Egon Sommer

*Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen*

Egon Sommer

# *Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen*

Herausgegeben von
Professor Dr.-Ing. Klaus Feldmann,
Lehrstuhl für
Fertigungsautomatisierung und Produktionssystematik
**FAPS**

HANSER

Carl Hanser Verlag München Wien

Als Dissertation genehmigt von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

| | |
|---|---|
| Tag der Einreichung: | 31. 05. 1991 |
| Tag der Promotion: | 29. 07. 1991 |
| Dekan: | Prof. Dr.-Ing. habil. G. Kuhn |
| Berichterstatter: | Prof. Dr.-Ing. K. Feldmann |
| | apl. Prof. Dr.-Ing. habil. W. Bär |

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

**Sommer, Egon:**
Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen / Egon Sommer. - München; Wien: Hanser, 1992
(Fertigungstechnik - Erlangen; 21)
Zugl.: Erlangen, Nürnberg, Univ., Diss., 1991
ISBN 3-446-17062-6


Herstellung: Copy Center 2000, Erlangen-Eltersdorf
Printed in Germany

# Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Universität Erlangen - Nürnberg.

Herrn Professor Dr.-Ing. K. Feldmann, dem Leiter des Lehrstuhls für Fertigungsautomatisierung und Produktionssystematik am Institut für Fertigungstechnik, danke ich für die wohlwollende Förderung bei der Durchführung dieser Arbeit.

Herrn apl. Professor Dr.-Ing. habil. W. Bär, Lehrstuhl für Regelungstechnik der Universität Erlangen-Nürnberg, danke ich für die Übernahme des Korreferates und für die Unterstützung bei der Betreuung von Studien- und Diplomarbeiten am Institut für Elektrotechnik.

Ferner gilt mein Dank allen Kolleginnen und Kollegen, speziell jenen der Forschungsgruppe für Steuerungen und Sensoren ('SUS'-Gruppe), für die ständige Diskussionsbereitschaft. Stellvertretend für die Unterstützung seitens des Instituts für mathematische Maschinen und Datenverarbeitung danke ich der Forschungsgruppe E für die Hilfe bei der Realisierung von Feldbuslösungen.

Ein besonderer Dank gebührt allen Studenten und wissenschaftlichen Hilfskräften für ihren engagierten Einsatz ( inbesondere Nachtschichten ) bei der Umsetzung meiner Konzepte in einer Modellanlage. Ebenso danke ich allen Beteiligten für ihre Hilfe bei der Fertigstellung dieser Arbeit.

Egon Sommer

# Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen

## - Inhaltsübersicht -

# 1 Einführung

Die fortschreitende Automatisierung verlangt nach immer leistungsfähigeren und flexibleren Komponenten. In vielen Anwendungsfeldern wandelt sich das Einsatzgebiet der mechanisierten Montage zu immer höherwertigeren und komplexeren Fertigungsaufgaben, nachdem viele Industriebereiche bereits mit Einfachgeräten ausgerüstet sind. Die Rechnerintegration muß mit dieser Entwicklung Schritt halten /96/. Gerade im Bereich der Montage ergibt sich ein hohes Anforderungspotential, da nicht nur der Fertigungsprozeß selbst meist schwer beherrschbar ist, sondern zudem auch eine große Varianz im Produktspektrum besteht. Eine Vorreiterrolle in der Montageautomatisierung spielte schon früh die Elektronikproduktion /28/, da hier der eigentliche Montagevorgang vergleichsweise einfach ist.

Andere Fügeaufgaben, wie zum Beispiel das Bahnschweißen oder das Bolzen-Loch-Problem stellen bei der automatischen Durchführung komplexe Anforderungen an die Steuerungen und deren Leistungsumfang zur Führung von Montagegerät und zugehörigem Prozeß. Hieraus resultiert die Notwendigkeit, leistungsfähige Bewegungsfunktionen mit umfassendem Sensoreinsatz zu kombinieren. Die große Zahl von Produktvarianten bedingt leistungsfähige dispositive Funktionen zur Beherrschung der Materialverwaltung. Dazu ist eine enge Anbindung an übergeordnete Steuerungsinstanzen unumgänglich /32/. Gerade im Bereich der Integration von Gerätesteuerungen in eine CIM-Hierarchie gibt es sehr viele Entwicklungsbemühungen /52/, die jedoch bislang noch nicht zu breiter Anwendung oder Standardisierung geführt haben.

Die Steuerungen von Montagegeräten befinden sich in einem Spannungsfeld zwischen der Forderung nach hoher Rechenleistung auf der Ebene der Bewegungsführung und der Forderung nach Integrationsmöglichkeiten in ein rechnergeführtes Gesamtsystem. Da die benötigte Leistungsfähigkeit das Vermögen heutiger Mikroprozessoren übersteigt, sind effiziente und funktional reduzierte Softwaresysteme in Maschinensprache notwendig. Die Rechneraufgaben sind zusätzlich gekennzeichnet durch strenge Echtzeitbedingungen.

Beim Aufbau einer rechnerintegrierten Fertigung hat sich ein hierarchischer Ansatz etabliert. Ebenso setzt sich auf der operativen Ebene die Zelle als Strukturierungsmerkmal durch /24,37/. Der Zellenrechner hat dabei neben seinen primären Verwaltungs- und Steuerungsaufgaben auch dafür zu sorgen, daß die unter-

schiedliche Funktionalität der Einzelsteuerungen kompensiert wird (Bild 1.1, hinterlegter Bereich). Durch Punkt-zu-Punkt-Verbindungen zu jedem einzelnen Gerät ist er zudem mit den unterschiedlichen Kommunikationsmitteln und Protokollen belastet. Ein Großteil der verfügbaren Rechenkapazität geht durch diese Anpassungen verloren, und sie verlangen häufig nach Sonderlösungen im Zellenrechner in Form von Zusatzausstattungen.

Die derzeitige Situation bei der Integration von Montagegeräten in die CIM-Umgebung ist wesentlich durch drei Schwachstellen gekennzeichnet. Zum einen ist hier die fehlende Leistungsfähigkeit der Steuerung selbst zu nennen. Eng gekoppelt damit sind zweitens fehlende Funktionalität und nicht zuletzt mangelnde Kommunikationsmöglichkeiten. Ein Eingriff in die Basissoftware (Firmware) zur Erweiterung von Funktionen oder Einbindung von Hardwarezusätzen ist von Herstellern nicht vorgesehen und deshalb kaum möglich.

Für flexible Anlagen ist nicht nur die Art der Verbindung zwischen den Geräten relevant; mitentscheidend sind auch die zu transportierenden Daten. Über die heutige Standardverbindung (binäre Signalleitungen) können nur wenige, genau festgelegte Zustandsinformationen oder Kommandos ausgetauscht werden. Dies entspricht in keinster Weise den Forderungen, die in einer rechnerintegrierten Fertigung bestehen. Rechnerintegriert bedeutet aber, daß sich gerade auch eine Montagezelle als verteiltes Rechnersystem betreiben läßt und so freier Zugriff zu Informationen in allen Geräten besteht. Für eine systemtechnische Sichtweise und eine fertigungsbegleitende Qualitätssicherung ist die durchgängige Verknüpfung aller Steuerungselemente die entscheidende Voraussetzung. Eine Gerätesteuerung muß somit erweiterte Möglichkeiten zur Koordination mit anderen Komponenten einer Montageanlage besitzen. Die Integration aller Aktoren und Sensoren einer Zelle soll anwendungsbezogen und transparent möglich sein.

In Zukunft haben heute eingesetzte Steuerungen mit ihrem festen, funktionsgleichen Programmumfang keinen Platz mehr. Auch die verwendeten Sprachen für die Programmierung eines Roboters sind nicht hinreichend mächtig /10/. Verglichen mit Standardsprachen bewegen sich viele noch nicht auf Hochsprachenniveau /39/. Neuere Entwicklungen nähern sich dem Niveau gängiger Sprachen an /34/, bieten jedoch meist kaum Unterstützung für parallel ablaufende Prozesse.

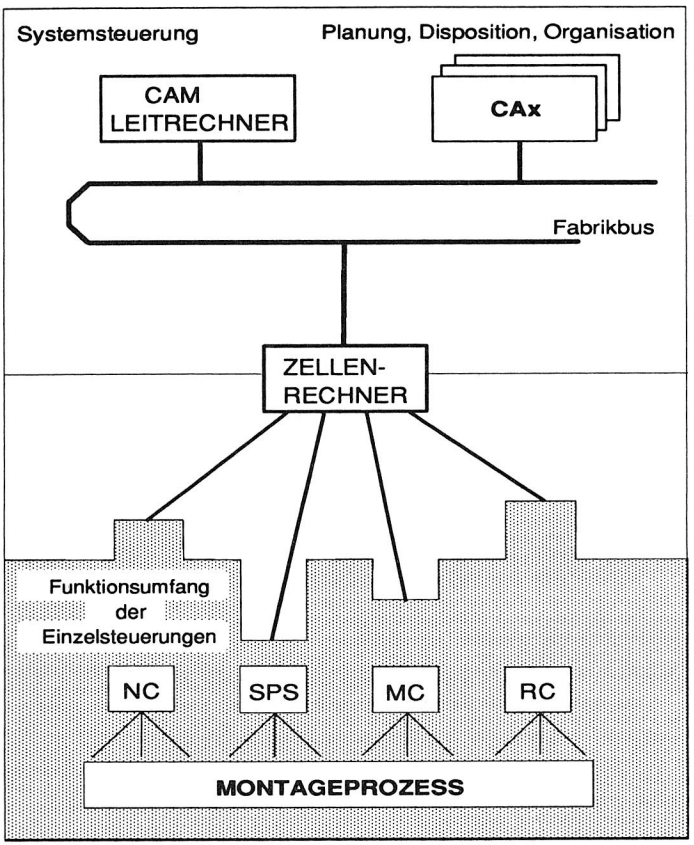



Bild 1.1 : Struktur der Fertigungssteuerung

Waren bislang Montageaufgaben mit nur einem einzelnen Handhabungsgerät zu lösen, so wird es in Zukunft immer mehr Anwendungen geben, die eine enge Koordination oder sogar Kooperation von mehreren Montagegeräten erfordern /30/. Die Möglichkeiten der Kommunikation mit anderen Elementen einer Montagezelle bedarf demnach ebenso einer Weiterentwicklung. Eine ähnliche Funktionalität wird benötigt für die schnelle Berücksichtigung von höherwertiger Sensorinformation. Das Montagegerät kann also auch aus fertigungstechnischer Sicht nicht isoliert behandelt werden. Es ist vielmehr als integraler Bestandteil der gesamten Montageumgebung zu sehen.

Das Fernziel beim Betrieb von Montagezellen und -geräten läßt sich in folgende Stichpunkte fassen:

- arbeitsplanbasierte Eingabe von Aufträgen,
- selbständige Planung und Ausführung von Operationen,
- Berücksichtigung von Umgebungsinformationen (Sensorik, benachbarte Geräte),
- schnelle und gesicherte Ausführung von Fügeoperationen,
- vollständige Integration in die CIM-Hierarchie.

Auf dem Weg zu diesen Zielvorgaben gilt es zunächst einmal, Teilschritte zu unternehmen. Speziell auf der Ebene der Gerätesteuerungen muß ein Ansatz gefunden werden, der Funktionalitäten aus dem Bereich der Standardcomputer mit der Forderung der Echtzeitverarbeitung von Steuerinformationen bei gleichzeitiger Berücksichtigung von Sensordaten verbindet. Das gewählte Konzept muß als Basis dienen können, um neue Funktionen, wie zum Beispiel den kooperierenden Betrieb von mehreren Geräten oder auch die Integration von Multisensorik zu gewährleisten.

Die vorliegende Arbeit befaßt sich mit dem Entwurf eines Steuerungssystems für Montagegeräte, das die angesprochenen Anforderungen erfüllt. Im Kern steht die Konzeption eines modularen und universellen Steuerungssystems, das die Integration von neuen Funktionen im verteilten System einer Montagezelle erlaubt. Anhand von ausgewählten Beispielen mit einem repräsentativen Anforderungsspektrum für künftige Steuerungsgenerationen wird die Leistungsfähigkeit des erstellten Konzepts in einer Modellanlage nachgewiesen.

Bevor jedoch mit dem Entwurf der notwendigen Strukturen auf Geräteebene begonnen werden kann, muß geklärt sein, welche Funktionalität die Elemente einer Zelle aufweisen müssen. Dazu ist eine hierarchische funktionale Gliederung der Montageaufgabe durchzuführen. Vor diesem Hintergrund gilt es, Anforderungen an Steuerungen auf der Geräteebene abzuklären. Diese lassen sich weiter präzisieren durch die Betrachtung von Anforderungen in der Montage, die speziell beim Betrieb von Robotern entstehen. Das zu erstellende Konzept muß die Betrachtung der Hardwarestruktur einer Steuerung, die Betriebssystemebene, die Kommunikationsanbindung und die objektorientierte Erweiterung auf das gesamte Steuerungssystem einer Zelle umfassen. Die Aspekte der Kommunikation in Montagezellen mittels Feldbus werden in dieser Arbeit nur teilweise behandelt, da

sich parallel laufende Aktivitäten ausschließlich mit dieser Thematik befassen /6/ und die Ergebnisse direkt übernommen werden können.

Zielsetzung der Arbeit ist also der Entwurf von neuen Funktionen zur Kooperation von Montagegeräten unter Einsatz von Sensoren auf der Basis eines allgemeinen Steuerungskonzepts für Montagezellen und einer modularen und universellen Multiprozessor-Steuerung.

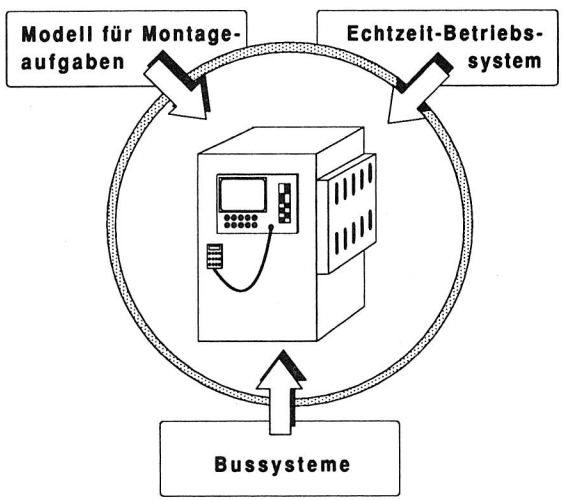



Bild 1.2: Werkzeuge zur Modularisierung von Steuerungen

Die notwendige Gliederung und Modularisierung läßt sich dabei mit drei Hauptwerkzeugen erreichen (Bild 1.2). Zur Strukturierung des Hardwareaufbaus bieten sich Standard-Bussysteme an. Ihre Eignung für den Einsatz in Robotersteuerungen gilt es zu bewerten. Die Analyse von Montageanlagen bedarf einer rein funktionalen Betrachtung aller Aufgaben und sollte zu einem abstrakten Modell für Montagezellen führen. Als Hilfsmittel für die Strukturierung der Steuerungssoftware (viele Einzelprogramme) muß ein leistungsfähiges Betriebssystem zum Einsatz kommen. Da es sich bei fast allen Aufgaben innerhalb einer Gerätesteuerung um zeitkritische Programme handelt, bedarf es besonderer Beachtung der Echtzeitaspekte. Die Entwurfskriterien für Steuerungen müssen sich an den Zeitrestriktionen orientieren und unterscheiden sich deshalb von Konzepten für Standardrechner.

# 2 Charakterisierung von Echtzeit-Betriebssystemen

Die Entwicklung im Bereich der Robotersteuerungen und deren Programmierung muß sich den Standardanwendungen annähern, jedoch mit dem großen Unterschied, daß die zeitlichen Randbedingungen eine große Rolle spielen. Es ist dennoch wünschenswert, daß sich Programmodule integrieren lassen, die für andere Anwendungen schon einmal erstellt wurden. Die Wiederverwendbarkeit von Software hat als Voraussetzung den Einsatz eines Betriebssystems und höherer Programmiersprachen.

Ein Forschungsschwerpunkt liegt derzeit auf der Entwicklung von Echtzeitsystemen (real time systems), die sich durch ihre Multiprogrammfähigkeit und schnelle Reaktionen auf externe Ereignisse auszeichnen. Sie sind prinzipiell dazu geeignet, Steuerungssysteme zu unterstützen. Maßgeblich ist die Fähigkeit, mehrere Programme (tasks) quasi gleichzeitig zu bearbeiten. Da viele Systeme den Begriff 'Echtzeit' im Titel führen und keine entsprechende Funktionalität bieten, hat sich der Begriff des 'harten Echtzeitsystems' (hard real time system) gebildet. Die Eigenschaft mancher Betriebssysteme, mehrere Benutzer gleichzeitig zu unterstützen (multiuser), ist in der Steuerungstechnik ohne Relevanz, da die zugehörigen Verwaltungs- und Schutzmechanismen nicht benötigt werden.

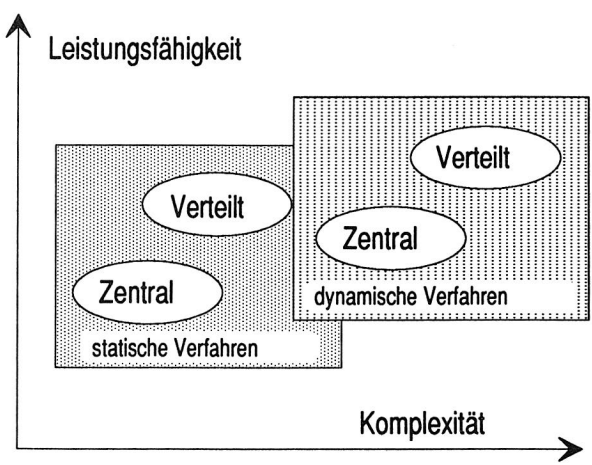



Bild 2.1: Echtzeit-Betriebssysteme

Im folgenden werden Systeme und Entwicklungen auf dem Echtzeitsektor betrachtet, um Rückschlüsse auf die Anwendbarkeit in der Robotik zu ziehen.

Die betrachteten Betriebssysteme zur Steuerung von Prozeßabläufen haben alle die Eigenschaft der Multitasking-Fähigkeit. Diese wird benötigt, um ein Gesamtsoftwaresystem modular aufbauen zu können. Bild 2.1 zeigt das Verhältnis von Komplexität und Leistungsfähigkeit beim Einsatz verschiedener Echtzeitsysteme. Man kann hierbei speziell für Multiprozessorsysteme zwischen verteilten und zentralen Systemansätzen unterscheiden. Die Komplexität steigt dabei sehr schnell an, ohne daß automatisch eine große Leistungssteigerung erreicht wird.

Ein Multitasking-Betriebssystem realisiert die quasiparallele Abarbeitung von Programmen durch Verfahren zur abwechselnden Zuteilung des Prozessors (Scheduling-Verfahren) an die einzelnen Tasks. Mit einem entsprechenden Algorithmus versucht das Betriebssystem normalerweise ein Gleichgewicht zwischen den einzelnen Tasks zu halten.

Im Echtzeitsystem sind jedoch hauptsächlich zeitliche Restriktionen Hauptaugenmerk von Optimierungen. Es lassen sich drei Hauptklassen für Zeitangaben identifizieren /19/.

- Maximum: Es darf zwischen zwei Ereignissen nicht mehr als eine bestimmte Zeit $t$ vergehen.
- Minimum: Zwischen zwei Ereignissen muß mindestens die Zeitspanne $t$ liegen.
- Dauer: Ein Ereignis oder Zustand muß für eine bestimmte Dauer $t$ bestehen.

Diese Grundtypen von zeitlichen Randbedingungen sind nicht zwangsweise exklusiv, sondern können auch in Kombination vorkommen. Es läßt sich ein allgemeiner Sprachentwurf durchführen /19/, der eine Formulierung der zeitlichen Restriktionen zuläßt.

Es gilt eine Unterscheidung zwischen 'harten' und 'weichen' Echtzeitsystemen zu machen. Eine exakte Definition läßt sich leider nicht finden, deshalb muß die Beschreibung an Eigenschaften oder Forderungen fixiert werden. Für ein hartes Echtzeitsystem müssen neben der Multitaskfähigkeit noch folgende Eigenschaften gefordert werden:

Ein 'hartes Echtzeitsystem' muß nicht nur das richtige Ergebnis einer Berechnung, sondern dieses auch noch in einer bestimmten, vorher festgelegten Zeit liefern /98/.

Mit dieser Forderung kann eine Trennung zwischen Echtzeit und "wirklichem" Echtzeitbetriebssystem versucht werden. Die harten Echtzeitsysteme haben ihre Einsatzgebiete in der Steuerungs- und Regelungstechnik, wo Reaktionszeiten im Milli- und Mikrosekundenbereich benötigt werden und diese Zeiten konstant sein müssen. Klassische Bereiche sind zum Beispiel Flugkontrollsysteme sowie Flugsimulatoren und Robotersysteme. Aufgrund des rasanten Geschwindigkeitsgewinns der Prozessoren sind Anwendungen, die vor fünf Jahren noch als Echtzeitanwendungen angesehen wurden, in ihrer Reaktionszeit für heutige Echtzeitanforderungen nicht mehr tragbar. Es läßt sich daher die Grenze zwischen Echtzeit- und 'harten' Echtzeitsystemen nicht bei einer bestimmten Zeitschranke ziehen.

Harte Echtzeitbetriebssysteme müssen

- schnell,
- vorhersagbar,
- zuverlässig und
- anpassungsfähig sein.

Wenn eine Task ihre Arbeit aufnimmt, muß gewährleistet sein, daß sie ihre Aufgabe in einer vorher bestimmten und garantierten Zeit erledigt, gleichgültig welche Randbedingungen (zum Beispiel Speicherplatz, Betriebsmittel, Systemlast) gerade bestehen. Diese Zuverlässigkeitsaspekte betreffen nicht nur das eingesetzte Betriebssystem, sondern alle verwendeten Hard- und Softwarekomponenten und ist in sicherheitsrelevanten Anwendungen unumgänglich.

Anpassungsfähig bedeutet, daß ein System den speziellen Anforderungen einer Aufgabe angepaßt werden kann, egal ob ein vorher einplanbares oder ein nicht vorhersehbares Ereignis (Fehler, Überlast) eintritt. Diese Forderung geht weit über das Stichwort flexibel hinaus. Flexibel bedeutet Anpassungsfähigkeit bei einplanbaren Ereignissen. Werden jedoch noch Fehler mit einbezogen, so läßt sich die Anpassungsfähigkeit nur durch softwaremäßig implementierte Maßnahmen erreichen. Echtzeitsysteme, die keinen harten Zeitanforderungen genügen, werden im folgenden als Multitasksystem bezeichnet.

## 2.1 Prozessor-Vergabestrategien

Im Kern eines Betriebssystems mit Multitaskingfähigkeiten steht die Art und Weise, mit der Tasks zur Bearbeitung durch den Prozessor eingeplant werden /13/. Für eine Schedulingstrategie gibt es prinzipiell folgende fünf Möglichkeiten zur Behandlung von Echtzeitanforderungen:

- Zeitscheiben-Verfahren (Round - Robin),
- prioritätsgesteuerter (nicht unterbrechender) Scheduler,
- unterbrechender, auf Unterbrechungspunkten basierender Scheduler,
- voll unterbrechender Scheduler,
- Deadlinescheduling.

Die folgenden Abbildungen (Bild 2.2, 2.3, 2.4) beschreiben die möglichen Arten der Prozessorvergabe.

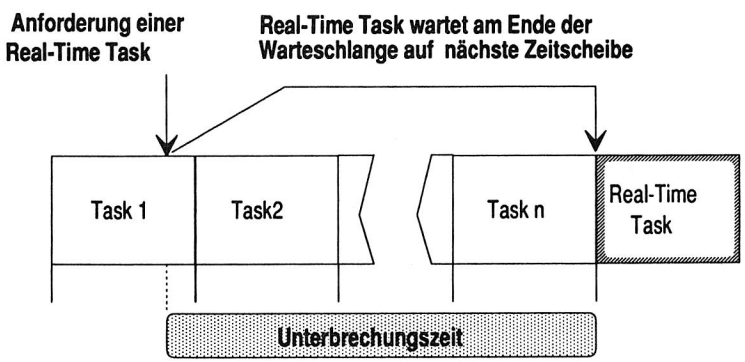



Bild 2.2: Prozessorvergabe im Zeitscheibenverfahren

Das reine Zeitscheibenverfahren (Round-Robin Scheduling) findet häufig in reinen Multitasksystemen Anwendung (Bild 2.2). Die Anforderung einer eintreffenden Task wird an das Ende der Warteschlange aller den Prozessor begehrenden Tasks gestellt. Die Task muß somit solange warten, bis alle vor ihr liegenden Tasks ihre Zeitscheibe erhalten und eigenständig wieder aufgegeben haben. Mit dem Round-Robin Verfahren können, je nach Anwendung, Reaktionszeiten von mehreren Sekunden auftreten. Diese Zeit ist nicht abschätzbar und vorhersagbar, so daß dieses Verfahren nicht für ein Echtzeitsystem geeignet ist.

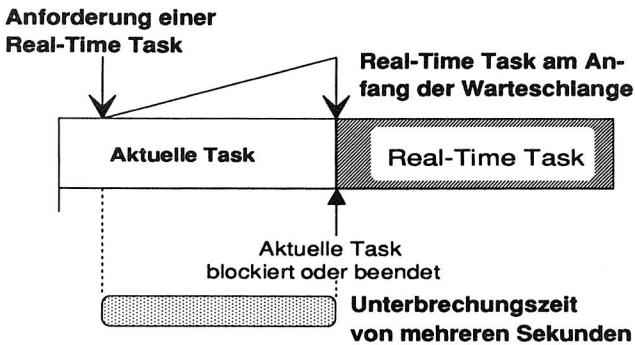



Bild 2.3: Prioritätsgesteuerte Prozessorvergabe

In prioritätsgesteuerten Betriebssystemen, wie zum Beispiel im UNIX™-System, wird die Anforderung einer eintreffenden Task gemäß ihrer Priorität in die Warteschlange aller Tasks eingetragen (Bild 2.3). Eine wichtige Task wird folglich aufgrund ihrer hohen Priorität direkt hinter der momentan aktiven Task eingereiht. Da die Laufzeit dieser Task (Systemroutinen) jedoch nicht vorhersagbar ist, kann die Reaktionszeit auch zu lang sein. Zudem werden die Prioritäten von Tasks durch das System meist nach Alterskriterien eigenständig geändert und so das Prioritätsgefüge gestört. Für Standardanwendungen ist dies sinnvoll, damit eine niederpriore Task auch einmal in der Prozessorzuteilung berücksichtigt wird. Somit ist dieses Schedulingverfahren nur bedingt für Echtzeitanforderungen geeignet.

Aufgrund der Möglichkeit, auf Ereignisse zu reagieren, eignen sich unterbrechende Verfahren (Bild 2.4) für Echtzeitapplikationen. Dieses Verfahren trägt die ankommenden Anforderungen entsprechend ihrer Priorität in die Warteschlange der aktiven Task ein. Sofern die Priorität der neu eintreffenden Task höher ist als die der gerade aktiven, wird die momentane Task unterbrochen. Dieses Verfahren stellt besondere Anforderungen an das verwaltende Betriebssystem.

Die beiden Verfahren in Bild 2.4 unterscheiden sich in der Zeitdauer bis zum nächsten Unterbrechungspunkt. Bei voll unterbrechendem Scheduler (Bild 2.4 unten) kann zu jeder Zeit die Bearbeitung des aktuellen Programms ausgesetzt werden, hingegen kann beim Scheduler mit Unterbrechungspunkten nur nach Eintreffen eines bestimmten Ereignisses, z.B. des Systemaufrufs, unterbrochen werden.

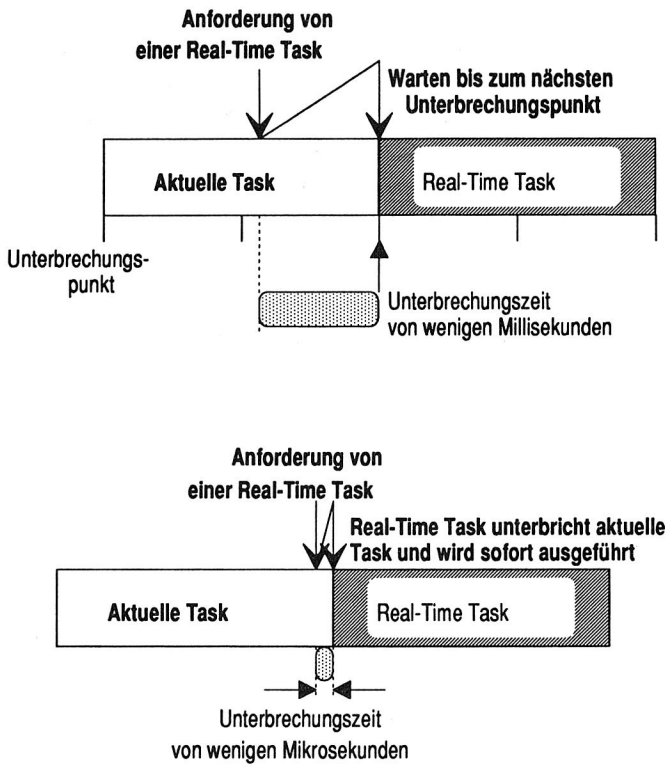



Bild 2.4: Unterbrechende Prozessorvergabe

Je nach Anforderung des Echtzeitsystems an die Reaktionszeit auf externe Ereignisse ist zwischen diesen beiden Verfahren zu wählen.

Ein weiteres in Echtzeitsteuerungen eingesetztes Einplanungs- und Garantieverfahren ist das Deadline-Scheduling /87/. Hauptmerkmal sind maximale Antwortzeiten, die der Programmierer zusätzlich zu einer Task angeben muß. Im Fall des Deadlinescheduling kann das Betriebssystem die Einhaltung der vorgegebenen Antwortzeiten überwachen und mit der Verlegung nicht mehr zu garantierender Tasks auf andere Prozessoren auch auf eine Prozessorüberlastung reagieren.

Das Deadline-Scheduling ist insofern optimal, als es Tasks, die sich in einem Intervall durch irgendeinen Algorithmus so einplanen lassen, daß ihre Zeitbedingungen

erfüllt werden, auch durch das Deadline-Verfahren eingeplant werden können /63/. Theoretisch läßt sich damit eine Prozessorauslastung von 100% realisieren. Sind die angenommenen Daten für die einzelnen Programme (Taktrate, Laufzeit) nicht korrekt, entsteht ein nicht vorhersehbares Systemverhalten. Mit der Abarbeitung in festen Prioritäten (Priorität entsprechend der Taktrate) können Prozessorauslastungen von 70% erreicht werden, ohne Zeitbedingungen von einzelnen Tasks zu verletzen /63/.

Der Verwaltungsaufwand im System ist dabei sehr gering, so daß für viele Anwendungen die Einplanung mit festen Prioritäten vernünftige Ergebnisse liefert. Der Hauptvorteil des Deadline-Scheduling ist, daß der Programmierer nicht sämtliche Prioritäten aller vorhandenen Tasks kennen muß, um seine neue Task in das System einzuplanen. Der Anwender gibt nur eine Zeitspanne an, in der die Task abgearbeitet sein muß. Hauptnachteil dieses Verfahrens ist die Tatsache, daß die benötigte Rechenzeit einer Task fix und vorher bekannt sein muß. Bei vielen Anwendungen innerhalb einer Robotersteuerung kann aber gerade davon nicht ausgegangen werden.

## 2.2 Interprozeßkommunikation

In einem System mit mehreren Prozessen muß es Möglichkeiten zur Kommunikation zwischen diesen geben. Nicht nur der reine Austausch von Informationen ist notwendig, sondern auch die gegenseitige Steuerung des Ablaufs durch Signalisierung von Ereignissen oder Zuständen. Dies kann über Meldungen geschehen oder für Synchronisationsaufgaben auch über Ereignismerker und Semaphoren. Letztere erlauben die Realisierung von gegenseitigem Ausschluß der Bearbeitung, zum Beispiel bei kritischen Berechnungsphasen. Dazu muß es sogenannte 'atomare Operationen' geben, das heißt der Zugriff darf nicht unterbrochen werden. Zur Erhöhung der Effizienz sollten schon auf unterster Ebene (Hardware, Maschinenbefehle) entsprechende Unterstützungsfunktionen zur Verfügung stehen.

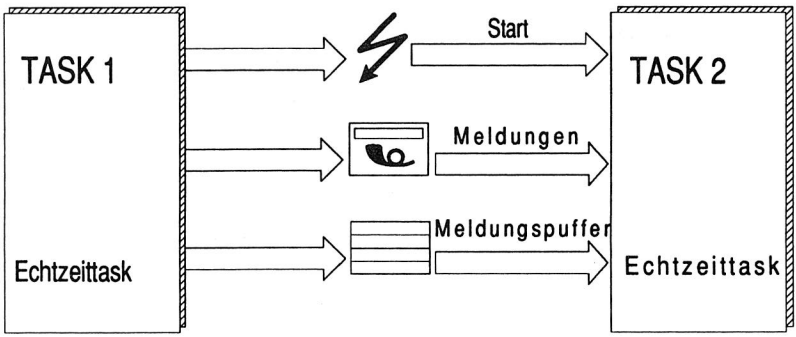



Bild 2.5: Interaktion zwischen Prozessen

Bei verteilten Prozessen sollten die Kommunikationsmechanismen in unterschiedlicher Ausprägung vorhanden sein /61/, um eine Adaption der Anwendung an die Konfiguration zuzulassen. Von den Mechanismen muß es verschiedene Ausprägungen geben, die für den jeweils verfügbaren Verbindungsweg optimiert sind.

## 2.3 Bestehende Echtzeit-Systeme

Die folgenden Darstellungen beschränken sich auf die speziellen Eigenschaften der ausgewählten Systeme. Eine ausführlichere Beschreibung ist in den angegebenen Quellennachweisen zu finden. Bei vielen Systemen handelt es sich um Simulationsmodelle /64, 97/ oder Entwürfe /99/. Bestehende Systeme sind zum Großteil durch Erweiterungen bestehender Betriebssysteme realisiert und auf die vorhandenen Grundfunktionen aufgesetzt. Als Basis dient in aller Regel UNIX™, dessen Prozeßmodell durch entsprechende Konstrukte erweitert wird /3, 77/. Die erreichbaren Leistungsdaten sind durch das unterlagerte UNIX™ stark beschränkt /36/ oder erfordern zusätzliche Rechnerausstattung (eigener Systemprozessor /77/). Vorteilhaft erscheint die Möglichkeit zum Erhalt einer weitgehenden Portabilität /3/. Ein anderer Ansatz ist eine Modifikation des UNIX™-Kerns (preemption-points) /33/, die schnellere Prozeßwechsel erlaubt. Der Einsatz unter strengen Echtzeitbedingungen läßt sich damit jedoch nicht erreichen.

Um ein hartes Echtzeit-Betriebssystem zu kreieren, gibt es zwei prinzipielle Möglichkeiten, Unterbrechungen (Interrupts) zu bearbeiten.

Typ 1: Verwendung möglichst weniger Interrupts. Meist wird nur noch ein Interrupt zugelassen. Die Realisierung der Prioritäten in der Taskabfolge geschieht durch Softwarekomponenten.

Typ 2: Realisierung der Taskprioritäten über eine große Zahl von Interrupteingängen. Je nach Systemzustand sind Interrupt-Ebenen gesperrt. Dieses Verfahren wird vielfach angewandt, wenn ein bestehendes Betriebssystem oder vorgegebene Hardware bestimmten Zeitanforderungen angepaßt werden soll.

Die beiden Verfahren unterscheiden sich in zwei wesentlichen Punkten. Die interruptgesteuerte Version (Typ 2) ist schwer in ihrem Verhalten vorhersagbar, dafür ist der Entwurf der einzelnen Module einfach. Die Sperrung von Interrupts in gewissen Systemzuständen birgt jedoch die Gefahr, daß in Systemebenen, in denen sich Anwenderapplikationen befinden, keine kurzen Reaktionszeiten mehr garantiert werden können. Kurze Reaktionszeiten und Vorhersagbarkeit sind jedoch Grundpfeiler eines harten Echtzeitsystems.

Unter Berücksichtigung einfacher Regeln bei der Erstellung von Programmen für ein System vom Typ 1 kann der Ablauf des Systems im voraus bestimmt werden, ist also planbar. Dies ist ein entscheidender Vorteil. Die Verarbeitung von Eingangssignalen (Interrupts) kann durch Realisierung als Scheduling-Ereignis, zum Beispiel auf einem eigenen Prozessor, erfolgen /99/. Das Wissen über Abhängigkeiten und Restriktionen von einzelnen Tasks (Prezedenz-Graph) läßt sich bei diesem Ansatz gut berücksichtigen.

Als typischer Vertreter des Typ 2 soll das System *EUROS* (Enhanced Universal Realtime Operating System) /53/ betrachtet werden (Bild 2.6). Durch Erweiterung von Diensten aus einer Bibliothek für die einzelnen Systemzustände ist es gelungen, auch Anwendertasks bedingt echtzeitfähig zu machen. Weiterhin wurde die Systembibliothek so ergänzt, daß sie ein echtzeitfähiges Softwaresystem unterstützen kann. Diese Systemaufrufe sowie der modulare Aufbau des Betriebssystems ermöglichen die Kompatibilität des EUROS-Systems mit MS-DOS™ und UNIX™.

*EUROS* stellt dem Anwender keine Routinen zur Verfügung, die ihm bei der Verteilung der einzelnen Programme auf das Gesamtsystem unterstützen. Auch für

das Scheduling des Objektsystems hat der Anwender selbst Sorge zu tragen und erfährt keine Unterstützung durch das Betriebssystem. *EUROS* ist im Gegensatz zu den anderen Hard-Real-Time Lösungen im Handel erhältlich. Augenblicklich gibt es aber lediglich eine Monoprozessor Testversion.

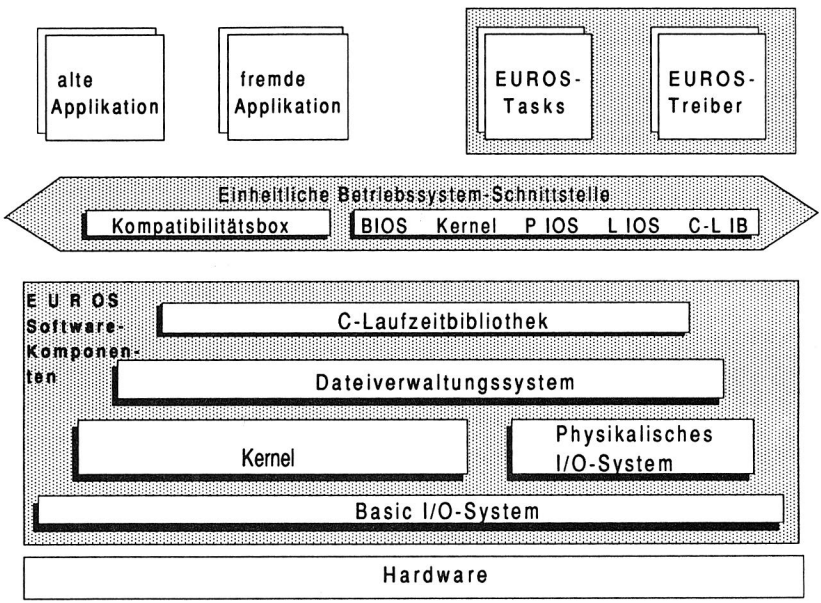
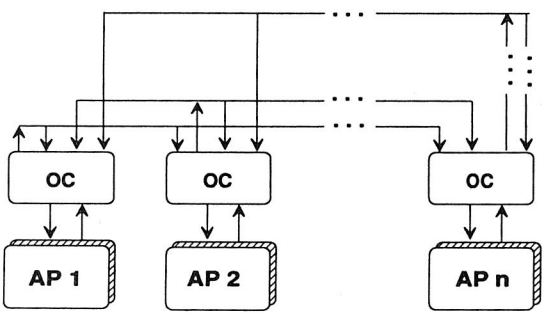



Bild 2.6: Aufbau des Betriebssystems *EUROS* /53/.

Beim zweiten Beispielsystem *MAFT* (Multicomputer Architecture for Fault Tolerance) /103/ bzw. *SIFT* (Software Implemented Fault Tolerance) /104/ wurde Fehlertoleranzaspekten Priorität geben. *SIFT* diente dabei *MAFT* als Vorlage. Solche Systeme werden hauptsächlich für flugkritische Kontrollfunktionen in der Luft- und Raumfahrttechnik eingesetzt /12/. Hier müssen Systemfehlerwahrscheinlichkeiten unterhalb einer Schranke von $10^{-10}$ während Missionszeiten von vielen Stunden garantiert werden. Gleichzeitig muß das verwendete Rechnersystem über eine hohe Rechenleistung verfügen. Daraus folgt die Vorgabe, Fehlertoleranz so in das System zu integrieren, daß die Rechenleistung nicht unter die Anforderungen sinkt.

Das System geht beim Architekturentwurf von einem fehlertoleranten Multiprozessorsystem aus. Die Fehlertoleranz bezieht sich dabei rein auf die Hardware und die eingesetzten Kommunikationsmodule. Die Fehlertoleranz wird erreicht durch dreifach modulare Redundanz (TMR), prioritätsgesteuertes Scheduling und eine genaue Synchronisation. Diese Maßnahmen werden noch durch eine exakte Auswahl der Programme und der zu benutzenden Prozessoren unterstützt, um zu verhindern, daß defekte Software und/oder defekte Hardware verwendet werden.




OC = Operation Controller
AP = Application Processor

Bild 2.7: Aufbau eines *MAFT*-Systems /103/

Die Architektur von *MAFT* geht weiterhin von einer strikten funktionalen und physikalischen Trennung des Verwaltungsaufwands (erzeugt durch die Fehlertoleranz und die Verwaltung des Gesamtsystems) und der Anwenderprogramme aus. Der gesamte Verwaltungsaufwand eines Netzknotens wird von einem sogenannten *'Operation Controller'* (**OC**) bewältigt, der exklusiv auf einem Prozessor läuft, während die Anwendungen auf einem anderen Prozessor, dem *'Application Processor'* (**AP**) ablaufen. Die Controller sind über mehrere serielle Busse miteinander verbunden (Bild 2.7). Beim Entwurf der *'Operation Controller'* wurde besonderer Wert auf die Fehlertoleranz gelegt. Der Stellenwert der Fehlertoleranz liegt in diesem System über allen anderen Entwurfskriterien. Jede ankommende Anfrage wird zuerst auf Konsistenz überprüft (*Message Checker*) und wird, sofern sie die Prüfung bestanden hat, dem Fehlertoleranzmodul zur weiteren Verarbeitung zugeführt. Dieser entscheidet dann, wie mit der ankommenden Anfrage verfahren wird. Ist diese korrekt, wird bei anfallenden Speicheränderungen zusätzlich ein Entscheider (*Voter*) zwischengeschaltet.

Tritt trotz aller Fehlertoleranzmaßnahmen doch ein Fehler auf, so wird dieser gemäß einer vorgegebenen Strategie behandelt. Dazu wurde ein übergeordnetes Automatenmodell für die Fehlerbehandlung eingeführt, um gegen aufgetretene statische Fehler "immun" zu sein. Immun bedeutet in diesem Fall, daß nach einem aufgetretenen und bemerkten Fehler das Gesamtsystem in einen konsistenten Zustand überführt wird und man versucht, das System so zu konfigurieren, daß es auch nach Ausfall einer Komponente weiterarbeiten kann. Transiente Fehler werden in diesem Fehlermodell nicht behandelt.

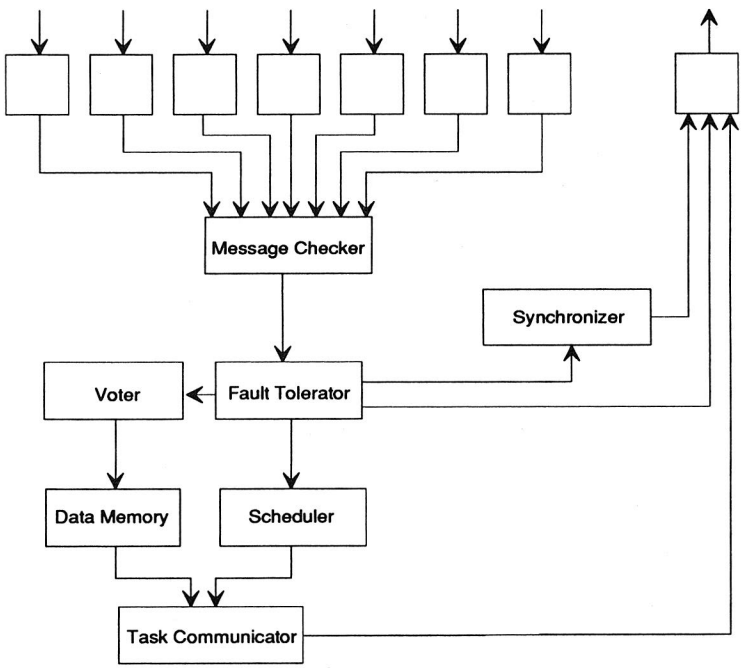



Bild 2.8: Aufbau eines *Operation Controllers* /103/

Trotz aller Vorzüge besteht der Mangel der Systeme *SIFT* und *MAFT* in der Nichtbehandlung von transienten Fehlern. Im Falle einer Störung oder des Teilausfalls einer Komponente wird davon ausgegangen, daß diese nicht reparabel ist und bei Wiederholung des Softwareabschnitts der Fehler erneut auftritt. Der Hardwareaufwand zur Erreichung der Fehlertoleranz ist zudem sehr groß.

