schnittstelle. Solcher Puffer ist meistens der entscheidende Faktor der kontinuierlich hohe Datenraten erst ermöglicht. Der Puffer, ein sogenannter FIFO Speicher, wird zum zeitweisen speichern von Eingabewerten benutzt, während der Systembus besetzt oder die CPU nicht verfügbar ist.

Alle diese Faktoren sind zu berücksichtigen, um eine Echtzeitapplikation mit maximaler Leistung auf einem Rechnersystem einsetzen zu können.
Bei geeigneter Abstimmung der entscheidenden Faktoren ist jedoch auf VAX-Systemen sogar eine höhere Durchsatzrate zu erreichen als auf leistungsfähigen 16-Bit Rechnern.
Zu einer Echtzeitanwednung gehören jedoch außer der reinen Erfassung auch die Programmentwicklung und die Auswertung, die zum Teil bis zu 70% des Gesamtaufwandes ausmachen können.

Auf diesem Gebiet ist die Softwareunterstützung in VMS eindeutig den 16-Bit Rechnern überlegen.

Implementierung von PEARL auf IBM PC unter PC-DOS

Erwin Kneuer, Werum GmbH
Glogauer Straße 2 A, 2120 Lüneburg

Zusammenfassung

Mit der steigenden Leistungsfähigkeit von Personal-Computern haben sich auch deren Einsatzgebiete erweitert. In zunehmendem Maße werden PCs auch in Echtzeitanwendungen eingesetzt, für deren Programmierung die Sprache PEARL besonders geeignet ist.
Die Firma Werum bietet bereits seit einigen Jahren ein PEARL-Programmiersystem /4/ auf IBM PCs unter dem Echtzeit-Betriebssystem AT/RTX an. Die Nachfrage nach diesem System war erstaunlich groß, der insgesamt hohe Preis verhinderte jedoch eine entsprechende Verbreitung. Außerdem hat IBM im März dieses Jahres eine neue PC-Familie, das System PS/2, mit dem multitasking-fähigen Betriebssystem OS/2 angekündigt, das ein gemeinsames Programmierinterface zu PC-DOS (ab Version 3.3) besitzt.

Aus diesen Gründen beschloß Werum, PEARL "auf dem Wege zu OS/2" unter PC-DOS zu implementieren. Im folgenden Beitrag wird diese Implementierung vorgestellt.

1. Implementierter Sprachumfang

Auch für die PC-Implementierung unter DOS wurde das von der Firma Werum GmbH, Lüneburg, entwickelte portable PEARL-Programmiersystem portiert. Damit steht unter DOS der gleiche Sprachumfang zur Verfügung wie für die PEARL- Implementierungen von Werum auf den folgenden Rechnersystemen:

- Apollo Domain Workstation mit UNIX
- Hewlett Packard HP 9000-3xx mit HP-UX
- Hewlett Packard HP 3000 mit MPE IV
- IBM PC AT mit AT/RTX
- IBM 43xx mit CMS
- intel-Systeme mit iRMX86 oder iRMX286
- Motorola 68000 mit VERSADOS
- Norsk Data NORD 100 mit SINTRAN
- PCS CADMUS 9000-Serie mit MUNIX
- PEARL Engine 68000 mit BAPAS-K
- Siemens PC16-20 mit FLEXOS
- Siemens R/M-Serie mit ORG 300 PV / ORG-M
- Siemens 7.xxx mit BS 2000
- SUN Workstation mit UNIX
- VAX 11/7xx mit VMS
- Zilog Z 8000 mit BAPAS-K (Compilierung auf VAX 11/7xx)

Konkret umfaßt der Sprachumfang Basic PEARL /2/ mit mächtigen Erweiterungen in Richtung Full PEARL /3/, die hauptsächlich aus den folgenden Sprachkonstrukten bestehen:

- Benutzerdefinierte Datentypen (TYPE)
- Referenzen (REF)
- BOLT als weiterer Datentyp für Synchronisationsvariablen
- Definition von neuen Operatoren (OPERATOR)
- Felder und Strukturen bzw. benutzerdefinierte Datentypen als Komponenten in STRUCT- bzw. TYPE-Definitionen
- Lokale Prozeduren in Tasks bzw. Prozeduren
- Listen von SEMA- bzw. BOLT-Variablen in Synchronisationsanweisungen
- Variable CHARACTER- bzw. BIT-Selektion.