Ein weiteres System zur Steuerung und Verwaltung von verteilten Objektsystemen stellt *CHAOS* (Concurrent Hierarchical Adaptable Object System) /90/ dar.

Es wurde entwickelt, um eine bessere Methode zur Strukturierung von komplexer Echtzeit - Anwendungssoftware zu erreichen und um gleichzeitig die statische und dynamische Anpassung solcher Software an die Zielanlage zu gewährleisten. Ein paralleles Programm besteht bei *CHAOS* aus einer Menge von abstrakten Objekten, die zusammenarbeiten, indem sie untereinander Operationen anfordern.

Jedes Objekt besteht aus einem Typ, einem eindeutigen Namen und einer Menge gültiger Operationen. Es gibt zwei Haupttypen eines Objekts:

- Passive Objekte: Operationen sind als Programmteile implementiert, die im Anforderungsfall innerhalb des Adreßbereichs des aufrufenden Objekts ablaufen. Dies entspricht dem "Thread"-Prinzip.
- Aktive Objekte: die Operationen sind entweder als ein einzelner Prozeß oder als eine Menge untereinander agierender Prozesse implementiert.

Es können die verschiedenartigsten Objekte kreiert werden. Die Spannweite geht vom leichtgewichtigen passiven Objekt, das keine internen Prozesse besitzt, bis hin zum zeitkritischen Objekt, das aus einer Vielzahl von internen Prozessen bestehen kann.

Die Koordinierung solcher Multiprozeßobjekte übernimmt dabei ein einzelner Koordinatorprozeß, der Objektaufrufe entgegennimmt und sie auf die ihm unterstellten Dienstprozesse verteilt. Durch diese Koordinierung sowie durch die unterschiedlichen Aufrufarten gewährleistet *CHAOS* auch den Aufbau von hierarchischen Objekten.

Die Prozesse eines Objekts interagieren untereinander mit Hilfe von Aufrufen. Sie können somit Operationen von ihren Partnerprozessen anfordern. Hierfür stehen eine Reihe von *CHAOS*-Systemaufrufen zur Verfügung. Sie lassen sich in drei Hauptklassen einteilen.

- ObjFastInvoke : Schnelle Kontrollaufrufe.
- ObjInvoke : Aufrufe, die Parameter enthalten und/oder die Kontrolle übergeben.
- ObjStreamInvoke : Reihenaufrufe, bei denen Wert auf geringe Zunahme der Übertragungszeit bei steigender Übertragungsrate gelegt worden ist.

Das gesamte Kommunikationsmodell von *CHAOS* beruht auf diesen drei Aufrufarten. Je nach Dringlichkeit wird eine der drei Aufrufarten verwendet. Dadurch läßt sich anhand der verwendeten Kommunikationsart schon die Priorität eines Teilobjekts erkennen. *CHAOS* ist das Echtzeitsystem, das die bei der Steuerung von Montagegeräten gestellten Anforderungen am ehesten erfüllt. Es fehlt in diesem Ansatz jedoch der Aspekt der Fehlertoleranz. Bei dem vorgestellten System handelt es sich um einen Entwurf einer Universität, und es ist deshalb nicht kommerziell verfügbar.

Die vorgestellten Systeme und Überlegungen zeigen ein Fehlen eines Betriebssystems für Roboter, das leistungsfähig genug ist, um die Echtzeitanforderungen zu erfüllen und gleichzeitig die Integration mit Standardrechnern einer Montagezelle erlaubt.

# 3 Strukturierung von Montageaufgaben

## 3.1 Steuerungsstrukturen in der Montageautomatisierung

In der Rechnerführung wird sich das Konzept der hierarchischen Gliederung der einzelnen Komponenten und eine entsprechende Funktionsaufteilung durchsetzen /31/. Der Einsatz eines Zellenrechners wird aufgrund von sichtbaren Kapazitätsgrenzen auf den elementaren Steuerungen und den wachsenden Ansprüchen in bezug auf Organisations- und Informationsstrukturen im Betrieb sowie der Verwendung von Hochsprachen immer sinnvoller.

Von speziellem Interesse sind also im folgenden die Zellenebene und die daraus resultierenden Anforderungen an die Gerätesteuerungen. Die Betrachtung einer Zelle als teilautonome Einheit eines Fertigungsunternehmens scheint gut dazu geeignet zu sein, die erforderliche Gliederung vorzunehmen und entsprechende Strukturierungsmechanismen zu entwickeln /65/. Dabei sollen auch prozeßnahe Aufgaben miteinbezogen werden /41/.

Eine flexible Montagezelle kann als ein Teilbereich des CAM - Gebildes verstanden werden, innerhalb dessen alle Betriebsmittel, die benötigt werden, um eine bestimmte Anzahl von Montageschritten automatisch durchführen zu können, organisatorisch und räumlich zusammengefaßt sind. Kernelemente sind automatische Montage- und Prüfstationen sowie ein Materialflußsystem zur Verkettung der einzelnen Stationen. Die Aufteilung einer Anlage oder Fabrik in einzelne Zellen hängt von Strukturen und Produkten des jeweiligen Unternehmens ab.

Aus informationstechnischer Sicht kann eine Montagezelle als ein Verbund von Einzelprogrammen verstanden werden, die den Gesamtablauf realisieren. Diese sind auf modulare Steuerungskomponenten in der Zelle verteilt, welche wiederum über Feldbusse miteinander verbunden werden.

Die Zelle bekommt von einer übergeordneten Institution (der Leitebene oder dem Bediener) einen ihren Möglichkeiten entsprechenden Auftrag mit den dazugehörigen technischen und organisatorischen Informationen und den erforderlichen Materialien (Rohstoffe, Halbzeuge, Werkstücke, Einzelteile, Baugruppen,..) zugeteilt. Zur Durchführung des Auftrags kann dann lokal in der Zelle geplant

werden /25/. Das Ausgangsprodukt einer Zelle stellen auf der Materialseite die montierten Baugruppen bzw. Fertigprodukte und auf der Informationsseite die auf Prozeßebene erfaßten und zellenintern bereits vorverarbeiteten Betriebsdaten dar.

Zum Erreichen einer Flexibilität mit Anforderungen wie gleichzeitiges Ausführen mehrerer Montageaufträge in der Zelle, Reaktion auf Ausfall von Stationen und automatische Arbeitsverteilung an die Stationen sind modulare Steuerungskonzeptionen notwendig /24/. Zusätzlich müssen die starren Programme für die einzelnen Geräte zustandsabhängigen, generischen Komponenten weichen /37/.

## 3.2 Gliederung von Montageaufgaben

Eine Voraussetzung zur Gliederung ist die systematische Analyse von Montageaufgaben und die Ermittlung von beschreibenden Parametern /17/. Die Parameter sind abhängig von den Haupt- und Nebenfunktionen, die in der Montage durchzuführen sind /95/.

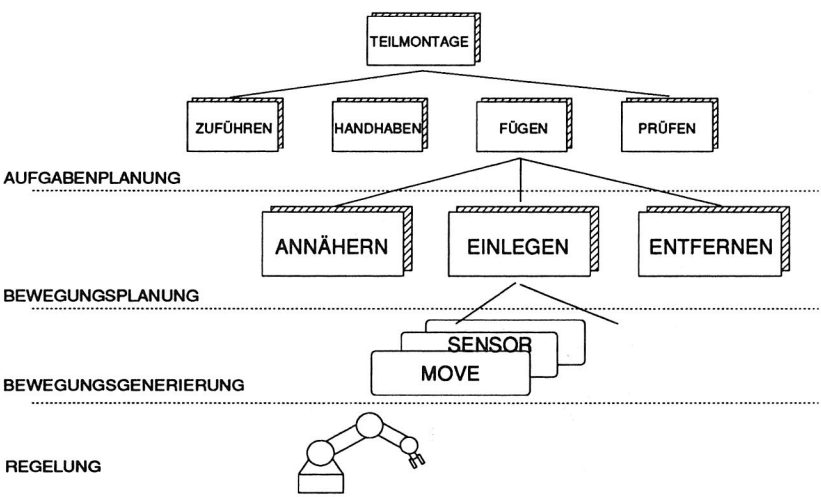



Bild 3.1: Funktionen in der Montage

Ziel der Bestrebungen zur Modularisierung der Aufgaben ist die Möglichkeit, auch die gesamte Steuerung einer Zelle entsprechend weiter zu modularisieren.

Für die systematische Dekomposition der gesamten Montagezellensteuerung muß diese auf einer *Abstraktionshierarchie* beruhen, d.h. die gesamte Steuerung der Zelle wird so in Komponenten zerlegt, daß diese funktional klar getrennt jeweils einer Ebene zugeordnet werden können /84/. Dieser Ansatz bildet die Grundlage für die folgenden Darstellungen zum hierarchisch gegliederten Aufbau der Zellensteuerung.

### 3.2.1 Modell einer Montagezelle

Das verwendete Modell beinhaltet in der höchsten Abstraktionsebene nur Stationen; die weitere Zergliederung erfolgt in *logische Orte* wie Bearbeitungsplätze, Puffer, Ein-/ Ausgänge (Bild 3.2). Die Abstraktion auf logische Orte erleichtert die Realisierung von Fehlerbehandlungen und lokaler Neuplanung innerhalb der Zelle, denn der Arbeitsplan enthält nur abstrakte, zellenunabhängige Angaben.

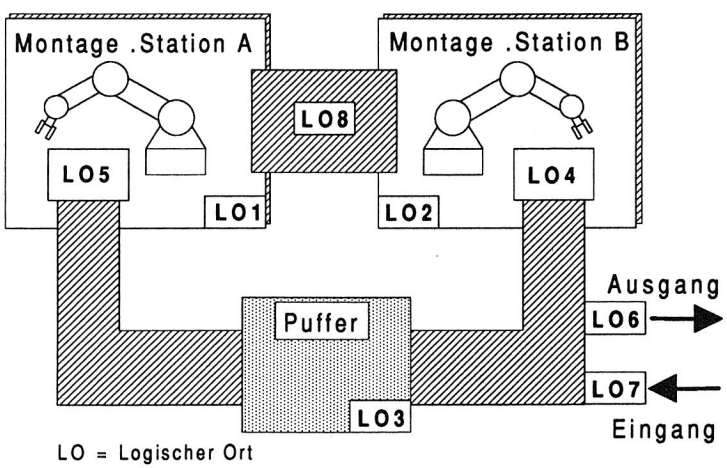



Bild 3.2: Logische Struktur einer Zelle

Wesentlich ist hierbei die strikte Trennung zwischen logischer und physikalischer Ebene, denn damit läßt sich ein Maß an Universalität erreichen, die eine allge-

meine Anwendung des Konzepts zuläßt. Erst auf einer niedrigen Ebene (Geräteebene) werden die logischen Elemente auf reale Komponenten der jeweiligen Zelle abgebildet.

Die Definition einer logischen Montagezelle erlaubt den Entwurf einer allgemeinen Arbeitsplanstruktur, in der auch alternative Abläufe berücksichtigt werden können. Die Strukturierung der Arbeitspläne erleichtert den Entwurf von Softwaresystemen für Montagezellen. Die Tragfähigkeit des Entwurfs wird in einer parallelen Arbeit nachgewiesen /85/.

### 3.2.2 Funktionshierarchie für Montagezellen

Für die Steuerungsaufgaben in einer flexiblen Montagezelle wurde ein Architekturmodell mit sechs Ebenen entwickelt. Dieses Modell liegt allen weiteren Überlegungen zugrunde und dient später auch zur Einordnung der Aufgaben der Gerätesteuerungen. Es dient ferner dazu, einen begrifflichen Rahmen für die Abläufe zu schaffen. Im folgenden wird die Funktion der sechs Hierarchieebenen bei der Steuerung einer Montagezelle aufgezeigt (Bild 3.3).

- ***EBENE 6***

  Die ***Ebene 6*** repräsentiert die Gesamtmontage in einer Zelle, dafür liegt entweder ein expliziter Arbeitsplan (APL) vor, oder es wird eine spezielle APL-Ausprägung aus Rahmenarbeitsplänen für Produktfamilien generiert. Randbedingung ist hier, daß sich die genannte Gesamtmontage vollständig innerhalb der Zelle ausführen läßt (keine zellenübergreifenden Angaben!). Eine entsprechende Aufteilung der Arbeitsinhalte muß auf der Fertigungsleitebene stattfinden. Entscheidend auf ***Ebene 6*** ist also die Aufgabe: Zellauftrag abwickeln!

  Diese Schicht braucht demnach Fähigkeiten zur Übernahme von Aufträgen, zu Durchführbarkeitsuntersuchungen, zur Verwaltung von Aufträgen, zur Stammdaten- und Arbeitsplanverwaltung, zur Einlastungsplanung von Aufträgen und zur Zellenüberwachung /76/. Diese Überlegungen zeigen, daß nur Fähigkeiten gebraucht und genutzt werden, die den Gesamtzellauftrag bzw. das Betriebsmittel Zelle betreffen.

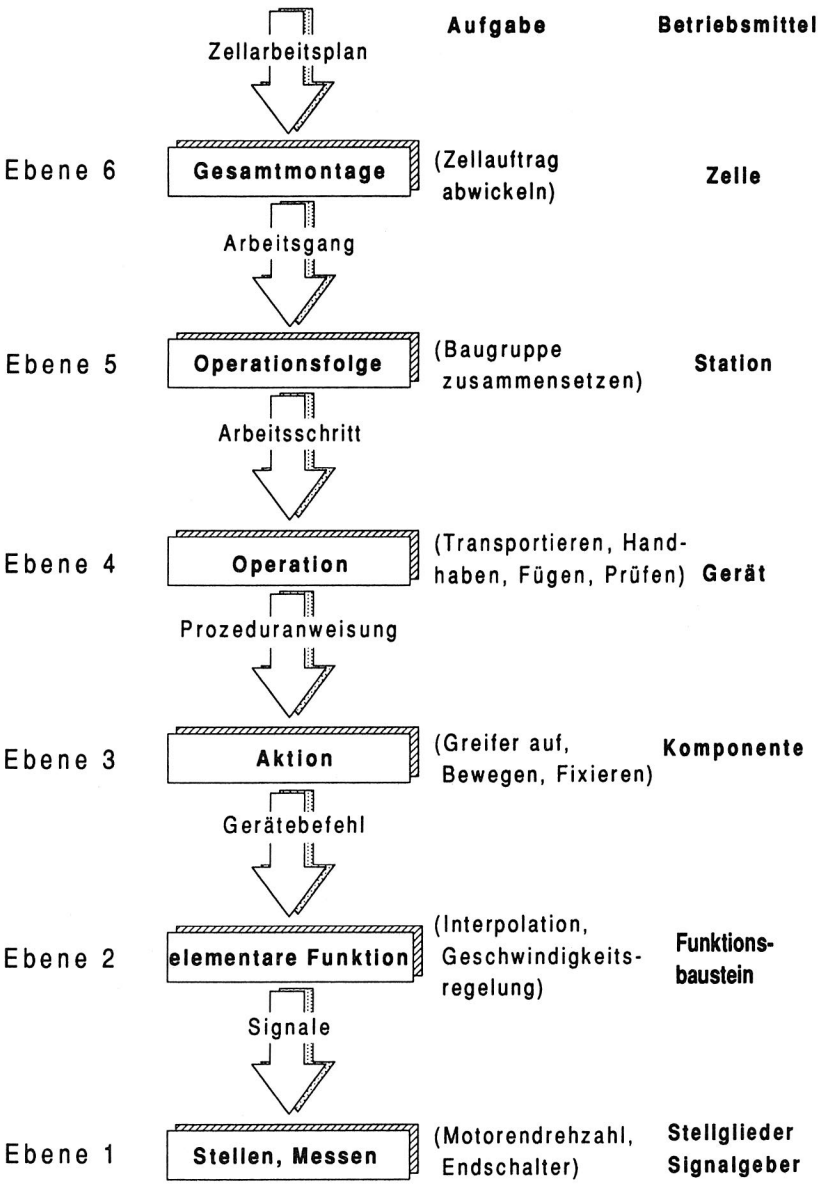



Bild 3.3: Funktionshierarchie für Montagezellen

- ***EBENE 5***

Auf der ***Ebene 5*** (Baugruppenmontage) ist das zentrale Element - die Ablaufsteuerung - angesiedelt. Sie bildet die Kernstelle für die zeitliche und logische Koordinierung der gegebenen Arbeitsgänge sowie der jeweils daraus abzuleitenden logischen Befehle für die Montagegeräte, beziehungsweise das Materialflußsystem.

Unter dem Betriebsmittel Station wird maschinenbaulich ein Roboter oder eine NC-Maschine mit zugehöriger Peripherie und Steuerung verstanden.

Die Grundlagen für die durchzuführende Synchronisation, Koordinierung und mögliche Parallelität der Arbeitsgänge bilden Fähigkeitsangaben zu den Stationen, die Gliederung der Zelle in logische Orte, eine entsprechende Arbeitsplanstruktur und logische Werkzeug- und Werkstückbezeichner. Von der Zuordnung zwischen logisch ansprechbaren Komponenten wie Bearbeitungsplätze, Übergabeplätze, Puffer, Ein-/Ausgänge und der konkreten physikalischen Realisierung wird auf dieser Ebene abstrahiert. Die Ablaufsteuerung ist also in diesem Konzept nicht eine spezielle Applikation, sondern unabhängig von technologischen und auftragsspezifischen Gegebenheiten. Besonders die heute geforderte Flexibilität wird durch eine freie Arbeitsgangauswahl nach Optimierungskriterien, durch eine freie Zuordnung der Aktivitäten im Rahmen der Möglichkeiten zu den Stationen (Planungsaktivitäten) und durch die generelle Vermeidung von Verklemmungen unterstützt /94/.
Die Aufbereitung des Arbeitsganges, das heißt die Aufteilung der Operationsfolge in einzelne Operationen basiert auf der funktionalen Gliederung des Montageablaufes. Neben dem Bereitstellen, Fügen und Entsorgen treten noch die Funktionen "Identifizieren" und "Überwachen" auf. Den Fügevorgang kann man weiter zergliedern in eine grobe Annäherung, eine Feinpositionierung (eventuell sensorunterstützt) und den eigentlichen Fügeprozeß (siehe auch ***Ebene 4***). An der Schnittstelle stehen nach Auflösung der Operationsfolgen für die ***Ebene 4*** erweiterte Solldaten zur Verfügung, die für die entsprechenden Phasen mit aktueller Zustandsinformation angereichert wurden. Für die Istzustandserfassung müssen von der ***Ebene 4*** IST- Informationen über Geräte , Werkzeuge und Werkstücke auf einem passenden Abstraktionsniveau bereitgestellt werden.

- ***EBENE 4***

In der ***Ebene 4*** gilt es, eine Operation in eine Reihe von Einzelaktionen aufzulösen. Diese klare Trennung und Modularisierung ist in heutigen Roboterprogrammen nicht enthalten. Ziel muß es daher sein, zu generatorischen Verfahren der Steuerinformationserzeugung überzugehen. An dieser Stelle im Steuerungsablauf wird die vorgegebene Operation unter Zuhilfenahme einer Datenbasis in entsprechende Aktionen umgesetzt. Als Beispiel soll 'Fügen' näher betrachtet werden. Für den Fall des Fügens durch Einlegen können die Aktionen durch eine Reihe von verfahrensbeschreibenden Parametern bestimmt werden. In der Regel wird in ein feststehendes größeres Teil (Basisteil) ein Teil eingelegt (Fügeteil). Die Art und Weise, wie dies zu geschehen hat, ist durch die Operation gegeben. Die räumliche Beziehung der beiden Komponenten bestimmt im wesentlichen den Gesamt- und den Bewegungsablauf. Für das Basisteil können relevante Parameter sein:

- Teilgeometrie,
- Teiltoleranzen,
- Fügeort auf dem Basisteil,
- Fügerichtung,
- maximal zulässige Fügekraft.

Diese Punkte sind zwar aus der Konstruktionsphase bekannt, könnten teilweise jedoch ebenso aus Material und Fügeverfahren plus Geometrieinformation bestimmt werden. Diese implizite Steuerung (Programmierung) läßt sich in verschiedenen Stufen der Auflösung durchführen. Es ist jedoch ersichtlich, daß Angaben und Parameter, welche in der Konstruktion entstehen, explizit an die Steuerdatengenerierung übergeben werden sollten. So könnten Annäherungs- und Fügerichtung prinzipiell aus den Konstruktionsdaten ermittelt werden. Heutige Systeme stellen allerdings diese Informationen noch nicht zur Verfügung. Die Entwicklung zeigt aber einen Trend in die Richtung an, zusätzliche Daten den eigentlichen Konstruktionsunterlagen anzugliedern /83/. Neben der impliziten Beschreibung der Steuerungsaufgabe ist die Angabe der Fügeteile durch logische Bezeichner eine entscheidende Voraussetzung auf dem Weg zu mehr Flexibilität. So sollte z.B. auf die explizite Angabe der Lage eines Teils auf einer Palette in Koordinaten verzichtet und statt dessen die logische Angabe des Ortes verwendet werden. Die eigentliche Fügeoperation kann dann in vielen Fällen explizit angegeben sein,

und man erreicht dennoch (für ein bestimmtes Fügeverfahren) einen hohen Grad an Flexibilität und ein leistungsfähiges "implizites Konzept", das für zukünftige Erweiterungen offen ist. Wesentlicher Gesichtspunkt der ***Ebene 4*** ist also die Umsetzung der logischen Teilebezeichner auf die aktuelle Konstellation der Montagestation und die darin befindlichen Geräte. Im Idealfall wird somit den überlagerten Ebenen eine stationsunabhängige Schnittstelle geboten, die eine Ausführung von Arbeitsplanangaben zuläßt. Entscheidend ist dabei außerdem, daß keine Transportanweisungen im Arbeitsplan enthalten sind (im Gegensatz zu /105/), sondern erst in der jeweiligen Instanz erzeugt werden . Die Abfolge der Gerätezuordnung wird damit so flexibel wie möglich gehalten.

- ***EBENE 3***

In der ***Ebene 3*** wird nun die einzelne Aktion in elementare Funktionen zerlegt. Dies ist eine mögliche Schnittstelle zu existierenden Robotersteuerungen. Eine Aktion ist z.B. die Bewegung von einem Start- zu einem Zielpunkt.

Befehlsbeispiel: *MOVE x, y, z, a, b, c*

Bei der Handhabung besteht die Aufgabe, ein Teil in einer bestimmten Lage und Orientierung an den Zielpunkt zu bringen. Im allgemeinen unterliegt der Weg dorthin einer Vielzahl von Randbedingungen und Grenzen, die jedoch für die eigentliche Aktion nicht von Interesse sind. Statt also explizit z.B. alle Hilfs- und Zwischenpunkte oder Beschleunigungen und Geschwindigkeiten festzulegen, sollte an dieser Stelle auf Funktionen zugegriffen werden, welche aus Start- und Zielpunkt und einem entsprechenden Umweltmodell die Bewegung generieren. Ein wesentlicher Bestandteil dieses Modells muß die dynamische geometrische Beschreibung der Umwelt sein, um Kollisionen zu vermeiden und den optimalen Verfahrweg zu bestimmen.
Als Ausgabe der ***Ebene 3*** können so einfache Gerätebefehle entstehen, die auch von vielen heutigen Steuerungen zur Verfügung gestellt werden. Für Anweisungen, die nicht unmittelbar die Bewegung betreffen (Sensoreinsatz) oder eine spezielle Bewegungsform implizieren (Pendeln, Fahre bis Kraft > Grenze), ist die Einheitlichkeit nicht mehr gegeben. Ebenso bestehen selten Möglichkeiten, parallele Aktionen zu initiieren.

Gegenstand der bisherigen Überlegungen war nur die Richtung der Daten von oben nach unten im Sinne der vorgestellten Hierarchie. In jeder Ebene muß es Rückmeldungen mit entsprechendem Abstraktionsniveau aus den unterlagerten Schichten geben. Vor allem auch aus diesem Grund lassen sich heutige Robotersteuerungen kaum integrieren, da sie keine ausreichende Informationstransparenz besitzen.

### 3.2.3 Prozeßstruktur für eine Montagezelle

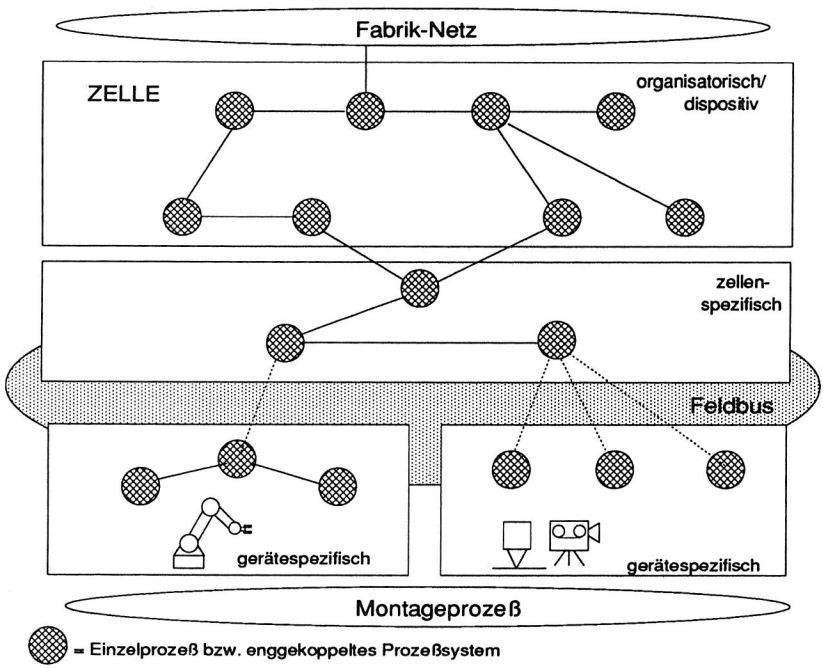



Bild 3.4: Allgemeine Prozeßstruktur für eine Montagezelle

Ein Verbund von Einzelprogrammen realisiert den Gesamtablauf einer Montagezelle (Bild 3.4). Man kann dabei vier Hauptbereiche trennen. Das Gebiet der Aktorik, welches Stellvorgaben und Bewegungen in der Anlage realisiert und den sensorischen Abschnitt der Information aus dem Montageprozeß liefert. Überlagert findet man einen Bereich, der zellen- und aufgabenspezifische Funktionen über-

nimmt. Die organisatorisch-dispositive Schicht übernimmt den Anschluß an die Informationsstruktur der Fabrik und führt zudem lokale Planungsaufgaben für die Ressourcen durch. Als Kommunikationsmedium ist in der Zelle ein Feldbus zum Datentransport vorgesehen, wenn ein Prozeßgebilde Gerätegrenzen überschreitet.

Die Prozeßstruktur läßt sich gemäß der in Abschnitt 3.2.3 vorgestellten Funktionshierarchie weiter spezifizieren, so daß man zu einer genaueren Aufteilung gelangt (Bild 3.5). Das zentrale Element der Steuerungssoftware bildet die Ablaufsteuerung, genauer betrachtet ein Prozeß zum Scheduling der Aufgaben. Ausgehend vom logischen Aufbau der Montagezelle und den vorliegenden Arbeitsgängen stößt er die entsprechenden Handhabungs- und Fügevorgänge an und löst die zelleninternen Transporte aus.

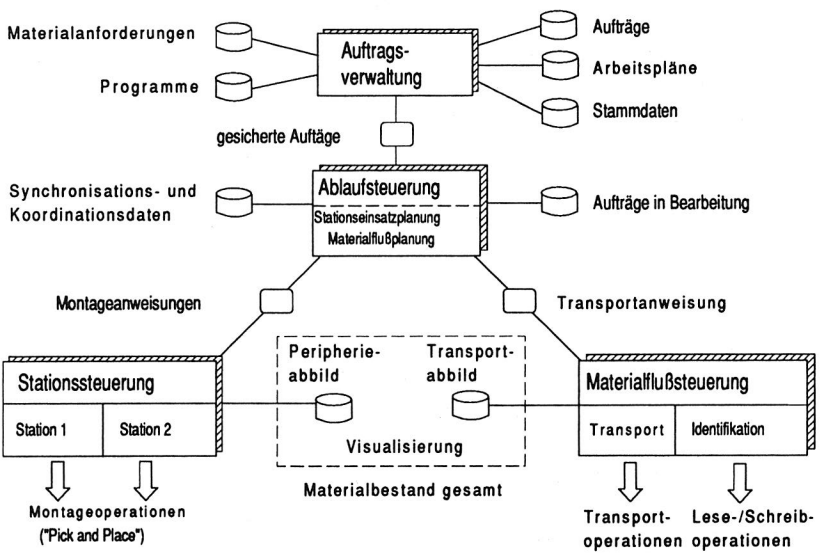



Bild 3.5: Softwaresystem gemäß der Funktionshierarchie

Die einzelnen Teilaufgaben stehen dabei häufig in direkten Abhängigkeitsverhältnissen. Die Koordinierung eines Arbeitsganges mit einem anderen Arbeitsgang kann der Herstellung einer räumlichen oder zeitlichen (chronologischen) Beziehung der Arbeitsgänge untereinander dienen. Eine räumliche Beziehung der Arbeitsgänge wird durch die Auswahl der ausführenden Stationen erreicht. Die

Gründe, den Start eines Arbeitsganges oder andere Vorgänge mit einem Ereignis zu synchronisieren, sind zellenspezifisch unterschiedlich aufgrund der Fähigkeitslisten der Stationen und der Abhängigkeiten der logischen Orte. In einer flexiblen Montagezelle mit zwei Stationen haben sich die folgenden wesentlichen Koordinierungs- und Synchronisationsmechanismen als notwendig erwiesen: Koordinierung zweier Arbeitsgänge bezüglich der Stationen, Koordinierung eines Arbeitsganges mit dem Vorhandensein eines Teiles an einem bestimmten Ort der Montagezelle, Koordinierung eines Arbeitsganges mit freier Lagerkapazität an einem Ort, Koordinierung von Transporten untereinander - entsprechend den Erfordernissen des Transportsystems, Synchronisation des Starts zweier Arbeitsgänge. Die dynamische, aber blockierungsfreie Zuordnung von Arbeitsgängen sowie die Generierung und Aufteilung von Transportanweisungen werden somit durch diesen Prozeß gewährleistet. Eine weitergehende Koordinierung, zum Beispiel die gemeinsame Bewegung zweier Roboter, obliegt dann den Gerätesteuerungen.

Über eigene Online-Planungsprozesse zum Stationseinsatz und Materialfluß (Schnittstelle zur unterlagerten Ebene) erfolgt dann die Übermittlung der Arbeitsgänge als logische Aufträge an die Steuerdatengenerierung beziehungsweise an die Materialflußsteuerung, die dann ihrerseits die Befehle interpretieren. Der angegebene Arbeitsschritt mit Fügeoperationskennung, Fügeteilnamen, Basisteilnamen und zugehörigem Fügeort sowie notwendigen technologischen Daten wird durch Auswertung der Zustandsdaten mit entsprechenden Parameterwerten aufgefüllt.

Als Beispiel sei der Fügeort, d.h. der Ort, an dem das Fügeteil auf das Basisteil montiert wird, angeführt. Aus Gründen der Flexibilität ist es zweckmäßig, alle Angaben von Fügeorten relativ zum Basisteil anzugeben; so können die Daten direkt von der Konstruktion übernommen werden, und auch bei Konstruktionsänderungen sind keine Folgeauswirkungen auf die Steuerung zu erwarten. Im jeweiligen Stationsprozeß werden diese relativen Ortsangaben ins Roboterkoordinatensystem umgesetzt und als Parameter an die Bewegungssteuerung übertragen. Da bei der Erstellung des Arbeitsplanes für die Zelle weiterhin nicht vorhersehbar ist, welche Fertigungsaufträge in einer Station zu einem bestimmten Zeitpunkt in Bearbeitung sind, enthält der Arbeitsplan keine an Fertigungsaufträge gebundene Objekte, sondern nur logische Teilebezeichner. Eine weitere Aufgabe ist also die Zuordnung zu in der Station vorhandenen Teilen bzw. Baugruppen.

### 3.2.4 Steuerdatengenerierung für Montagegeräte

Im vorangegangenen Abschnitt wurde die Aufteilung der Aufgaben aus der Funktionshierarchie auf einzelne Prozesse vorgenommen. Diese lassen sich intern weiter in drei wesentliche Teile zergliedern. So besteht die Notwendigkeit, planende Funktionen von der Ausführung zu trennen und die Kommunikation zu den beiden angrenzenden Softwareebenen zu separieren. Bei der Generierung der Steuerinformationen für Montagegeräte besteht die Möglichkeit, ein Abstraktionsniveau zu erreichen, das den Einsatz der Softwarekomponenten für verschiedene Anwendungen erlaubt. Die Betrachtung von Funktionen und Sensoraufgaben, die unmittelbar mit dem Fügeprozeß korrelieren, sei im folgenden ausgenommen. Die Steuerdatengenerierung muß primär in der Lage sein, das Umfeld (Peripherie) des Roboters zu modellieren und zu manipulieren. Dazu sind drei Grundfunktionen zu realisieren: das Einschleusen von Bauteilen, der Fügevorgang und das Ausschleusen von Baugruppen.

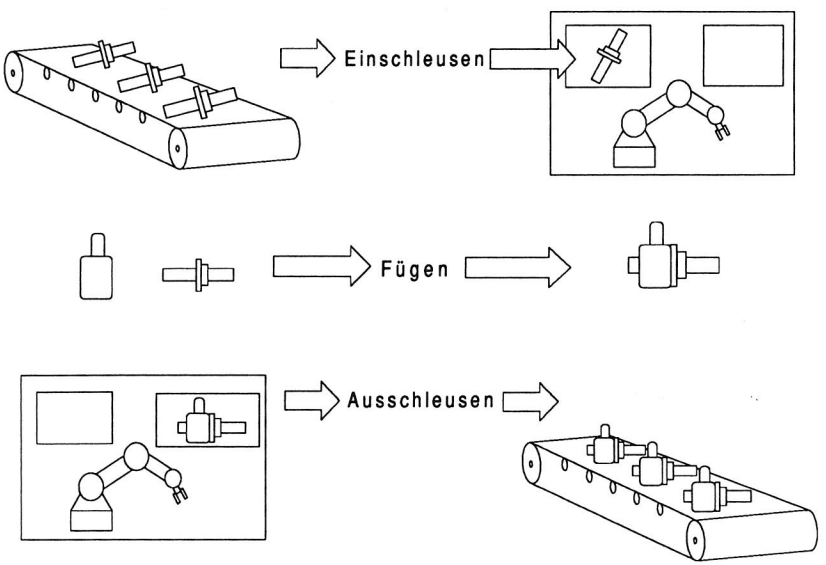



Bild 3.6: Grundfunktionen eines Roboterarbeitsplatzes

Diese Betrachtungsweise ist sinnvoll für die Integration des Roboters samt seiner Umgebung als eine Komponente einer Zelle. Man kann dadurch eine klare Tren-

nung der Informationen für Produkt, Montagegerät und Fügevorgang erreichen. Dies erleichtert den Austausch beliebiger Komponenten und steigert die Universalität der Softwarekomponenten.

Maßgeblich ist dabei die Identifikation der Teile oder Baugruppen über logische Bezeichner. Es muß eine Abstraktion stattfinden zwischen Bezeichnung und Lokation. Dazu ist eine klare Trennung durchzuführen zwischen den Angaben zu Anzahl und Art der Teile, die sich in der Geräteperipherie befinden und der physikalischen Lage und Anordnung der Magazine.

Für den eigentlichen Fügevorgang ist die Strukturierung der Steuerinformation der Einzelaufgaben noch nicht weit fortgeschritten, so daß sich die relevanten Daten meist in nicht isolierter Form im Bearbeitungsprogramm des Montagegerätes befinden. Spezielle Sprachentwürfe können dabei nur bedingt Abhilfe schaffen /78/.

Die vorgestellten Überlegungen zur systematischen Gliederung von Montageaufgaben und -anlagen sind der Ansatzpunkt zur Betrachtung der internen Abläufe in Robotersteuerungen mit dem Ziel einer Modularisierung und Strukturierung auch in der Gerätesteuerung selbst.

# 4 Anforderungen an Gerätesteuerungen

Die Steuerungen der einzelnen Geräte sollen sich leicht in das System einer Montagezelle integrieren lassen, das die vorgestellte Struktur besitzt. Dabei sind nicht nur die erforderlichen Funktionalitäten von Interesse, sondern neben der Möglichkeit weitergehender Kommunikation auch der Grad der Modularität der Steuerung. Gerade im Bereich der Montage ist durch die Robotersteuerungen ein sehr breites Spektrum an Anforderungen abzudecken, abhängig von der jeweiligen Anwendung. Eine klare Strukturierung und hohe Modularität der Steuerung bergen deshalb ein besonders hohes Flexibilisierungspotential für Roboter.

## 4.1 Planungsaspekte

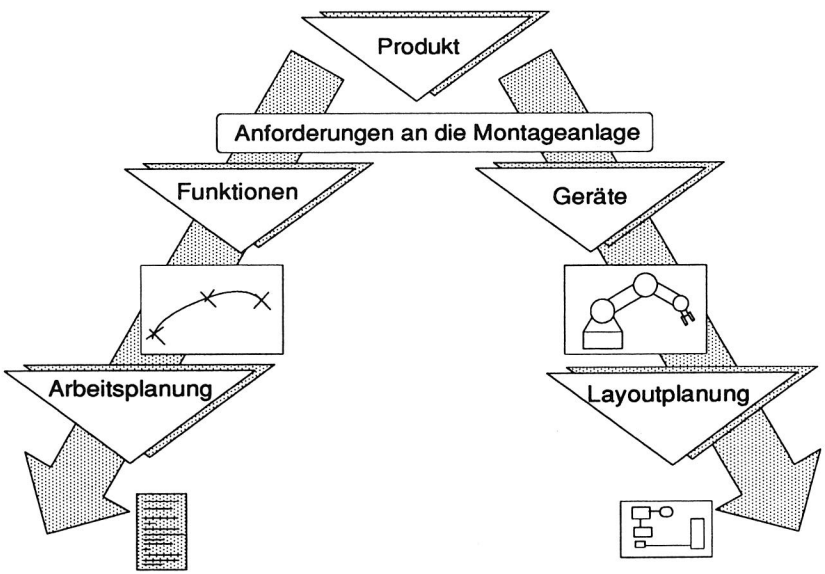



Bild 4.1: Getrennte Planung von Abläufen und Geräten

Im Bereich der rechnergestützten Planung gibt es schon weitergehende Ansätze zur Konzeption, Detailplanung und auch Programmierung von Montageanlagen.

Ausgehend von der Analyse oder auch Synthese der geforderten Fertigungsoperation stehen Werkzeuge zur Auswahl des passenden Gerätes und eine automatisierte Bestimmung der Aufgabenreihenfolge zur Verfügung.

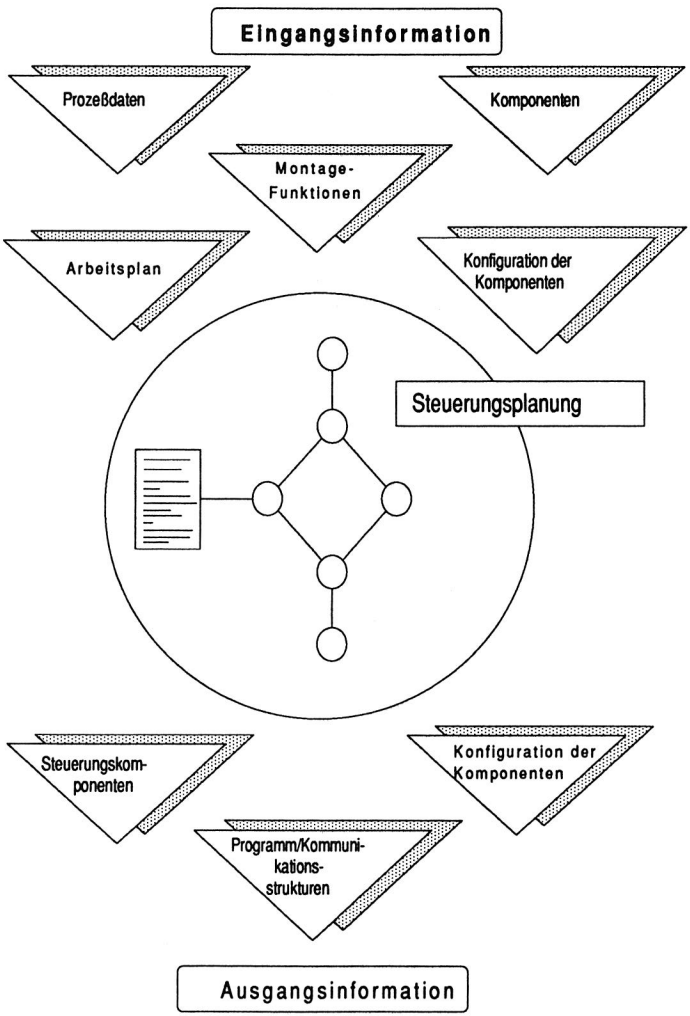



Bild 4.2: Rechnergestützte Steuerungsplanung

Durch Simulationen können der Materialfluß, Art und Größe von Puffern in der Anlage ermittelt werden. Als Planungsergebnis erhält man das Layout der Anlage, Vorschriften zum Montageablauf und Konstruktionsunterlagen für Fertigungshilfsmittel. Einige Arbeiten befassen sich zudem mit den Möglichkeiten der Generierung von Steuerprogrammen für die einzelnen in der Anlage verwendeten Geräte (Offline-Programmierung) /60, 66/. Als hinderlich für den allgemeinen Einsatz erweist sich dabei das Fehlen eines Standards für Programmiersprachen von Robotern. Neben firmenpolitischen Interessen liegt dieses Manko vor allem begründet im fixen und unterschiedlichen Leistungsumfang heutiger Steuerungen. Es kann aufgrund der unterschiedlichen Mächtigkeit der Steuerungen keine einheitlichen Oberflächen geben.

Im Planungsablauf einer Anlage sind deshalb zwei große Schwachpunkte auszumachen. Zum einen die fehlende Behandlung von Sensorintegration und zum anderen die starre Kopplung von Roboter und Steuerung. Der Funktionsumfang wird in der Regel schon durch die Auswahl des Geräts festgelegt und nicht durch die Anforderungen aus der Anwendung oder aus Integrationsgesichtspunkten.

Das Steuerungssystem einer Anlage wird auf Zellenebene nicht realitätsnah geplant, weil keine planbaren oder konfigurierbaren Elemente zur Verfügung stehen, die sich frei von Restriktionen kombinieren lassen.

Die Steuerung eines Gerätes muß sich in Zukunft genauso aus Bauelementen zusammensetzen und planen lassen (Bild 4.2), wie das heute mit den mechanischen Anlagenteilen der Fall ist. Dazu sind Werkzeuge und Hilfsmittel aus dem Bereich der Informatik sicher hilfreich. Künftige Steuerungen müssen ein Höchstmaß an Flexibilität und Modularität besitzen, um die ganzheitliche Planung einer Anlage zu gewährleisten.

## 4.2 Modularität der Steuerungen

Um einen Entwurf der Gerätesteuerung in der Planungsphase zu ermöglichen, ist Modularität auf zwei Ebenen anzustreben. Naheliegend ist die Erweiterbarkeit auf Hardwareebene; aber ebenso wichtig erscheint die Möglichkeit zur Strukturierung der Softwarekomponenten. Der Modulgedanke muß dabei bis in direkte Hardwarenähe verfolgt werden, um eine ausreichende Transparenz des Systems zu erreichen.

### 4.2.1 Hardwaremodularität

Die Gerätesteuerungen sind über Sensor- und Aktorelemente direkt mit dem Montageprozeß verbunden. Analog zur Vielzahl von Fügeverfahren existiert eine große Anzahl von unterschiedlichen Prozeßelementen. Diese erfordern in der Regel auch eigene Anschaltungen auf Hardwareebene. Diese Schnittstellen lassen sich sinnvoll nur über Einsteckkarten in einem Bussystem realisieren. Die unterste Ebene in der Hardewaremodularität stellt deshalb ein Rückwandbus dar.

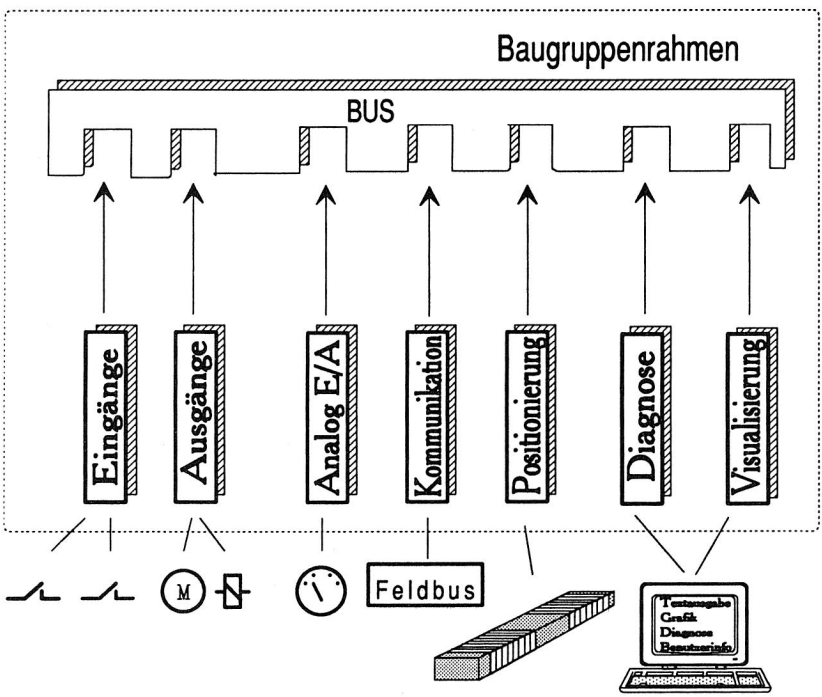



Bild 4.3: Hardwaremodularität mittels Bussystem

Es gibt verschiedenste Ausführungen von parallelen Bussen, jedoch haben sich in der gehobenen Leistungsklasse (mindestens 16 bit Datenwortbreite) mit Multi-Bus und VME-Bus zwei Hauptvertreter herauskristallisiert. Von der Leistungsfähigkeit in etwa vergleichbar, sind sie jedoch mit unterschiedlichen Rechnerfamilien assoziiert (Multibus = Intel, VME-Bus = Motorola). In der industriellen Anwendung hat

der VME-Bus einen deutlichen Vorsprung in der Verbreitung, da für dieses System die weitaus größte Anzahl von Steckkarten der verschiedensten Hersteller erhältlich ist. Davon unberührt gibt es viele Produzenten von abgeschlossenen Steuerungen, die auf eigene Systeme setzen. Im Sinne einer weitergehenden Modularisierung - auch über Herstellergrenzen hinweg - sollte man jedoch zu einem einheitlichen Standard kommen. Zur Zeit entwickelt sich der VME-Bus im Bereich der industriellen Steuerung ebenso zu einem 'de-facto-Standard' /15, 20/, wie der ISA-Bus im Bereich der Personalcomputer (PC).

Gerade das Beispiel der PC's illustriert nachhaltig, wie universell ein Rechner eingesetzt werden kann, wenn nur eine ausreichend breite Palette von Hardwarezusätzen zur Verfügung steht.

Während sich also auf dem Gebiet der Rückwandbusse eine langsame Standardisierung erkennen läßt, sieht die Lage bei der Kopplung verschiedener Steuerrechner, zum Beispiel in einer Montagezelle, nicht so günstig aus. Es konkurrieren auf der Ebene der Feldbusse eine große Zahl von unterschiedlichen Systemen. Die Spanne reicht von Zweidrahtleitungen mit niedriger Übertragungsrate bis hin zu Lichtwellenleiterverbindungen hoher Leistungsfähigkeit. Jedoch selbst bei gleichem physikalischen Medium (RS485) existieren viele verschiedene Protokolldefinitionen. Eine erster Ansatz einer weiterreichenden Normung (MAP), initiiert durch General Motors /52, 67/, ist etwas ins Stocken geraten, da wegen des komplexen Ansatzes sehr hohe Anschlußkosten pro Gerät entstehen. Hinzu kommen natürlich die grundlegenden Probleme beim Versuch, eine weltweite Normung durchzuführen. So entstanden aus der Grundidee von MAP eine Reihe von Folgeprojekten, die alle zum Ziel haben, auf Basis eines Feldbusses (geringere Kosten) eine standardisierte Kommunikation der Geräte einer Zelle mit vertretbarem Aufwand zu realisieren. Dabei werden vereinfachte Architekturen verwendet (EPA = Enhanced Performance Architecture), die einen größeren Durchsatz an Informationen erlauben /21/. Eine Normung ist hierbei noch nicht zu erkennen, wenngleich sich auf der physikalischen Ebene Hauptformen von Feldbussen abzuzeichnen beginnen. In der vorliegenden Arbeit wird deshalb auch das Medium Feldbus als Verbindung in der Zelle in die Konzeptüberlegungen miteinbezogen.

### 4.2.2 Softwaremodule in Gerätesteuerungen

Analog zur Hardware muß auch auf dem Gebiet der Software eine Standardisierung und Modularisierung stattfinden. Ein erster richtungsweisender Ansatz wurde in einem Normentwurf Mehrprozessor-Steuersystem für Arbeitsmaschinen (MPST) /69/ gemacht. Hier werden Module definiert, die Teilaufträge abarbeiten - sogenannte beauftragbare Funktionen.

Wesentlich ist, daß es sich dabei um Aufgaben handelt, die innerhalb heutiger Steuerungen, d.h. im Echtzeitbereich der Steuerungsaufgaben, liegen. Es fehlt zum Komplettsystem jedoch noch die Betrachtung von Verwaltungsaufgaben für die Funktionen im Betriebssystem. Ein wesentlicher Gedanke zur Strukturierung von Softwaresystemen läßt sich damit jedoch schon realisieren - nämlich Teile der Idee der Objektorientierung. Dieses Schlagwort aus dem Fachgebiet der Softwaretechnologie hat in diesem Zusammenhang nicht die volle Bedeutung wie in der sonstigen Rechnerwelt. Unbeachtet bleiben die hierarchischen Abhängigkeiten von Merkmalen eines Objekts (Vererbung). Vielmehr läßt sich die Kernidee extrahieren, daß Informationen verborgen werden und Manipulationsvorschriften (Methoden) mit zum Inhalt eines Objekts gehören. Objekte korrespondieren untereinander also rein über Meldungsverkehr, das heißt Aktivierung von Methoden. Eine unzulässige oder nicht typgerechte Manipulation von Informationen ist dadurch ausgeschlossen. Eine objektorientierte Betrachtung eines Steuerungssystems erscheint deshalb als sinnvolles Strukturierungshilfsmittel für weite Bereiche. Eine Einschränkung der Verwendbarkeit oder eine Erweiterung des Objektbegriffes ist aber in den Teilen einer Steuerung nötig, in denen Echtzeitanforderungen gestellt werden.

## 4.3 Kommunikationsfähigkeit

An den Übergangsstellen der Rechnerhierarchie ist nicht nur ein einheitlicher physikalischer Anschluß an ein Kommunikationssystem notwendig, sondern auch weitgehend einheitliche Semantik. Um eine umfassende Transparenz zu erreichen, müssen system- beziehungsweise steuerungsinterne Zustandsinformationen über ein standardisiertes Kommunikationssystem zugänglich sein. Diese einheitliche Kommunikation kann auch über Feldbusse oder Verbunde von Feldbussen erreicht werden /6/. Für Echtzeitaufgaben ist die Anwenderschnittstelle der

'Manufacturing Message Specification' (MMS) /67/ kaum geeignet, da die erreichbaren Zeiten für einen einzelnen Auftrag noch bis zu 10 ms betragen /26/.

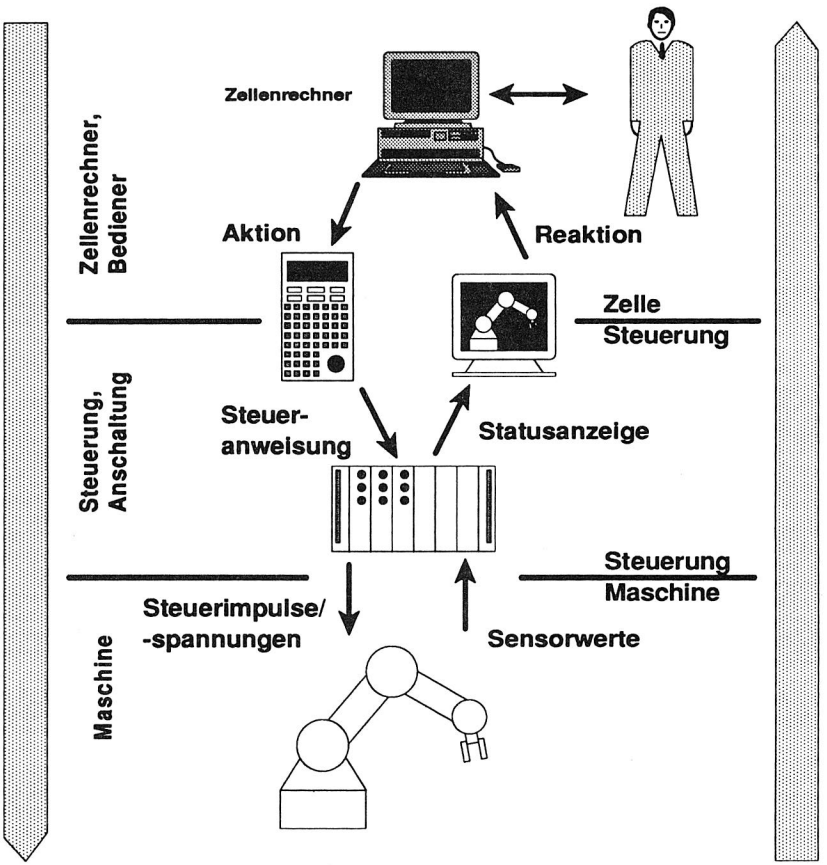



Bild 4.4: Integration in die Rechnerhierarchie

Ungeachtet der Problematik des Entwurfs einer einheitlichen Norm (verschiedene Standards für unterschiedliche Geräte 'weichen die Norm auf') bleibt die Frage der Semantik. Ein erster Teilerfolg kann durch die Festlegung von Variablendiensten (Normen, Formate und Inhalte) für Systemvariablen eines Roboters erreicht werden (Bild 4.5). Durch symbolische Referenzierung können dann diese Werte einheitlich vom Zellenrechner manipuliert werden.

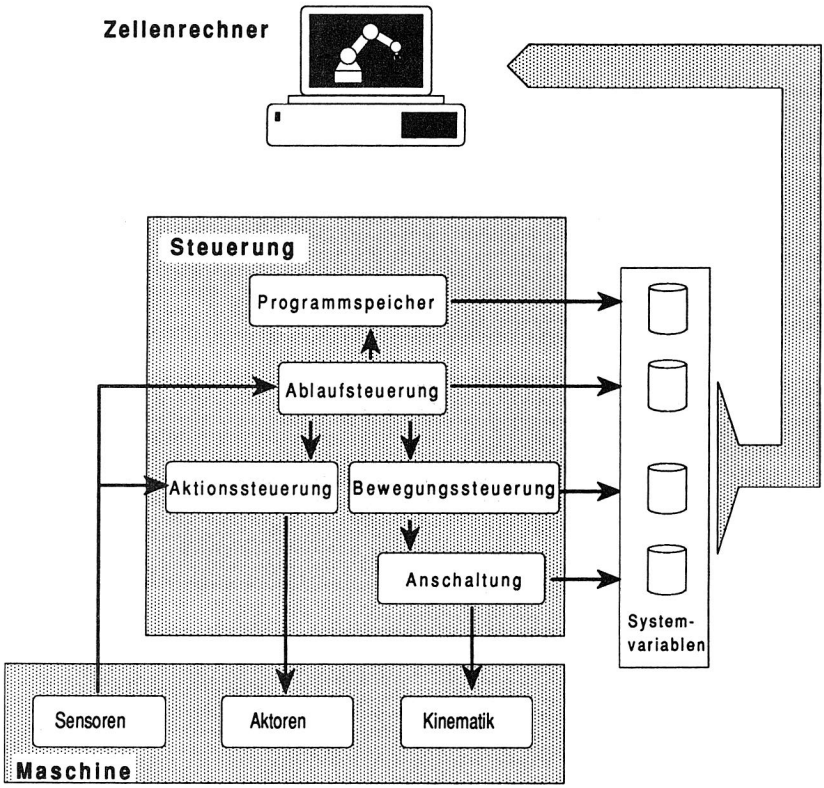



Bild 4.5: Informationstransparenz

Nicht nur Standard-Systemvariablen (zum Beispiel Position, Geschwindigkeit) sollten auf diesem Wege zugänglich sein, sondern auch Variablen, die im Anwendungsprogramm des Roboters definiert sind. Damit lassen sich dann Abläufe in der Zelle besser koordinieren und applikationsspezifische Daten austauschen. In einem weiteren Schritt muß sich die Steuerung als eine Komponente eines verteilten Rechnersystems einer Zelle betreiben lassen.

## 4.4 Zeitanforderungen in Steuerungen

Auf der Ebene der Gerätesteuerungen existiert eine ausgezeichnete Klasse von zeitkritischen Anwendungen. Dazu gehört insbesondere der Betrieb von Montagegeräten, der durch aufwendige Mechanismen zur Steuerung und Regelung des Bewegungsablaufs gekennzeichnet ist. Es resultieren spezielle Anforderungen an das Zeitverhalten des Steuerungssystems /92/. Die Bedeutung von Zeiten beziehungsweise Gleichzeitigkeit wird für Montagegeräte genauer betrachtet.

Für einen Großteil der Steuerungsaufgaben ist die Abarbeitung in Echtzeit gefordert. Unter Echtzeit wird im allgemeinen die Fähigkeit verstanden, innerhalb einer definierten maximalen Zeit auf ein externes Ereignis zu reagieren. Dabei interessiert weniger die Zeit bis zur Aktivierung einer Serviceroutine (Interrupt-Latenz), sondern aus Anwendungssicht die Zeitspanne, bis eine Reaktion an den Ausgängen sichtbar wird. Für Überwachungsaufgaben sind solche Zeitangaben sinnvoll, da es sich um aperiodische Signale handelt.

Viele interne Steuerungsfunktionen eines Montagegeräts sind jedoch periodischer Natur. Im Kern besteht die Steuerung aus einem starr getakteten System.

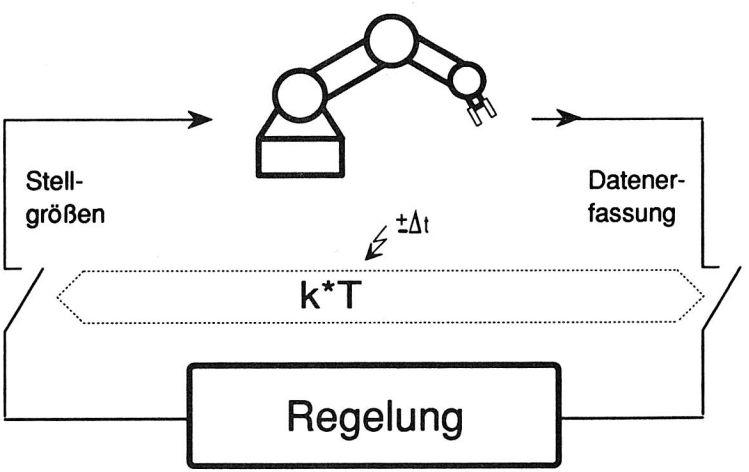



Bild 4.6: Getaktete Verarbeitung in einer Steuerung

Neben der starren Taktung besteht eine weitere Besonderheit im Vergleich zu Standardrechnern. Der Betrieb einer Steuerung ist gekennzeichnet durch verschiedene Phasen unterschiedlicher Auslastung der Rechnerkapazitäten.

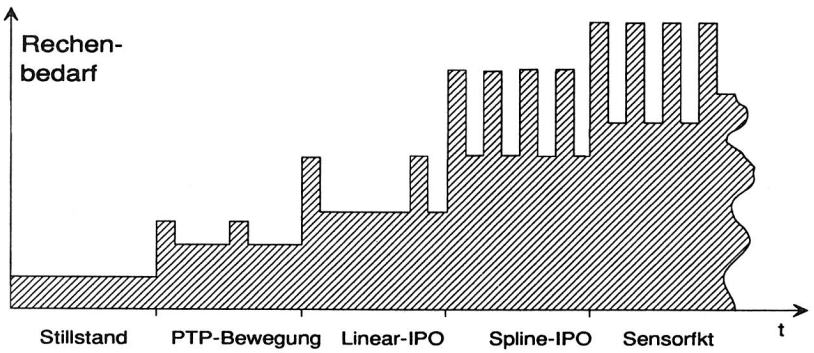



Bild 4.7: Rechnerauslastung während Bewegungen

Es sind nicht nur Grobphasen zu beachten (zum Beispiel unterschiedliche Interpolationsverfahren), sondern zudem noch Belastungsspitzen beim Übergang zwischen einzelnen programmierten Stützpunkten (neuer Programmsatz). Eine kritische Situation ergibt sich, wenn diese sogenannten Satzwechsel sehr dicht aufeinander folgen (rechter Abschnitt im Bild 4.7). Dies ist immer dann der Fall, wenn Kurven mit kleinen Stützpunktabständen und hoher Geschwindigkeit gefahren werden sollen. Also hat auch die jeweilige Anwendung starken Einfluß auf die erforderliche Leistungsfähigkeit der Steuerung.

Der ***'pulsierende Betrieb'*** der Steuerung ist beim Entwurf aller an der Bewegung beteiligten Softwarekomponenten zu beachten. Die Optimierung des Gesamtsystems muß also nicht nach Kriterien, wie zum Beispiel gleich hoher Belastung aller Recheneinheiten erfolgen, sondern muß für die optimale Bearbeitung der Spitzenlasten konfiguriert werden. Da die reale Belastung erst zur Laufzeit feststellbar ist, muß durch Monitoringkomponenten die tatsächliche Be- oder Überlastung erfaßt werden.

Eine Gerätesteuerung ist weiterhin gekennzeichnet durch eine große Anzahl paralleler Aufgaben. Diese lassen sich in drei Hauptkategorien einteilen (Bild 4.8). Zwei zyklische Programmgruppen mit verschiedenen Zykluszeiten $t_1$ und $t_2$

(mit $t_1 = n * t_2$) bilden den Kern, der von aperiodisch arbeitenden Programmen mit Daten und Aufträgen versorgt wird.

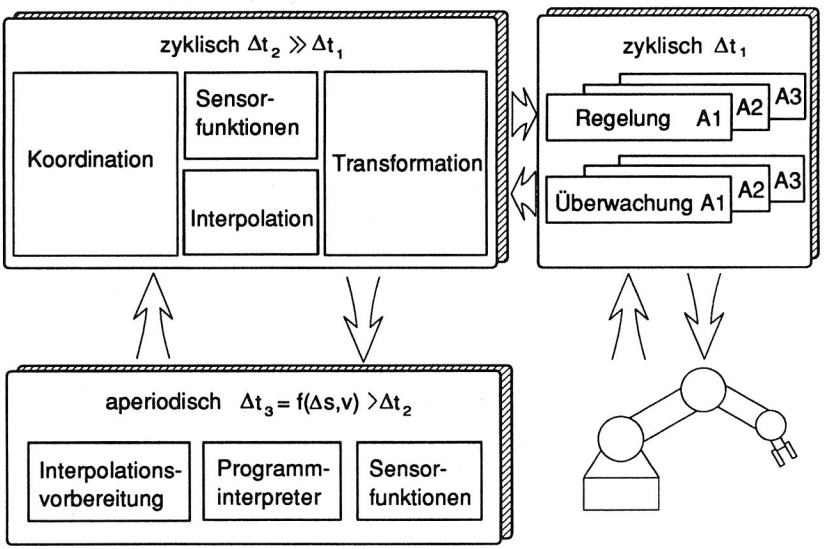



Bild 4.8: Kernfunktionen einer Gerätesteuerung

Die Regelung der einzelnen Achsen ist primär unabhängig voneinander. Auch die Überwachung von Verfahrweg, maximaler Geschwindigkeit und Beschleunigung je Achse können unabhängig ausgeführt werden. Die Aufgaben sind deshalb gut für eine Multiprozessorkonfiguration geeignet. Die zeitkritischsten Steuerungsfunktionen sind dabei durch einen kontinuierlichen Datenfluß verbunden. Es kann deshalb bezüglich der Relevanz zur Erzeugung der Bewegung für die einzelnen Prozesse eine Priorität angegeben werden, die über deren Laufzeit invariant ist.

### 4.4.1 Lageregelung

Alle Regelungsaufgaben und ein Teil der Bewegungsgenerierung laufen in festen Zeitabständen ab. Für Regelungsfunktionen und Bewegungsgenerierung ist die bisherige Echtzeitdefinition nicht ausreichend, sondern muß um die Forderung nach möglichst konstanten Zeitdauern zwischen den einzelnen Durchläufen er-

weitert werden. Eine Verschiebung oder Dehnung des Rasters aufgrund hoher Rechnerbelastung führt zu Fehlern in der Bewegungsgeschwindigkeit und zu Störungen der Reglergüte /51/. Die Forderung nach konstantem, deterministischem Zeitverhalten ist ein Kennzeichen für 'harte Echtzeitforderungen' ($\Delta t \cong 0$ in Bild 4.9).

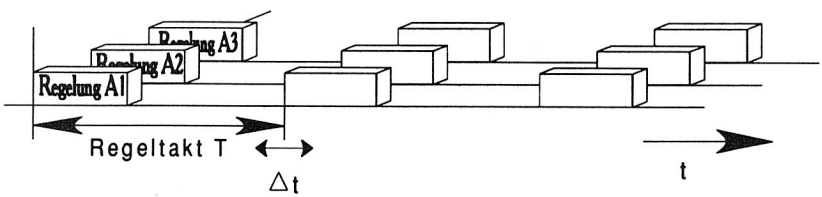



Bild 4.9: Konstanter Regeltakt T

Ein Ausbleiben oder die Verzögerung der Steuerinformationen für einen Systemtakt bewirkt ein Fehlverhalten des Roboters. Der temporäre Stop der Bewegung erzeugt zum Beispiel eine Reaktion der Beschleunigungsüberwachungen der Achsen.

Die folgende Überschlagsrechnung kann als Richtwert für ein Maß der erforderlichen Zeit dienen: Um in einer digitalen Regelung keine Fehler durch die getaktete Arbeitsweise des Reglers zu erhalten, sollte die Abtastzeit deutlich (Faktor 5) unter der Hauptzeitkonstante des zu regelnden Systems liegen. Als ein Näherungswert für einen Montageroboter kann dabei eine Zeit von etwa 0.1 sec dienen. Daraus resultiert ein gefordertes Zeitintervall von maximal 20 msec zwischen zwei Abtastungen. Für kartesische Montagegeräte und hochdynamische Roboter liegt dieser Wert bei etwa 2 msec. In neuesten Entwicklungen wird nicht nur die Lage des Arms, sondern auch die Geschwindigkeit des Antriebs digital geregelt. Hier ist nochmals eine Zehnerpotenz weniger als Abtastzeit gefordert. Diese extremen Anforderungen lassen sich durch eigenständige Subsysteme mit Spezialprozessoren abfangen /11, 40/. Deshalb werden in dieser Arbeit keine Funktionen betrachtet, die der Lageregelung unterlagert sind. Auch die Aspekte von 'höherwertigen Regelungen' (Zustandsregler, Inverses Modell, /46, 62/) lassen sich durch dedizierte Rechnerkomponenten realisieren. Innerhalb der genannten Zeiten für den Lageregeltakt müssen eine Anzahl von Aufgaben durchgeführt

werden, die im Sinne einer guten Modularisierung und funktionalen Trennung als eigenständige Programme realisiert sein sollten.

### 4.4.2 Führungsgrößenerzeugung

Die Generierung von Bewegungsbahnen ist nicht ganz so zeitkritisch, denn hier spielt eine konstante Zeitverschiebung (Totzeit) dann keine Rolle, wenn sie für alle beteiligten Berechnungen gleich ist und es sich nicht um einen geschlossenen Regelkreis handelt. Mittels Spezialprozessoren /81/ können heute schon schnelle kartesische Regelungen (t<1ms) aufgebaut werden. Damit lassen sich höhere Bahngenauigkeiten als mit der bisherigen kartesischen Steuerung und Regelung auf Achsebene erreichen.

Die Taktrate in der Führungsgrößenerzeugung kann etwa um Faktor 10 niedriger ausfallen als in der Regelungsebene. Wünschenswert ist dabei dennoch eine Reduzierung der Verarbeitungszeiten, um bessere Bahngenauigkeiten zu erreichen. Die Berechnung von Bahnen für den Roboter und die Transformationen von kartesischen Koordinaten in das Roboter-Koordinatensystem sind geprägt von umfangreichen mathematischen Berechnungen mit einer großen Anzahl von trigonometrischen Rechenanweisungen /62/. Der Führungsgrößenrechner muß daher so dimensioniert werden, daß er Spitzenlasten mit mathematischen Funktionen verarbeiten kann. Zudem ist eine ausreichende Genauigkeit der Zahlendarstellung für alle Bahnberechnungen zu fordern. Als unterer Grenzwert für die Darstellung von Gleitkommazahlen können 32 bit angesehen werden, zumal effektiv nur etwa 20 bit Genauigkeit resultieren /82/. Für komplexere Bewegungsbahnen (Kreise, Spline) ist diese Auflösung nicht mehr ausreichend. Sinnvoll und meist auch notwendig ist deshalb der Einsatz einer Darstellungsform mit 64 bit. Damit verbleibt auch bei längeren Rechenausdrücken oder in der Umgebung von mathematisch singulären Stellen eine ausreichende Genauigkeit erhalten.

Zur Sicherung des Gesamtverhaltens und zur Erkennung einer eventuellen Überlastung sind kritische Zeiten in der Führungsgrößenerzeugung vom Betriebssystem zu überwachen.

## 4.5 Koordination und Synchronisation innerhalb der Steuerung

Im Sinne einer feinen Granularität aller Steuerungselemente ist die Gesamtaufgabe in möglichst viele einzelne Programme zu zerlegen. Für eine Standardanwendung eines Montagegerätes kann deshalb von etwa 20 Taskwechseln pro Robotersystemtakt ausgegangen werden, die ein Betriebssystem durchführen muß. Wesentliche Aufgabe der Synchronisation ist der gleichzeitige Start all der Programme, die einen direkten Einfluß auf die aktuelle Aktion haben. Aber auch der Stop, beziehungsweise Abbruch von Programmen sollte im Fehlerfall gleichzeitig geschehen.

**Gleichzeitige oder synchrone Bearbeitung heißt Ausführung im gleichen Regel- oder Interpolationszyklus.**

Die 'gleichzeitige' Bearbeitung ist auch mit nur einem Prozessor möglich, da die ermittelten neuen Sollvorgaben an den Roboter erst am Ende eines Takts ausgegeben werden oder die Ausgabe erst zu Beginn des nächsten Zyklus erfolgt.

Eine weitere Besonderheit von Steuerungssystemen ist die Notwendigkeit, Programme abzubrechen und dabei noch bestimmte Aktionen durchzuführen, die vom aktuellen Systemzustand abhängig sind. Durch den funktionalen Verbund von Einzelprogrammen kann es vorkommen, daß an irgendeiner Stelle im Gesamtsystem ein Fehler detektiert wird, der Auswirkungen auch auf andere Module hat. Es muß deshalb möglich sein, Programmen extern die sofortige Beendigung der aktuellen Aktion zu signalisieren (zum Beispiel der Bewegungsgenerierung). Das Programm kann dann seine internen Variablen auf einen konsistenten Zustand bringen. Ein Abbruch ohne Datensicherung führt beim Versuch, die Anwendung fortzusetzen, in aller Regel zu einem Fehlverhalten.

## 4.6 Integration von Sensorinformation

Sensoren werden in der Fertigungstechnik für ein breites Anwendungsgebiet benutzt /80/. Ähnlich vielschichtig sind die Möglichkeiten, Sensordaten in Gerätesteuerungen einzusetzen. Haupteinsatz ist die Steuerung des Ablaufs von Fertigungsaufgaben. Dazu sind Bearbeitungsprogramme oder Abschnitte aus-

zuwählen und eventuell auch noch zu korrigieren. Der zweite Bereich umfaßt die überwachenden und regelnden Anwendungen. Hier sind schnelle Datenpfade und ein möglichst direkter Eingriff der Sensorinformationen notwendig, da es sich entweder um einen geschlossenen Regelkreis handelt oder Prozeßüberwachungen zeitkritischen Charakter besitzen.

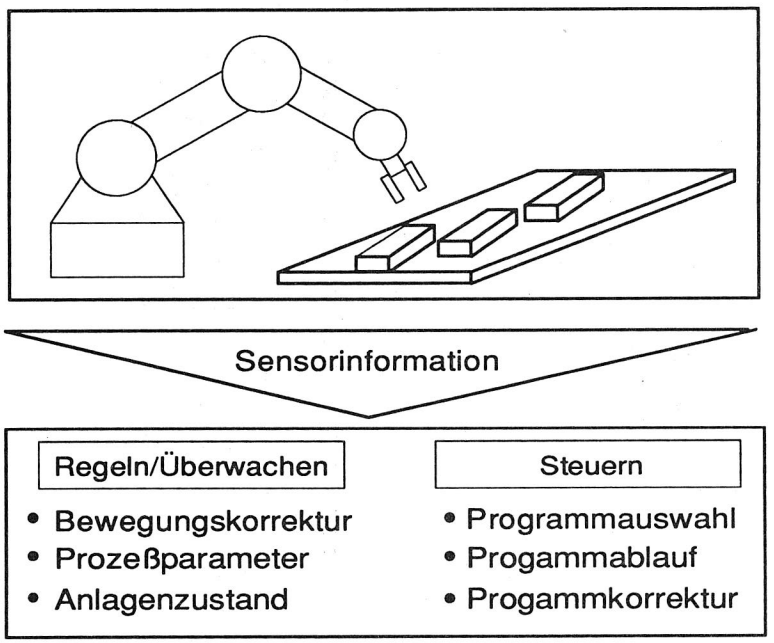



Bild 4.10: Sensoraufgaben in Gerätesteuerungen

Die Integration von Sensorinformation kann auf verschiedenen Ebenen der Gerätesteuerung relevant sein. Bei der Prozeßführung innerhalb der Steuerung von Montagegeräten sind die in Bild 4.11 gezeigten Eingriffspunkte zu berücksichtigen. Die notwendige Sensorinformation kann von einem Sensor geliefert werden oder aber auch durch die Fusion mehrerer Informationsquellen entstehen. Die Sensordaten können auf oberster Ebene dazu verwendet werden, um die nächsten Aktionen zu planen. Dazu wird versucht, ein möglichst vollständiges Umweltmodell der Zelle zu erhalten. In der nächsten Stufe können die gewonnenen Informationen dazu verwendet werden, einzelne Bewegungen zu planen. Vor der Ausführung können in der Präparationsphase letzte statische Korrekturen

an den Zielvorgaben durchgeführt werden. In der Ausführung besteht die Möglichkeit zur Beeinflussung der gefahrenen Bahn, entweder auf Ebene der kartesischen Vorgaben (Interpolation) oder direkt auf Achsebene. Die Zeitanforderungen an die Sensordatenverarbeitung steigen dabei in der Reihenfolge der Aufzählung.

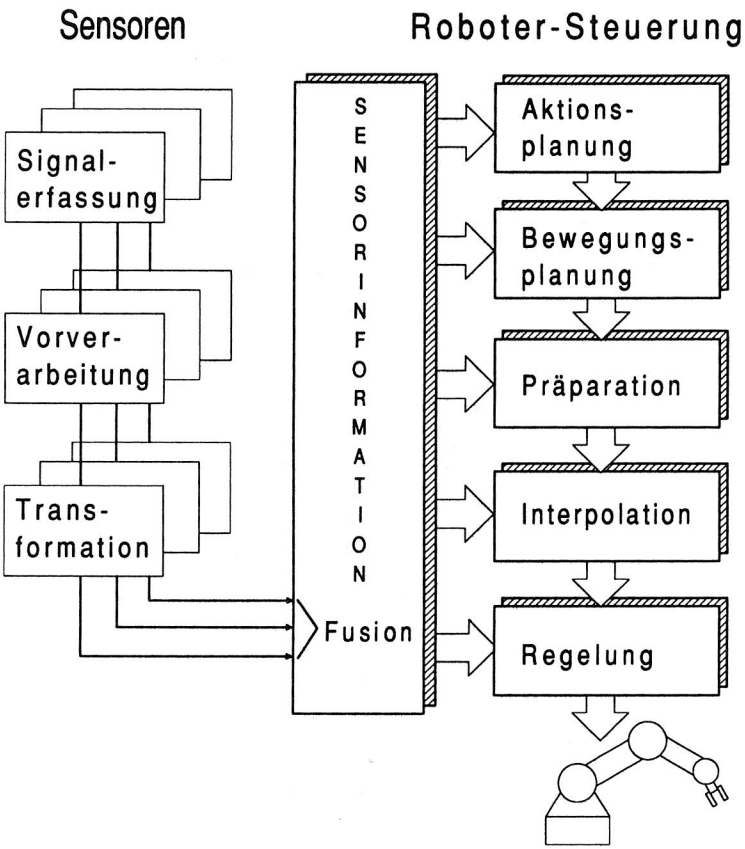



Bild 4.11: Hierarchische Sensorkorrekturen

Bei der Bearbeitung von Sensordaten spielt die Summe aller Verzögerungen im Datenpfad dann eine Rolle, wenn es sich um einen geschlossenen Regelkreis handelt und die Totzeit die Regelungsgüte beeinflußt /101/. Auch für die Bahngenerierung sind deshalb kurze Taktzeiten zu fordern.

## 4.7 Koordination mit anderen Geräten

Die Integration von Robotern in flexiblen Fertigungszellen bedingt eine Kooperation mit anderen Elementen der Zelle. Für mehrere Roboter sind drei wesentliche Betriebsarten denkbar:

- unabhängiger Betrieb,
- unabhängiger Betrieb mit Kollisionsüberwachung = synchronisierter Betrieb,
- koordinierter Betrieb.

Unter synchronisiertem Betrieb ist in diesem Zusammenhang der gleichzeitige Start oder Ausschluß von Operationen oder Bewegungen zu verstehen. Bei Koordination sollen sich die Bewegungen zu jedem Zeitpunkt in einer definierten Relation zueinander befinden.

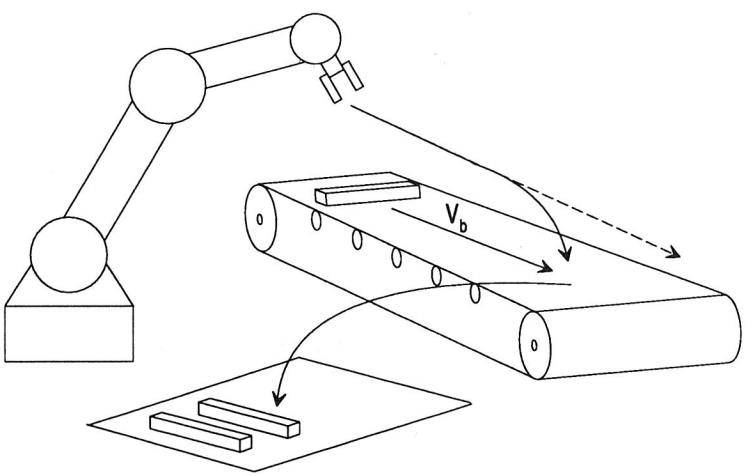



Bild 4.12: Koordination mit einem Transportband

Eine weitverbreitete Anwendung ist die Koordination mit einem Transportband (Bild 4.12). Dazu ist der zyklische Transfer der aktuellen Koordinaten des Bandes im Takt der Robotersteuerung notwendig. Ähnliche Anforderungen an die Kommunikation entstehen bei der Bearbeitung von Montageaufgaben mit mehreren Robotern.

# 5 Konzept einer universellen Gerätesteuerung

In der Montage möchte man Roboter als möglichst universelle Automatisierungsgeräte einsetzen. So vielfältig wie die Montageaufgaben sind deshalb auch die eingesetzten Geräte und die jeweils erforderlichen Funktionen.

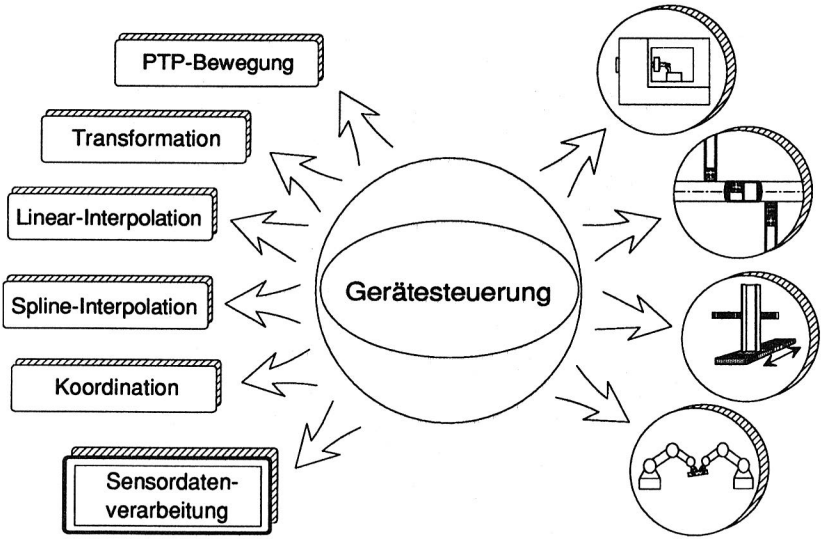



Bild 5.1: Steuerungsaufgaben für unterschiedliche Montagegeräte

Das Spektrum reicht von einfachen Einachsgeräten bis hin zu Systemen mit Doppelrobotern oder auch Geräten mit einer großen Anzahl von Achsen /22/.

Eine Steuerung, die möglichst viele Anwendungsgebiete abdeckt, muß daher variabel im Hardwareausbau, der vorhandenen Rechenleistung und im Funktionsumfang sein. Beim Entwurf ist außerdem die Integration der Einzelsteuerung in das Gesamtsystem 'Zelle' zu beachten. Nur so kann ein flexibles Gesamt-Steuerungssystem einer Zelle oder Anlage entstehen.

## 5.1 Hardwarestrukturen

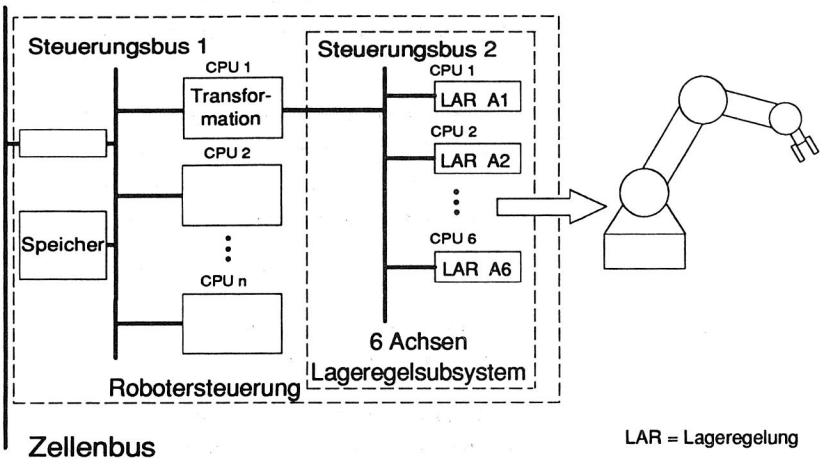



Bild 5.2: Allgemeine interne Busstruktur für Robotersteuerungen

Ein hierarchischer Aufbau aus parallelen Bussen ist die ideale Struktur für eine Robotersteuerung, da sich damit die Ebenen Bewegungsführung und Regelung klar trennen lassen und eine Übertragungsleistung möglich ist. Für die beiden Bereiche können so jeweils optimierte Rechnerkomponenten eingesetzt werden /1/.

In extremen Anwendungen wie zum Beispiel Roboter in der Raumfahrt sollten redundante Konfigurationen von Verbindungen verwendet werden, um das System ausfallsicher zu machen. Der erhöhte Aufwand für die parallele Kommunikation muß dabei durch eigene Prozessoren abgefangen werden.

Ein paralleler Bus ist zwischen Rechnerknoten realisierbar, die innerhalb eines Baugruppenträgers angeordnet sind. Mit marktüblichen Bussystemen lassen sich heute schon hohe Datenraten erreichen /20/. Der Vorteil dieser Rückwandbusse liegt in der Möglichkeit zum DMA-Betrieb (Direct-Memory-Access), das heißt, die Adreßbereiche aller am Rückwandbus angeschlossener Recheneinheiten werden zu einem großen Adreßbereich zusammengefaßt. Jede am Bus angeschlossene Recheneinheit kann über den Bus jede angeschlossene Speicherzelle lesen und beschreiben.

Somit können von allen Teilnehmern gemeinsam genutzte Speicherbereiche (Shared Memory) eingerichtet werden. Mit Hilfe dieser Bereiche können einzelne Tasks rechnerübergreifend miteinander kommunizieren. Neben dem schnellen Datenaustausch können Tasks über diesen Parallelbus auch synchronisiert werden.

Der Zusammenschluß von mehreren Rechnern wird häufig als Cluster bezeichnet. Unter einem Cluster wird üblicherweise eine Menge lose miteinander gekoppelter CPU's verstanden, die über einen gemeinsamen Massenspeicher verfügen. Diese Rechner sind meist von einem Hersteller und/oder von der gleichen Bauart.

Der Begriff Cluster wird für die Betrachtungen in dieser Arbeit wie folgt definiert:

Ein **Cluster** ist ein Zusammenschluß von ein oder mehreren Prozessoren, die über einen gemeinsamen Speicher und Multitaskingfähigkeiten verfügen.

Diese Definition wurde gewählt, da in dieser Arbeit primär Rechnersysteme mit hoher Leistung betrachtet werden. Der schnellste Zusammenschluß zweier Prozessoren ist der Anschluß an einen parallelen Bus und die Kommunikation über gemeinsamen Speicher (Shared Memory).

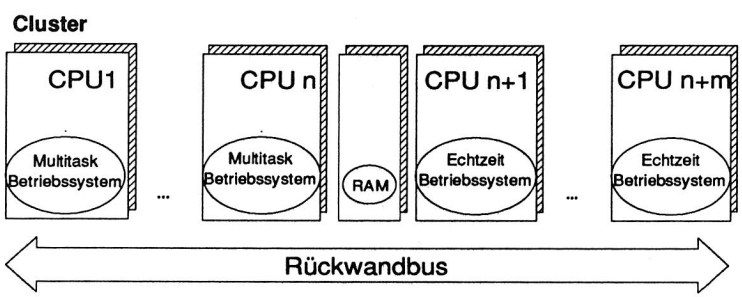



Bild 5.3: Prinzipieller Aufbau eines Clusters

Die Minimalausführung eines Clusters besteht aus einem Prozessor, der mit einem Multitaskbetriebssystem ausgestattet ist. Kommt ein weiterer Prozessor hinzu, so ist das Cluster um den gemeinsamen Speicher und um den Rückwandbus zu erweitern (Bild 5.3).

## 5.2 Hybrides Rechnerkonzept

In modernen Gerätesteuerungen werden Funktionalitäten gefordert, die von Steuer- und Regelungsaufgaben bis hin zu Datenverwaltung und Bedienerführung reichen. Für die resultierenden Aufgabenfelder einer Steuerung sind deshalb divergierende Unterstützungsfunktionen des Betriebssystems notwendig. Im Bereich der dispositiven und graphischen Bereiche ist es sinnvoll, ein Standardbetriebssystem einzusetzen (zum Beispiel UNIX™). Dieses bietet jedoch nicht die geforderte Effizienz, die im eigentlichen Steuerungsteil notwendig ist.