Der genaue Sprachumfang ist in dem Buch "Introduction to PEARL" /5/ beschrieben.

2. DOS und Echtzeit, Einschränkungen hinsichtlich Multitasking

Das Betriebssystem DOS wurde für single-user - single-task - Betrieb entworfen und ist eigentlich damit als Host-Betriebssystem für eine Echtzeitsprache ungeeignet. Wenn aber ein Programm, das unter DOS abläuft, aus einem Betriebssystem mit Multitaskingfähigkeiten und seinen Anwendungstasks besteht, so gibt es solange keinerlei Einschränkungen im Echtzeitverhalten wie keine DOS-Funktion benutzt wird.
In der in diesem Beitrag vorgestellten Implementierung wird DOS für die Ein- und Ausgabe mit den Standard-Geräten benutzt. Die Benutzung einer DOS-Funktion, was auf Anwenderebene

E/A-Operationen mit Standardgeräten entspricht, hat dann zur Folge, daß solange kein Taskwechsel in der Anwendung stattfindet, bis DOS wieder verlassen wurde. Hinzu kommt, daß prozessorfreie Zeit, die während der Ein-/Ausgabe-Operation im Gerätetreiber anfällt, nicht genutzt werden kann. Für schnelle E/A-Vorgänge im Millisekundenbereich, z.B. E/A mit Harddisk oder Console-Ausgabe, spielt dies kaum eine Rolle; es kann hier sogar durch die Einfachheit des Betriebssystems zu Performance-Vorteilen gegenüber Echtzeit-Betriebssystemen kommen. Der Nachteil wird natürlich umso größer, je länger eine E/A-Operation dauert. Fatal wird dies bei Eingabe vom Terminal, wo die Dauer der Operation davon abhängt, wie schnell eine Keyboard-Taste gedrückt wird. Für Eingabe von der Console gilt diese Einschränkung allerdings nicht, da DOS die Möglichkeit bietet, den Console-Eingabe-Buffer auf vorhandene Eingabe abzufragen. Eine PEARL-Task kann also auf Eingabe von der Console warten, während andere Tasks der Anwendung ausgeführt werden.

Weitere Einschränkungen hat diese PEARL-Implementierung gegenüber anderen unter Host-Betriebssystemen mit Echtzeiteigenschaften nicht. Auch PEARL-Schedules werden, basierend auf einem 18 ms Takt, den die batteriegepufferte Uhr des PC's liefert, zeitgerecht aktiviert, wenn zu diesem Zeitpunkt ein Taskwechsel erlaubt ist.

Eine zeitkritische Prozeßdatenerfassung (nicht Verarbeitung) wird beim Einsatz von interruptgesteuerten und DOS-unabhängigen Treibern von dem PEARL-System durch die Bereitstellung einer Treiberschnittstelle, auf die in diesem Beitrag noch näher eingegangen wird, unterstützt.

3. Implementierung

3.1 Hardware- und Software-Voraussetzungen zur Programmentwicklung

Zur Compilierung von PEARL-Moduln muß der PC mindestens die folgenden Betriebsmittel bereitstellen:

- 512 KB Speicher
- 10 MB Festplatte
- Arithmetischer Coprozessor
- PC-DOS oder MS-DOS ab Version 3.00
- Microsoft Assembler ab Version 2.00
- Console.

Zur Ausführung von PEARL-Programmen wird mindestens benötigt:

- Arithmetischer Coprozessor
- PC-DOS oder MS-DOS ab Version 3.00
- Console

Der Speicher hängt von der Größe der Anwendung ab, das PEARL-Laufzeitsystem benötigt maximal 50 KB.

3.2 Portierung

Das portable PEARL-Programmiersystem von Werum besteht aus folgenden Komponenten:

- PEARL-Compiler Front End
- PEARL-Codegeneratoren für eine Vielzahl von Rechnerarchitekturen, die C, Assembler oder Objectcode erzeugen
- PEARL-Betriebssystem BAPAS-K
- PEARL-E/A-Routinen BAPAS-EA

Alle hier aufgeführten Komponenten sind in PEARL programmiert und deshalb mit einem geeigneten Codegenerator sehr einfach auf ein beliebiges Zielsystem zu portieren. Für diese Implementierung konnte auf einen bereits existierenden Codegenerator (aus der INTEL-Implementierung), der Code für den 8086-Prozessor erzeugt, zurückgegriffen werden.
Für eine vollständige Implementierung waren dann noch folgende Interfaces zu erstellen:

- E/A-Interface für den PEARL-Compiler
- E/A-Interface als Treiber für die PEARL-E/A-Anweisungen
- BAPAS-K-Interface wie Timer, Taskwechselroutine usw.