Eine Realisierung aller geforderten Funktionen auf einem Rechner ist nicht sinnvoll durchzuführen. Die Versuche, mit "Untersätzen" oder "Aufsätzen" eine gewisse Echtzeitfähigkeit bei Standardsystemen zu erreichen, hat nur bedingt Erfolg /36, 53/. Dies liegt vor allem daran, daß die verwendeten Betriebssysteme relativ große Abschnitte besitzen, in denen sie nicht unterbrechbar sind. Dies gilt in besonderem Maß für ein UNIX™-System.

Eine Trennung der Aufgabenfelder und die Zuordnung zu getrennten Prozessoren bietet die Möglichkeit, diese Engpässe zu umgehen /35/. Deshalb wird die Aufteilung auf mindestens zwei unterschiedliche, aber ähnliche Rechner und Betriebssysteme vorgeschlagen. Diese Zweiteilung entspricht der Vorgehensweise bei Hybridrechnern, die Vorteile von Analog- und Digitalrechner verbinden. Im Rahmen dieser Arbeit wird für den Begriff Hybridrechner folgende Definition gewählt:

Ein **Hybridrechner** ist ein Rechnersystem, das über mindestens zwei Prozessoren mit einem gemeinsamen Speicher verfügt und mindestens je ein Echtzeit- und ein Standardbetriebssystem auf getrennten Prozessoren besitzt.

Zwischen den beiden Betriebssystemteilen sind eigene Kommunikationsmechanismen notwendig, die eine Verbindung von Programmen ermöglichen. Aus Gründen der Effizienz sollten die verwendeten Rechner die gleiche Repräsentation der Datenstrukturen im gemeinsamen Speicher verwenden. Damit entfallen aufwendige Konversionenm, und es wird ein direkter strukturierter Zugriff auf die gemeinsamen Daten ermöglicht.

Die Realisierung eines gemeinsamen Speichers macht das Kopieren von Informationen unnötig und erleichtert die Konsistenzerhaltung der Daten. Moderne Mikroprozessoren unterstützen dabei die Implementierung von Zugriffsverwaltungen

durch leistungsfähige Koordinierungsanweisungen im Befehlssatz /45/. Damit lassen sich zum Beispiel gegenseitige Ausschlüsse ohne aufwendige Software-Protokolle umsetzen.

## 5.3 Echtzeit-Betriebssystem

Unter Echtzeitbedingungen werden im folgenden die harten Anforderungen der Steuerungsebene verstanden. Der Betriebssytemkern muß die Koexistenz von mehreren abgeschlossenen Programmen (Tasks) effizient verwalten können. Zur Realisierung sind zwei konzeptionelle Ansätze entscheidend. Der eigentliche Systemkern, der nicht unterbrechbar ist, muß möglichst kurz gehalten werden. Die zweite Überlegung ist die Auslagerung von "höherwertigen" Funktionen des Betriebssystems in eigene Tasks (Servertask). Damit lassen sich kurze Antwortzeiten bei vielen Systemaufrufen erreichen, da die eigentliche Ausführung des Auftrags durch eine Servertask realisiert wird. Die Schnittstelle stellen dabei Bibliotheksaufrufe dar.

Eine essentielle Forderung ist die Möglichkeit zur Unterbringung des Systemkerns und der Tasks in einem Festwertspeicher (ROM-fähig). Nach dem Einschalten durchsucht eine spezielle Initialisierungsroutine den Speicher nach Tasks, die zur Identifikation mit einem eindeutigen Taskkopf (Header) versehen sein müssen. Mit den Informationen aus diesen Headern können die Systemtabellen zur Task- und Speicherverwaltung initialisiert werden.

Der Systemkern sollte weiterhin eine einfache Struktur des Taskmanagements besitzen, um eine einfache Synchronisation des Schedulingablaufs zu ermöglichen (vgl. Kap. 5.5).

Für die Gerätesteuerung wird eine Aufteilung in vier wesentliche Prozeßebenen vorgenommen. Die Ebene mit der höchsten Priorität und im allgemeinen auch der kürzesten Taktrate umfaßt Aufgaben der Regelung und Überwachung. In der nächsten Ebene befinden sich all jene Tasks, die unmittelbar die Erzeugung von Führungsgrößen der Bewegung durchführen. Dies sind vor allem Interpolations- und Transformationsfunktionen. Aber auch Sensorfunktionen zur Bahnregelung und zur dynamischen Korrektur von Bewegungsabläufen fallen in diese Kategorie.

Für die bisher genannten Ebenen gilt die Forderung, daß alle involvierten Tasks innerhalb einer starren Zeitvorgabe ihre Berechnungen abgeschlossen haben müssen. In der nächst niederen Ebene befinden sich Tasks, die mittelbar Bewegungen beeinflussen. Vorbereitende Berechnungen von Bahnparametern und Korrekturen von Bewegungsstützpunkten (auch durch Sensoren) sind hier zu nennen. In dieser Ebene von Tasks ist die zur Verfügung stehende maximale Bearbeitungszeit abhängig von der jeweiligen Anwendung des Montagegeräts. Ausschlaggebend ist der zeitliche Abstand, in dem einzelne Bahnstützpunkte durchfahren werden (Stützpunktabstand, Verfahrgeschwindigkeit), die Komplexität der Interpolationsart und der gleichzeitige Aufwand zur Berechnung von Sensorkorrekturen (vgl. Bild 4.7).

Eine Überlastung des Rechners kann dabei von den überlagerten Tasks durch das Fehlen von Daten festgestellt werden. Um eine weitere Stufung der verbleibenden Aufgaben durchführen zu können (zum Beispiel Interpreter, Anzeige, Hintergrundaufgaben), sind weitere zwei Prioritätsebenen vorzusehen. Da innerhalb einer Prioritätsebene vielfach Abhängigkeiten im Datenfluß bestehen (Erzeuger -> Verbraucher) kann in einer Ebene ein sequentielles Bearbeiten der Tasks erfolgen (Round-Robin-Strategie). Der Ablauf eines Scheduling-Ablaufs für die geforderten sechs Prioritätsebenen mit je 32 Tasks in jeder Ebene zeigt Bild 5.4.

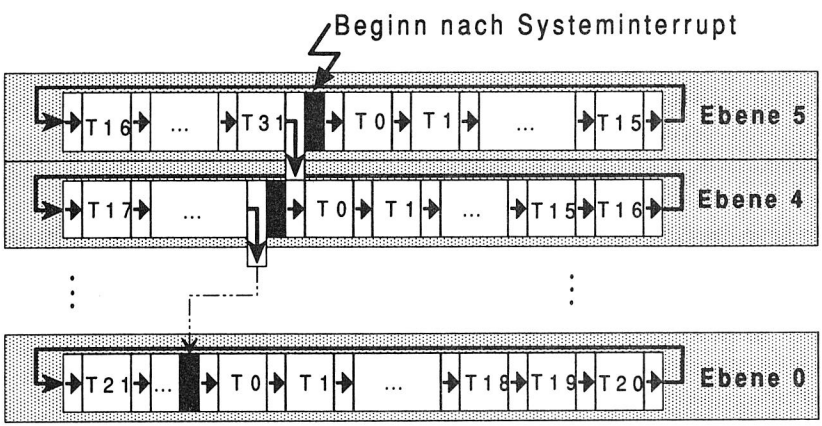



Bild 5.4: Scheduling (Ebenen/Tasks)

Mit den so maximal möglichen 192 Tasks ist eine feine Gliederung aller Steuerungsaufgaben möglich. Der Ansatz mit den nach Taktraten geordneten Prioritätsebenen wird durch theoretische Untersuchungen für periodische Programme gestützt /63/. Diese haben eine vergleichsweise hohe mögliche Auslastung des Rechners (70 %) ohne Verletzung von Zeitbedingungen ergeben.

### 5.3.1 Task-Verwaltung

Der Verwaltung von Tasks einer Gerätesteuerung kommt große Bedeutung zu, dient sie doch der Adaption des Geräts an wechselnde Aufgabenstellungen. Prinzipiell müssen sich alle Programme schon vor dem Aufruf im Hauptspeicher befinden. Das Ein- und Auslagern auf Massenspeicher kostet zuviel Zeit, wenn es in der üblichen Art und Weise abläuft. Normalerweise wird durch ein hochpriores Modul (Lader, Swapper) des Betriebssystems der Austausch von Programmen zum Zeitpunkt des Aufrufs realisiert. Während eines Ladevorgangs ist damit das Echtzeitverhalten des Systems gestört. Wenn in Steuerungen auch der Großteil der Tasks zum festen Funktionsumfang gehört und deshalb im Festwert-Programmspeicher abgelegt werden kann, so gibt es dennoch viele Funktionen, die änderbar sein sollten. Das Laden und das Nachladen von Programmen muß deshalb unterstützt werden. Im Falle einer unaktiven Steuerung stellt dies kein großes Problem dar, da kaum zeitliche Restriktionen zu beachten sind.

Sollen Programme zur Laufzeit hinzugefügt werden, so muß darauf geachtet werden, daß das restliche System durch den Ladevorgang nicht beeinträchtigt wird. Dies bedeutet insbesondere eine Notwendigkeit zur Durchführung des eigentlichen Ladens mit relativ niedriger Priorität.

Die Aktualisierung der Tasktabellen des Systems muß dann schnell erfolgen, sobald sich die Task im Speicher befindet. Die Verwendung von festen Tabellen statt Listen erleichtert diese Aufgabe, da keine aufwendige Daten-Reorganisation durchgeführt werden muß. Unter den angesprochenen Voraussetzungen kann das Laden oder auch der Austausch von Tasks durch eine niederpriore Servertask (*Taskverwalter*) ausgeführt werden. Gleichzeitig kann der *Taskverwalter* die Aufgabe eines Auskunftdienstes (*Nameserver*) übernehmen und auf Anfrage Angaben über die vorhandenen Tasks übermitteln. Dies ist vor allem für die Kommunikation zwischen Programmen wichtig (Kap. 5.4).

### 5.3.2 Tasksteuerung

Der quasi-parallele Ablauf von mehreren Programmen (Multitasking) erfordert ein Verfahren, welches eine effiziente Umschaltung zwischen den einzelnen Tasks ermöglicht. Der zugrunde liegende Algorithmus stellt ein prioritätsgesteuertes 'preemptive scheduling' dar (vgl. Kap. 2.1). Zu einem festen Zeitpunkt (Takt) wird die Abarbeitung unterbrochen und in die höchste Prioritätsebene gewechselt. Durch einfache Entwurfsregeln bei der Programmierung von Tasks kann innerhalb einer Prioritätsebene ein Round-Robin Verfahren realisiert werden, ohne daß eine komplette Kontextumschaltung durchgeführt wird. Es ist also kein Zeitscheibenprinzip zur Prozessorvergabe notwendig.

Die Strukturierung von Tasks in kurze Teilabschnitte gestaltet sich für viele Anwendungsaufgaben einer Gerätesteuerung einfach, da es sich ohnehin um iterative Prozesse handelt. Das Verlassen einer Iterationsschleife (= Rückkehr zur Tasksteuerung) muß deshalb durch effiziente Systemaufrufe unterstützt werden, damit die Fortsetzung ohne großen Zusatzaufwand im nächsten Regelungstakt möglich ist.

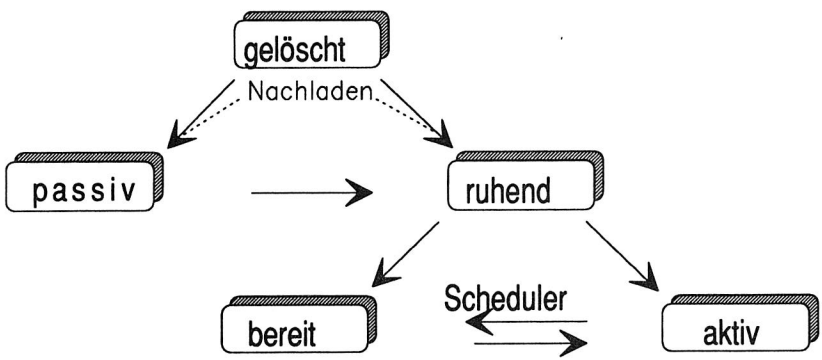



Bild 5.5: Taskzustände im Echtzeitsystem

Neben den üblichen Zuständen von Programmen macht die Art des Nachladens einen neuen Zwischenzustand erforderlich. Dieser ist in Bild 5.5 mit *passiv* gekennzeichnet. Das entsprechende Programm befindet sich zwar im Hauptspeicher, ist aber nur für den *Taskverwalter* und nicht direkt für das Betriebssystem zugänglich (vgl. Kap 5.3.1).

Dem einfachen Entwurf der Tasksteuerung entsprechend sind daneben nur wenige Taskzustände vom Systemkern zu berücksichtigen. Weitergehende Funktionalität wird durch Bibliotheksroutinen und eigene Dienstprogramme realisiert.

Es ist jedoch eine besondere **Funktion zum Abbruch einer Task** vorzusehen. Aus der Anwendung heraus kann es notwendig sein, eine Task zu einem beliebigen Zeitpunkt abzubrechen und in einen definierten Zustand zu überführen.

Die notwendigen Operationen zum Rücksetzen sind im allgemeinen nur der Task selbst bekannt und können zudem auch noch von der Vorgeschichte abhängig sein. Es muß demnach möglich sein, eine in der Task definierte Abbruchroutine von außen zu aktivieren. Diese Routine kann im Minimalfall keine Operationen enthalten und ermöglicht andererseits die Herstellung eines konsistenten Zustands aller internen Daten einer Task.

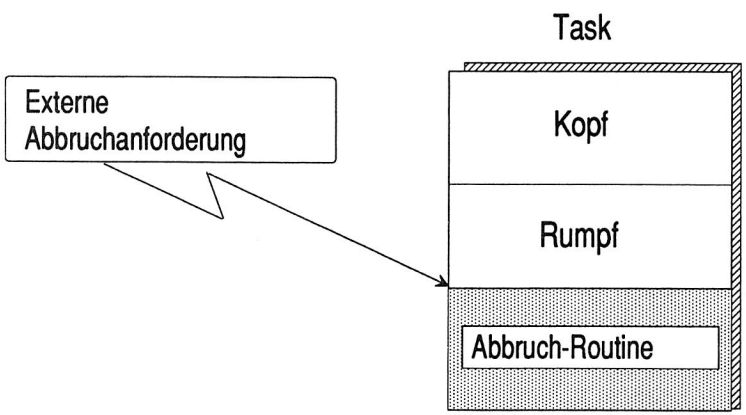



Bild 5.6: Taskabbruch

Da sich alle Programme mit ihren Daten permanent im Hauptspeicher befinden, bedeutet eine spätere Aktivierung nicht zwangsläufig einen Neustart, sondern erlaubt zum Beispiel auch die Fortsetzung an definierter Stelle mit vorher festgelegten Parametern. Ein klassisches Beispiel für die Notwendigkeit eines solchen Vorgehens ist die Fortsetzung einer Bewegungsbahn, die mit Hilfe einer Sensorregelung durchfahren wurde. Eine genaue Fortsetzung läßt sich nur erreichen, wenn die in der Regelungs-Task vor dem Abbruch gültigen Korrekturwerte und Zustandsparameter für das Wiederaufsetzen beibehalten werden.

### 5.3.3 Zeitdienste

Innerhalb einer Steuerung sind viele Vorgänge zeitbehaftet oder bedürfen einer zeitlichen Überwachung. Die erforderliche Funktionalität läßt sich durch entsprechende Routinen mit Unterstützung durch eine Task erreichen, welche die Zeit-Aufträge verwaltet (*Timeserver*).

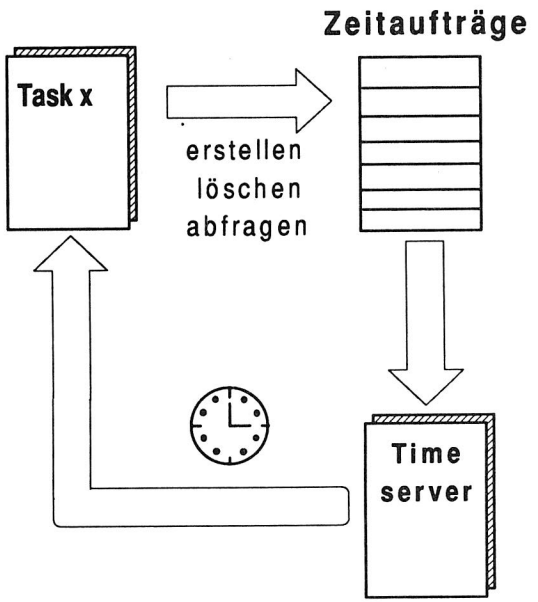



Bild 5.7: Zeitdienste über eigenes Dienstprogramm

Es lassen sich dabei zwei Hauptanwendungen identifizieren. Die erste Anwendung ist das Pausieren einer Task (delay, suspend) für einen bestimmten Zeitraum. Hierbei muß der *Timeserver* nach Ablauf der vorgegebenen Zeit dafür sorgen, daß die entsprechende Task wieder angefordert wird, das heißt im gleichen Zyklus wieder aktiviert wird.

Die zweite Anwendung ist die Überwachung von Zeitlimits durch eine Task. Dazu muß es möglich sein, Zeitzähler zu setzen, zu löschen oder auch abzufragen. Prinzipiell sollte der Ablauf eines Zeitzählers durch den *Timeserver* direkt an die jeweilige Task gemeldet werden (Bild 5.7).

Dies kann am effizientesten durch die Manipulation von lokalen Variablen einer Task erfolgen, deren Referenz dem *Timeserver* im Auftrag mitgeteilt wurde. Der Wert dieser Variablen kann dann direkt zur Steuerung des internen Ablaufs einer Task verwendet werden. Die Realisierung seitens des *Timeservers* erfordert ebenfalls nur geringen Aufwand. Es lassen sich damit sehr gut Zeitintervalle, wie zum Beispiel die Reaktionszeiten anderer Tasks, überwachen. Auf jedem Rechnerknoten ist lokal ein solches Serverprogramm vorhanden.

## 5.4 Programmkommunikation

Um ein verteiltes Echtzeitsystem zu unterstützen, ist es notwendig, mehrere unterschiedliche Kommunikationsmöglichkeiten zur Verfügung zu stellen. Diese unterscheiden sich in der räumlichen Distanz der Partner und in den erreichbaren Geschwindigkeiten.

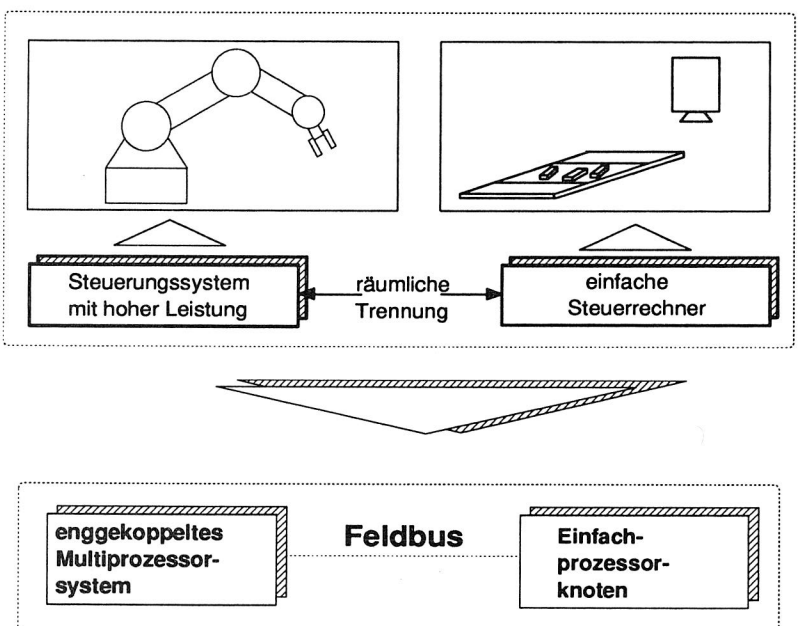



Bild 5.8: Verteilte Gerätesteuerung

In diesem Konzept werden drei verschiedene Kommunikationsarten angeboten:

- CPU-intern,
- Cluster-intern,
- Cluster-extern.

Die Unterstützung ist notwendig, um für verschiedene Konfigurationen (Allocation von Tasks) optimale Kommunikationspfade zu schaffen.

### 5.4.1 Lokale Kommunikation

Die lokale Kommunikation wird von jedem Multitasking-Betriebssystem mit Systemdirektiven unterstützt. Der Datenaustausch erfolgt dabei über einen gemeinsamen lokalen Speicherbereich der Kommunikationspartner. Der Speicherbereich kann über eine Systemdirektive (*malloc()*) temporär angefordert, und die Adresse wird in der Initialisierungsphase dem Partner mitgeteilt werden. Dies ist die schnellste Art der Kommunikation und wird im folgenden mit *FAST_CALL* bezeichnet.

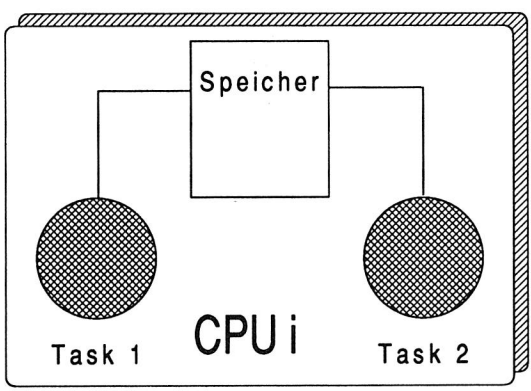



Bild 5.9: Kommunikation über gemeinsamen lokalen Speicher

Die Kommunikation kann auch über einen Meldungsbereich im gemeinsamen Speicher (shared memory) aller Rechnerkarten erfolgen. Dies ist der normale Kommunikationspfad und wird im folgenden mit *NORMAL_CALL* bezeichnet

(Bild 5.10). Zusätzlich zum *FAST_CALL* wird hier noch die Zeit für die Zustellung der Nachricht durch einen Kommunikationsdienst benötigt.

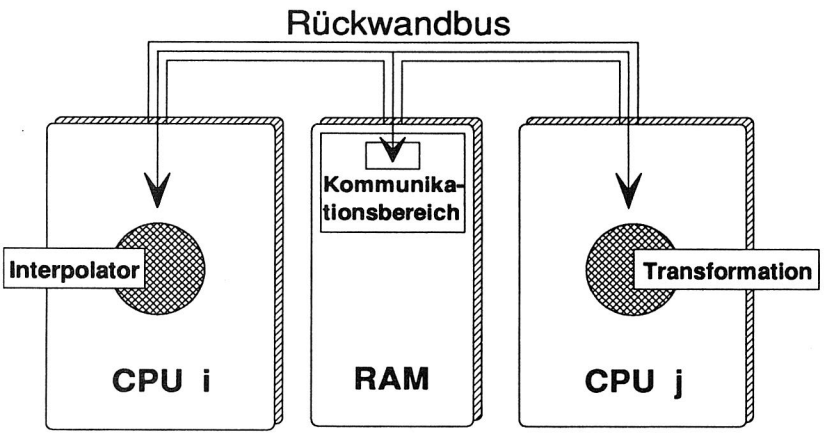



Bild 5.10: Kommunikation über globalen Speicher

Der Programmierer hat die Möglichkeit, zwischen temporärem Meldungspuffer und statischem Puffer zu unterscheiden. Je nach beabsichtigter Nutzung durch die Kommunikationspartner (einzelne Meldung oder dauerhafte Verbindung) kann der passende Typ verwendet werden. Über einen Aufrufparameter wird dem Kommunikations-Dienstprogramm der Typ mitgeteilt.

### 5.4.2 Zellenweite Kommunikation

Die clusterexterne zellenweite Kommunikation erfolgt über den Feldbus der Zelle. Dieser übernimmt die Aufgabe, die Meldungen vom Cluster *A* in den entsprechenden Bereich des Clusters *B* zu transportieren.

Bei dieser Variante entfällt die Möglichkeit des direkten Speicherzugriffs, da der Transfer über den Feldbus realisiert werden muß. Die Koordinierung muß demnach durch Protokolle in der Software realisiert werden. Für kleine Speicherbereiche läßt sich ein virtueller gemeinsamer Speicher auch zwischen verschiedenen Clustern aufbauen. Dieses Verfahren wird im folgenden mit *EXTERN_CALL* bezeichnet und sollte nur verwendet werden, um einem Prozeß in einem anderen Cluster eine Nachricht zu schicken.

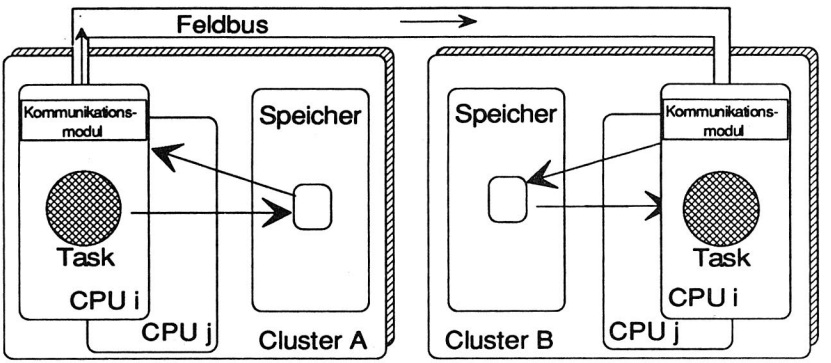



Bild 5.11: Clusterexterne Kommunikation

Es ist sinnvoll, die verschiedenen Kommunikationsarten nach der zeitlichen Dauer der Verbindung zu betrachten. Dies ist insofern wichtig, da im folgenden immer wieder zwischen Initialisierungsphase und Auftragsphase (Bearbeitung) von Programmsystemen unterschieden wird. Sofern zwischen den Partnern eine permanente Verbindung bestehen soll, wird diese in der Initialisierungsphase aufgebaut. Unter permanent ist eine Verbindung zu verstehen, die von einer Initialisierungsphase bis zur nächsten aufrecht erhalten wird, zum Beispiel für die Dauer eines Bewegungsabschnitts. Die Einrichtung von statischen Verbindungen (= konstant über eine Bearbeitungsphase) bedeutet deshalb keine Einschränkung, da eine Verlegung eines Programmes auf einen anderen Prozessor nicht während einer aktiven Phase vorkommen kann. Für eine solche Umkonfigurierung fehlt in diesen Betriebsphasen die Rechenkapazität.

Nach dem Verbindungsaufbau haben die rechnerinternen Kommunikationsmodule keinen Einfluß mehr auf diese Datenpfade. Somit entstehen wenig System-Verwaltungsaufwand und keine weiteren Verzögerungen durch die Zustellung der Meldungen.

Die Verwendung eines externen Busses stellt die größte Einschränkung in einem verteilten Echtzeitsteuerungssystem dar. Der Engpaß liegt nicht nur in der erhöhten Laufzeit für Meldungen, sondern auch in den vom Bussystem bestimmten Formaten (maximale Blockgröße). Je nachdem, welche Alarmbehandlungszeiten vom Anwender für das System gefordert werden, erhöht oder erniedrigt sich die Anzahl der maximal einzusetzenden Cluster. Messungen /56/ an einem realen System haben für einen Feldbus eine maximale Clusteranzahl von etwa drei er-

geben ($t_r$ < 5 ms). Bei Erhöhung der Clusteranzahl kann eine Alarmmeldung nicht mehr in der angegebenen Zeit weitergereicht werden. Diese Zeitgrenze wird sich aber dank neuer Feldbusmedien bald nach unten verschieben lassen und so kürzere Reaktionszeiten oder eine größere Zahl von Rechnerknoten ermöglichen.

## 5.5 Multiprozessorunterstützung

Ein breites Anwendungsfeld einer Steuerung bedingt Möglichkeiten zur sukzessiven Erweiterung. Der **Steuerrechner muß** deshalb **skalierbar sein** bezüglich der Rechenleistung. Deshalb sind vom Betriebssystem Funktionen bereitzustellen, welche die Konfiguration und den Betrieb eines Multiprozessorrechners erlauben. Beim Einsatz mehrerer Prozessoren läßt sich die Last gleichmäßig auf alle Recheneinheiten verteilen.

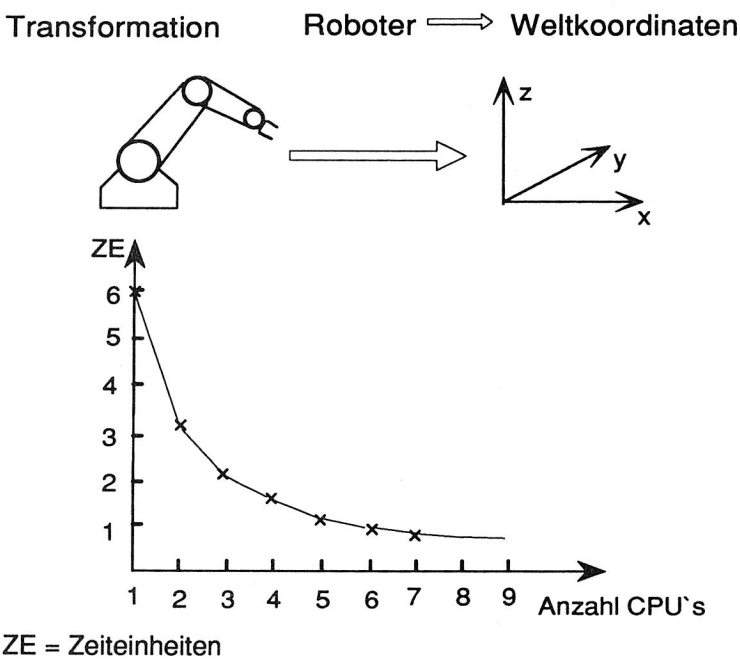



Bild 5.12: Leistungssteigerung durch Parallelisierung

Durch den Einsatz mehrerer Prozessoren läßt sich die Verarbeitungszeit vieler Steuerungsaufgaben stark senken, da die Programme praktisch schon in paralleler Form vorliegen (Bild 5.12). Durch den Einsatz von Planungshilfsmitteln lassen sich viele Kinematikberechnungen (speziell für 6-achsige Roboter) zusätzlich parallelisieren /54/. Man kann so eine gleichmäßige Auslastung aller Prozessoren erreichen und hat keinen großen Aufwand bei der Parallelisierung der Programme.

### 5.5.1 Allokation von Tasks

Die Flexibilität von Multiprozessorsystemen wird unter anderem dadurch erreicht, daß die Tasks wahlfrei auf die Prozessoren verteilt werden können (Allokation). Diese Zuordnung kann nach verschiedenen Gesichtspunkten und Randbedingungen durchgeführt werden. Wesentlichster Aspekt ist die gleichmäßige und optimale Nutzung von Betriebsmitteln des Rechnersystems. Allen voran ist die Rechen- und Speicherkapazität zu nennen. Da sich in einem Echtzeitsystem alle Tasks permanent im Hauptspeicher befinden, ist der verfügbare Speicher einer Prozessoreinheit ein wesentliches Verteilungskriterium. Ebenso spielen Fragen der gleichmäßigen Auslastung der Rechenleistung eine entscheidende Rolle. Im Gegensatz zu Standardsystemen kommt es jedoch nicht darauf an, im Integral eine hohe Auslastung aller Prozessoren zu erreichen, sondern die Verteilung muß so gewählt werden, daß es in Phasen hoher Belastung nicht zu einer Überlastung eines Prozessors kommt. Im Normalbetrieb kann dabei durchaus eine Unsymmetrie der Last bestehen.

Eine weitere Randbedingung bei der Zuordnung stellen die Verbindungen zum Montageprozeß dar. Hat ein Prozessor exklusiven Zugriff auf bestimmte Betriebsmittel (Sensoren), weil sie hardwaremäßig direkt angeschlossen sind, so sollten (müssen) Programme, die eine Verarbeitung der Sensordaten durchführen, ebenfalls auf dem entsprechenden Prozessor plaziert werden (Sensor 'A' und Task 'A' Bild 5.13)

Wie zwingend die gemeinsame Lokation ist, hängt vom Umfang der Informationen des Sensors (Datenrate, Taktzeit) und der Leistungsfähigkeit des die Prozessorknoten verbindenden Bussystems ab.

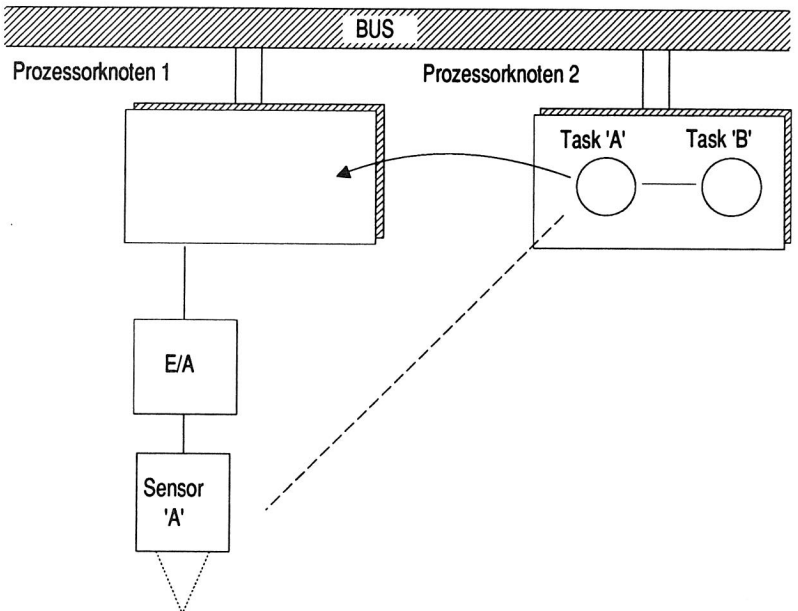



Bild 5.13: Zuordnung Task - Sensor

Nicht nur die Assoziation von Tasks zu Hardwareelementen, sondern auch die Beziehungen von Tasks untereinander beeinflussen die Allokation. Stehen zwei Programme in enger Verbindung (Task 'A' und 'B') mit häufiger oder umfangreicher Kommunikation, so sollten diese zusammengehörenden Tasks eine gemeinsame Lokation besitzen. Würde man die Verteilung der Programme durchführen, ohne die gegenseitigen Beziehungen zu beachten, so ist mit einer Überlastung der Verbindungswege zwischen den Prozessorknoten zu rechnen. Voraussetzung für eine effektive Ausnutzung von Lokationsbeziehungen ist die Möglichkeit der Adaption von Kommunikationsverbindungen an die aktuelle Konfiguration (vgl. Kap. 5.4). Dazu kann durch eigene Verwaltungsmodule (Migrationssteuerung) die Allokation auch im Betrieb noch geändert werden /86/.

### 5.5.2 Konfiguration und Monitoring

Die im vorigen Abschnitt erläuterten Möglichkeiten zur Verteilung von Tasks müssen durch entsprechende Konfigurationshilfsmittel unterstützt werden. Es sind

dabei zwei Unterscheidungen zu treffen. In vielen Fällen wird eine statische Konfiguration ausreichend sein. Durch entsprechende Sprachhilfsmittel wird die Anordnung aller Tasks eines Systems beschrieben und vor der Aufnahme des Betriebs in der Initialisierungsphase durch eine Taskverwaltung hergestellt. Die Programme werden von einem Massenspeicher in die einzelnen Prozessorknoten geladen oder sie befinden sich im jeweiligen Festwertspeicher.

Die zweite Möglichkeit besteht in der dynamischen Rekonfiguration. Hier muß die Taskverwaltung zur Laufzeit Tasks nachladen oder austauschen. Da es in Steuerungen immer Phasen gibt, in denen ganze Taskgruppen inaktiv sind, bietet sich die Änderung der Konfiguration in genau solchen Abschnitten an. Dadurch reduziert sich der Aufwand Referenzen, zum Beispiel zur Kommunikation, zu aktualisieren. In harten Echtzeitsystemen ist es ohnehin schwer realisierbar, aktive Tasks zu verlagern. Für nicht zeitkritische Programme bereitet es dagegen wenig Probleme.

Die Möglichkeit, die Konfiguration zur Laufzeit zu ändern, birgt natürlich die Gefahr, daß es zu Überlastungen einzelner Prozessorknoten kommen kann. Der Ablauf der Tasks mit harten Echtzeitanforderungen (Prioritätsebenen 5 und 4) ist deshalb zu kontrollieren. Im einfachsten Fall kann sich das Monitoring auf die Überwachung der verfügbaren Restzeit der Prioritätsebene 4 bis zum nächsten Systemtakt beschränken. Diese Zeitspanne ist ein Maß dafür, wie stark der jeweilige Rechnerknoten mit der Erzeugung von Bewegungsdaten ausgelastet ist. Die Beobachtung von zusätzlichen Zeiten ist sinnvoll, erlaubt aber keinen unmittelbaren Rückschluß auf die Gefahr einer Überlastung.

Eine flexible Monitoringkomponente im Systemkern gestattet die Messung von Laufzeiten und Zeitspannen und kann damit zur Effizienzanalyse einzelner Module eingesetzt werden. Die Beobachtung muß sich dabei jedoch auf wenige Ereignisse beschränken, um den Zeitablauf des Systems nicht zu sehr zu verändern. Eine beeinträchtigungsfreie Zeitmessung ist nur durch aufwendige Zusatzausstattung (Hardwaremonitoring) zu verwirklichen. Für die vorgeschlagene Softwaremessung genügt das Vorhandensein eines globalen Systemtaktes in Verbindung mit lokalen Zeitgebern, um eine ausreichende zeitliche Auflösung und Synchronität auch in einem Multiprozessorsystem zu erreichen.

### 5.5.3 Synchronisation von Uhren

Die Synchronisation in einem verteilten heterogenen Echtzeitsteuerungssystem besteht nicht nur aus der eigentlichen Synchronisation der freilaufenden Systemuhren, sondern auch aus der Strategie, mit der die einzelnen Task untereinander synchronisiert werden.

Die Synchronisation muß mit Hardwaremaßnahmen und nicht mit Software gelöst werden, wenn hohe Genauigkeitsanforderungen bezüglich der Synchronität der freilaufenden Uhren gestellt werden. In dieser Arbeit werden die verschiedenen Einzelkomponenten so synchronisiert, daß die geforderte Gleichzeitigkeit aus Kapitel 4 erreicht werden kann.

Die Synchronisation zerfällt in einem heterogenen System in zwei Teile :

- Clusterintern und
- Clusterextern.

Innerhalb eines Rechnerknotens ist eine Synchronisation von Uhren nicht notwendig. Als Uhr kann ein zentraler Taktzähler (Systemtakt) benutzt werden. Die Synchronisation der einzelnen Tasks führt der Scheduler durch.

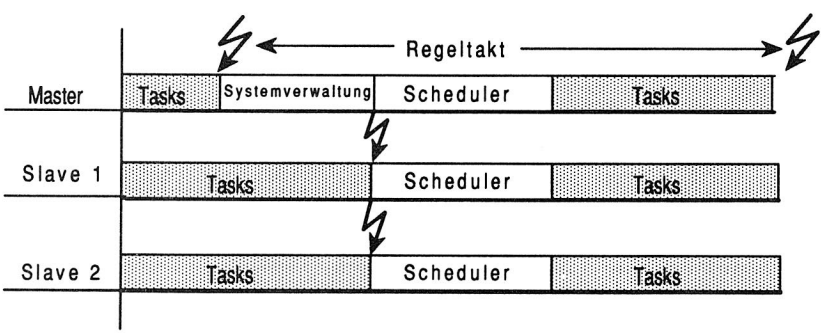



Bild 5.14: Synchronisation der Scheduler (Multiprozessor)

Mehrere Recheneinheiten an einem Bus lassen sich über eine zentrale Uhr versorgen. Die erreichbare Genauigkeit liegt hierfür sicher besser als die meisten Anwendungen es erfordern. Die Synchronisation von mehreren Rechnerknoten bereitet dann keine Schwierigkeit, wenn die Systemtakte clusterweit von einer Uhr generiert werden. Die Realisierung des Echtzeitkerns mit nur einem System-

interrupt und 'synchronen' Schedulern (Bild 5.14) erleichtert die Synchronisation zusätzlich.

Um Abschnitte gleichzeitig systemweit starten zu können, müssen die freilaufenden lokalen Uhren synchronisiert werden. Hierzu gibt es in der Literatur mehrere Lösungsansätze. Man geht jedoch immer davon aus, daß die einzelnen verwendeten Rechnerknoten identisch sind, das heißt sie besitzen den gleichen Systemtakt und die gleiche Taktfrequenz. Ferner müssen die zu synchronisierenden Rechnerknoten eng miteinander gekoppelt sein, das heißt sie besitzen mindestens einen gemeinsamen Speicher, besser ist jedoch, wenn sie eine direkte Registerkopplung haben /58/.

Keiner der oben erwähnten Punkte trifft auf ein heterogenes Steuerungssystem einer Zelle zu. Auch viele der in der Automatisierung eingesetzten Systeme können diese Bedingungen nicht erfüllen. Die einzelnen Cluster sind über einen Feldbus miteinander gekoppelt (lose Kopplung), und es ist deshalb nicht möglich, die Cluster zum Beispiel über einen Interrupt zu synchronisieren.
Mit dem folgenden Algorithmus können die Uhren der Cluster jedoch ausreichend genau synchronisiert werden. Alle Aktionen erfolgen über den Feldbus im Single-Taskbetrieb der zu synchronisierenden Cluster. Die Koordination hat dabei der ausgewählte Hauptverwalter auf dem Zellenrechner. Der Algorithmus ist nur für zwei Cluster beschrieben, er läßt sich jedoch in iterativer Form auf eine beliebige Anzahl von Clustern erweitern (Round-Trip-Delay ist dabei die Laufzeit einer Meldung zum Partner und wieder zurück):

- Anmeldung beim Partner für die Synchronisationsphase,
- Messung des Round-Trip-Delay,
  - Feststellen der Systemzeit,
  - Warten auf Rückantwort,
  - Feststellen der Systemzeit,
  - Berechnung der Differenz (= *Round-Trip-Delay*),
- Partner die ermittelte Zeit mitteilen,
- Partner Synchronisationsauftrag ankündigen,
- Beim Systemtakt Synchronisationauftrag auslösen,
- Partner synchronisiert sein Cluster, indem er sich für den restlichen, Systemtakt deaktiviert (Zeit= *Takt - ((Round-Trip-Delay) / 2)* und dann den
- Timer Reset für das gesamte Cluster über den Rückwandbus ausführt.

Daraus ergibt sich folgende Synchronisationsreihenfolge:

- die Cluster untereinander,
- CPU's innerhalb eines Clusters.

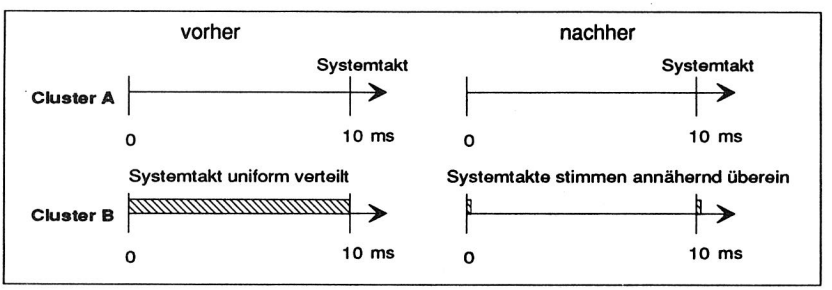



Bild 5.15: Systemuhren vor und nach der Synchronisation

Sofern die Notwendigkeit besteht, systemweite, uhrzeitbezogene Echtzeitkontrollbefehle wie zum Beispiel *ACTIVATE AT xx* einzuführen, ist es erforderlich, nicht nur den gleichen Taktzeitpunkt, sondern auch dieselbe Systemzeit zu haben.

Hierzu wird der oben beschriebene Algorithmus um die folgenden Punkte erweitert:

- Partner die eigene Systemzeit in der Ankündigung des Synchronauftrags mitteilen,
- Partner liest die Systemzeit,
- Partner stellt seine Uhr auf Partnerzeit ein.

# 6 Zustands- und Objektorientierte Task-verwaltung

In Kapitel 5 wurden die Grundlagen für eine effiziente Behandlung verteilter Echtzeitaufgaben gelegt. Die prozeßnahe Steuerung eines Montagevorgangs besteht aus einer großen Zahl von einzelnen Programmen.

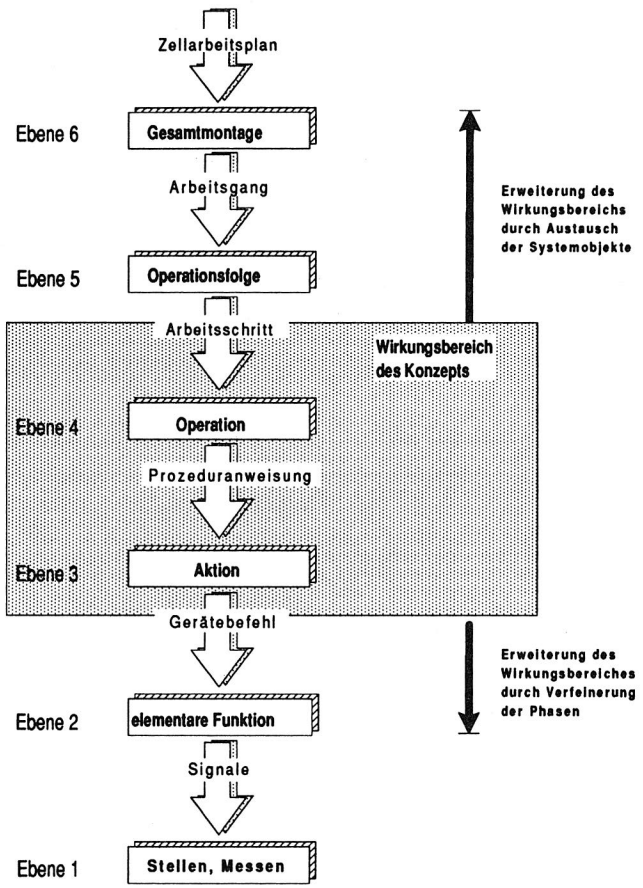



Bild 6.1 : Konzeptbereich im Hierarchiemodell

Zur besseren Handhabung sollte die Möglichkeit bestehen, Gruppen von Tasks zu Objekten zusammenzufassen. Mit der Bildung von Objekten erreicht man eine Abstraktion von Einzelprogrammen hin zu anwendungsorientierten Prozeßgebilden. Das gesamte Prozeßsystem soll Aufgaben entsprechend der Funktionshierarchie (Kap. 3) übernehmen.

Das folgende Konzept deckt den gesamten Bereich von ***Ebene 4*** bis hin zur ***Ebene 3*** ab (Bild 6.1). In der ersten Konzeption konnte die ***Ebene 4*** mit den beiden Eingangs- und Ausgangsschnittstellen abgedeckt werden. Durch Entwurf eines zusätzlichen Phasenkonzepts (vgl. Kap. 6.1.2) ist es möglich, auch noch die ***Ebene 3*** einschließlich ihrer Schnittstelle zur ***Ebene 2*** zu realisieren. Das heißt, die Steuerungsfunktionen bis in die Ebene der Bewegungs- und Sensorfunktionen, wo strenge Echtzeitanforderungen gestellt werden.

Die Möglichkeit des Objektaustausches sowie der Verteilung von Objekten auf einem Multiprozessorsystem /89, 98/ ermöglicht ein Höchstmaß an Flexibilität bei gleichzeitig hoher Leistungsfähigkeit unter Echtzeitbedingungen.

Die Einzelfunktionen des Gesamtsystems erfordern aus programmiertechnischer Sicht häufig keine so starke Gliederung, wie sie bei genauer Betrachtung der Anwendung notwendig wird. So kann zum Beispiel die Durchführung einer Fügebewegung, die sich prinzipiell als eine geschlossene Aufgabe darstellt, noch unterteilt werden in einzelne Phasen (Beschleunigung, Konstantfahrt, Sensorkorrektur...), deren Kenntnis speziell im Fehlerfall von Bedeutung sein kann.

## 6.1 Modellvorstellungen

### 6.1.1 Objektmodell

Zur Klärung der erstellten Ansätze sind zunächst Definitionen und Modellvorstellungen zu erläutern, auf denen die eigentliche Objektverwaltung basiert.

Die objektorientierte Programmierung ist ein derzeit sehr gebräuchlicher Ansatz zur Aufteilung und Strukturierung von großen Programmpaketen in kleine überschaubare Einheiten. Man unterscheidet dabei Methoden und zugehörige Daten. Die Methoden (Teilprogramme) sind die einzige Möglichkeit, um Zugriffe und Manipulationen an den Daten vorzunehmen.

In dieser Arbeit soll unter einem Objekt eine Ansammlung von eigenständigen Prozessen bzw. Tasks verstanden werden, die gemeinsam eine Aufgabe übernehmen. Mit der Auffassung von Tasks als Teilobjekte und mit dem Zusammenschluß dieser Teilobjekte zu einem Gesamtobjekt lassen sich komplizierte Handhabungs- und Montageaufgaben flexibel durchführen. Hierbei wird bei der Zusammenfassung der Teilobjekte besonderer Wert auf die in der Steuerung geforderten Echtzeitbedingungen gelegt. Die resultierende Kommunikation bei der Interaktion von Objekten wird in der Regel der Systemumgebung überlassen. Restriktionen bezüglich der Zeitanforderungen sind dabei nicht definierbar und können deshalb nicht berücksichtigt werden.

Unter Beachtung der Kommunikationsstruktur eignen sich Objekte auch in der Steuerungstechnik sehr gut zur

- Strukturierung komplexer Echtzeitsoftware,
- Zerlegung paralleler Programme in Teilprogramme,
- Verteilung von Programmpaketen auf heterogene Systeme.

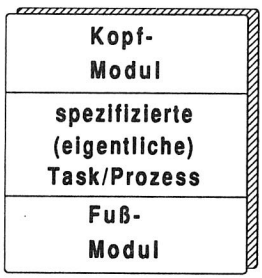
Wie aus der Aufstellung zu entnehmen ist, bietet sich der Objektbegriff für die Lösung der gestellten Anforderung an. Der Objektansatz alleine macht jedoch nur einen Teil des Lösungskonzepts aus.

Neben den Funktionen für die Anwendung müssen Teile eines Objektes auch noch Aufgaben der Objektverwaltung übernehmen. Es empfiehlt sich daher, die Objektteile in verschiedene Klassen einzuteilen. Diese Einteilung erfolgt entsprechend den Aufgaben, die ein Teilobjekt in der Objekthierarchie übernimmt.

Ein Objekt kann wahlweise aus folgenden Teilen bestehen :

- Atomares Objekt :
  Dies sind Tasks, die neben der spezifizierten Aufgabe sowie der Anforderung und Verwaltung von Betriebsmitteln nur positive bzw. negative Rückmeldung an übergeordnete Objekte übernehmen. Diese Tasks sind die eigentlichen Ressourcen des Systems.

- Zusammengesetzte Objekte :
  Diese Objekte setzen sich aus einem Objektvater und einem oder mehreren Objektsöhnen zusammen. Jeder Objektsohn kann ein atomares oder zusammengesetztes Objekt sein.




Verwaltungsmodul besteht aus :
- Kopf-Modul +
- Fuß-Modul

Kopf-Modul übernimmt
- Kommunikation mit übergeordneten Objekten
- Starten, Stoppen und Ermitteln der eigenen Phasen
- Fehler- und Ausnahmebehandlung

Fuß-Modul übernimmt
- Kommunikation mit den unterstellten Objektsöhnen
- Erstellung des Phasenablaufplans
- Steuerung des unterstellten Teilobjekts

Bild 6.2: Aufgabenblöcke eines Teilobjekts

Objektväter sind Tasks, die neben Aufgaben von atomaren Objekten noch die Steuerung der untergeordneten Teilobjekte übernehmen. Man kann die zusammengesetzten Objekte aufgrund ihrer Struktur als hierarchische Objekte bezeichnen, und das hier vorgestellte Objektmodell ist somit ein hierarchisches Objektmodell.

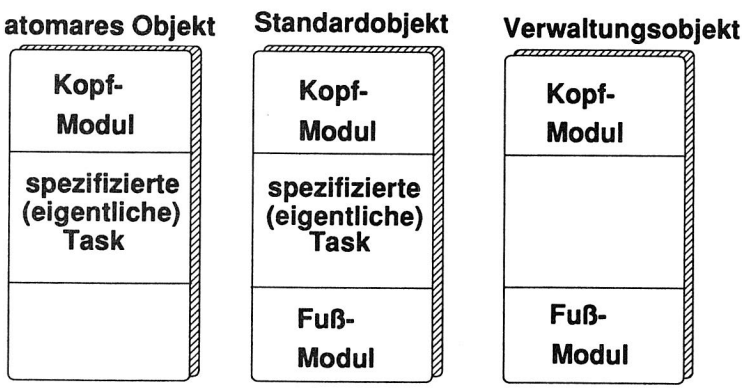



Bild 6.3: Objektarten

Um ein Objekt komplett verwalten zu können, gehört neben dem Starten und Stoppen von Teilobjekten auch das Verteilen und Synchronisieren des Gesamtobjekts auf eine Multiprozessorumgebung. Dazu sind in der Regel auch reine Verwaltungsprozesse nötig.

Solche Prozesse werden im folgenden als künstliche Prozesse oder Verwaltungsobjekt bezeichnet (Bild 6.3), da sie reinen Verwaltungsoverhead erzeugen und keine dem Objekt zugeschriebene Arbeit verrichten.

Für das Objektmanagement sind auf jeden Fall folgende künstliche Objekte notwendig :

- Pro Rechnerknoten (CPU) ein Objektverwalter,
- Pro Cluster ein Cluster-Objektverwalter. Dieser Verwalter ist auf einer CPU des Clusters anstelle des normalen Verwalters installiert.
- Einer dieser Cluster-Objektverwalter ist System-Objektverwalter.

Die Verwalter sind Objektväter für jedes in ihrem Verantwortungsbereich ablaufende Teilobjekt. Sie übernehmen dabei folgende Hauptaufgaben:

- Verbindungsaufbau,
- Ablaufsteuerung der Teilobjekte,
- Starten / Stoppen / Synchronisieren,
- Meldung an den übergeordneten Verwalter.

Sofern alle Komponenten eines Objekts direkt von einem Verwalter aus gesteuert werden können, wird das Gesamtobjekt als *natürliches Objekt* bezeichnet. Wenn jedoch zum Management eines Objekts mehrere Hierarchieebenen von Objektverwaltern benötigt werden, ist dies ein *virtuelles* oder *verteiltes Objekt.*

Diese Aufteilung in *virtuelle* und *natürliche Objekte* erlaubt es, eine weitere Einteilung der Objekte bezüglich ihres Echtzeitverhaltens zu machen. Diese ist speziell für die Einteilung und Einplanung der einzelnen Objekte erforderlich. Ferner stehen für bestimmte Objektarten nicht alle Kommunikationsmöglichkeiten zur Verfügung. *Natürliche Objekte* überschreiten keine Prozessorgrenzen. Je nachdem, welches Betriebssystem auf diesem Prozessor existiert, handelt es sich bei dem Objekt um ein

- Multitask-Objekt (Real Time) oder ein
- Objekt mit hartem Echtzeitverhalten (Hard Real Time).

Verteilte Objekte haben eine Ausdehnung von mindestens zwei Prozessoren. Sie müssen nochmals unterteilt werden in *clusterinterne* und *clusterübergreifende Objekte.*

Die *clusterinternen Objekte* können über den Rückwandbus und einen gemeinsamen Speicher (shared memory) kommunizieren. Die Einschränkung, die bezüglich des Echtzeitverhaltens zu machen ist, begründet sich in der möglichen Heterogenität der eingesetzten Betriebssysteme. Befindet sich in einem *clusterinternen Objekt* eine unter einem normalen Multitasksystem zu steuernde Task, so ist das gesamte Objekt als Multitaskingobjekt mit Echtzeitverhalten zu sehen. Auf keinen Fall erfüllt dieses Objekt die Anforderung eines harten Echtzeit-Systems. Sofern jedoch nur Steuerungsrechner eingesetzt werden, die unter einem wirklichen Echtzeitsystem betrieben werden, so kann das *clusterinterne Objekt* die erhöhten Anforderungen erfüllen.

Die *clusterübergreifenden Objekte* zeichnen sich dadurch aus, daß die notwendige Kommunikation über das in der Montagezelle eingesetzte Feldbussystem durchgeführt werden muß. Dieser Bus ist der Engpaß des gesamten Systems. Objekte, die systemweit installiert sind, erfüllen nur die Eigenschaften eines Multitaskingsystems, es muß jedoch besonderer Wert auf die Synchronisation gelegt werden, um eine ausreichende Leistung zu erreichen.

### 6.1.2 Phasenmodell

Die Einführung von Phasen im Ablauf einer Task ist die entscheidende Erweiterung des Objekt-Konzepts (Bild 6.4). Sie bringt folgende Vorteile :

- Funktionale Gliederung ohne großen zusätzlichen Aufwand.
- Einfachere Synchronisation mehrerer Teilprozesse eines Objekts.
- Phasen können als Kontrollpunkte eingesetzt werden und erleichtern die zeitliche Lokalisierung von Fehlern .
- Phasenwechsel können als Wiederaufsetzpunkte nach Fehlern dienen.

Jedes Programm durchläuft in seinem Lebenszyklus mehrere Phasen (Bild 6.4). Werden diese vor der eigentlichen Ausführung in ihrem Ablauf geplant, so reduziert sich die Synchronisation im Bearbeitungsabschnitt auf einfache Triggersignale für den nächsten Abschnitt. Besitzt das eingesetzte Feldbussystem Broadcast-Fähigkeiten, kann eine Steuerung der Phasen sogar zellenweit in Echtzeit erfolgen.

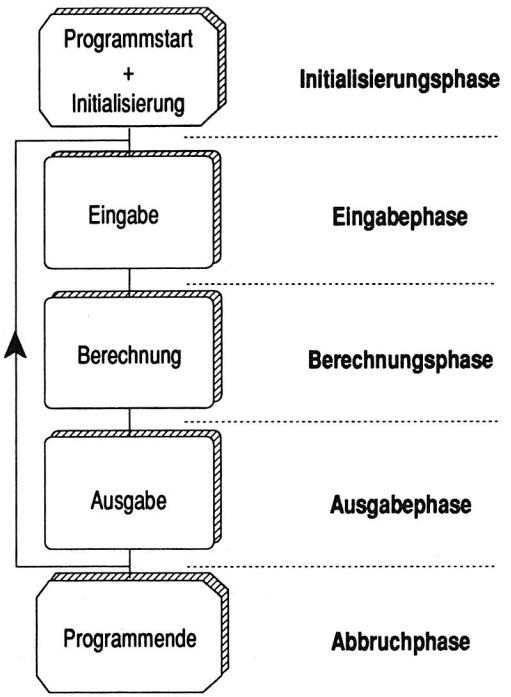



Bild 6.4: Einteilung in Programmphasen

An einem Beispielprogramm soll die Phasendefinition erläutert werden. Es besteht aus einem wiederholten Zyklus, der sich aus einer Eingabe-, einer Berechnungs- und einer Ausgabephase zusammensetzt. Eingerahmt wird dieser Zyklus durch die Initialisierungsphase und eine Programmabbruchphase.

Diese Phasen oder Zustände ergeben sich häufig direkt aus der Anwendung heraus und lassen sich entsprechend in Programmstrukturen umsetzen. Um sie nutzen zu können, kann man das vorher beschriebene Programm auch gemäß Bild 6.5 darstellen. Man erkennt, daß durch Anforderung der Phasen 1, 2, 3,.., 1, 2, 3 der Programmablauf aus Bild 6.4 durchgeführt wird.

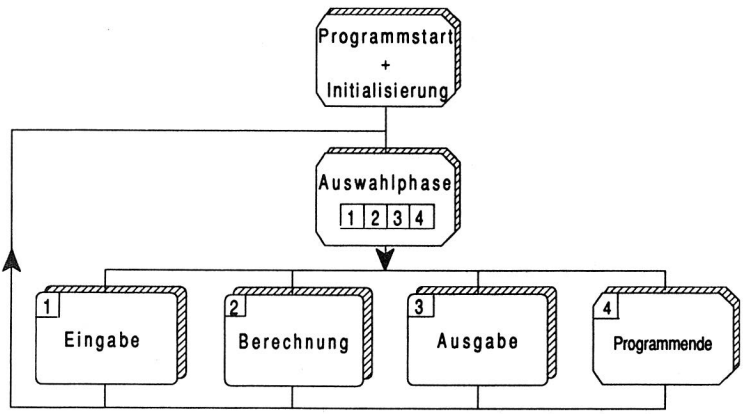



Bild 6.5: Beispielprogramm in Phasendarstellung

Das Programm besteht jetzt nur noch aus Prozeduren, wobei jede Prozedur einer Phase entspricht. Der Programmablauf wird über ein Feld von Funktionszeigern gesteuert. Dadurch wird der Eingriff in den Programmablauf von außen ermöglicht. Die externe Steuerung durch einen übergeordneten Vaterprozess im Rahmen eines System-Objekts wird ermöglicht. Die Phasen werden somit frei wählbar. Sie sind zum Betriebsmittel des eigenen und auch des übergeordneten Programms geworden (Bild 6.6). Der Programmablauf läßt sich frei aus den angebotenen Phasen eines Prozesses zusammensetzen.

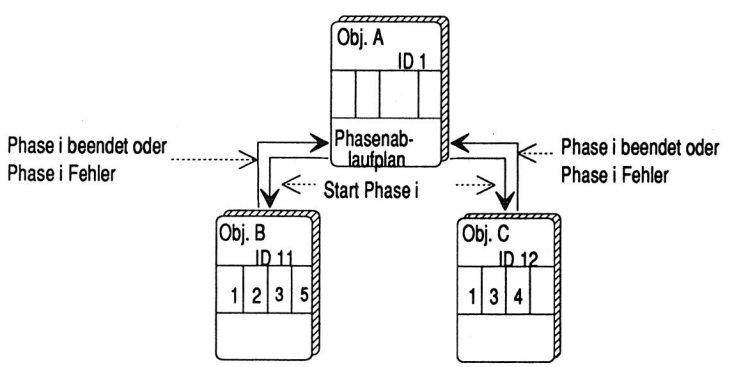



Bild 6.6: Externe Ansteuerung einer Phase.

Eine Phase sollte einem Bearbeitungabschnitt einer Handhabungs- und Montageaufgabe entsprechen. Es ist zusätzlich zwischen

- *internen Phasen* und
- *externen Phasen*

zu unterscheiden.

Intern kann ein Programm mehr Phasen haben als es nach außen in der Initialisierungsphase bekannt gibt. *Interne Phasen* sind nur lokal bekannt, sie sind von außen nicht ansteuerbar. Sie dienen unter anderem zur genauen Lokalisierung von Fehlern (rein für Diagnosezwecke). Je feiner man die internen Phasen wählt, um so näher kommt man dem in Kapitel 3 beschriebenen Modell der ***Ebene 2*** des Hierarchiemodells für Montagezellen.

Mit Hilfe der *externen Phasen* kann ein Programm durch ein anderes (übergeordnetes) Programm gestartet, gestoppt und zurückgesetzt werden. Alle Phasen, die extern ansteuerbar sein sollen, müssen systemweit bekannt sein, damit Eindeutigkeit bezüglich des Namens und der Aktionen der einzelnen Phasen herrscht. Diese sichtbaren Phasen dienen dem Ablauf und der Synchronisation mehrerer Prozesse. Im konkreten Fall werden die Phasen auch als Kontrollpunkte benutzt. Um die Möglichkeiten auszunutzen, die sich durch Einführung der Phasen ergeben haben, ist es notwendig, die vorhandenen Phasen global bekannt zu machen. Diese sichtbaren Phasen können von anderen Tasks dann als Softwarebetriebsmittel angefordert werden.

Die Koordinierung der einzelnen Teilprozesse (Teilobjekte) und die Verteilung der einzelnen Tasks in einem ereignisgesteuerten Objektsystem machen es erforderlich, einen Phasenablaufplan zu erstellen. Nur so läßt sich die Forderung nach hoher Effizienz zur Laufzeit erfüllen.

In Bild 6.7 sind die Phasen 1 und 3 *synchrone Phasen*, das heißt beide Objekte müssen diese Abschnitte gleichzeitig durchlaufen. Die Erstellung eines Phasenablaufplans geschieht bottom-up, von den atomaren Objekten der untersten Hierarchiestufe bis hin zum Objekt-Hauptverwalter auf dem Zellenrechner. Die untergeordneten Objekte melden in der Initialisierungsphase all diejenigen Phasen an ihren Objektvater, die sie zur Erfüllung ihrer spezifizierten Aufgabe durchlaufen müssen. Der Objektvater stellt anhand aller von seinen Söhnen gemeldeten

Phasen den Phasenablaufplan seiner untergeordneten Objekte auf und entscheidet anhand der eindeutigen Phasennamen, ob eine Phase

- synchronisierend oder
- nebenläufig ist.

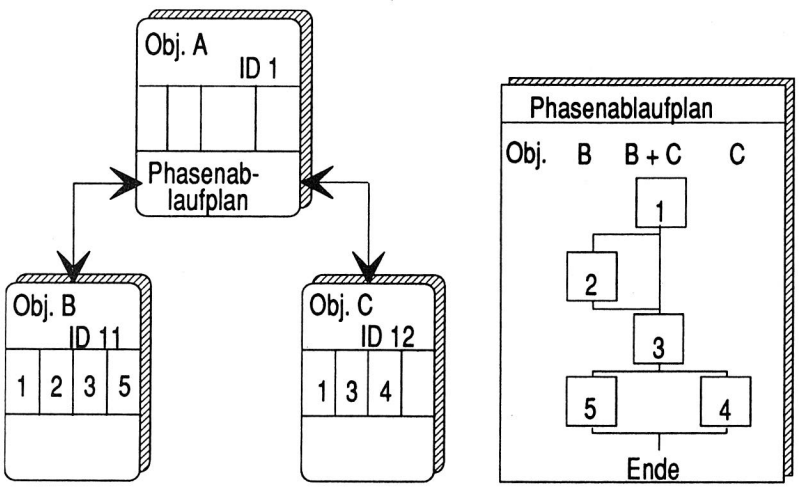



Bild 6.7: Phasenablaufplan für zwei atomare Objekte

Bei der synchronisierenden Phase muß der Objektvater eine synchrone Aktivierung der beteiligten Prozesse gewährleisten und darf deshalb eine darauf folgende Phase nicht eher freigeben, bis sich alle beteiligten Prozesse zurückgemeldet haben. Die unterschiedliche Laufzeitlänge ein und derselben Phase kann sich daraus ergeben, daß in den Einzelobjekten nicht das gleiche Codesegment bearbeitet werden muß.

### 6.1.3 Fehlerkonzept

Um das Fehlerkonzept zu verdeutlichen, ist es erforderlich, einige Definitionen aus dem Bereich der betrachteten Fehler darzulegen. Bei der Betrachtung dieser Definitionen sowie des gesamten vorgestellten Konzepts ist zu beachten, daß der Ansatz für eine Handhabungs- und Montageumgebung konzipiert wurde. Daher ist

besonderer Wert auf die Erkennung und ***Verarbeitung von transienten Fehlern*** gelegt worden. Diese Fehler treten in einer Industrieumgebung, zum Beispiel durch den Einfluß anderer Geräte, häufiger auf als permanente Fehler. Durch elektrische Interferenz, magnetische Störungen oder Vibrationen in der Steuerungsanlage kann ein Fehlverhalten entstehen. Es können Daten verändert oder Instruktionen falsch ausgeführt werden. Es liegt bei einem transienten Fehler kein Hardwarefehlverhalten vor, so daß bei Restauration der Daten oder bei einem wiederholten Ablauf der Instruktionen das Programm sehr wahrscheinlich ein korrektes Ergebnis liefert. Die Wahrscheinlichkeit für Datenfehler ist, bedingt durch Einflüsse aus dem Fertigungsprozeß, sehr viel höher als die Wahrscheinlichkeit für falsche Anweisungen oder gar Störungen im Programmfluß.

Für die folgenden Betrachtungen bilden einige Definitionen die begriffliche Grundlage:

**Fehler** (fault) :
Physikalische oder algorithmische Ursache für einen Fehlerzustand.

**Fehlerzustand** (error) :
Zustand eines Systems, der nicht korrekt ist, das heißt bei weiterer Verarbeitung zu einem Fehlverhalten führt.

**Fehlverhalten** (failure) :
Abweichen des Systems von seinem spezifizierten Verhalten zu seiner Umwelt ('Ausfall' des Systems).

Prinzipiell gibt es zwei komplementäre Ansätze zur Erhöhung der Zuverlässigkeit und Verfügbarkeit eines Systems :

- Fehlerintoleranz: Das Auftreten von Fehlern wird a priori ausgeschaltet.
- Fehlertoleranz: Auftretende Fehler werden toleriert, das heißt sie führen nicht zu einem Fehlverhalten.

Da in einem Steuerungsrechnersystem das Nichtauftreten von Fehlern wohl wünschenswert ist, jedoch nicht ausgeschlossen werden kann, ist der Ansatz über die Fehlerintoleranz wenig praktikabel. Es gibt eine wesentliche Definition für das Maß der Fehlertoleranz:

Der ***Überdeckungsfaktor C*** (Coverage) eines Systems ist die bedingte Wahrscheinlichkeit, daß ein System fehlerfrei auf einen Fehler reagiert.

In vielen Anwendungen wird ein Überdeckungsfaktor von annähernd 1 (0,9999...) verlangt, um Fehlverhalten auszuschließen, die Schaden an Personen und Einrichtungen verursachen könnten.

Für die Objektverwaltung in einem verteilten System muß ein für diese Spezialaufgabe passender Fehlertoleranzansatz konzipiert werden. Oberstes Ziel der einzusetzenden Fehlertoleranzmaßnahmen ist die Sicherung der Anwendung, d.h die Vermeidung von Fehlverhalten und Schäden in der Montageanlage. Eine Hauptforderung ist daher das rechtzeitige Stoppen des Ablaufs bei einem Fehlerzustand (fail stop).

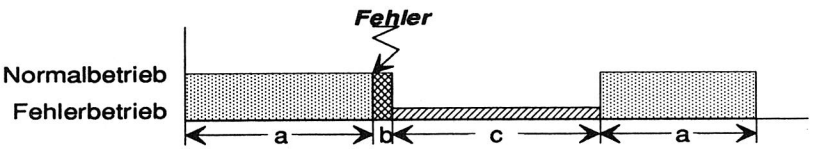
Unter Fehlertoleranz werden folgende Punkte verstanden :

- Fehlererkennung,
- Fehlerlokalisierung,
- Fehlerklassifizierung,
- Fehlerbehebung (Recovery).

Fehlererkennung bedeutet in diesem Zusammenhang, daß eine Komponente selbst ein von dem normalen Systemverhalten abweichendes Verhalten wahrnimmt.

Die Fehlerlokalisierung umfaßt nicht nur die örtliche, sondern auch die zeitliche Lokalisierung einer fehlerhaften Komponente. Zeitlich heißt, daß die Objektphase (Zeitpunkt) festgestellt werden muß, in der sich die fehlerhafte Komponente bei der Entdeckung des Fehlers befindet. Hierbei müssen auch solche Fehler berücksichtigt werden, die in einem normalen Programmsystem eigentlich keinen Fehler darstellen. Die zu späte Bereitstellung von Informationen eines Prozesses für einen anderen aufgrund einer zeitlichen Überbelastung stellt für ein Echtzeitsteuerungssystem eine wesentliche Fehlerquelle dar.

Die Fehlerklassifizierung bezieht sich primär auf die Zuordnung des aufgetretenen Fehlers in Fehlerklassen, die den Grad der Gefahr für das zu überwachende System beschreiben. Die Fehlerklassen lösen Sofortmaßnahmen aus, die ein mögliches Fehlverhalten verhindern sollen. Sie sind ein erster Schritt in Richtung Fehlerbehebung. Alle weiteren Schritte versuchen das System in einen konsistenten Zustand zu überführen, so daß entweder das System von sich aus oder der Bediener die spezifizierte Arbeit wieder aufnehmen kann.




a) normaler Objektbetrieb

b) Gefahr in Verzug => zeitkritische Direktmaßnahmen
   -je nach Fehlerklasse reagieren

c) keine Gefahr mehr => nicht zeitkritische Folgemaßnahmen
   - Fehler melden
   - auf Checkpoint zurücksetzen

Bild 6.7: Fehlerbehandlung.

Das verwendete Fehlermodell sieht folgendes vor:
Es werden nur Fehler berücksichtigt, die von irgendeiner Komponente des Objekts erkannt und weitergemeldet werden. Die Rechtfertigung für diesen Ansatz liegt in der Tatsache der hohen Wahrscheinlichkeit von Datenfehlern und der relativ einfachen Erkennung durch das Programm selbst. Moderne Mikroprozessoren gestatten die Detektion und Behandlung (Error-Traps) einer ganzen Reihe von Fehlern und sichern so den Ablauf schon auf unterster Programmebene /79/.

## 6.2 Ablaufsteuerung eines Objektsystems

Es ist in den vorherigen Abschnitten schon eine Trennung zwischen Initialisierungsphase und Bearbeitungsphase eines Objekts gemacht worden. Der Grund für die Trennung in diese beiden Hauptphasen ist die Forderung nach garantierten Zeiten. In der Bearbeitungsphase müssen einige Teilobjekte bzw. Phasen dieser Teilobjekte unter strengen Realzeitbedingungen ablaufen. Diese Zeiten können aber nur garantiert werden, wenn alle dafür notwendigen Voraussetzungen gegeben sind.
Bei der Herstellung dieser Voraussetzungen können Konflikte zeitlicher und resourcenmäßiger Natur auftreten. Aus diesem Grund werden Maßnahmen, die zur Herstellung eines konsistenten Objekt-Startzustandes notwendig sind, in der Initialisierungsphase ablaufen.

Zusätzlich kann durch die klare Trennung in eine Vorbereitungs- und Konfigurierungsphase (Initialisierung) und eine Ausführungsphase (Bearbeitung) die Effizienz einer Steuerungsanlage für Handhabungsgeräte noch erhöht werden. Der Einsatz von Optimierungsalgorithmen oder Heuristiken ist in der Vorbereitungsphase durchführbar /86/, so daß die anschließende Konfigurierung für die folgende Bearbeitungsphase optimal verläuft.

### 6.2.1 Initialisierung

Die einzelnen Teilaufgaben der Initialisierungsphase werden im folgenden näher beschrieben. Vorteilhaft ist, daß in der Initialisierungsphase meist keine Echtzeitbedingungen gefordert werden. In dieser Vorbereitungsphase müssen nur alle notwendigen Maßnahmen getroffen werden, um die Anforderungen, welche in der nachfolgenden Bearbeitungsphase bestehen, zu erfüllen.

Der Objekt-Hauptverwalter auf dem Zellenrechner übernimmt Aufgaben wie Konfigurierung und Laden der einzelnen Teilobjekte auf die Prozessoren des Gesamtsystems.

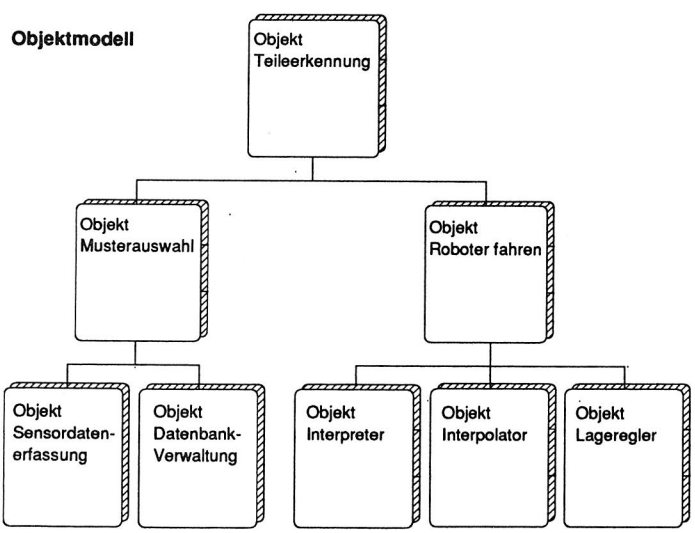



Bild 6.8: Beispiel-Objektmodell

Alle Tasks des Gesamtobjekts werden anhand eines Objekt-Konfigurationsfiles eingelesen und gemäß Scheduling- und Garantiealgorithmen auf die einzelnen Cluster und Prozessoren des Systems verteilt. Aufgrund der Varianz in der Verteilung wird zwischen

- ***Objektmodell*** und
- ***Arbeitsmodell*** unterschieden.

Das ***Objektmodell*** ist in seiner Hierarchiestruktur eindeutig und im Zellenrechner hinterlegt.

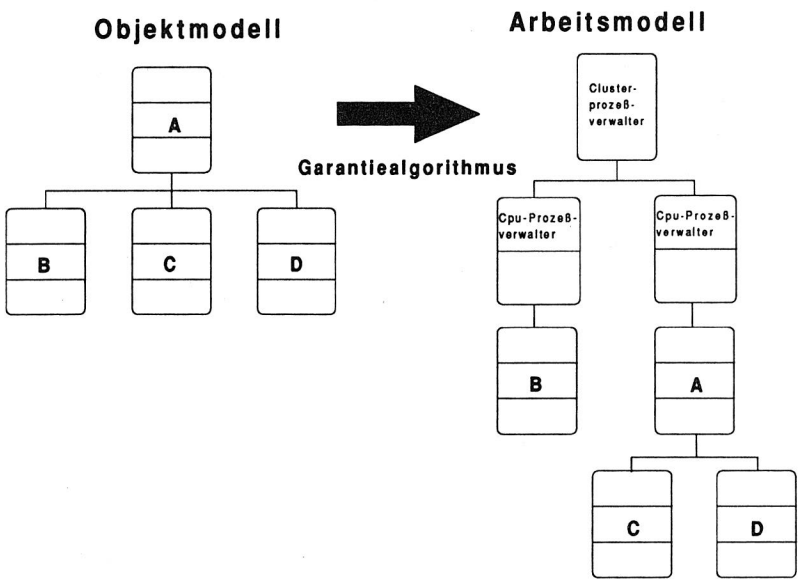



Bild 6.9: Durch Rechnerzuordnung geänderte Objektstruktur

Sofern einige Prozesse einen speziellen Rechnerknoten aufgrund von Ressourcen benötigen, werden sie dort geladen. Die Überprüfungen des Objektmodells und das Garantieren von Endzeiten übernimmt ein Garantiealgorithmus. Derartige Algorithmen sind in /90, 97, 99/ angegeben. Ein Garantiealgorithmus kann die Struktur des Objektmodells zwar verschieben, der prinzipielle Aufbau bleibt jedoch erhalten. Die resultierende Struktur mit der Rechnerzuordnung wird im *Arbeitsmodell* abgebildet. Der Objekt-Hauptverwalter besitzt ein Abbild aller geladenen Objekte bzw. ein komplettes Systemabbild.

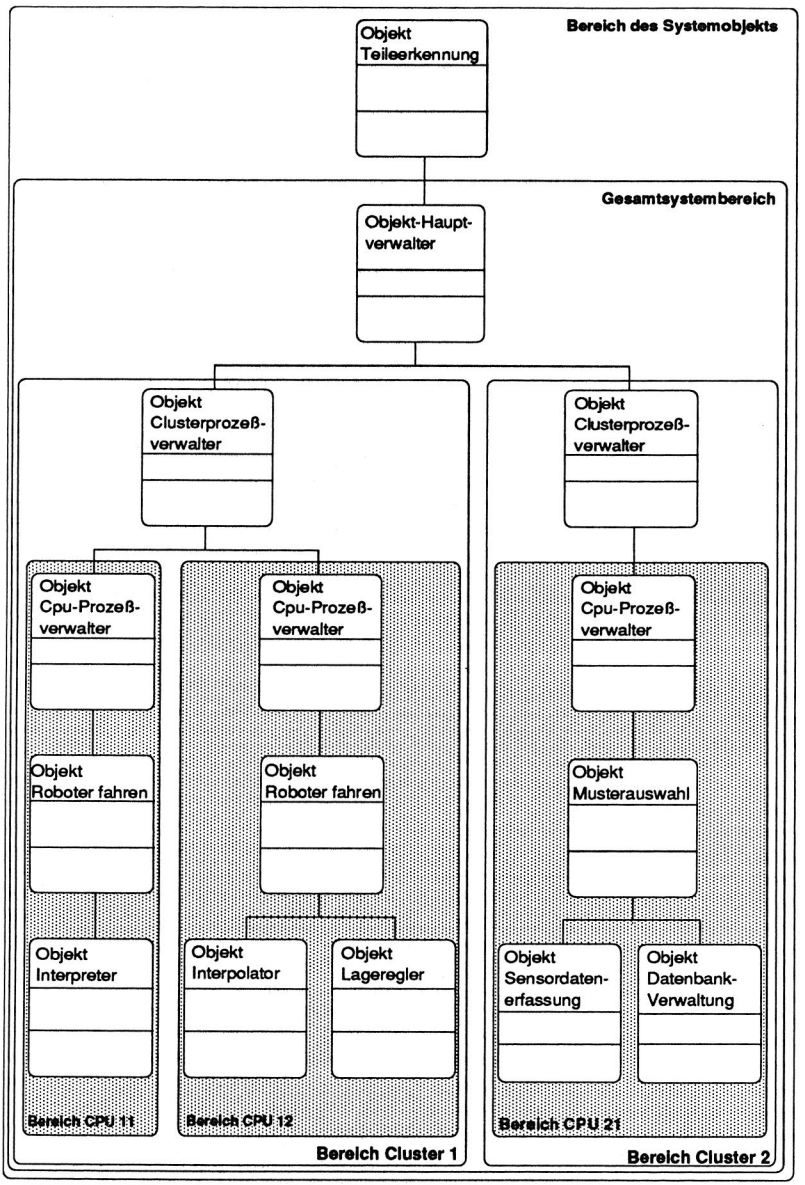



Bild 6.10: Arbeitsmodell eines Objekts.

Die Initialisierungsphase läuft Top-Down vom Objekt-Hauptverwalter über alle Hierarchiestufen bis hin zum atomaren Teilobjekt. Die hierfür benötigten Informationen bezüglich der Lage der einzelnen Tasks des Gesamtobjekts entnimmt der Objekt-Hauptverwalter den in der Ladephase erstellten Informationen, die im *Arbeitsmodell* enthalten sind.

### 6.2.2 Verbindungsaufbau

Vor der Bearbeitung müssen die einzelnen Tasks Verbindungen zu ihren Partnertasks herstellen. Hierfür und für die damit verbundene Lokalisierung des Partners stehen im wesentlichen zwei Strategien zur Verfügung. Der Prozeß fragt direkt beim Objekt-Hauptverwalter auf dem Zellenrechner nach **(1)** oder der Prozeß richtet die Anfrage an seinen Vaterprozeß **(2)**.

Vor- und Nachteile von **(1)**:

+ Ein Aufruf genügt,
– Konzentration der Kommunikation beim Objekt-Hauptverwalter,
– Keine Informationsabstraktion.

Vor- und Nachteile von **(2)**:

+ Informationsabstraktion, das heißt jeder kennt nur seinen Vaterprozeß,
+ Dezentrales Kommunikationsverhalten,
+ Der bereits aufgebaute Kommunikationsweg zum jeweiligen Vaterprozeß kann später genutzt werden,
+ Einheitlicher Verbindungsaufbau möglich,
– Eventuell mehrere Aufrufstufen.

Zur Adressieung der Partner muß ein Dienst zur systemweiten Namensauskunft (*Nameserver)* zur Verfügung stehen.

Dem Systemprogrammierer bleibt es überlassen, welchen Kommunikationsweg er zwischen den Tasks aufbaut. Entweder er läßt nur die Lokation durch den *Nameserver* ermitteln und baut in der Bearbeitungsphase eine temporäre Verbindung auf oder er wählt eine stationäre Kommunikation über den gemeinsamen Speicher. Beim Entwurf ist die Intensität der Verbindung schon bekannt und es

kann deshalb eine Auswahl getroffen werden. Eine Entscheidung erst zur Laufzeit verbietet sich durch den zu hohen Zusatzaufwand im Betriebssystem.

Die einzelnen Teilobjekte eines Gesamtobjekts müssen für die Bearbeitungsphase aufeinander abgestimmt werden. Diese Aufgabe umfaßt neben der Arbeitsaufteilung auch die Synchronisation der einzelnen Prozesse.

Zu diesem Zweck teilen die Sohnprozesse nach Aufforderung dem Vaterprozeß alle zu durchlaufenden Phasen mit. Zusätzlich teilen sie mit, welche Betriebsmittel in welcher Phase benötigt werden. Die Rückmeldung des Sohnprozesses beim Vaterprozeß erfolgt Bottum-up, das heißt erst nachdem sich alle Sohnprozesse beim Vaterprozeß als initialisiert und startbereit zurückgemeldet haben, meldet er sich wiederum bei seinem Vaterprozeß zurück. Der Vaterprozeß steuert die Abarbeitung dieser ihm mitgeteilten Phasen nach dem Phasenablaufplan. Dabei sind folgende Punkte berücksichtigt:

- synchronisierende Phasen,
- nicht synchronisierende (nebenläufige) Phasen und
- zu allokierende Betriebsmittel.

Dieser Plan ist dann der Ablaufplan für das gesamte systemweite Objekt. Anhand dieses Plans kann der Objekt-Hauptverwalter den Ablauf des gesamten Objekts, das heißt aller seiner Phasen, garantieren. Falls der Ablaufplan und das Arbeitsmodell für wiederkehrende Aufgaben und somit für immer gleiche Systemabbilder permanent gehalten wird, können die Vorbereitungs- und Aktivierungsphase entfallen. Sobald der Objekt-Hauptverwalter seinen Ablaufplan erstellt hat, ist die gesamte Vorbereitungsphase abgeschlossen.

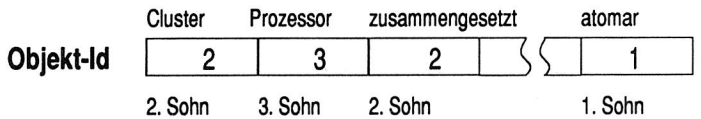
### 6.2.3 Aktivierung

Die Aktivierungsphase zerfällt in $2 + n$ Phasen, wobei $n$ die Anzahl der Hierarchieebenen unterhalb der CPU-Objektverwalter ist.

- Der Objekt-Hauptverwalter aktiviert die CPU-Prozeßverwalter. Hierbei werden nur die betroffenen Prozeßverwalter aktiviert. Ihnen wird auch der für sie relevante Teil des Systemabbilds (Namen, Anzahl und Lage der ihnen unterstellten Nachkommen (Teilobjekte)) übertragen.

- Die CPU-Prozeßverwalter aktivieren die ihnen direkt untergeordneten Sohnprozesse. Auch sie bekommen den relevanten Teil des CPU-Abbilds übermittelt. Sofern diese Sohnprozesse wiederum Sohnprozesse besitzen, verfahren sie sinngemäß.

In der Aktivierungsphase findet eine rekursive Top-Down-Verteilung der Information statt. Abbruchkriterium der Rekursion ist das Nichtvorhandensein weiterer Sohnprozesse.



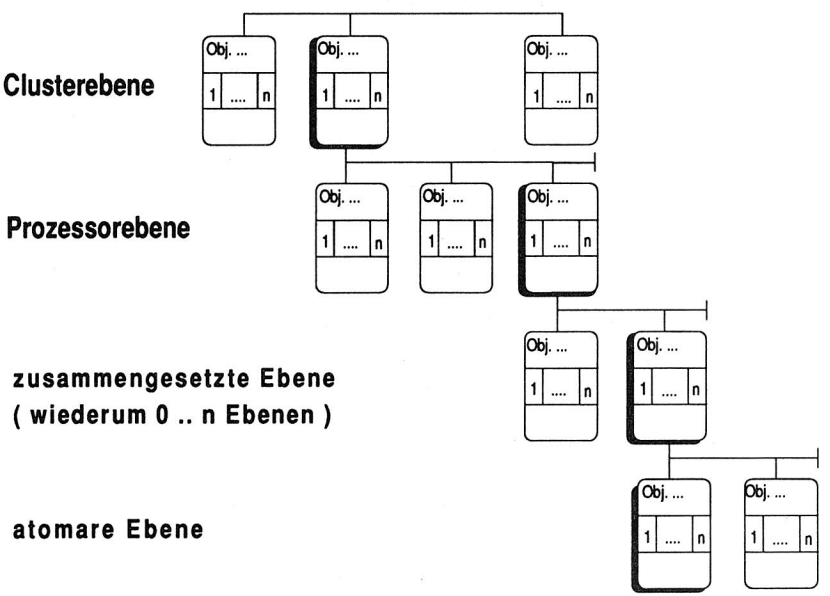




Bild 6.11: Ebenen der Objektverwaltung

Die Aktivierung der einzelnen Prozesse geschieht wie folgt:

Der Vaterprozeß aktiviert seine Sohnprozesse immer über das Kommunikationsmodul des Rechnerknotens. In Abhängigkeit von der Lokation des Partners wird die jeweils leistungsfähigste Kommunikationsverbindung hergestellt (Shared Memory, Message Passing). Clusterübergreifend dürfen keine permanente Verbindungen zwischen zeikritischen Komponenten hergestellt werden (Feldbus !).