Außerdem benötigen PEARL-Programme noch eine Laufzeitunterstützung für bestimmte Sprachkonstrukte wie z.B. CHARACTER- und BIT-Selektion. Hier konnte auch auf einen Assembler-Modul aus der INTEL-Implementierung zurückgegriffen werden.

3.3 Eigenschaften von PEARL-DOS

Der Compiler übersetzt einen PEARL-Modul in einen Assembler-Modul, der dann noch assembliert werden muß, um einen linkbaren Objekt-Modul zu erhalten. Im erzeugten Assembler-Modul werden als Kommentar Hinweise auf die Ursprungzeile im PEARL-Modul eingefügt. Im Benutzerhandbuch ist außerdem die PEARL-Laufzeitorganisation genau beschrieben, so daß leicht in Assembler oder auch anderen Sprachen geschriebene Module zu einem PEARL-Programm hinzugefügt werden können.

Steuerbar durch Compiler-Parameter, wird ein Quell-Listing und/oder eine Cross-Referenzliste erzeugt. Zudem kann die Laufzeit-Überwachung von Feldgrenzen und Referenzen durch Compiler-Parameter ein- oder ausgeschaltet werden.

Ein Preprozessor gestattet das Einfügen von Programmtexten aus Files (%INCLUDE) und bedingte Compilierung (%IF). Damit können zum Beispiel in mehreren Moduln verwendete Schnittstellen aus einer getrennten Textdatei zur Übersetzungszeit textuell einkopiert werden, so daß die Einhaltung der Modulschnittstellen implizit gewährleistet wird.

Mit Hilfe des DOS-Programms LINK und einer PEARL-spezifischen Library kann dann ein ablauffähiges Programm erzeugt werden, das wie jedes andere DOS-Programm gestartet wird.
Der Anwender kann die Stackgröße jeder Task in einfacher Weise festlegen, ansonsten wird vom PEARL-Laufzeitsystem ein Default-Wert angenommen. Stack-Overflow wird vom Laufzeitsystem erkannt und gemeldet.

PEARL-DOS unterstützt alle DOS-Standardgeräte wie

- Console (CON)
- Serial Adapter (COM1, COM2)
- Parallel Adapter (LPT1, LPT2, LPT3)
- Harddisk und Floppy (A:, B:, C:, D:)

Um ein laufendes Programm jederzeit unterbrechen und beliebige Aktionen starten zu können, wurde ein sogenannter Software-Interrupt implementiert, der mit Hilfe von CONTROL-C und der Eingabe einer Zahl von der Console aus getriggert werden kann.

Beispiel:

```
    MODULE (MAIN);

    SYSTEM;
        SWI2:   SOFTIN (2) * 0;

    PROBLEM;

STRT:   TASK;
    WHEN SWI2 ACTIVATE DIALOG;
        .
        .
        .
    END;

DIALOG: TASK PRIO 10;
        .
        .
        .
    END;
    .
    .
    .
```

Während des Ablaufs der Applikation kann die Task DIALOG durch Eingabe von CONTROL-C und der Zahl 2 an der Console aktiviert werden.

3.4 Die offene Treiberschnittstelle für Prozeß-E/A

Für den IBM PC gibt es bereits eine Vielzahl von Interface-Karten zur Meßdatenverarbeitung mit steigender Tendenz. Es ist deshalb für Implementatoren von höheren Programmiersprachen unmöglich, Treiber für alle Interface-Karten anzubieten. Außerdem gibt es viele Anwender, die eigene Treiber oder sogar Interface-Karten einsetzen möchten. Deshalb stellt diese PEARL-Implementierung von Werum eine Schnittstelle zur Verfügung, die es erlaubt,

beliebige Treiber in das PEARL-Programmiersystem zu integrieren und mit den PEARL-Anweisungen TAKE / SEND anzusprechen.
Diese Schnittstelle ist im Detail im Benutzerhandbuch des PEARL-Programmiersystems für DOS beschrieben, deshalb wird in diesem Beitrag nur das Prinzip dargestellt.