### 6.2.4 Bearbeitung

Die Bearbeitungsphase bereitet bei der Gliederung mehr Schwierigkeiten als die Vorbereitungsphase. Sie ist abhängig von dem in der Vorbereitungsphase erstellten Phasenablaufplan. Im fehlerfreien Fall steuert der Objekt-Hauptverwalter den systemweiten Ablauf sequentiell anhand dieses Plans. Im Fehlerfall versucht er das Gesamtobjekt mit Hilfe von Recovery-Verfahren auf einen Kontrollpunkt des Phasenablaufplans zurückzusetzen, um die weitere Abarbeitung zu ermöglichen.

Einige Maßnahmen sind erforderlich, um diese Aufgaben in einem verteilten, heterogenen Echtzeitsystem durchführen zu können. Es handelt sich hierbei um das Stoppen von Phasen im Fehlerfall und das Zurücksetzen auf einen Kontrollpunkt. Zum Start von verteilten Objekten in einem heterogen System gehört neben der eigentlichen Aktivierung auch die Behandlung des Phasenplans.

Es bietet sich folgende Vorgehensweise an :

- Ankündigung der zu durchlaufenden Phasen (Vorbereitung),
- Freigabe der Phasen (Ausführung),
- nach Abarbeitung der Phasen Rückmeldung an Objektvater (Ende).

Der Objektvater schickt seinen Objektsöhnen jeweils eine Freigabeankündigung für eine oder mehrere Phasen. Die Anzahl und die Namen der Phasen entnimmt er seinem Phasenablaufplan. Die Ankündigung der Freigabe gilt immer bis zur nächsten synchronisierenden Phase aller seiner Objektsöhne. Dies hat zur Folge, daß der Verwaltungsoverhead für Ankündigung, Freigabe und Rückmeldung minimal bleibt. Bei der Synchronisationsphase muß der Objektvater die Rückmeldung aller Teilobjekte abwarten, da alle seine Söhne gleichzeitig in die nächste Phase eintreten müssen. Nachdem alle Objektsöhne sich bei ihrem Vaterprozeß als "bereit" zurückgemeldet haben, überprüft dieser, ob er alle für den ange-

kündigten Phasenblock notwendigen Betriebsmittel besitzt und meldet sich, wiederum bei seinem Objektvater, zurück.

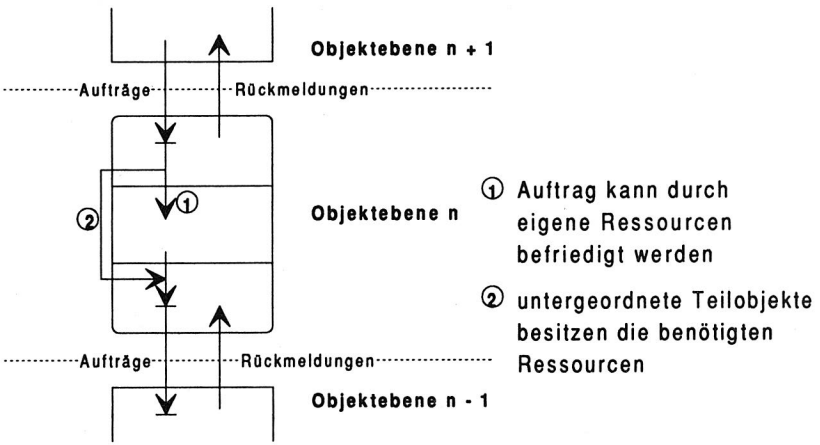



Bild 6.13: Taskablauf (Auftragsabwicklung)

Die Freigabe der vorher angekündigten Phasen kann zentral vom Objekt-Hauptverwalter auf dem ausgewählten Zellenrechner erfolgen. Er versendet dazu über alle ihm zur Verfügung stehenden Kommunikationswege eine Sammelmeldung, die die Freigabe enthält. Hierbei muß er die eventuelle Verzögerung der Freigabemeldung über den Feldbus an andere Cluster berücksichtigen.

Die einzelnen Tasks sind nach der Aktivierung freilaufend. Die Steuerung des Ablaufs wird nur durch die Prioritäten, die in der Vorbereitungsphase vom Objektvater den Söhnen zugeteilt wurde, bestimmt.

Damit im Fehlerfall die Fehlerstelle möglichst genau lokalisiert werden kann, setzt ein Teilobjekt nach jeder Phase eine positive bzw. negative Rückmeldung an das jeweilige Vaterobjekt ab. Daraus folgt, daß die Rückmeldungen wesentlich feiner gestuft sein können als die Freigabe (interne Phasen). Nach jeder Rückmeldung kann der Vaterprozeß seine Sohnprozesse im Fehlerfall stoppen.

Zur systemweiten Synchronisation können sogenannte Kontrollpunktphasen eingesetzt werden. Das Zurücksetzen im Fehlerfall auf eine schon vergangene Phase unterscheidet sich von der normalen Phasenfreigabe nur durch das eventuelle

Rückgängigmachen von bereits durchgeführten Aktionen. Ein Problem kann hier auftreten, sofern die Fehlerroutinen der Teilobjekte nicht mehr ausreichen, um das Systemobjekt zurückzusetzen. Das ist immer dann der Fall, wenn eine nicht vorher geplante Fehlersituation eintritt, für die keine Behandlungsstrategie entworfen wurde. Aus Sicht der Anwendung kann es notwendig sein, nicht nur das System in einen definierten Zustand zurückzusetzen, sondern auch umfangreiche Zusatzvorgänge durchzuführen (zum Beispiel Ausschleusen von Ausschußteilen).

## 6.3 Fehlerbehandlung

Die Fehlerbehandlung kann auf unterster Ebene durch Hardwareredundanz erfolgen. Die Verwendung von drei parallelen Systemen mit Entscheider ist die Minimalvoraussetzung für hardwarerealisierte Fehlertoleranz. Auf einer höheren Ebene der Anwendungssoftware kann man Softwaretoleranzmaßnahmen einsetzen. Das kann auch Vermeidung von Softwarefehlern durch Vervielfältigung und getrenntes Ablaufenlassen der Programme (Softwareredundanz /104/) bedeuteten. Eine softwaremäßige Unterstützung der Fehlertoleranz kann durch eine geeignete Programmiersprache, eine erweiterte Systembibliothek und/oder eine modulare Struktur der verwendeten Softwaremodule (Anwendertasks) geschehen.

Eine weitere Möglichkeit, ein Fehlverhalten des Systems festzustellen, ist die Überprüfung des Programmflusses. Stellt der eigens dafür installierte Watchdog-Prozessor einen unerlaubten Ablauf fest, so erzeugt er eine Fehlermeldung. Somit kann der Steuerfluß und dadurch auch der korrekte Ablauf in einer Task überprüft werden. Datenfehler lassen sich natürlich damit nicht erfassen.

Auf der Anwendungsebene gibt es konzeptionelle Ansätze zur Gliederung eines Gesamtsystem, damit es für den Bediener transparent wird. Diese Ansätze beziehen sich meist auf die Definition einer Sprache und auf die Festlegung von geeigneten Schnittstellen und somit der Systematisierung der vorliegenden Abläufe. Im Übergang zu Echtzeitsystemen fehlen diese Strukturierungshilfsmittel. Eine Möglichkeit zur Realisierung ist ein rechtzeitiges Stoppen (fail-stop) des Echtzeitsystems und die Behebung des Fehlers durch die angesprochenen Methoden.

In einem Echtzeitsystem muß für jede relevante Verbindung eine Überwachung bestehen. Da das gesamte Objektsystem auf mehreren Objekthierarchien aufgebaut ist, erfolgt die Überprüfung der untergeordneten Objekte (Sohnprozesse) durch das übergeordnete Objekt (Vaterprozeß). Jeder Vaterprozeß aktiviert pro Sohnprozeß eine Zeitüberwachung (Bild 6.14). Die Zeitspanne dieser Überwachung ist abhängig von der Priorität (Art) des Sohnprozesses und der Phase, in der sich der Masterprozeß gerade befindet. Als leistungsfähige Systemkomponente muß dafür ein eigener Zeitdienst (Kap 5.3.3) zur Verfügung stehen.

Nach jeder Rückmeldung des Sohnprozesses wird die Zeitüberwachung neu aktiviert. Läuft ein Zeitzähler ab, so sendet der Vaterprozeß dem Sohnprozeß eine Anfrage. Quittiert der Sohn diese Anfrage, wird die Zeitüberwachung neu aktiviert, anderenfalls gilt der Sohnprozeß als gestört.


Bild 6.14: Zeitüberwachung der Verbindung

Es kann bei dieser Methode nicht festgestellt werden, ob der Prozeß oder nur der Kommunikationskanal gestört ist. Diese Tatsache ist jedoch irrelevant, da der Prozeß dem Vaterprozeß als Betriebsmittel nicht mehr zur Verfügung steht. Es werden dabei nicht alle geschaffenen Kommunikationswege überprüft, sondern nur die Wege, die bei der Initialisierung aufgebaut wurden. Hierdurch wird der Überwachungsaufwand auf das erforderliche Maß beschränkt.

Die Fehlererkennung basiert auf Fehlermeldungen und auf ausbleibenden Quittungen der Sohnprozesse. Die örtliche Lokalisierung, das heißt die Ermittlung der fehlerhaften oder gestörten Komponente ist nicht vorgesehen, da sie zusätzlichen Aufwand erfordert.

Da jedoch die fehlerhafte Komponente in ihrer aktuellen Bedeutung für das System eingeordnet werden soll, ist es notwendig, den Fehler zeitlich zu lokalisieren (Bild 6.15). Dies ist deshalb erforderlich, da der Ausfall ein und derselben Komponente in verschiedenen Phasen des Objektablaufs auch verschiedenartige Reaktionen auslösen kann.

Fehler einer Komponente können zum Beispiel in einer Phase tolerierbar sein, in einem anderen Abschnitt des Objektablaufs jedoch zu einem sofortigen Programmabbruch führen. Die zeitliche Lokalisierung übernimmt das beschriebene Phasenkonzept.

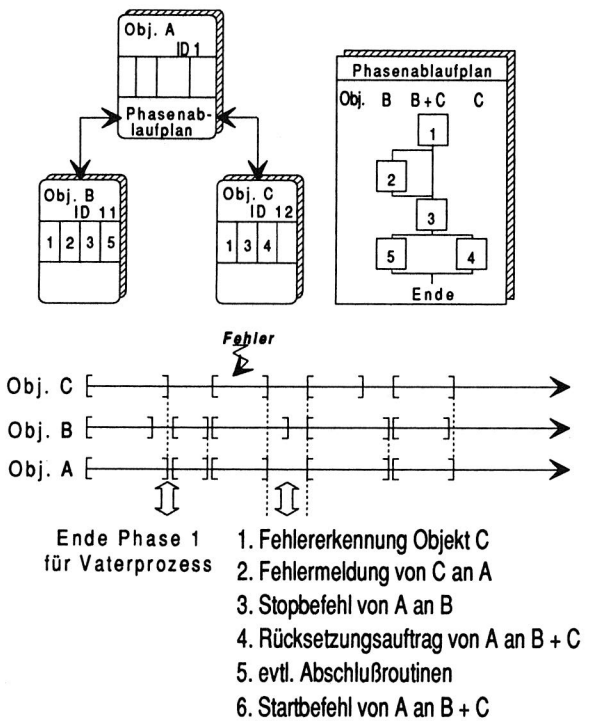



Bild 6.15: Zeitliche Fehlereinordnung

Die Einführung von Fehlerklassen ist für ein Echtzeitbetriebssystem unerläßlich, denn nicht jeder Fehler führt unmittelbar zu einem Fehlverhalten des Systems. Durch die Fehlerklassen läßt sich ein einheitliches Fehlerkonzept einführen, das die Reaktion des Systems bei einem Fehlerzustand beschreibt.

Die Notwendigkeit von Fehlerklassen soll an zwei Beispielen erläutert werden. Im ersten Beispiel diagnostiziert ein Prozeß den Ausfall eines Betriebsmittels. Dieses wird aber in der aktuellen Objektphase nicht benötigt. Dieser Fehler gehört in die

Fehlerklasse "Keine direkte Gefahr. Reine Diagnose". Es kommt zum momentanen Zeitpunkt zu keinem Fehlverhalten.

In einem weiteren Beispiel stellt der Lageregler eine Überschreitung des Schleppabstands in einer Achse fest. Bei Nichtberücksichtigung dieser Fehlermeldung kann es zu einer Beschädigung der Montageeinrichtung kommen. Daraus folgt, daß diese Fehlermeldung in die Klasse "NOTHALT! Gefahr in Verzug" gehört. Es reicht nicht mehr aus, dieses Fehlverhalten nur zu melden, sondern der Lageregler muß sofort mit einem Stop des Antriebs reagieren.

Die in den beiden Beispielen vorgestellten Fälle bilden die Hauptfehlerklassen:

- Gefahrenklasse,
- Diagnoseklasse.

Bei der Gefahrenklasse müssen sofort Maßnahmen eingeleitet werden, um ein Fehlverhalten zu vermeiden.

In der Diagnoseklasse meldet eine Task einen diagnostizierten Fehler an ihren übergeordneten Objektvater. Dieser überprüft wiederum, in welcher Klasse der Fehler der Task eingetragen ist und verfährt gemäß dieser Fehlerklasse. Das heißt, daß ein leichter, nur zu meldender Fehler in einer unteren Objekthierarchie bei einem Objektvater zu einem Nothalt führen kann.

Die Hauptfehlerklassen beschreiben das prinzipielle Verhalten des Systems bei einem aufgetretenen und bemerkten Fehler. Um die Reaktionsweise des Systems noch zu verfeinern, lassen sich die Klassen weiter unterteilen.

Die Gefahrenklasse kann gegliedert werden in:

- NOTHALT — Gefahr in Verzug. Unspezifiziertes Fehlverhalten.
- HALT — Gefahr in Verzug. Spezifiziertes Fehlverhalten.
- ACHTUNG — Gefahr in Verzug. Teilausfall, Störung einer benötigten Komponente.

Für die Diagnoseklasse kann folgende weitere Unterscheidung erfolgen:

- WARNUNG — Keine direkte Gefahr. Teilausfall, Störung einer momentan nicht benötigten Komponente.
- DIAGNOSE — Keine direkte Gefahr. Nur zu Diagnosezwecken.

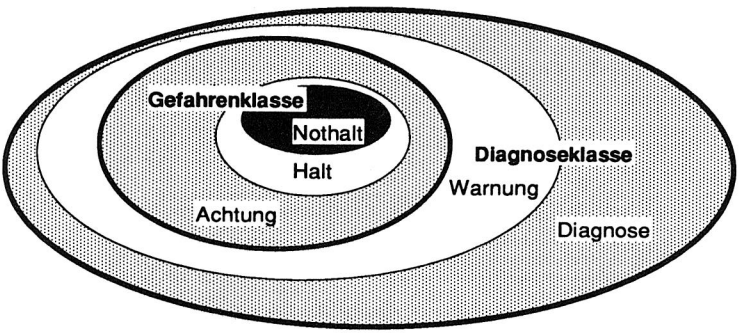



Bild 6.16: Einordnung in Fehlerklassen

Jedes Betriebsmittel wird einer Fehlerklasse zugeordnet. In welche Fehlerklasse das Betriebsmittel eingeordnet wird, hängt davon ab, wie kritisch der Ausfall dieser Komponente für die jeweilige Task im aktuellen Zustand ist.

Jeder Fehlerklasse läßt sich eine bestimmte Menge von Reaktionen zuordnen. Da das Auftreten eines Fehlers nicht während des gesamten Lebenszyklus eines Objekts die gleiche Auswirkung hat, werden die Betriebsmittel je nach Phase des Objekts einer Fehlerklasse zugeordnet. Bei Wechsel von einer in eine andere Phase werden die Betriebsmittel den Fehlerklassen neu zugeteilt, wodurch ein dem Objektstatus angepaßtes Fehlerreaktionsverhalten möglich wird. Nachdem der Fehler von einem Teilobjekt bemerkt worden und in eine Fehlerklasse eingeordnet ist, beginnt die Fehlerbehandlung. Zielsetzung ist, das Gesamtobjekt in einen konsistenten Zustand zu überführen, so daß die Anwendung möglichst wieder weitergeführt werden kann. Dieser Wiederaufsetzpunkt des Objekts kann eventuell auch in einer früheren Objektphase erfolgen.

Die Fehlerbehebung muß über das intern im Rechnersystem notwendige Verfahren noch um die Zusatzmaßnahmen erweitert werden, die notwendig sind, um den Montageprozeß in einen Zustand zu versetzen, der eine Fortführung des Montageprozesses erlaubt. Es lassen sich dabei keine allgemeingültigen Strategien benennen, vielmehr sind in diesem Bereich Heuristiken und Regeln ein denkbarer Ansatz /43/.

Als Beispiel soll der Nothalt eines Roboters dienen. Der Roboter befindet sich noch an dem Punkt, an dem er angehalten wurde. Eventuell muß er wieder zu der Position gebracht werden, die er vor Beginn der aktuellen Montagephase hatte. Dies kann insofern schwierig sein, da während der soeben unterbrochenen Phase

Montageteile so im Arbeitsbereich des Roboters angeordnet wurden, daß er seine Startposition nicht mehr anfahren kann.

Ein Montagevorgang läßt sich ohne Bedienereingriff nur dann zurücksetzen, wenn in der unterbrochenen Phase keine irreversiblen Operationen durchgeführt wurden. Vielfach ist auch ein mehrfaches Ausführen von Operationen nicht zulässig (zum Beispiel Bahnschweißen).

Ob die Behebung des Fehlers möglich ist, hängt von der zu bearbeitenden Aufgabe ab und von der Leistungsfähigkeit des Steuerungssystems auf dem Zellenrechner. Bei vielen Aufgaben lassen sich mögliche Umkehrmaßnahmen für die aktuelle Konstellation bestimmen, jedoch sind sie meist fertigungstechnisch nicht durchführbar. Es ist deshalb zu unterscheiden, ob und wie sich der Montageprozeß überhaupt zurücksetzen läßt. Dieses hängt ab von der Phase des Montageprozesses, in der ein Fehler aufgetreten ist. Zur Behebung einer bereits durchgeführten Aktion muß unter Umständen der gesamte Montageprozeß umstrukturiert werden (zum Beispiel Ausschleusen des aktuellen Teils als Schlechtteil). Die notwendige Neuplanung hat dann auf dem Zellenrechner zu erfolgen und übt keinen direkten Einfluß auf das Echtzeitsystem aus. Das Zurücksetzen oder Reparieren der Montageaufgabe stellt die eigentliche Schwierigkeit bei der Fehlerbehebung dar.

Die Grundfunktionen (schneller Stop) zur Schadensbegrenzung werden über das Kommunikationsgerüst hierarchisch ausgelöst (Vater-Sohn-Beziehungen). Das Fehler- und Phasenkonzept unterstützt in allen Fällen die Feststellung der Objektphase und der Fehlerklasse, den Einsprung in den vorhandenen Fehlerbaustein und die Abarbeitung der Sofortmaßnahmen sowie die Ausgabe von Fehlermeldungen.

# 7 Kooperierender Betrieb von Doppelrobotern

Die Leistungsfähigkeit des vorgestellten Rechnerkonzepts erlaubt den Entwurf und die echtzeitfähige Realisierung neuer Steuerungsfunktionen für den kooperierenden Betrieb zweier Roboter. Im folgenden Abschnitt wird eine kartesische Kopplung in einer Duplexroboterzelle vorgestellt, deren Roboter mit fließenden Übergängen zwischen autonomem und koordiniertem Betrieb wechseln können. Zusätzlich ist auch im Synchronbetrieb die Integration von Sensorinformation möglich. Mit Hilfe dieses Konzepts können komplexe Tätigkeiten, die den Einsatz von zwei Robotern erfordern, realisiert werden.

Das Hauptaugenmerk liegt dabei weniger auf den Entscheidungen über die auszuführenden Bewegungen /2, 14, 102/, als vielmehr auf einer schnellen Umsetzung der geforderten Funktionen.

Zum Erreichen dieses Zieles müssen folgende Aspekte in Betracht gezogen werden:

- Sowohl im autonomen Einzelbetrieb, als auch im koordinierten Synchronbetrieb sollen bahnkorrigierende Sensorsignale berücksichtigt werden.

- Einer der beiden Roboter soll bei der koordinierten Bewegung der Anordnung die Regie führen (Master) und dem zweiten, abhängigen Roboter (Slave) online Bewegungsanweisungen geben. Der regieführende Roboter hat im Synchronbetrieb die Aufgabe, den zweiten Roboter direkt zu steuern. Zudem soll der Slave während des Synchronbetriebes in der Lage sein, Bewegungen relativ zu der Bahn auszuführen, die ihm vom Master vorgegeben wird.

- Bei einem Wechsel zwischen autonomem und synchronem Betrieb der beiden Roboter ist eine Synchronisation des Slave-Roboters mit den Sollvorgaben des Master-Roboters nötig.

Die Synchronisation soll beschleunigungsgesteuert erfolgen, um einen fließenden Übergang zwischen den voneinander unabhängigen Betriebsarten sicherzustellen. Die zur späteren Berechnung notwendigen mathematischen Grundlagen finden sich zum Beispiel in /4, 62, 75/ und in einer Vielzahl ähnlicher Veröffentlichungen. Die Kenntnis der Repräsentationsformen von Werkzeugorientierungen und der Umgang mit Transformationshilfsmitteln zwischen verschiedenen Koordinaten-

systemen werden als bekannt vorausgesetzt. Zur Illustration der verwendeten Koordinatensysteme dient Bild 7.1.

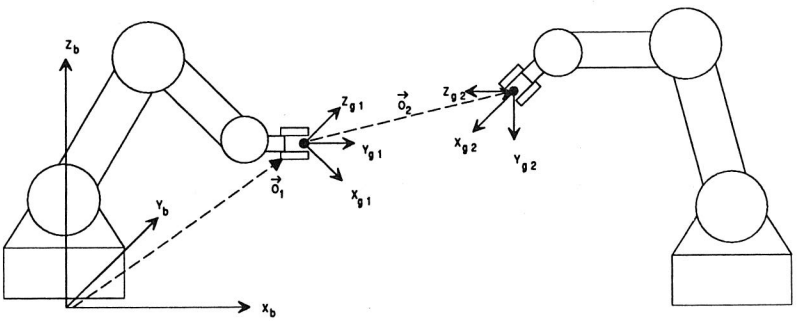



Bild 7.1: Koordinatensysteme der Doppelroboter

Eine spezielle Ausprägung der mathematischen Berechnungen für den skizzierten Fall findet sich bei /9/.

## 7.1 Anwendungsgebiete einer kartesischen Roboterkopplung

Betrachtet man den Stand der Technik in der Montage, so liegt die Schlußfolgerung nahe, daß durch Kooperation von zwei Robotern eine ganze Reihe von Aufgaben bearbeitet werden können, für die bislang nur Spezialgeräte in Frage kommen. Die Leistungsfähigkeit und Flexibilität von Montageanlagen läßt sich so entscheidend verbessern.

### 7.1.1 Handhabungsprozesse mit zwei Robotern bei konstantem Greiferverhältnis

Die Handhabung forminstabiler Werkstücke, wie Schläuche, Kabel oder auch große Dichtungen zählt heute noch zu den Aufgabengebieten, welche nur manuell oder mit spezialisierten Einzweckautomaten bewältigt werden können. Eine flexible Automatisierung dieser Tätigkeiten erfordert den koordinierten Einsatz

mehrerer frei programmierbarer Handhabungsgeräte unter Sensorkontrolle. Dabei genügt es meist, das Werkstück mit den beiden Robotern zu greifen, im Synchronbetrieb mit konstantem Verhältnis der beiden Robotergreifer zueinander zu handhaben und danach das Werkstück loszulassen.

Zur selben Kategorie von Anwendungsfällen gehören Handhabungsprozesse, bei denen Gegenstände mit anderen extremen Eigenschaften zu bewegen sind, welche die Verwendung zweier Greifarme erfordert. Denkbar ist hier beispielsweise eine extreme Massenverteilung des Objekts, so daß beim Anheben an der Greiffläche hohe Drehmomente oder Anpreßkräfte entstehen, welche die Roboterkinematik oder das Teil zu stark belasten.

### 7.1.2 Anwendungsfälle für zwei Roboter bei veränderlichem Greiferverhältnis

Einen möglichen Anwendungsfall stellt das Einhängen von Federn dar. Hier genügt es für die beiden Roboter nicht, mit einem konstanten Greiferverhältnis zu verfahren; es ist zusätzlich eine Bewegung zu überlagern, welche die Feder an ihrem Bestimmungsort einhängt. Diese Bewegung muß während der Kopplung relativ zu der vom Master vorgegebenen Bahn ausgeführt werden. Auch in dieser Anwendungskategorie sind natürlich zahlreiche Abwandlungen denkbar: Statt einer Feder kann ebensogut die Montage von Antriebsriemen bei Plattenspielern oder das Einsetzen von Gummizügen usw. vorgenommen werden /30/.

### 7.1.3 Koordiniertes Fügen mit zwei Industrierobotern

Von großer Bedeutung ist in der derzeitigen Entwicklung der flexiblen Montageanlagen das koordinierte Fügen wie es in Bild 7.2 dargestellt ist. Dabei ahmen die beiden Roboter den Bewegungsbalauf nach, den ein Mensch mit seinen beiden Armen bei der manuellen Bearbeitung derselben Montageaufgabe ausführen würde. So können Fügeprozesse automatisiert werden, die bislang nur manuell möglich waren. Interessant sind dabei insbesondere Fügevorgänge, welche zusätzlich zu dem zu fügenden Element die Handhabung von Verbindungselementen wie Stifte, Bolzen, Schrauben oder Muttern erfordern. Hier erschließt der koordinierte Einsatz zweier kartesisch gekoppelter Industrieroboter ein weites Anwendungsgebiet für die Automatisierung im Montagebereich. Neben dem Be-

reich der Handhabung forminstabiler Objekte stellt das koordinierte Fügen sicherlich die häufigste Anwendung der kartesischen Roboterkopplung dar. Der Fügevorgang kann dabei in Ruhe oder auch während einer Bewegung erfolgen und senkt so die Taktzeit einer Montageaufgabe.

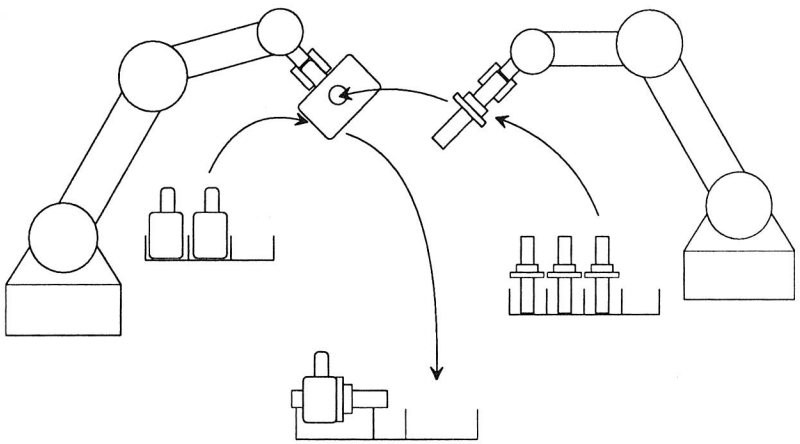


Bild 7.2: Koordiniertes Fügen einer Baugruppe

Eine weit verbreitete, aber nicht sehr komplexe Anwendung stellt die sogenannte Bandsynchronisation dar. Dabei dient ein Förderband als Master, ein frei programmierbares Handhabungsgerät als Slave. Die Aufgabe besteht in diesem Falle darin, Master und Slave bezüglich ihrer Verfahrgeschwindigkeit on-line zu synchronisieren. Danach soll der Slaveroboter ein Werkstück vom laufenden Band abnehmen und an einem anderen Ort abstellen. Das synchrone Annähern an das Werkstück und sein Greifen und Abheben werden vom Slaveroboter als Bewegung relativ zu der vom Band vorgegebenen Bewegungsrichtung während des Synchronbetriebes ausgeführt. Das Ablegen des Werkstücks kann dagegen nach Beendigung des synchronen Betriebs vom Slaveroboter autonom ausgeführt werden.

## 7.2 Bestehende statische Verfahren zur Steuerung von Doppelrobotern

Im Bereich der Mehrrobotersteuerungen gibt es eine Vielzahl verschiedenartiger Ansätze zur Lösung der aufgezeigten Problematik. Dabei werden unterschiedliche Verfahrensweisen angewendet, welche mehr oder weniger leistungsfähig hinsichtlich der komplexen Anforderungen eines sensorkontrollierten Mehrroboterbetriebes sind. Eine Hauptgattung stellen die planenden Verfahren dar. Die Bahnplanungsverfahren legen offline bereits in der Programmierungsphase die vollständige Abstimmung der Bewegungen aller beteiligten Roboter fest /23/. Demgegenüber werden bei einer Roboterkopplung die koordinierten Bewegungen der Roboter online durch einen ausgewählten Roboter in Echtzeit gesteuert. Einen Kompromiß stellt die schritthaltende Planung von Bewegungsparametern für bekannte Bahnkurven dar /16/. Der Hauptunterschied liegt dabei jeweils im Grad der gegenseitigen Abhängigkeit der einzelnen Robotersteuerungen. Diese reicht vom Austausch von Synchronisationssignalen für die Bahnplanungsverfahren bis hin zur Übermittlung von Koordinaten in Echtzeit für kartesische Kopplungen. Sehr häufig wird zwischen den beiden Robotern eine Kraftregelung eingesetzt, da sich so kinematische und bedingt auch dynamische Fehler der Geräte ausregeln lassen /49/.

### 7.2.1 Bahnplanungsverfahren

Die Mehrrobotersteuerung gliedert sich bei Anwendung eines Bahnplanungsverfahrens in Subsysteme, welche autonom die Steuerung der einzelnen Roboter durchführen. Diese Substeuerungen können durchaus identisch sein /16/. Statusinformation im globalen Datenfeld der Steuerungshierarchie dient dem Austausch von Synchronisationssignalen, über die die Substeuerungen logisch verknüpft sind /57/. Das Hauptproblem bei einem synchronen Einsatz der Roboter liegt in der Offline-Programmierung des Gesamtsystems. Dabei ist für jede Substeuerung ein eigenes Anwendungsprogramm bereitzustellen, deren synchroner Ablauf den koordinierten Einsatz der beteiligten Roboter bewirkt. Es gibt spezielle Programmiersysteme /23/, die, top-down von einer bestimmten Mehrroboteranwendung ausgehend, eine zeitliche und räumliche Dekomposition durchführen und so eine allgemeine Zerlegung der Aufgabenstellung in synchrone Bewegungsabläufe der einzelnen Roboter vornehmen. Diese Bewegungsabläufe

werden schrittweise auf Realisierbarkeit, Synchronisierbarkeit, singuläre Stellen und Kollisionsgefahr hin überprüft. Das fertige Programmsystem kann schließlich durch eine Parametrisierung an die reale Hardware der Roboteranordnung angepaßt werden. Um bei der Einleitung synchroner Bewegungsabschnitte mögliche Verzögerungszeiten, die durch das zeitlich versetzte Erreichen der Startpositionen der einzelnen Roboter hervorgerufen werden, zu vermeiden, existieren spezielle Synchronisationsverfahren für bewegte Roboterkonfigurationen /57/.

Der Vorteil der Bahnplanungsverfahren liegt in der einfachen Steuerungsstruktur der Anordnung. Ein Nachteil besteht neben der aufwendigen Programmierung des Systems in seiner geringen Flexibilität während des Betriebes. Da Sensorsignale als statistische Größe betrachtet werden müssen, ist es für eine derart starre Abstimmung der einzelnen Roboter unmöglich, Sensorik zur Bahnkorrektur in den Programmablauf miteinzubeziehen. Als logische Folge ergibt sich eine Einengung des Anwendungsspektrums der Bahnplanungsverfahren im Bereich der sensorkontrollierten Mehrroboteranwendungen.

### 7.2.2 Online Kopplungsverfahren

Steuert man zwei Roboter durch eine Roboterkopplung, also durch eine in Echtzeit betriebene Master-Slave-Abhängigkeit, so werden die Bewegungen des Slaves online aus den Bewegungen des Masters abgeleitet /8/. Manipuliert zusätzlich ein Sensorsignal die Bahn des Masters, so wirkt sich diese Manipulation genau gleich auf die Bahn des Slaves aus. Zudem kann die Bahn des Slaves durch eine eigene Sensorik beeinflußt werden, da die Roboterkopplung nur in Master-Slave-Richtung, nicht aber umgekehrt, wirksam ist. Somit ermöglicht die online betriebene Roboterkopplung ein maximales Maß an Bahnkorrekturen durch Sensorsignale. Bewegungen können mit Hilfe der Slave-Sensorik gesteuert und überlagert zur Masterbewegung ausgeführt werden. Die Überlegenheit der Online-Kopplung von Robotern gegenüber den Bahnplanungsverfahren ist in diesem Punkt offensichtlich.

## 7.3 Dynamische Kopplung für Doppelroboter

Die Verfahren zur direkten Kopplung von Roboterbewegungen müssen durch eine dynamische Komponente erweitert werden, die die Einleitung und Auflösung des koordinierten Betriebs auch bei bewegten Robotern gestattet. Dies ist ein sehr wesentlicher Faktor zum Erreichen kurzer Gesamttaktzeiten einer Anlage.
Unter einer dynamischen Kopplung soll eine spezielle Form der Online-Verbindung verstanden werden, die es erlaubt, daß sich beide Roboter bewegen, bevor sie in den gemeinsamen Bereich eintreten. Bild 7.4 zeigt ein Beispiel für einen Bewegungsablauf, wie er beim koordinierten Fügen während der Fahrt entsteht.

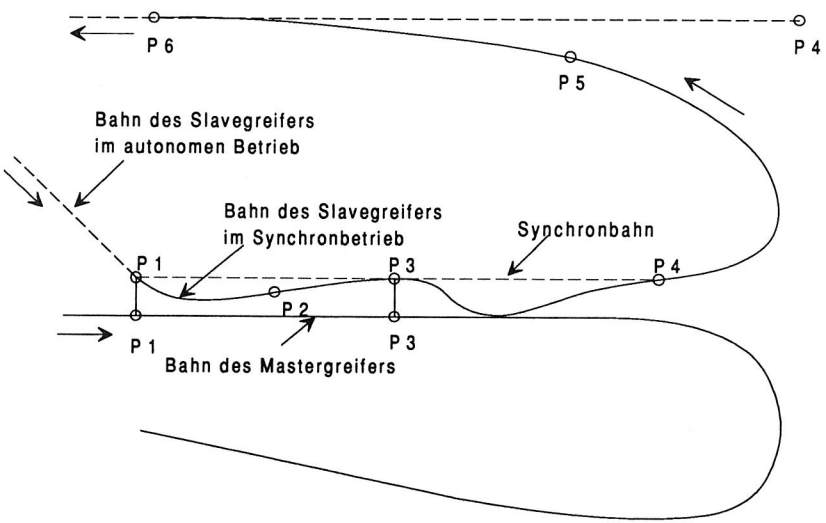



Bild 7.4: Synchronisation von Doppelrobotern

In der Abbildung sind einige ausgezeichnete Punkte eingetragen:

P1 Startpunkt des Synchronbetriebes. Hier beginnt das Synchronisationsintervall.
P2 Der Slaveroboter erreicht den kritischen Bereich der Synchronisation.
P3 Ende der Synchronisation. Ab diesem Punkt bewegt sich der Slaveroboter auf oder relativ zur Synchronbahn.

P4 Endpunkt der Slavebewegung relativ zur Synchronbahn. Der Synchronbetrieb wird hier abgebrochen, die Desynchronisationsphase beginnt.

P5 Der Slaveroboter erreicht den kritischen Bereich der Desynchronisationsphase.

P6 Ende der Desynchronisation. Der Slave arbeitet wieder autonom.

Durch die unbestimmten Vorbewegungen zum Zeitpunkt der Synschronisationsanforderung kann es vorkommen, daß die Herstellung des Synchronzustands in einem vorgegebenen Raumsegment nicht möglich ist. Hier können planende Werkzeuge Unterstützung leisten, um die Realisierbarkeit zu sichern.

### 7.3.1 Grobstruktur der Kopplung

Die beiden betrachteten ***Robotersteuerungen*** sollen zunächst unabhängig voneinander, aber ***gleich getaktet*** in einer gemeinsamen Fertigungszelle arbeiten. Die Forderung nach einem gemeinsamen internen Takt für beide Steuerungen spart Synchronisationsaufwand beim dynamischen Ablaufgeschehen der Anordnung, da bei einem fortwährenden Wechsel zwischen Synchronbetrieb und dem autonomen Betrieb keine Taktabgleichungen für die Robotersteuerungen nötig sind. Das in Kapitel 5 vorgestellte Konzept unterstützt diese Forderung.

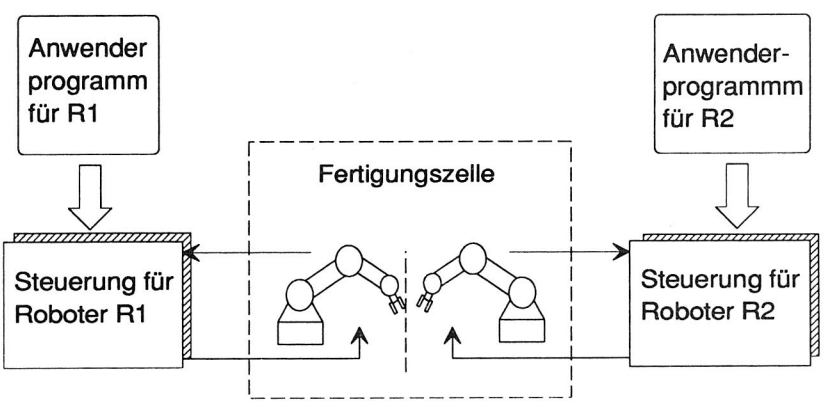



Bild 7.5: Symmetrische Steuerungsstruktur für zwei Roboter

Die Arbeitsbereiche beider Roboter überschneiden sich, denn ohne einen gemeinsamen Arbeitsraum können die Roboter im Synchronbetrieb nicht zusammenarbeiten. Dadurch wird eine umfangreiche Kollisionskontrolle nötig, welche die Steuerungen beider Roboter umfaßt. Die komplexen Aspekte einer Kollisionskontrolle in Mehrrobotersystemen /14/ sollen nicht Gegenstand dieser Arbeit sein, sie bleiben daher im folgenden unberücksichtigt. Sieht man also von der Kollisionskontrolle ab, arbeiten beide Robotersteuerungen gemäß dem jeweiligen Anwenderprogramm unabhängig voneinander. So entsteht die in Bild 7.5 dargestellte symmetrische Steuerungsstruktur für zwei unabhängig arbeitende Roboter.

Sollen die beiden Roboter *R1* und *R2* eine gemeinsame Aufgabe im Synchronbetrieb erfüllen, so müssen beide Steuerungen durch eine überlagerte Komponente verkoppelt werden. Dabei wird ein Roboter (Slave) durch die Steuerung des Masterroboters mitgeführt. Der Roboter *R1* sei der steuernde Masterroboter und *R2* der vom Master mitgesteuerte Slave.

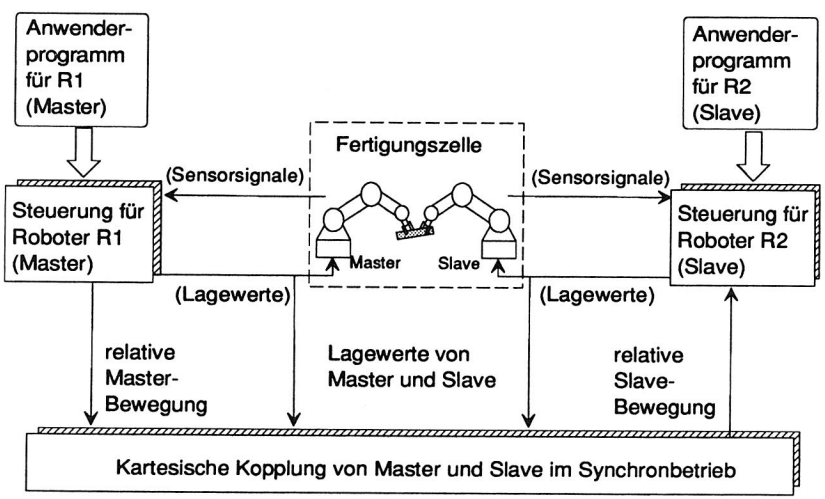



Bild 7.6: Überblick zur kartesischen Master-Slave-Kopplung für einen synchronen Betrieb beider Roboter

Der Greifer des Masterroboters bewegt sich im Synchronbetrieb auf einer kartesisch interpolierten Bahn im gemeinsamen Arbeitsraum von Master und Slave.

Diese Bahn wird durch das Anwenderprogramm für *R1* vorgegeben. Die Führungsgrößenerzeugung des Slaveroboters stellt sich etwas vielschichtiger dar. Prinzipiell wird die Bahn des Slavegreifers durch eine konstante Sollposition relativ zum Mastergreifer bestimmt. Der Slavegreifer muß also so geführt werden, daß seine Stellung relativ zum Mastergreifer immer dieselbe ist. Diese Vorgabe kann aber in aller Regel nicht von Anfang an erfüllt werden: Der Slave muß bei Beginn des Synchronbetriebes mit den Bahnvorgaben des Masters erst noch synchronisiert werden. Ist schließlich die Synchronität von Master und Slave hergestellt, so sollte der Slave in der Lage sein, zusätzliche Bewegungen relativ zu der vom Master vorgegebenen Bahn auszuführen. Diese Bewegungen werden durch das bei Synchronität gestartete Slaveprogramm bestimmt und von der Slavesteuerung ausgeführt. Als Beispiel können hier einfache Fügevorgänge an einem vom Master gehandhabten Werkstück gelten.

Die kartesische Kopplung bezieht die Lage-Istwerte der absoluten Greiferstellungen von den Sollvorgaben der Steuerungen (Bild 7.6) und nicht von den Istwerten der Roboter.

Bei der Konzipierung der kartesischen Roboterkopplung als Betriebssystemkomponente eines Zweirobotersystems wird das Verhalten der beiden Roboter hinsichtlich ihrer Bahntreue als ideal betrachtet. Es wird also angenommen, daß der Robotergreifer stets dort ist, wo er gemäß der Stellungsvorgaben durch die Steuerung sein soll. In der Praxis einer Roboterzelle ist das nicht immer so: Schleppfehler können zu kleinen Abweichungen der tatsächlichen Istposition des Robotergreifers von den Vorgaben der Steuerung führen. Diese sind jedoch für die prinzipielle Funktionsweise der Kopplung nicht von Bedeutung. Bei ähnlichem Verhalten der beiden Roboter können sich die Effekte sogar kompensieren.

### 7.3.2 Funktionsweise der Kopplung

Die kartesische Kopplung von Master und Slave hat als Teil der Steuerung eines Zweirobotersystems zum einen die Aufgabe, Master und Slave zu synchronisieren und zum anderen, die Berechnung der Bahn des Slavegreifers im Raum bei synchronem Betrieb durchzuführen. Sollen dagegen Master und Slave als unabhängige Roboter betrieben werden, so darf die Kopplung keinen Einfluß auf eine der beiden Robotersteuerungen haben. Sie muß also vollständig abschaltbar sein.

Diese Voraussetzungen bestimmen die funktionale Grobgliederung des Steuerungsbereiches zur Roboterkopplung. Der Bereich der Gesamtsteuerung gemäß Bild 7.6 sei im folgenden einfach als ***Kopplung*** bezeichnet. Die ***Kopplung*** gliedert sich nach Bild 7.7 in drei Funktionsbereiche, die einmalige Berechnung der ***Koppelmatrix***, die Berechnung des Sollwertes der Slavelagewerte und den Regler zur Geschwindigkeits- und Lageanpassung.

- Einmalige Berechnung der ***Koppelmatrix***:
  Die ***Koppelmatrix*** beschreibt das konstante Stellungsverhältnis zwischen den Greifern von Master und Slave im Synchronbetrieb. Dieses Stellungsverhältnis wird entweder bereits bei der Anforderung des Synchronbetriebs durch den Master im Programm explizit angegeben, oder muß zum Zeitpunkt der Synchronisierungsanforderung durch den Master einmalig berechnet werden. Dazu wird das aktuelle Stellungsverhältnis der Robotergreifer von Master und Slave zu einem Triggerzeitpunkt berechnet und in der ***Koppelmatrix*** gespeichert. Während des gesamten Synchronbetriebes wird dieses Stellungsverhältnis als Grundvorgabe konstant bleiben und trägt maßgeblich zur Bahnbestimmung des Slavegreifers im Raum bei.

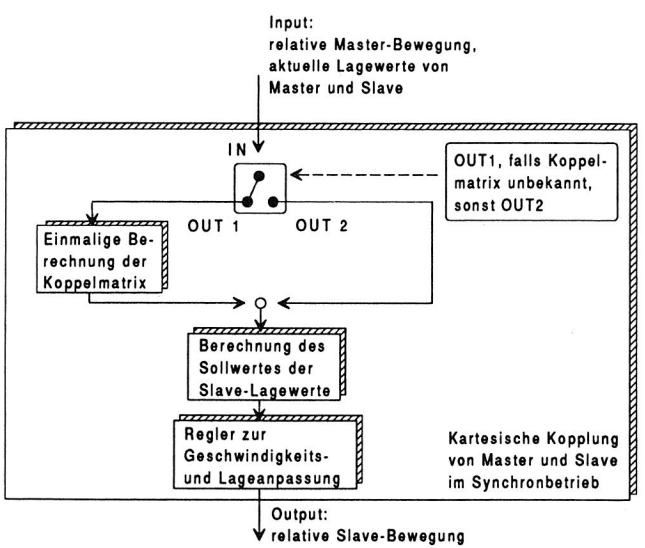



Bild 7.7: Funktionale Grobgliederung des Steuerungsbereiches ***"Kopplung"***

- Berechnung des Sollwertes des Slaveroboters:
  Die Berechnung der Sollwerte für die Slavelagewerte faßt die Einflußgrößen auf die Greiferbahn des Slaves zusammen und errechnet daraus die Sollstellung des Slavegreifers. Zu berücksichtigen sind dabei zunächst nur die ***Koppelmatrix*** und die relative Masterbewegung während des aktuellen Taktes.
  Aus diesen Einflußgrößen wird die Zielstellung des Slavegreifers berechnet, in welcher er nach dem aktuellen Takt stehen muß, um das konstante Greiferverhältnis zum Mastergreifer beizubehalten. Wie in Abschnitt 7.1.2 jedoch bereits festgestellt wurde, ist es häufig nicht ausreichend, Master- und Slavegreifer in einem fest definierten Verhältnis zueinander zu bewegen. Vielmehr sollte der Slaveroboter relativ zu diesem Greiferverhältnis eigenständige Bewegungen ausführen können, die durch ein spezielles Slaveprogramm für den Synchronbetrieb gegeben sind. Diese Bewegungen müssen also noch als dritte Größe, die die Bahn des Slavegreifers beeinflußt, berücksichtigt werden. Die resultierenden Slavebewegungen ergeben sich demnach als Summe der einzelnen Komponenten.

- Komponente zur Geschwindigkeits- und Lageanpassung:
  Dieser Baustein führt mit einer beschleunigungsgesteuerten Führungsgrößenerzeugung die nötigen Synchronisierungen des Slaves bei Beginn und bei Beendigung des synchronen Betriebes durch. Ist der Slave bezüglich des Sollwertes der Slavelagewerte synchronisiert, so kann auf andere Reglerstrukturen (zum Beispiel Kraftregelung) umgeschaltet werden. Für die Aufgabe der Synchronisierung wurde ein Regler mit PI-Verhalten und Begrenzung vorgesehen.

### 7.3.3 Realisierung der Kommunikation zwischen den einzelnen Steuerungsbereichen

Um einen reibungslosen Ablauf bei den Wechseln zwischen autonomem Betrieb und Synchronbetrieb der Zweiroboteranordnung gewährleisten zu können, sind Kommunikationsmechanismen für den aktuellen Zustand notwendig.

In der Programmiersprache der Roboter müssen Befehle zur Handhabung des Ablaufs integriert werden. Die assoziierten internen Aktionen können direkt aus Prozeduraufrufen bestehen, oder aber auch nur über Zustandsmerker Auswirkungen in der gesamten Steuerungstruktur veranlassen.

Der Befehl *SYNC* (Master) fordert den Slave zum Synchronbetrieb an. Das augenblickliche Stellungsverhältnis bestimmt dabei auch die Beziehung während der koordinierten Phase. Soll eine vorher bekannte relative Position eingenommen werden, kann der Befehl um die entsprechenden Parameter zur Definition der Beziehung erweitert werden ( *SYNC* (x,y,z,Φ,Θ,Ψ) ). Mit dem Befehl *READY_FOR_SYNC* kann der Slaveroboter die Bereitschaft zu synchronem Verfahren anzeigen. Soll im koordinierten Betrieb vom Slave eine zusätzliche Bewegung durchgeführt werden, muß außerdem ein Konstrukt bestehen, das die bedingte Verzweigung in das Synchron-Programm ermöglicht (*REACT ON SYNC_STATUS = '1' WITH GOSUB <prog_name>*). In diesem Programm muß die erste Anweisung *WAIT_FOR_SYNCHRONIZED* heißen, damit vor dem Start der Zusatzbewegung die Synchronisation abgeschlossen wird. Durch die Anweisung *UNSYNC* kann der Synchronbetrieb durch den Masterroboter wieder aufgehoben werden. Die Synchronisation innerhalb der Steuerung erfolgt über Variablen, welche die jeweiligen Zustände charakterisieren.

Die Reihenfolge, in der die einzelnen Steuerungsbereiche auf die Zustandsmerker (Flags) zugreifen können, muß während des aktuellen Taktes jederzeit eindeutig definiert sein. Die Tatsache, daß Master und Slave gleich getaktet sind, kommt dieser Forderung sehr entgegen. Aus der jeweiligen Struktur der steuernden Hardware für die Gesamtanlage ergibt sich die Notwendigkeit weiterer Zugriffskontrollen. Verwendet man beispielsweise nur eine Recheneinheit für die gesamte Steuerung der beiden Roboter, so folgt bereits aus der Zuteilung der Rechenzeit für Mastersteuerung, Slavesteuerung und ***Kopplung*** eine eindeutig definierte Zugriffsreihenfolge der steuernden Komponenten auf die Kommunikationsflags. Verwendet man dagegen mehrere Rechnerknoten, so muß die Zugriffsreihenfolge mit Hilfe von Semaphoren synchronisiert werden, um eindeutige Prozeßverhältnisse zu bewahren.

Die nachfolgende Auflistung aller notwendigen Kommunikationsflags der kartesischen Kopplung gibt darüber Auskunft, welche Flags innerhalb der Steuerungsstruktur der Kommunikation dienen.

***SYNC_READY:***
Dieses Flag wird durch den Slave gesetzt. Er zeigt dem Master dadurch seine Bereitschaft zum gemeinsamen Synchronbetrieb an. *SYNC_READY* wird durch die Aufforderung des Masters an den Slave zum Synchronbetrieb zurückgesetzt.

***SYNC_STATUS:***
Mit Hilfe dieses Flags wird die aktuelle Betriebsart der Roboteranordnung angezeigt. Das Flag wird durch die Aufforderung des Masters an den Slave zum Synchronbetrieb gesetzt. Die Zurücksetzung erfolgt, wenn aus irgendeinem Grund der Abbruch des Synchronbetriebes notwendig erscheint, oder wenn der Synchronbetrieb durch den Master - beziehungsweise den Slave - ordnungsgemäß beendet wird.

***SYNCHRONIZED:***
Sobald die Lagewerte und die Bahngeschwindigkeit des Slaveroboters nach gestartetem Synchronbetrieb mit den vom Master geforderten Sollwerten synchronisiert sind, wird dieses Flag gesetzt. Es zeigt der Slavesteuerung an, daß nun die Abarbeitung des Slaveprogramms für den Synchronbetrieb gestartet werden kann. Die Rücksetzung erfolgt bei Beendigung des Synchronbetriebes zusammen mit der Rücksetzung des *SYNC_STATUS*-Flags.

***OUT_OF_SYNC:***
Wenn nach Beendigung eines Synchronbetriebes der Einfluß des Masters auf die Bahn des Slaves vollständig ausgeregelt ist und der Slave zur Abarbeitung seines eigenen Anwenderprogrammes zurückgekehrt ist, wird das *OUT_OF_SYNC*-Flag gesetzt. Seine Rücksetzung erfolgt gleichzeitig mit dem Setzen des *SYNC_STATUS*-Flags, also wenn eine Aufforderung zum Synchronbetrieb vorliegt.

***UNSYNC_TIMEOUT:***
Dieses Flag wird lediglich dann eingesetzt, wenn der Slave nach einem Synchronbetrieb nicht in sein eigenes Anwenderprogramm zurückfinden kann. Um dies zu diagnostizieren, setzt sich unmittelbar nach dem Rücksetzen von *SYNC_STATUS*, also bei Beendigung eines Synchronbetriebes, ein Timer in Gang. Läuft er ab und wurde das *OUT_OF_SYNC*-Flag bis zu diesem Zeitpunkt noch nicht gesetzt, so ist dies ein Indiz für die Unmöglichkeit der Synchronisation des Slaveroboters mit seinem eigenen Anwendungsprogramm. Dies führt nun zum Setzen des *UNSYNC_TIMEOUT*-Flags. Der Slavesteuerung wird somit signalisiert, daß sie nun selbst die definierte Wiederaufnahme des autonomen Betriebs

des Roboters durchführen muß. Die Slavesteuerung quittiert diese Mitteilung durch das Rücksetzen des *UNSYNC_TIMEOUT*-Flags und durch das Setzen von *OUT_OF_SYNC*.

### 7.3.4 Funktionaler Aufbau der Kopplung

Nachdem in Abschnitt 7.3.1 ein erster Überblick über die funktionalen Strukturelemente der kartesischen Kopplung gegeben wurde, wendet sich dieser Abschnitt einer detaillierten Erläuterung der einzelnen Funktionseinheiten zu. Dabei stehen die internen Strategien und Vorgehensweisen, die nötig sind, um diese Funktion ausführen zu können, im Vordergrund. Es sollen auch auftretende Probleme und deren Lösungsmöglichkeiten kurz diskutiert werden.

Zuvor müssen die für die gesamte Steuerungsstruktur geltenden Konventionen hinsichtlich der Darstellung der Bausteine von Funktionseinheiten vereinbart werden. Für die Namen von Strukturvariablen muß ein systematischer und einheitlicher Aufbau eingeführt werden, um die Erläuterung der Funktionseinheiten und deren Variablen nicht unnötig zu verkomplizieren. Im folgenden gelten nachstehende Konventionen:

Variablen, deren Namen mit

- ***"FR"*** beginnen, stellen Frames mit Richtungswinkeln dar, also Objekt aus $\mathbf{R}^6$.
- ***"FE"*** beginnen, verkörpern Frames mit Eulerwinkeln, also ebenfalls Objekte aus $\mathbf{R}^6$.
- ***"DH"*** beginnen, enthalten eine Denavit-Hartenberg-Matrix, ein Objekt aus $\mathbf{R}^{4x4}$.

Wird der Variablenname danach mit

- ***"M"*** fortgesetzt, so ist die Variable im Koordinatenraum des Masters dargestellt.
- ***"S"*** fortgesetzt, so handelt es sich um eine Größe in Slavebasiskoordinaten.

Variable, in deren Namen die Zeichenkette

- ***"DELTA"*** enthalten ist, bezeichnen immer die aktuelle Bewegung des Robotergreifers relativ zur bisherigen absoluten Greiferstellung. Anderenfalls

handelt es sich um die absolute Stellung des Greifers im Basiskoordinatenraum des Roboters.

Ist einem Variablennamen das Prädikat

- ***"(neu)"*** nachgestellt, so enthält diese Variable den im aktuellen Takt berechneten, beziehungsweise zu berechnenden Wert.
- ***"(alt)"*** nachgestellt, so enthält die Variable den bereits im Vortakt errechneten Wert der gleichnamigen, aktuellen Variable.

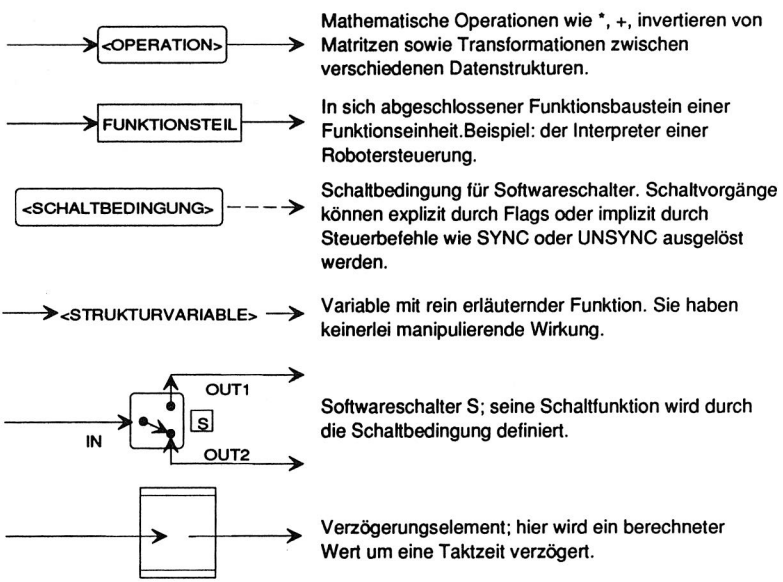



Bild 7.8: Komponenten der Struktur

Ein Beispiel soll die Funktionsweise der eingeführten Nomenklatur verdeutlichen. Die Strukturvariable *"FRM (alt)"* stellt die im Vortakt berechnete, absolute Greiferstellung des Mastergreifers im Basiskoordinatenraum des Masterroboters als Frame mit Richtungswinkeln dar. Weiterhin seien die in Bild 7.8 dargestellten Bausteine einer Funktionseinheit für die gesamte Steuerungsstruktur vereinbart.

Im Bereich der Robotersteuerungen kann man bei Konfigurationen, welche eine kartesiche Roboterkopplung zulassen sollen, zwei Komplexitätsstufen unter-

scheiden. Für den Master genügt dabei im Prinzip eine einfache Bauart der Steuerung, wie sie beim autonomen Betrieb des Roboters verwendet wird.

Die Steuerung des Slaves muß aber darüberhinaus über eine Reihe von Softwareschaltern und weiteren Datenpfaden verfügen, welche dem für die kartesische Kopplung verantwortlichen Steuerteil die nötigen Eingriffsmöglichkeiten gestatten. Aus den Betrachtungen in Abschnitt 7.3.1 geht hervor, daß sich beide Roboter in der Rolle des Masterroboters abwechseln können. Daraus folgt unmittelbar, daß somit beide Robotersteuerungen als potentielle Slave-Steuerungen über die entsprechenden Erweiterungen verfügen müssen. Bei der folgenden Erläuterung der Funktionseinheiten wird die Mastersteuerung jedoch nur als einfache Robotersteuerung dargestellt. Dies geschieht einerseits zur Vermeidung von Mißverständnissen und andererseits, um die Darstellung der gesamten Steuerungsstruktur der kartesischen Kopplung so übersichtlich wie möglich zu gestalten.

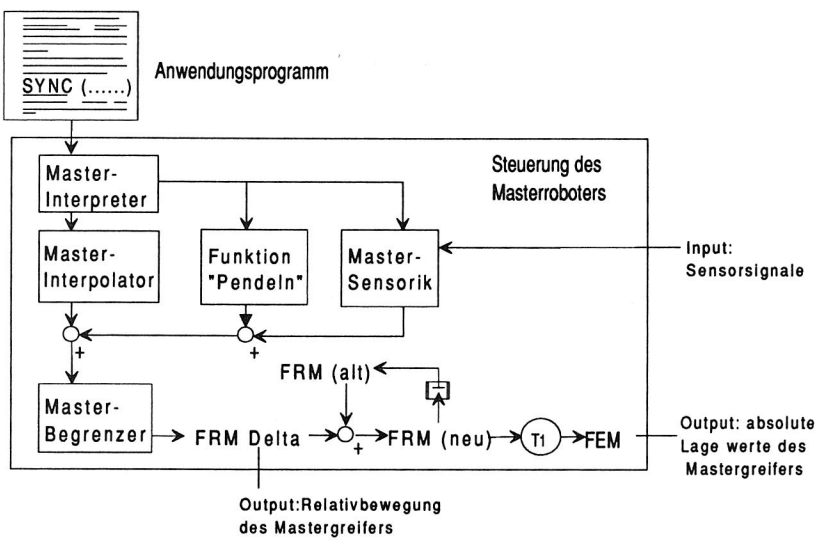



Bild 7.9: Steuerung des Masterroboters (vereinfacht)

Die Funktionseinheit "Mastersteuerung" gliedert sich in die Elemente Interpreter, Interpolator, Begrenzer, Pendelfunktion und Sensorik. Der Interpreter liest in einer Quelldatei den Text des Anwenderprogramms und leitet daraus die nötigen Steuersignale ab. Solche Steuersignale gehen insbesondere zum Beispiel an die Funktion "Pendeln" und an die Sensorik des Roboters. Die Pendelfunktion dient

der Ausführung pendelnder Bewegungen mit kleiner Amplitude im rechten Winkel zur Greiferbahn. Diese Funktion ist beispielsweise beim Bahnschweißen von großer Bedeutung und steht stellvertretend für alle der Bahn überlagerten Funktionen. Die eigentliche Bewegung des Roboters wird gemäß der vom Interpreter analysierten Programmanweisung vom Interpolator vorgegeben. Die Bewegungsanweisungen von Interpolator, Pendelfunktion und Sensorik werden additiv überlagert. Der nachfolgende Begrenzer hat die Aufgabe, die eintreffende Bewegungsanweisung - falls nötig - so zu manipulieren, daß die kinematischen Grenzen der einzelnen Roboterachsen hinsichtlich Beschleunigung und Maximalgeschwindigkeit nicht überschritten werden. Der sogenannte Begrenzer kann auch so ausgeführt werden, daß die gesamte Bewegung stoppt, wenn eine Überschreitung von Grenzwerten festgestellt wird. Man erhält am Ausgang die tatsächlich in diesem Takt auszuführende Relativbewegung des Mastergreifers als Frame mit Richtungswinkeln. Dieser Frame ist zu der absoluten Greiferstellung des Vortaktes zu addieren. Das Ergebnis ist die absolute Stellungsangabe des Greifers nach der anstehenden Bewegung, also das räumliche Ziel der Bewegung als Frame mit Richtungswinkeln.

Die Steuerung des Slaveroboters arbeitet im Prinzip genauso wie die Mastersteuerung. Da sie aber im Synchronbetrieb ihre Bewegungsanweisungen nicht vom eigenen Interpolator bekommt, sondern von der Roboterkoppelung, sind einige Umleitungen von Datenpfaden mittels Softwareschaltern vorzusehen:

Die Schalter *S1* und *S2* (Bild 7.10) stellen die Verbindung zu dem Steuerteil her, das die Kopplung der Roboter durchführt. Stehen *S1* und *S2* auf *OUT 1*, so arbeitet diese Steuerung autonom, also unabhängig von den Bewegungen des anderen Roboters. Das Umschalten beider Schalter auf *OUT 2* ist Teil der Umstellung der Gesamtanlage auf Synchronbetrieb. Die Bewegungsvorgaben durch den Slaveinterpolator, die Pendelfunktion und die Sensorik des Slaves werden zur Bahnberechnung nicht mehr an den eigenen Interpolator direkt weitergegeben, sondern an die kartesische Kopplung übermittelt. Diese überlagert die Slavebewegungen den Bewegungsvorgaben des Masters. Diese Überlagerung wird nach situationsabhängiger Manipulation durch den Regler als Synchronbewegung weitergegeben.

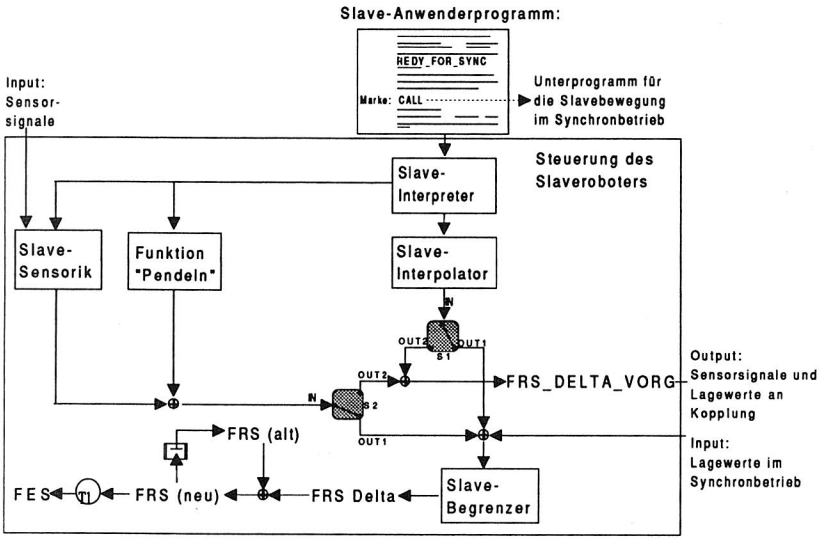



Bild 7.10: Steuerung des Slaveroboters

Die Vorgänge im übrigen Teil der Slavesteuerung entsprechen denen der Mastersteuerung. Ihre Beschreibung ist also identisch mit den entsprechenden Stellen in der obigen Beschreibung der Steuerungsabläufe einer Mastersteuerung.

Denavit-Hartenberg-Matrizen zur Bezugnahme zwischen den Koordinatensystemen (Bild 7.11):

***DHM***: Bezug zwischen Mastergreifer-und Masterbasiskoordinaten in Masterbasiskoordinaten

***DHS***: Bezug zwischen Slavegreifer- und Slavebasiskoordinaten in Slavebasiskoordinaten

***DHB***: Bezug zwischen Slavebasis und Masterbasis in Masterbasiskoordinaten

***DHG***: Bezug zwischen Slavegreifer und Mastergreifer in Mastergreiferkoordinaten. Diese Matrix entspricht der ***KOPPELMATRIX K*** bei Start eines Synchronbetriebes.

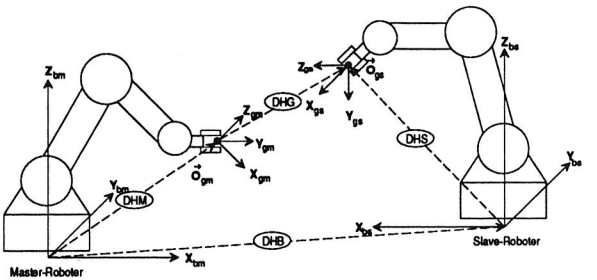



Bild 7.11: Transformationen zwischen den Bezugssystemen

Die Berechnung des Sollwertes der Lagewerte für den Slaveroboter während des Synchronbetriebes und der Phase, in der der Synchronbetrieb wieder aufgehoben wird, ist aufgrund der vielen Einflußfaktoren sehr komplex. Die Vorgaben des Masterroboters bestimmen zunächst eine Bahn, auf der die Beziehung zwischen Mastergreifer und Slavegreifer während des gesamten Synchronbetriebes konstant bleibt. Diese Beziehung ist durch die ***Koppelmatrix*** gegeben. Die beschriebene Bahn wird im folgenden mit Synchronbahn bezeichnet. Die Berechnung der Synchronbahn ist die Basis für nachfolgende Bewegungsüberlagerungen, welche relativ zu dieser Bahn vorzunehmen sind. Dabei stützt sich die Berechnung der Einzelschritte je Takt auf ähnliche Überlegungen wie die Berechnung der ***Koppelmatrix*** im vorigen Abschnitt.

Gegeben sind zwei Frames mit Richtungswinkeln, welche die auszuführende Bewegung des Masterroboters und dessen absolute Greiferposition vor Ausführung dieser Bewegung beschreiben. Mit Hilfe der Transformation ***T2*** berechnet man daraus die Bewegung des Mastergreifers in Form einer Denavit-Hartenberg-Matrix. Diese Matrix beschreibt die räumliche Stellungsänderung des Greiferkoordinatensystems des Masterroboters, welche aus der auszuführenden Bewegung in den vor der Bewegung gültigen Greiferkoordinaten resultiert. Diese Matrix wird mit ***DHM_DELTA*** bezeichnet. Eine Matrix mit derselben Funktion wird für den Slaveroboter gesucht. Dabei macht man sich die Tatsache zu Nutze, daß sowohl vor, als auch nach der Bewegung beider Roboter die Greifer immer noch in dem durch die ***Koppelmatrix K*** beschriebenen, konstanten Verhältnis zueinander

stehen müssen. Aus dieser Erkenntnis folgt unmittelbar die gesuchte Matrix, die mit ***DHS_DELTA_BAHN*** bezeichnet wird:

$$DHS\_DELTA\_BAHN = K^{-1} * DHM\_DELTA * K$$

Die Ausführung zusätzlicher Bewegungen des Slaveroboters relativ zur Synchronbahn ist möglich. Solche Bewegungen können durch Sensorsignale der Slavesensorik, Pendeln des Slavegreifers oder durch einen kompletten Bewegungsablauf relativ zur Synchronbahn entstehen, beispielsweise ein vom Slave auszuführender Fügevorgang an einem vom Master bewegten Werkstück. Die Sollwertberechnung erhält die gewünschte Relativbewegung von der Slavesteuerung in Form eines Frames mit Richtungswinkeln. Da der Frame stets eine Bewegungsanweisung relativ zur Synchronbahn darstellt, handelt es sich also um eine Positionsangabe in einem entlang der Synchronbahn mitgeführten Hilfskoordinatensystem, welches zum Zeitpunkt des Setzens des *SYNCHRONIZED*-Flags durch das Greiferkoordinatensystem definiert wird. Dieser Frame muß mit Hilfe einer weiteren Transformation in eine Denavit-Hartenberg-Matrix umgerechnet werden. Da die resultierende Matrix Bewegungen beschreibt, welche nur im Synchronbetrieb ausgeführt werden, wird sie mit *DHS_DELTA_SYNC* bezeichnet.

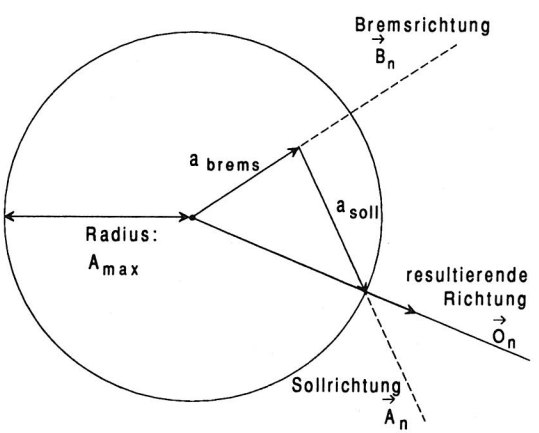



Bild 7.12:Begrenzung der Summe aller Beschleunigungen auf $A_{max}$

Zu Beginn einer Synchronisationsphase werden zwei voneinander unabhängige Bewegungen überlagert. Zum einen beginnt das Koppelmodul mit einer Beschleunigung in Richtung auf die eintreffenden Lagesollwerte. Diese Bewegung

geht von der Position aus, an der der Robotergreifer bei Start der Synchronisation stand. Dabei wird eine Startgeschwindigkeit $v_o = 0\ mm/s$ und eine Anfangsbeschleunigung $a_o = 0\ mm/s^2$ angenommen.

Zum anderen ergibt sich aus der vorherigen Bewegung des Roboters bis zum Start der Synchronisation die Notwendigkeit, den Roboter in der alten Richtung abzubremsen. Die zum Bremsen nötige Beschleunigung sei $a_{brems}$. Betrachtet man alleine den Bremsvorgang, so dauert es einige Zeit, bis der Roboter in dieser Richtung in den Stillstand heruntergebremst wird.

Während dieser Zeit darf der Betrag der vektoriellen Summe von $a_{brems}$ in Bremsrichtung und $max_a$ in Sollrichtung den Grenzwert $A_{max}$ nicht überschreiten (Bild 7.12). Dieselbe Bedingung gilt analog für $a_{brems}$ und der Beschleunigung $min_a$ in der zur Sollrichtung entgegengesetzten Richtung. Eine Verringerung der Beschleunigungsbelastung kann durch eine Beeinflussung der Mastergeschwindigkeit erreicht werden.

### 7.3.5 Zeitlicher Ablauf der Komponenten

Bei einer Steuerung der Duplexroboter-Konfiguration mit nur einem Rechner bildet die sequentielle Abfolge der zur Steuerung verwendeten Funktionen das zentrale Problem. Schließlich müssen innerhalb eines Robotertaktes sowohl die Steuerfunktionen von Master- und Slavesteuerung als auch die Funktionen der kartesischen Kopplung ausgeführt werden. Am Ende des Taktes erhalten dann beide Roboter ihre Bewegungsanweisungen für den kommenden Takt zugeteilt.

Legende für die folgenden Darstellungen:


Aktivitäten, welche R1 zugerechnet werden


Aktivitäten, welche R2 zugerechnet werden


Aktivitäten, welche der kartesischen Kopplung zugerechnet werden


CPU frei

Die zeitliche Reihenfolge, in der die einzelnen Funktionen aufzurufen und vom Rechner zu bearbeiten sind, wird anhand der in den Bildern 7.13, 7.14 und 7.15 dargestellten Gantt-Diagramme festgelegt. Diese Diagramme beschreiben die

zeitliche Auslastung des Rechners je Robotertakt durch eine Sequenz von auszuführenden Funktionen.

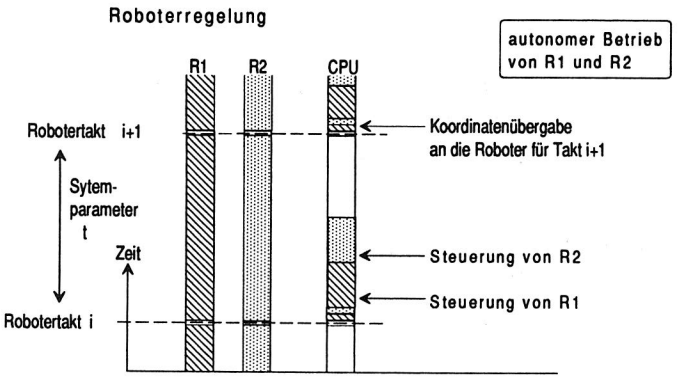



Bild 7.13: Gantt - Diagramm zur Aufrufreihenfolge der Steuerfunktionen bei autonomem Betrieb der Roboter.

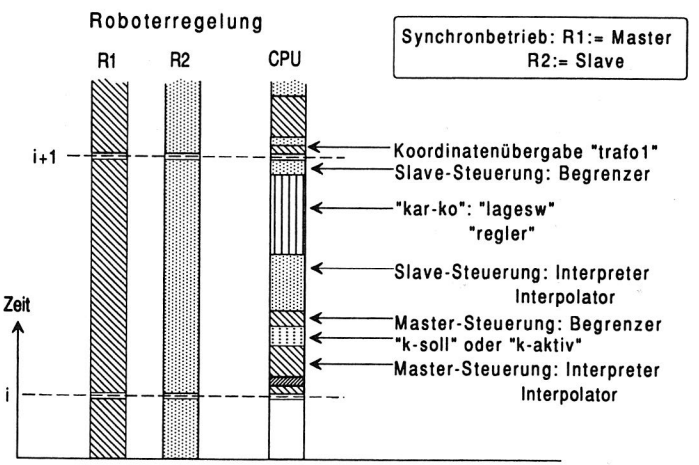



Bild 7.14: Gantt - Diagramm zur Aufrufreihenfolge der Steuerfunktionen im ersten Takt *i* eines Synchronbetriebs der beiden Roboter.

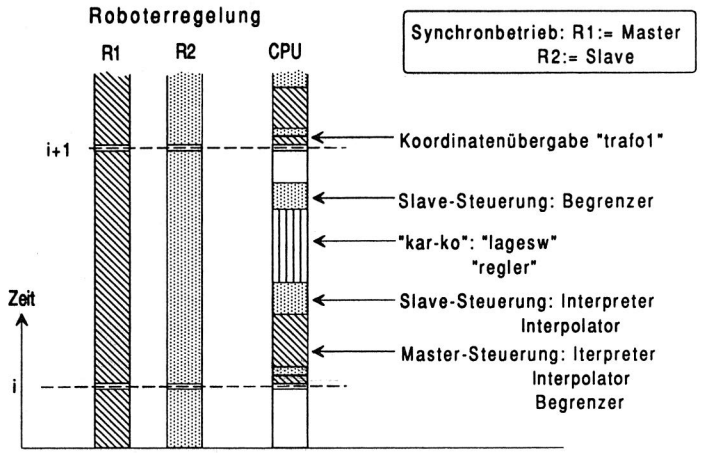



Bild 7.15: Gantt - Diagramm zur Aufrufreihenfolge der Steuerfunktionen im Synchronbetrieb der Roboterkonfiguration.

Die Berechnung der ***Koppelmatrix*** verlangt zusätzlichen Rechenaufwand (Bild 7.14).

Für den Einfluß von Sensorsignalen der Slave-Sensorik während des Synchronbetriebes auf die Synchronbahn läßt sich mit der Programmierung derartiger Bewegungen in Koordinaten eines Hilfskoordinatensystems und deren mathematische Berücksichtigung im Bereich der Lagesollwertberechnung eine befriedigende Lösung finden. Dabei wurden in erster Linie die besonderen Eigenschaften der Denavit-Hartenberg-Matrix verwendet, welche es gestatten, die gesamten Hilfskoordinatensysteme entlang der vom Master bestimmten Bahn mitzuführen, ohne die Koordinatenwelt dieses Koordinatensystems selbst zu manipulieren. Die Inverse der ***Koppelmatrix*** stellt einen festen Bezug zum Mastergreifer im Hilfskoordinatensystem des Slaveroboters her. Dieser Bezug bildet die Basis für die Bewegungen des Slavegreifers innerhalb des Hilfskoordinatensystems.

Im Bereich des *Koppel-Moduls* ist durch die beschleunigungsgesteuerte Synchronisation des Slaveroboters mit den von der Sollwertberechnung vorgegebenen Lagewerten der zweite große Problemkreis gegeben. Dabei liegt in diesem Bereich die größte Schwierigkeit in der Tatsache, daß das Ergebnis der Überlagerung der verschiedenen Vorgaben a priori nicht bekannt ist und es deshalb zu Überschreitungen von Grenzwerten kommen kann. Zudem müssen die

Verfahren zur Berechnung des Istweges so einfach wie möglich sein, da die Rechenzeit für die gesamte Steuerungsstruktur durch den Robotertakt beschränkt wird und somit keine beliebig komplexen Entscheidungsverfahren eingesetzt werden können.

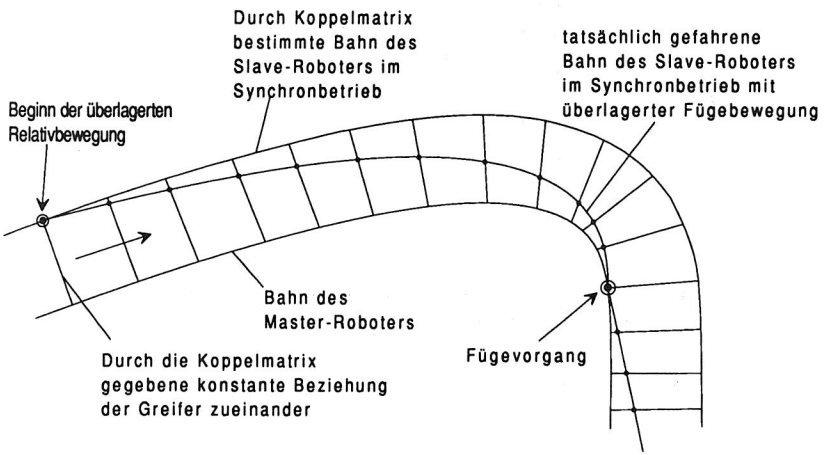



Bild 7.16: Beispiele für den Ablauf eines Synchronbetriebes im zweidimensionalen Fall.

Die vorliegende Auslegung des *Koppel-Moduls* als einfacher Regler lieferte in zahlreichen Simulationsläufen und -versuchen zufriedenstellende Ergebnisse, sofern der Verlauf der Sollwerte nicht ständig und extrem in Richtung und Geschwindigkeit differierte.

Die Einbindung der dynamischen Roboterkopplung in den sonst autonomen Betrieb zweier Handhabungsgeräte bildet ein weiteres Problemfeld. Im Mittelpunkt stehen hierbei die Übergänge zwischen synchronem und autonomem Betrieb.

Zum einen muß der Slave die Abarbeitung seines Anwenderprogramms an einer nicht immer exakt definierbaren Stelle unterbrechen, wenn ihn der Master zum Synchronbetrieb auffordert. Die eingeführten Kommunikationsflags gewährleisten, daß der Slave vom Master erst dann zu einem Synchronbetrieb herangezogen werden kann, wenn es dem Slave ermöglicht ist, sein Anwenderprogramm zu diesem Zweck zu unterbrechen. Zum anderen erfordert der Umstand, daß der Abbruch eines Synchronbetriebes statistischen und damit nicht deterministischen

Einflußgrößen unterliegen kann, die Angabe von Strategien, mit denen der Slaveroboter nach einem Synchronbetrieb den autonomen Betrieb in eindeutig definierter Weise wieder aufnehmen kann.

Versuche haben gezeigt, daß mit sinnvollen Randbedingungen und Bewegungen vor der Synchron-Phase eine dynamische Synchronisation möglich ist. Die vorgestellten Verfahren erlauben einen weichen Übergang zwischen den Betriebsphasen und sichern so kurze Taktzeiten in der Anwendung. Dieses Konzept konnte auf der Basis des im folgenden Kapitel beschriebenen Steuerungssystems umgesetzt und der Synchronbetrieb der beiden Roboter auch praktisch realisiert werden. Als hilfreich hat sich dabei die Möglichkeit zur weitgehenden Modularisierung der Funktionen erwiesen. Die Realisierung mit einem Multiprozessorsystem gestattete gleichzeitig kurze Taktzeiten für die gesamte Interpolation.

# 8 Anwendung des Steuerungskonzepts in einer Modellanlage

## 8.1 Aufbau der Anlage

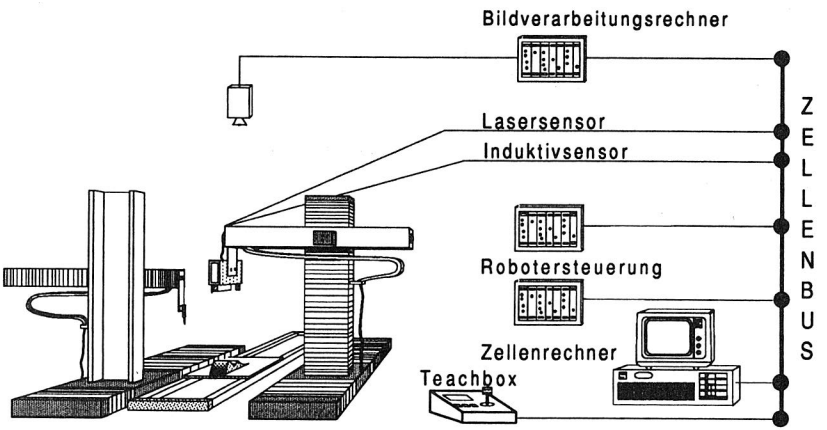



Bild 8.1: Komponenten der Duplexzelle

Als Testumgebung für die Realisierung wurde der in Bild 8.1 skizzierte Aufbau erstellt. Zwei kartesische Geräte mit einem sich überlappenden Arbeitsraum stellen den Kern der Anlage dar. Die Steuerrechner für die Roboter und auch der Bildverarbeitungsrechner basieren auf VME-Bus-Systemen mit Motorola-Prozessoren. Zur Erhöhung der Verarbeitungsleistung bei mathematischen Operationen sind alle Rechnerkarten mit Fließkomma-Koprozessoren ausgerüstet. Die Baugruppen zum Anschluß der beiden Roboter waren zu Beginn der Arbeiten nicht kommerziell verfügbar und wurden deshalb selbst entwickelt.

Alle Steuerungskomponenten sind über einen Feldbus (PDV-Bus /68/) miteinander verbunden. Für das Bedienfeld und die Einfachsensoren wurden besonders kompakte Anschaltungen realisiert. Auf das Kommunikationssystem, das auf dem Bus betrieben wird, wird an dieser Stelle nicht näher eingegangen. Das Themengebiet "Standardisiertes Kommunikationssystem für Feldbusse" wird in parallel laufenden Arbeiten behandelt /6/.

## 8.2 Die Robotersteuerung YARC

Auf der Basis der in Kapitel 5 vorgestellten Überlegungen wurde eine Steuerung für Roboter erstellt. Das hybride Rechnersystem **YARC** (**Y**et **A**nother **R**obot **C**ontrol) enthält die wesentlichen Merkmale zur Steuerung von Montagegeräten auf der Basis des in Kapitel 5 vorgestellten Konzepts. Als Multitask-Rechner fungiert eine Rechnerkarte unter dem kommerziellen Betriebssystem OS9™ /73, 74/ und dient gleichzeitig als Entwicklungssystem. Das echtzeitfähige Multiprozessor-Betriebssystem ROBOS9 wurde im Rahmen der Arbeit erstellt.

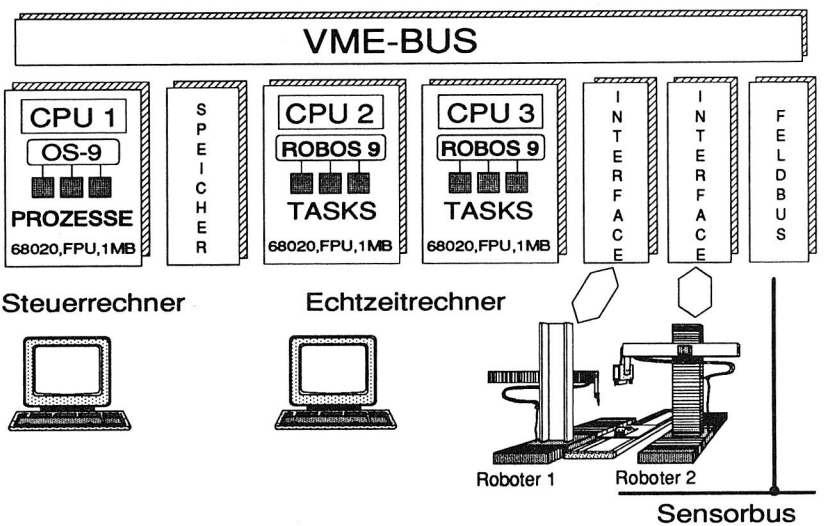



Bild 8.2: Multiprozessor-Steuerungssystem YARC

### 8.2.1 Hardwareaufbau

Beim Aufbau stand nicht maximale Leistung für direkte Regelungsaufgaben im Vordergrund. Deshalb wurde die Ebene der Regelung und Kompensation von dynamischen Effekten ausgenommen. Hierfür gibt es schon verschiedene Ansätze /11, 40, 50/ mit für den speziellen Aufgabenbereich optimierten Strukturen, die sich in das hier vorgestellte System als Subkomponente integrieren lassen. Die

Rechnerkarten bieten dazu einen zweiten Bus-Anschluß (VSB-Bus), der eine Erweiterung zuläßt /20/.

Beim Einsatz von drei Prozessorkarten (Bild 8.2) ist eine hierarchische Strukturierung innerhalb der Steuerung nicht unbedingt erforderlich, da der verwendete Bus einen ausreichenden Durchsatz bietet. Der physikalische Aufbau unterscheidet sich von der logischen Darstellung in Bild 8.1 dadurch, daß sich beide Steuerungen in einem Gehäuse (Cluster) befinden. Zudem kann keine strikte Zuordnung von Rechenaufgaben für einen Roboter zu einer bestimmten Rechnerkarte getroffen werden. Die einzelnen Tasks lassen sich wahlfrei auf die beiden Echtzeit-Rechner verteilen. Die Roboteranschaltungen und die Ein/Ausgabemodule besitzen keine eigene Intelligenz. Für den Anschluß an den Feldbus (PDV-Bus) sorgt ein spezieller Kontrollerbaustein, der eigenständig das Protokoll am Bus durchführt und so die Recheneinheiten wesentlich entlastet. Die Kommunikation mit den Prozessoren erfolgt über einen eigenen gemeinsamen Speicher, der sich auf der PDV-Bus-Anschaltung befindet.

### 8.2.2 Der Echtzeitkern ROBOS 9

Die wesentlichen Komponenten des Echtzeit-Systems werden in diesem Kapitel beschrieben. Der Kern des Echtzeit-Betriebssystems ROBOS9 (**Rob**ot-**O**perating-**S**ystem **9**) wurde aus Effizienzgründen in Assemblersprache erstellt. Die Ziffer 9 im Namen steht in Anlehnung an das Betriebssystem OS9™, da die gleiche Aufrufsynthax für die Betriebssystemdienste verwendet wurden. Viele Programme sind daher praktisch ohne Quellcode-Änderung auf beiden Systemen lauffähig. Das System realisiert ein prioritätsgesteuertes Multitasking mit 6 Prioritätsebenen und unterstützt 32 Tasks je Ebene. Die Priorität nimmt mit steigender Ebenennummer zu. Die Abarbeitung der Tasks geschieht wie folgt :

Es wird mit der höchsten Prioritätsklasse (Ebene 5) begonnen. In der Ebene werden die Tasks nach aufsteigender Nummer abgearbeitet. Sind alle Tasks, die den Prozessor erhalten wollten, abgearbeitet, erhält die nächstniederpriore Ebene den Prozessor.