Die Funktionen eines interrupt-gesteuerten Treibers für Prozeß-E/A lassen sich wie folgt charakterisieren:

1. Treiber-Start

   - Treiber-spezifische Variablen und Buffer initialisieren
   - Interrupt-Vektor für Interrupt-Handler setzen
   - Interface-Karte initialisieren.

2. Treiber-E/A

   - Interface-Karte entsprechend E/A-Auftrag programmieren
   - Laufende PEARL-Task suspendieren und Prozessor für andere lauffähige Task freigeben.

3. Treiber-Ende

   - Interface-Karte deaktivieren
   - Interrupt-Vektor zurücksetzen.

4. Interrupt-Handler

   - Meßwert-Übernahme
   - Wenn gesamter E/A-Auftrag schon bearbeitet,
     dann: suspendierte PEARL-Task fortsetzen
     sonst: Interface-Karte erneut programmieren.

Für die Funktionen

- Laufende PEARL-Task suspendieren und
- Suspendierte PEARL-Task wieder fortsetzen

stellt das PEARL-Programmiersystem Routinen zur Verfügung, die aus Treibern heraus aufgerufen werden können.

Für den Data Acquisition and Control Adapter von IBM (1 Interface-Karte mit 2 analogen Eingabe- und 4 analogen Ausgabekanälen sowie je einem 16 Bit breiten binären Eingabe- und Ausgabekanal) stellt Werum einen Treiber zur Verfügung.

Im folgenden Beispiel ist die Benutzung dieses Treibers aus Anwendersicht kurz dargestellt.

Beispiel 1: Es sollen mit einer TAKE-Anwendung jeweils 100 Werte von den Analog-Kanälen 0 und 1 mit einer Abtastrate von 1 KHz eingelesen werden.

```
.
.
.
SYSTEM;
      ANALOG_IN:   DACAAI  (0)  * 0  * 1 ;
```

Ende Kanalbereich Kanal 1

Anfang Kanalbereich Kanal 0

Adapternummer

Systemname für Analogeingabe

```
.
.
.
TYPE   T_ANALOG_MW   STRUCT  [ TAB (100)  STRUCT  [ Kanal_0  FIXED,
                                                    Kanal_1  FIXED ] ];

DCL      ANALOG_MW   T_ANALOG_MW ;
.


.
TAKE     ANALOG_MW   FROM ANALOG_IN
     BY  CONTROL (200, 1000) ;
```

Abtastrate 1 KHz

Anzahl Eingaben 100 * Kanal 0 + 100 * Kanal 1

Beispiel 2: Ein Gerät kann über den binären Ausgabe-Kanal ein- bzw. ausgeschaltet werden. Über den binären Eingabe-Kanal erfolgt eine Zustandsanzeige, die Ein, Aus oder Änderung (Wechsel von einem Zustand in den anderen, wozu das Gerät einige Zeit benötigt) signalisiert.

```
SYSTEM;
     DEVICE_SET:     DACABO (0) * 0 * 7, 1 ;    /* Bit 7 */
                                    |  |
                                    |  Anzahl Bits
                                    Start-Bit
     DEVICE_STATE:   DACABI (0) * 0 * 4, 2 ;  /* Bit 4 und 5 */
   .
   .
   .
   .
DCL  DEVICE_START    BIT (1)    INIT   ('1'B) ,
     DEVICE_STOP     BIT (1)    INIT   ('0'B) ;

DCL  DEVICE_STATUS   BIT (2) ;

   .
   .
   .
   .
SEND  DEVICE_START TO  DEVICE_SET ;
   .
   .
   .
   .
TAKE  DEVICE_STATUS FROM DEVICE_STATE ;
   .
   .
   .
   .
```

4. Charakterisierung des verfügbaren PEARL-Testsystems

Für den interaktiven Test von PEARL-Programmen auf Sprachebene hat Werum einen portablen, in PEARL programmierten Debugger entwickelt /1/, der auch unter DOS verfügbar ist. Die Kommunikation mit dem Debugger kann über die Console (CON) oder COM1 erfolgen; dies ist besonders hilfreich, wenn eine Anwendung umfangreiche Dialoge mit der Console durchführt.
Dieser Debugger bietet für den Einzeltest von Programmkomponenten komfortable Display-, Trace- und Unterbrechungsmöglichkeiten (Breakpoints):

```
DISPLAY Mess.Wert
Mess-Wert := '1101'B
LINE TRACE FROM 55 TO 75 ON
CALL TRACE IN Erfassung OFF
BREAK ON 457; HALT; END
BREAK ON ENTRY Regelung; DISPLAY Mess; END
```

Das letzte Beispiel zeigt einen Breakpoint auf den Eingang der Prozedur Regelung mit einem Schnappschuß der Variablen Mess, ohne daß der Debugger in den Dialogbetrieb übergeht, wie es beim vorletzten Beispiel durch das HALT-Kommando verlangt wird.

Über diese "sequentiellen" Möglichkeiten hinaus unterstützt der Debugger auch den Integrations- und Gesamttest durch die Möglichkeiten der Diagnose und Veränderung der Zustände von Tasks und Synchronisationsvariablen sowie durch Schnittstellen zur Simulation der Prozeßperipherie:

```
STATUS TASK
STATUS TASK Erfassung
ACTIVATE Erfassung
STATUS SEMA Puffer_Sema
RELEASE Puffer_Sema
```

Durch die STATUS-Kommandos kann der Anwender gewissermaßen auf PEARL-Sprachebene in das PEARL-Betriebssystem hineinschauen. Z.B. erhält er als Antwort auf das Kommando STATUS TASK Erfassung folgende Informationen im Klartext ausgegeben:

- Aktueller Zustand der Task Erfassung
  (nicht aktiv, aktiv, laufend, blockiert, suspendiert, wartend auf I/O)
- Gegebenenfalls Zeitpunkt der Aktivierung, Blockierung, Suspendierung
- Verursacher der Aktivierung, Blockierung, Suspendierung
- Zur Ausführung anstehende Anweisung mit Rückverfolgung bei (geschachtelten) Prozeduraufrufen.

Der Anwender kann nun den Zustand einer Task interaktiv mit den PEARL-Tasking- und Synchronisier-Anweisungen ändern (ACTIVATE, TERMINATE, SUSPEND, CONTINUE, PREVENT, REQUEST, RELEASE, ENTER, LEAVE, RESERVE, FREE).

5. Ausblick, PEARL unter OS/2

Sobald OS/2 auf dem Markt verfügbar ist, wird Werum sein portables PEARL-Programmiersystem auch unter OS/2 implementieren. Da OS/2 ein Echtzeit-Betriebssystem ist, fallen die in Kapitel 2 aufgeführten Einschränkungen natürlich weg und PEARL wird, wie in der Sprachdefinition angegeben, verfügbar sein.
Unter OS/2 können bis zu 16 Programme (jedes Programm kann wiederum mehrere Tasks (Threads) enthalten) gleichzeitig aktiv sein. Ein PEARL-Programm wird wie ein normales OS/2-Programm gestartet. Das in OS/2 vorhandene PIPE-Konzept wird direkt aus PEARL heraus benutzbar sein, so daß verschiedene PEARL-Programme über PIPES miteinander kommunizieren können.
Natürlich wird auch der PEARL-Debugger verfügbar sein sowie das Echtzeit-Datenbanksystem BAPAS-DB, das über eine SQL-Schnittstelle ansprechbar sein wird.
In etwa 1 bis 2 Monaten nach Verfügbarkeit von OS/2 werden das PEARL-Programmiersystem sowie BAPAS-DB verfügbar sein.

Literatur

/1/ Brunner, P.J.; Meffert, K.; Windauer, H.:
PEARL-Testsystem. PEARL Rundschau, Band 2, Nr. 6, Dezember 1981, S. 8 – 13.

/2/ DIN 66253, Teil 1:
Programmiersprache PEARL. Basic PEARL. Beuth Verlag, Berlin 1981.

/3/ DIN 66253, Teil 2:
Programmiersprache PEARL. Full PEARL. Beuth Verlag, Berlin 1982.

/4/ Kneuer, E.:
PEARL-PC – Ein Echtzeit-Programmiersystem für IBM PCs. Workshop Personal Realtime Computing, München-Neubiberg, September 1986.

/5/ Werum, W.; Windauer, H.:
Introduction to PEARL. Vieweg 1985, 3. Auflage.