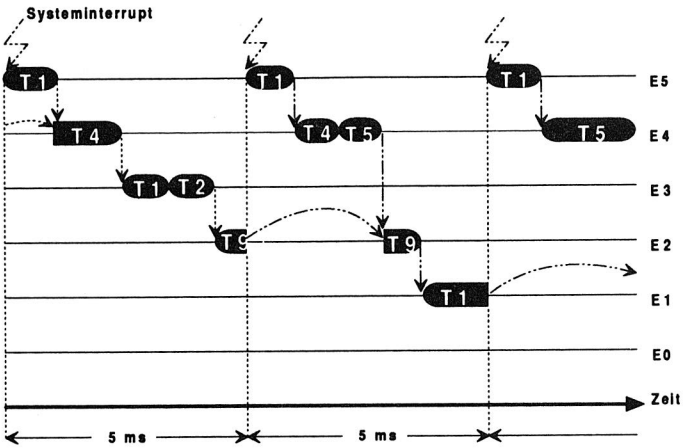



Bild 8.3: Beispielablauf Robos 9 - Tasks

Zu Beginn des Systemtakts wird wieder mit der höchstprioren Ebene begonnen, unabhängig davon in welcher Ebene die Unterbrechung durch den Systemtakt stattgefunden hat. Die Hauptkomponenten der Robotersteuerung, die in den Ebenen 5 und 4 eingeordnet sind, laufen pro Systemtakt einmal ab.
Bei Wiederaufnahme der Bearbeitung in einer Ebene wird diejenige Task fortgesetzt, welche im letzten Zyklus vom Systemtakt unterbrochen wurde (Round-Robin innerhalb der Ebenen).

Zur Steuerung des Taskablaufs existieren pro Prioritätsklasse drei Register:

**Anforderungsregister**
Hier sind die entsprechenden Anforderungen der Tasks vermerkt, die gestartet werden wollen.

**Aktivitätsregister**
Hier sind Tasks eingetragen, die momentan aktiv sind, d.h die Abarbeitung des Programmcodes hat schon begonnen. Dies sind auch Tasks, die den Prozessor freigegeben haben (Wartezyklus).

**Abbruchregister**
Hier sind solche Tasks eingetragen, die von anderen Programmen explizit abgebrochen werden. Diese Tasks durchlaufen, sobald sie das nächste Mal den Prozessor erhalten, ihre eigene Abbruchroutine (vgl. Kap. 5.3.2).

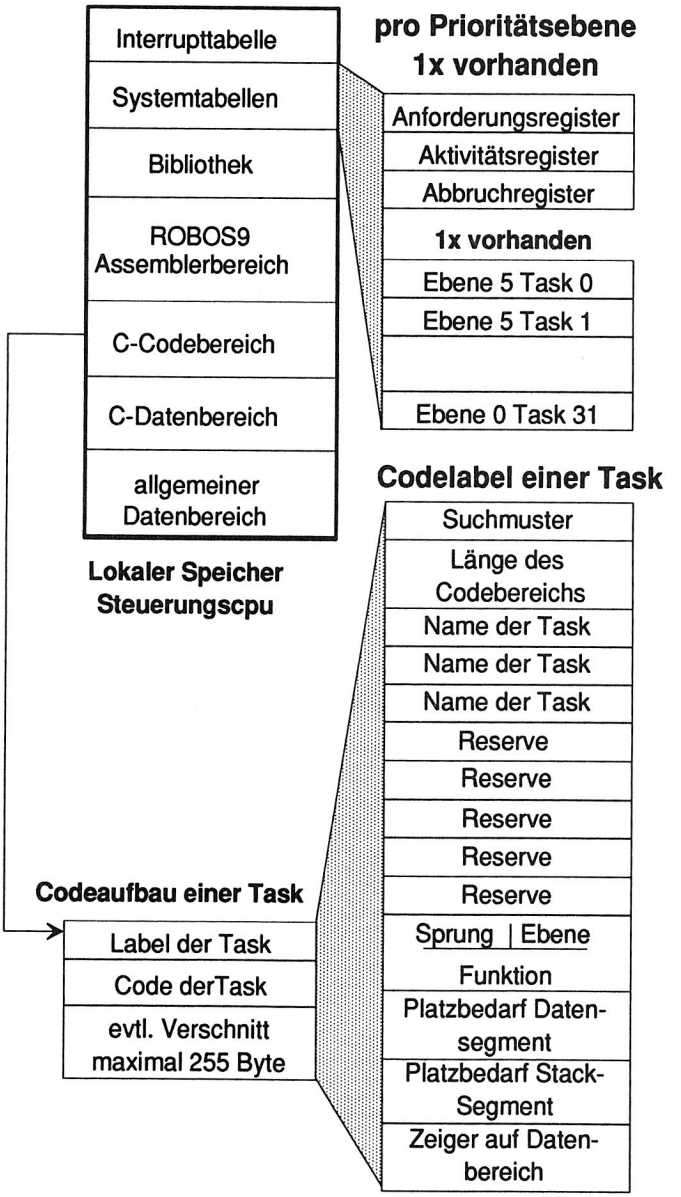



Bild 8.4: Speicheraufteilung eines Echtzeitrechners

**Tasktabelle**

ROBOS9 benötigt die Anfangsadressen des Code- und Datenbereichs der einzelnen Tasks, um diese starten zu können. Dafür verwendet ROBOS9 eine entsprechende Tabelle.

Diese Aufstellung wird bei der Initialisierung des Betriebssystems durch Durchsuchen des lokalen Speichers gefüllt. Die einzelnen Anfangsadressen sind nach abnehmender Ebenennummer und aufsteigender Tasknummer geordnet. Ist eine Tasknummer nicht belegt, so wird dies in der Tasktabelle vermerkt.

Existieren mehrere Versionen einer Task, wird die Startadresse der Version mit der höchsten Speicheradresse eingetragen, so daß die anderen Versionen für das Betriebssystem nicht mehr greifbar sind. Mit einem speziellen Dienstprogramm (*Taskverwalter*) ist es möglich, solche passiven Tasks in einen aktiven Zustand zu versetzen /18/. Damit ist ein schneller Austausch von Tasks möglich. Die Verwaltung aller Programme über Tabellen ist zwar speicherintensiv, bietet aber Verwaltungsvorteile. Die Synchronisierung der Scheduler der Echtzeitkerne erfolgt über einen Mailboxinterrupt, der auf allen Rechnerkarten durch Beschreiben einer Speicheradresse gleichzeitig ausgelöst werden kann (Broadcast).

In der Version 3.0 des Betriebssystems ROBOS9 ist der globale gemeinsame Speicher systematisch aufgeteilt.
In diesem globalen Speicher können alle am Bus angeschlossenen Rechnerkarten direkt lesen bzw. schreiben und auch Semaphoroperationen durchführen. Der Speicher wurde wie folgt partitioniert :

- systemweite Konstanten, Semaphore,
- pro CPU ein Block mit den globalen Variablen und Konstanten,
- pro Roboter ein Block mit allen globalen Daten,
- ein Kommunikationsbereich.

ROBOS 9 zeichnet sich als Echtzeitbetriebssystem durch seine einfachen, bei der Betriebssystementwicklung schon berücksichtigten Mechanismen zur Ansteuerung von Tasks aus. Höherwertige Systemdienste werden durch eigene Server-Tasks erbracht, so daß die Zeitintervalle, während denen der Systemkern nicht unterbrochen werden kann, sehr kurz sind. Die Dienste sind über eine C-Bibliothek erreichbar. Die entsprechenden Aufrufe sind kursiv in Klammern angegeben. Die im Programm verwendeten Bibliotheksroutinen werden nicht dazugebunden, sondern die Bibliothek existiert nur einmal an fester Adresse im Speicher jedes Rechners und ist reentrant programmiert.

Für jeden Mechanismus steht ein eigener Systemaufruf zur Verfügung.
Starten einer Task :

- Erstmalig oder nach Stop
  Durch Setzen der Anforderung der Task. (Systemaufruf : ***setanf(ebene, task))***.

Stoppen einer Task :

- Planmäßig
  Die Task suspendiert sich selbst, entweder für eine bestimmte Zeitdauer (Systemaufruf : ***tsleep(x>0)***) oder für eine von der Task selbst nicht absehbare Zeitspanne (Sytemaufruf : ***tsleep(0)***). Bei der ersten Alternative wird der Zeitserver beauftragt, nach ***x*** Systemtakten die Anforderung der Task zu setzen. Bei der zweiten Version kann eine andere Task zu jedem Zeitpunkt die Anforderung der Task setzen, um diese zu aktivieren.
- Unplanmäßig
  Eine Task kann jederzeit durch Setzen des Abbruchbits abgebrochen werden (Systemaufruf : ***setabb(ebene,task)***). Dies geschieht, indem bei der nächsten Aktivierung automatisch die Abbruchroutine der Task aufgerufen wird. In dieser vom System ROBOS9 vorgeschriebenen Pflichtroutine trägt der Programmierer die Aktionen ein, welche die Task in einen konsistenten Zustand überführen. Im Minimalfall muß die Routine eine "Endeanweisung" (***exit(0)***) enthalten. Im anderen Extrem kann die Abbruchanforderung auch ignoriert werden oder es können noch aufwendige Sicherungsmaßnahmen durchgeführt werden.

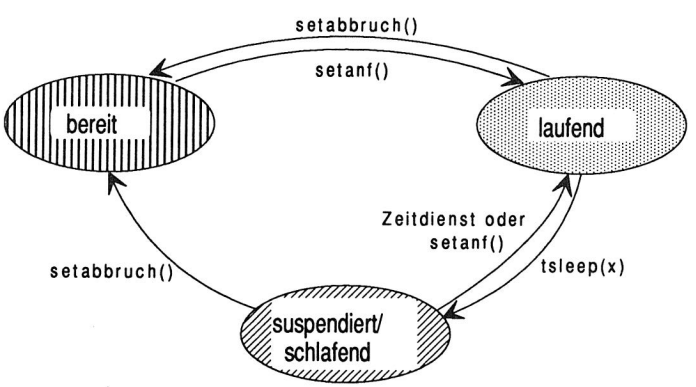



Bild 8.5: Mögliche Taskzustände und zugehörige Systemaufrufe

Im Falle von fatalen Fehlern, wie zum Beispiel Stacküberlauf, Busfehler oder illegalen Befehlen steht ein Notstop für das gesamte Softwaresystem (***panic ()***) zur Verfügung. Dabei werden automatisch auch die Vorgaben für den Roboter gelöscht und die Leistungsverstärker deaktiviert. In der Testversion wird zudem in den Debugger verzweigt und die Adresse des Prozessors sowie die Registerinhalte angezeigt.
Der eigentliche Systemkern umfaßt neben dem Scheduler nur noch eine Interruptroutine für die serielle Schnittstelle zum Füllen eines Zeichenpuffers. Alle weiteren Funktionen (auch das Einlesen von Zeichen ***getc ()***) werden durch Aufrufe einer Bibliothek realisiert.

### 8.2.3 Softwarekomponenten

Ein wesentlicher Bestandteil des Gesamtsystems sind die Hilfsmittel zur Erstellung von Tasks für das System ROBOS9.

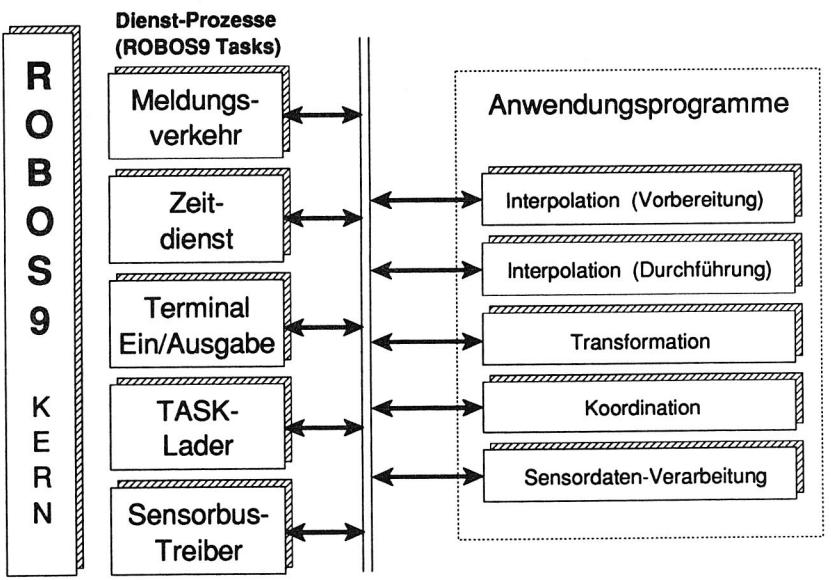



Bild 8.6: Softwarekomponenten des Systems

Editor und C-Compiler stimmen mit dem OS9™-Standard überein. In der Assemblerphase wird der Taskkopf mit den notwendigen Information automatisch

erzeugt. Im Code selbst sind Änderungen durchzuführen (Sprünge absolut), da die Bibliotheksroutinen nicht mehr dazugebunden werden, sondern sich nur einmal (an fester Stelle) im Speicher des Echtzeitrechners befinden. Auch diese Änderung erfolgt automatisch.

Die Bibliothek unterstützt alle Standard Ein-/Ausgabe - und Mathematikoperationen mit Ausnahme der Datei-Funktionen. Bei der Realisierung mußte besonderes Augenmerk darauf gerichtet werden, daß alle Routinen reentrant programmiert sind. Durch die Struktur des Systems ist theoretisch ein sechsfacher Aufruf der gleichen Bibliotheksfunktion möglich. Für kritische Codeabschnitte (zum Beispiel in der Verwaltung des lokalen Speichers mit ***malloc ()***) ist der gegenseitige Ausschluß deshalb über Semophoren geregelt.

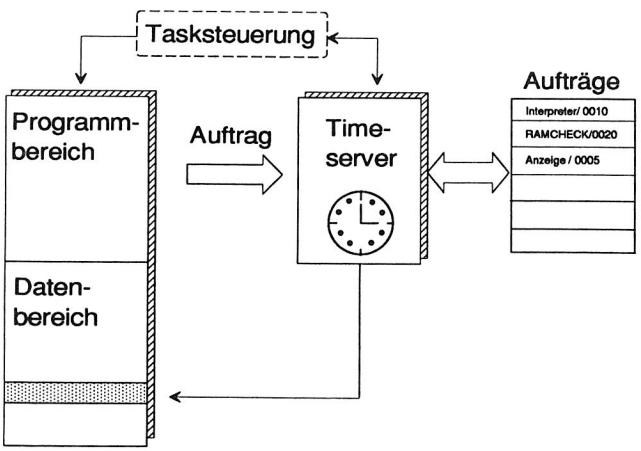



Bild 8.7: Direkter Zugriff des Timeservers auf Variable

Die notwendigen Zeitdienste werden über Bibliotheksfunktionen erbracht, die sich auf eine eigene Task abstützen (*Timeserver*, Ebene 5 - Task 2). Um eine effiziente Realisierung zu ermöglichen, stehen zwei wesentliche Arbeitsmodi zur Verfügung (Bild 8.7). Zum einen ist das die erneute Aktivierung einer Task nach einer bestimmten Zeit. Sehr häufig müssen parallel zur Programmabteilung noch zusätzlich Zeiten überwacht werden. Dies ist am einfachsten dadurch zu erreichen, daß der *Timeserver* direkt Variablen der entsprechenden Task manipuliert. Dazu werden im Auftragstelegramm an den Timeserver die Zeitdauer und die Adresse der Variablen mitgeteilt. An beliebiger Stelle im Programmablauf kann

dann der Ablauf der Zeitspanne anhand des Werts der Variablen überprüft werden. Für genauere Auskünfte kann zudem die verbleibende Restzeit über einen Bibliotheksaufruf (*telltimer ()*) ermittelt werden. Die Vorgehensweise mit der direkten Variablenmanipulation über eine Adresse ist möglich, da der gesamte physikalische Speicher direkt und linear adressiert wird (Motorola-Prozessor, keine Segmentierung oder virtuelle Verwaltung).

Das Dienstprogramm *Taskverwalter* liefert Auskünfte über geladene Programme oder kann Tasks löschen, deaktivieren oder nachladen. Der Ladevorgang wird in der Initialisierungsphase zudem durch einen speziellen Lader unterstützt, der Tasks, Systemkern und Bibliothek direkt vom Massenspeicher des OS9™-Rechners in den Hauptspeicher der betreffenden Rechnerkarte transferiert. Als Entwicklungsrechner diente ebenfalls die Rechnerkarte unter dem Betriebssystem OS9™. Es lassen sich nicht nur die Programme erstellen, die auf diesem Rechner ablaufen sollen, sondern durch entsprechend erstellte Softwareunterstützung auch die Programme für den Echtzeitkern. Mit den vorgestellten Hilfsmitteln können Echtzeitprogramme in der Sprache 'C' effektiv entwickelt und getestet werden.

## 8.3 Sensorgestützte Programmierung

Zur Erfassung von nicht exakt bekannten Konturen oder Konturverläufen, die sich schlecht analytisch beschreiben lassen, eignen sich Programmierverfahren mit Sensorunterstützung /7, 29, 88/. Dabei wird unter Sensoreinsatz der Roboter als Meßeinrichtung benutzt, um in einem Trainingslauf die unbekannte Geometrieinformation zu erfassen. Besonders interessant ist dieser Ansatz in Verbindung mit der Offline-Programmierung oder auch beim Übergang zu impliziten Verfahren, wo die reale Umwelt unbedingt erfaßt werden sollte. Durch den Einsatz geeigneter Sensoren können die Parameter und Geometriedaten den realen Verhältnissen angepaßt werden. Vielfach lassen sich dabei auch Genauigkeitsfehler des Roboters mitkorrigieren.

Im folgenden werden zwei Verfahren zur Ermittlung einer Werkstückkontur und ein spezielles Interpolationsverfahren vorgestellt, das die Erzeugung glatter Raumkurven erlaubt. Ein Programmierverfahren mittels Kraftsensor wird bei /16, 42/ aufgezeigt. Optische Verfahren haben demgegenüber den großen Vorteil, daß Reibungseffekte nicht auftreten und die aufwendigen Kompensationsrechnungen

entfallen. Die Erfassung der Kontur kann zudem meist mit einer wesentlich höheren Geschwindigkeit erfolgen, da kein geschlossener Regelkreis vorliegen muß.

### 8.3.1 Bildverarbeitung zur Konturerkennung

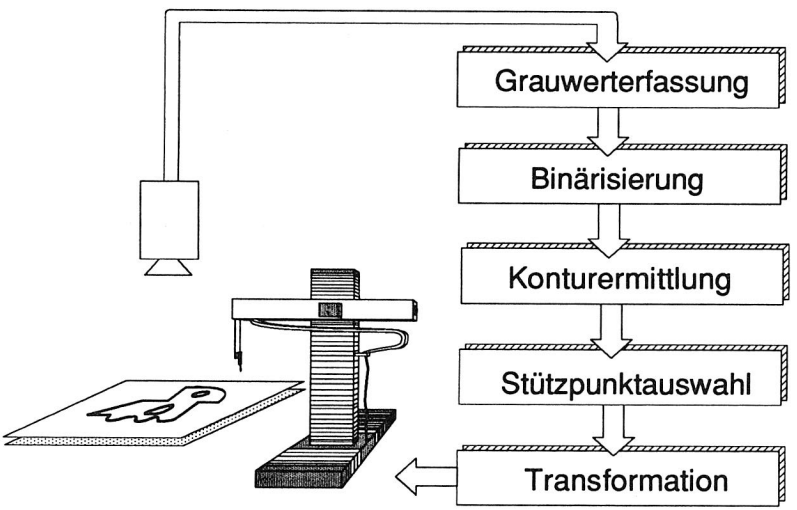



Bild 8.8: Grob-Ablauf der Konturerkennung

In vielen Anwendungsfällen liegen die zu bearbeitenden Werkstückkonturen in einer Ebene und sind somit der Inspektion durch ein Bildverarbeitungssystem zugänglich. Speziell wenn es sich um Außenkonturen handelt und ein ausreichender Kontrast zum Hintergrund existiert, wird eine Auswertung durch eine Matrixkamera sinnvoll. Aber auch Innenkonturen wie tiefere Einschnitte und Bohrungen lassen sich auf hellen oder metallischen Werkstücken sehr gut erkennen. Typische Anwendungen sind der Auftrag von Kleber oder Dichtmasse oder auch die Nachbearbeitung von Werkstückkanten. Für diese Anwendungsfälle wurde ein Kontursystem erstellt, das die besonderen Anforderungen des Anwendungsbereichs berücksichtigt. Nach der Auswertung der Konturlinien im Bildkoordinatensystem erfolgt eine Stützpunktreduktion, denn nicht alle gefundenen Punkte sind für die Regeneration der Konturen notwendig. Durch eine abschließende Trans-

formation der Koordinaten stehen die Konturzüge der Robotersteuerung zur Reproduktion zur Verfügung.

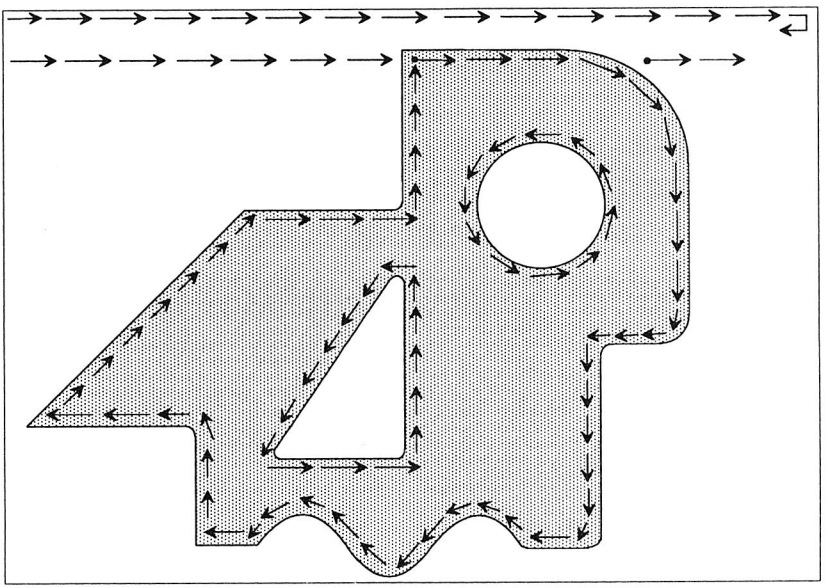
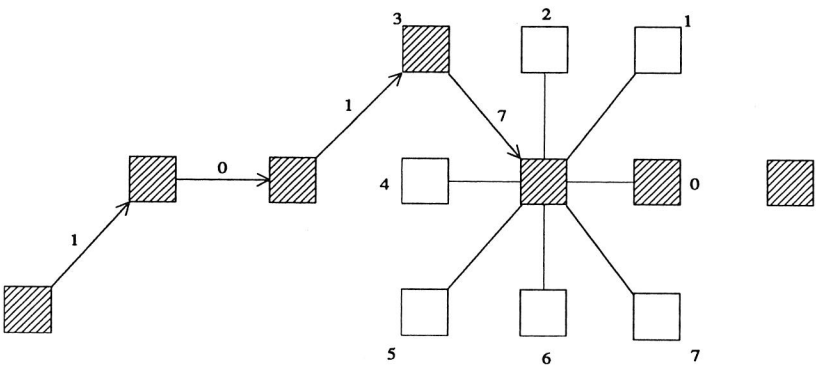


Bild 8.9: Konturermittlung (Grob-Ablauf)

Der Einsatz eines Grauwertsystems hat den entscheidenden Vorteil, daß die Helligkeitsschwelle der Binärisierung in weiten Bereichen der aktuellen Beleuchtungssituation angepaßt werden kann. Durch eine Glättung (Tiefpaßfilterung) im Grauwertbereich oder den Einsatz von Rangfolgefiltern werden Störungen verschliffen, aber auch der Kontrast des gesamten Bildes reduziert /47/. Die Anwendung von Maßnahmen zur 'Bildverbesserung' hängen deshalb sehr stark von der beobachteten Szene ab. Im aktuellen Fall wurde zur Glättung ein einfacher Tiefpaßfilter verwendet.

Auf der Basis des Binärbilds erfolgt die Durchführung der Konturdetektion. Ein Konturpunkt ist durch einen Übergang von '0' nach '1' in der Binärmatrix charakterisiert. Die Bestimmung der Konturen erfolgt gemäß dem in Bild 8.9 gezeigten Verfahren. Die Abarbeitung des Bildes erfolgt zunächst Zeile für Zeile, bis ein Objektpunkt erreicht wird. Von diesem ersten Punkt aus startet das Programm

eine systematische Suche in der Umgebung nach weiteren Objektpunkten (Bild 8.10). Dazu wird zunächst in einer Richtung von +90 Grad zur alten Suchrichtung geprüft, ob sich dort ein weiterer Objektpunkt befindet. Findet sich ein Bildpunkt, der zum Hintergrund gehört, so wird für die nächsten Suchschritte die Richtung jeweils um -45 Grad geändert, bis ein Objektpunkt erreicht wird. Daraus resultierend entsteht schrittweise der komplette Konturzug.




alte Richtung: 7 (315°)
-- > Suche startet in Richtung : 1 (45°) (alte Richtung + 90°)
kein Objektpunkt in Richtung 1 (45°) (vorige Suchrichtung - 45°)
-- > neue Suchrichtung : 0 (0°)
Objektpunkt in Richtung 0 (0°)
-- > Punkt in Richtung 0 (0°) wird zum neuen Konturpunkt

Bild 8.10: Verfahren zur Kontursuche

Das gewählte Verfahren erfaßt die Konturlinie stets so, daß das Objekt stets rechts von der Umlaufrichtung der Kontur liegt. Bei Innenkonturen ist demnach der Umlauf entgegengesetzt zu den Außenkonturen (Bild 8.9).

Bei einem Grauwertsystem können die gefundenen Konturpunkte direkt im Bildspeicher markiert werden (spezieller Grauwert), und so ist das Erreichen des Startpunkts einfach zu erkennen. Um Störungen zu unterdrücken, ist die Einführung einer minimalen Konturlänge sinnvoll. Damit lassen sich aus der Liste aller Konturen diejenigen entfernen, die unter einem bestimmten Umfang liegen. So ist

es möglich gezielt kleine Ausschnitte eines Teils (zum Beispiel Bohrungen) herausfiltern.

Sehr gute Ergebnisse lieferte auch die Glättung der erhaltenen Konturlinie durch einen 1-dimensionalen Medianfilter. Damit lassen sich Störungen in der Kontur mit wenig Rechenaufwand reduzieren.

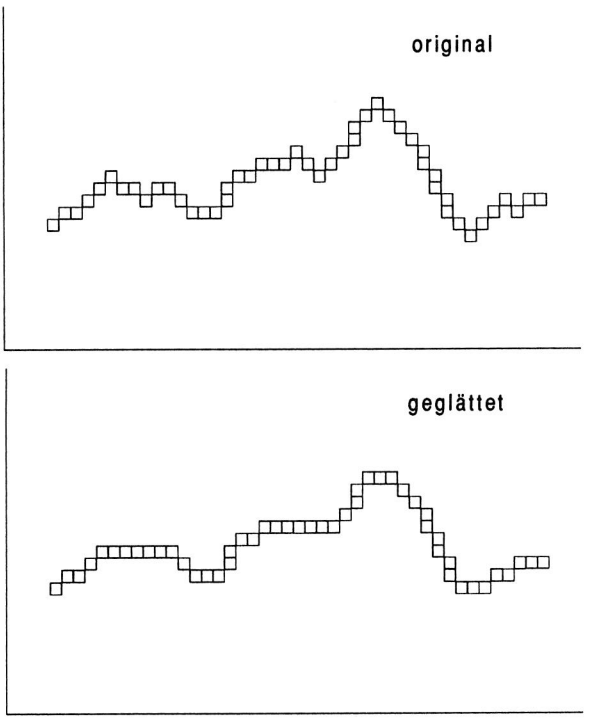



Bild 8.10: Konturglättung mit Medianfilter (1-dimensional)

Auf der Menge der ermittelten Konturzüge muß anschließend noch eine Stützpunktreduktion durchgeführt werden. Dabei wurde die Approximation durch Geraden gewählt. (Bild 8.11).

Zum Erreichen eines minimalen Fehlers in der späteren Ausführung durch den Roboter muß die Stützpunktauswahl mit dem gleichen Verfahren (Linear, Spline) durchgeführt werden, das auch bei der Reproduktion verwendet wird. Verfahren

der dynamischen Programmierung erzielen dabei sehr gute Ergebnisse /16/, allerdings bei extrem hohem Rechenaufwand. Die lineare Approximation (Bild 8.11) zeigte eine ausreichend gute Annäherung für die betrachteten Werkstücke.

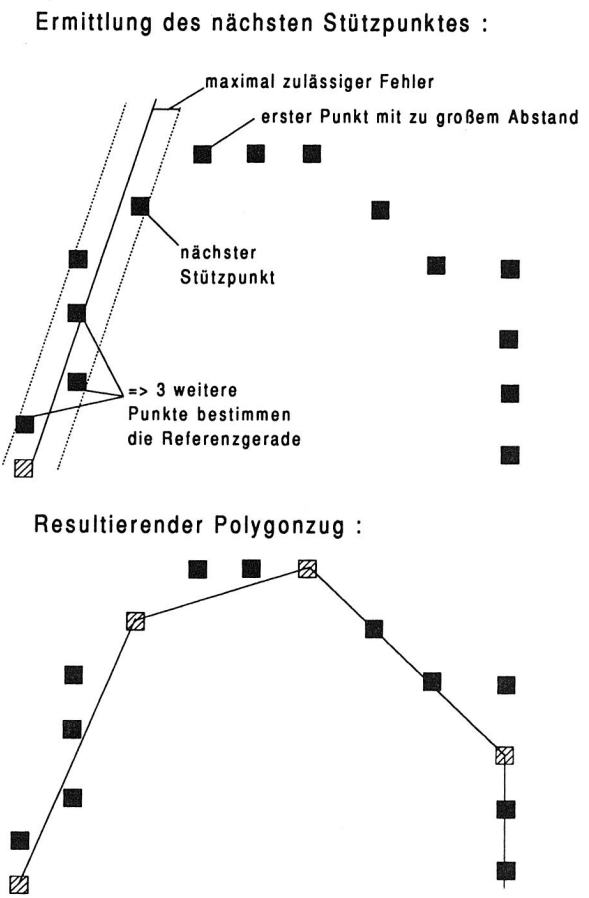



Bild 8.11: Lineare Approximation

Damit die Bildverarbeitungskoordinaten an das Koordinatensystem des Roboters angepaßt werden können, ist abschließend eine Transformation durchzuführen. Hierbei sind die Parameter für Skalierung, Rotation und Translation aus einer geeigneten Meßszene zu ermitteln. Mit je zwei korrespondierenden Punkten *P1*

und *P2* in den beiden Koordinatensystemen lassen sich die gesuchten Parameter bestimmen. Der genaue Ablauf und ein mögliches Meßbild findet sich bei /47/. Als Folge kann dann der Roboter die aufgenommene Kontur nachfahren.

Versuche mit klein gewählten Fehlerschranken zeigen eine Genauigkeit der Erfassung, die auf dem Niveau des theoretisch Möglichen (im konkreten Fall etwa 0,5 mm) liegt. Die erreichbare Auflösung ist demnach nur abhängig von der Pixelanzahl der Kamera und der Ausdehnung der beobachteten Szene.

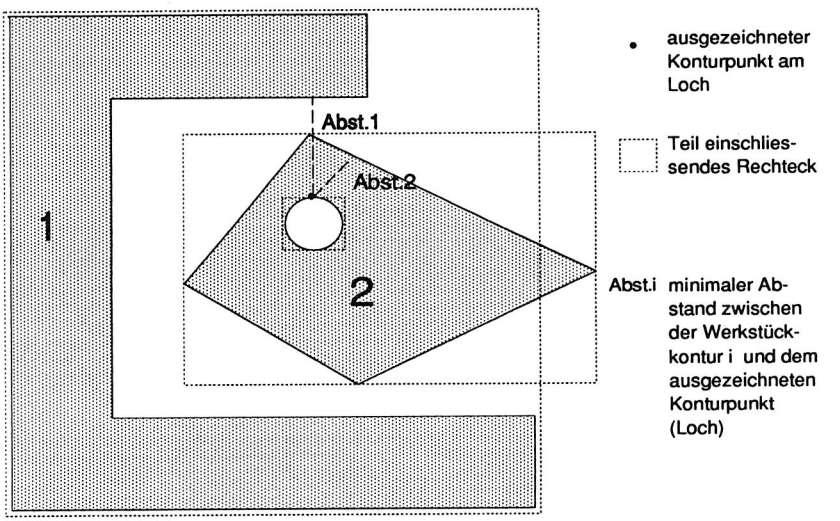



Bild 8.11: Zuordung von Konturen zu Objekten

Befinden sich mehrere Teile in der beobachteten Szene müssen die gefundenen Konturlinien, genauer die Innenkonturen, den jeweiligen Objekten zugeordnet werden (Bild 8.11). Dazu werden die Teile (=Außenkontur) mit einem einschließenden Rechteck umgeben. Der minimale Abstand zwischen Konturlinie und einschließendem Rechteck bestimmt die Zuordnung.

Es steht somit ein Hilfsmittel zur Verfügung, mit dem sich schnell und berührungslos Konturen erfassen lassen. Die erzielbare Genauigkeit ist für viele Anwendungen ausreichend und gestattet so zum Beispiel den Auftrag von Klebe- oder Dichtmasse auf Teilekonturen. Dazu muß nur noch eine kleine Verschiebung der ermittelten Linien in Richtung des Teile-Inneren vorgenommen werden. Dies ist

ohne Probleme möglich, da das vorgestellte Verfahren die Unterscheidung erlaubt, auf welcher Seite der Konturlinie sich das Teil befindet.

### 8.3.2 Profilerfassung mit Abstandssensor

Eine weitere Möglichkeit der sensorgestützten Programmierung wurde durch die Verwendung eines bewegten, eindimensionalen optischen Sensors realisiert (Lasertriangulation).

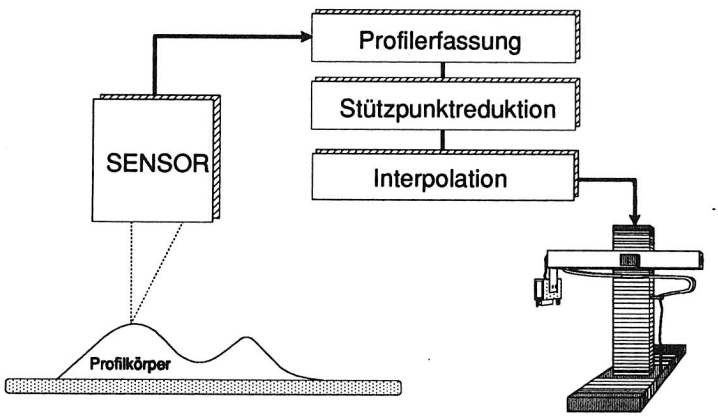



Bild 8.14: Profilerfassung mit Triangulationssensor

Durch den Einsatz eines Feldbusses (PDV-Bus) als Kommunikationsmedium existieren Restriktionen bezüglich der Übertragungsleistung. Die erreichbare Nutzdatenrate ist abhängig von der Länge der versendeten Datenpakete. Für kurze Informationen, wie zum Beispiel die 2 Byte Nutzdaten des Triangulationssensors, konnte dennoch ein Durchsatz erreicht werden (Bild 8.15), der sicher für viele Anwendungen ausreicht. Im vorliegenden Fall stellte der Feldbus keine Einschränkung dar. Voraussetzung ist dabei jedoch, daß die Kommunikation Sensor->Steuerung für die Dauer der sensorgeführten Bewegung.der einzige Vorgang am Bus ist. Das Kommunikationssystem muß in solchen Phasen der Anwendung im Single-Master-Betrieb arbeiten. Sonst lassen sich die in Bild 8.15 gezeigten Datenraten nicht realisieren. Ausführliche Genauigkeitsbetrachtungen des eingesetzten Sensors finden sich bei /27/. Durch Glättung der Stützpunkte und

Reproduktion der Bahn mittels Splineinterpolation konnten sehr gute Ergebnisse erreicht werden.

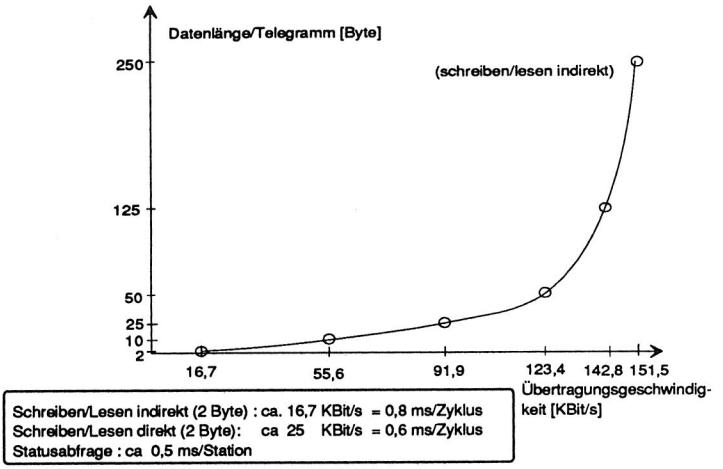



Bild 8.15: PDV-Bus Datenraten (Messung)

### 8.3.3 Konturverfolgung durch Spline-Interpolation

Durch die beiden vorgestellten Verfahren lassen sich Stützpunkte einer Kontur erzeugen. Da Hauptanwendungsgebiete einer Generierung von Geometrieinformationen mit Sensoren überwiegend Freiformkurven sein werden, ist es notwendig, die erhaltenen Kurveninformationen auch mit einem leistungsfähigen Interpolationsverfahren zu regenerieren. Bei den potentiellen Einsatzgebieten, wie zum Beispiel Kleberauftrag und Entgraten, sind häufig möglichst 'glatte' Kurvenverläufe erwünscht /93/. Knickstellen an den vorgegebenen Punkten sollen vermieden werden, um keine Geschwindigkeitssprünge in den Fahrvorgaben für den Roboter zu erhalten. Dies ist durch die Approximation der Kurve mit einem Polynom 3-ten Grades möglich. Diese Funktionen heißen auch kubische Splinefunktionen. Sie besitzen die Minimum-Norm-Eigenschaft /100/ und deshalb die kleinstmögliche Gesamtkrümmung in einem Intervall.

In einem Intervall $I_n = [a,b]$ mit der Stützstellenunterteilung $a = t_0 < t_1 < \ldots < t_{n-1} = b$ ist $S_k(t)$ interpolierende Splinefunktion vom Grad $k >= 0$ über $I^n$ wenn gilt:

$s_k(t) \in C_{k-1}([a,b])$: $s_k$ ist auf $I_n$ $(k-1)$-mal stetig differenzierbar

$s_k(t)$ stimmt in jedem Intervall $[t_{i-1}, t_i]$ $(i=1,\ldots,n-1)$ mit einem Polynom vom Grade höchstens $k$ überein.

$s_k(t_i) = f(t_i)$ $(i=0,1,\ldots,n-1)$: Die Splinefunktion durchläuft alle gegebenen Stützpunkte.

Für $k=3$ kann nach obiger Definition ein Polynom 3-ten Grades berechnet werden:

$$f_i(t) = A_i(t-t_i)^3 + B_i(t-t_i)^2 + C_i(t-t_i) + D_i$$

Zur Berechnung der Splinefunktionen $f_i(t)$ sind jeweils 5 Stützstellen notwendig. Eine Berechnung mit weniger Stützstellen führt zu Unstetigkeiten im Kurvenverlauf /55/. Untersuchungen haben ergeben, daß bei fünf Punkten die Krümmung am letzten Punkt keinen Einfluß auf den Kurvenverlauf im ersten Abschnitt besitzt und deshalb zu Null gesetzt werden darf. Eine Lösung des Gleichungssystems zur Bestimmung der Polynom-Parameter ist deshalb möglich /38/.

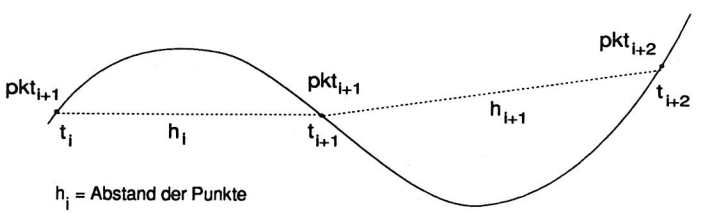



Bild 8.16: Ausschnitt aus einer Splinekurve

In der Definition wird verlangt, daß die Steigungen und Krümmungen der inneren Stützpunkte ineinander übergehen. Analoge Aussagen über die beiden Stützpunkte am Anfang beziehungsweise am Ende des Intervalls sind nicht möglich, da keine Informationen über den Kurvenverlauf außerhalb der Intervallgrenzen zur Verfügung stehen. Je nachdem, welche Bedingungen hierfür zugrunde gelegt werden, unterscheidet man zwischen *natürlichen kubischen Splinefunktionen* (Steigungen und Krümmungen werden vorgegeben - konstant) und *periodischen kubischen Splinefunktionen* (gleiche sich wiederholende Krümmungen an den Rändern). Der allgemeinere Fall sind die natürlichen Splinefunktionen, deshalb beschränken sich alle weiteren Aussagen auf diese Art. Für die Berechnungen zur

Führungsgrößenerzeugung ist eine Parameterdarstellung mit der Bogenlänge ($t$) als ungebundene Variable die geeignete Darstellung.

Die Bogenlänge $h_i$ zwischen zwei Stützpunkten errechnet sich näherungsweise zu:

$$h_i \approx \sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2+(z_{i+1}-z_i)^2}$$

Allgemein kann für die Berechnung jedes Teilintervalls ($[t_i,t_{i+1}]$; $i = 0,\ldots,n-2$) ein kubisches Polynom der folgenden Gestalt zugrunde gelegt werden:

$$f_i(t) = \begin{bmatrix} f_{ix} \\ f_{iy} \\ f_{iz} \end{bmatrix} = \begin{bmatrix} a_{ix} \\ a_{iy} \\ a_{iz} \end{bmatrix} (t-t_i)^3 + \begin{bmatrix} b_{ix} \\ b_{iy} \\ b_{iz} \end{bmatrix} (t-t_i)^2 + \begin{bmatrix} c_{ix} \\ c_{iy} \\ c_{iz} \end{bmatrix} (t-t_i) + \begin{bmatrix} d_{ix} \\ d_{iy} \\ d_{iz} \end{bmatrix}$$

Die Berechnungen für die drei Koordinatenrichtungen und eventuelle Orientierungswinkel laufen vollkommen analog, aber getrennt voneinander ab.
Für jedes Polynom $f_{i.}$ ($i=0,\ldots,n-1$) können die Funktionswerte am Anfang - beziehungsweise am Ende - des Definitionsbereichs berechnet werden. Dies ist ebenfalls für die ersten beiden Ableitungen möglich. In der folgenden Formel können für die Punkte in den Indices die jeweiligen Koordinaten (x,y,z) substituiert werden. Es gilt:

$$f_{i.}(t_i) = d_{i.} = pkt_{i.}$$

$$f_{i.}(t_{i+1}) = a_{i.}h_i^3 + b_{i.}h_i^2 + c_{i.}h_i + d_{i.} = pkt_{i+1.}$$

$$f'_{i.}(t_i) = c_{i.} = pkt'_{i.}$$

$$f'_{i.}(t_{i+1}) = 3a_{i.}h_i^2 + 2b_{i.}h_i + c_{i.} = pkt'_{i+1.}$$

$$f''_{i.}(t_i) = 2b_{i.} = pkt''_{i.}$$

$$f''_{i.}(t_{i+1}) = 6a_{i.}h_i + 2b_{i.} = pkt''_{i+1.}$$

Nach Umformung dieser Gleichung stellt man fest, daß die gesuchten Polynomkoeffizienten nur von den bekannten Stützpunktwerten und deren zweiten Ableitungen abhängen. Bislang wurde die Bedingung der Stetigkeit der Steigungen in den einzelnen Punkten noch nicht ausgenutzt. Wählt man zusätzlich die zweiten Ableitungen in den Rändern ($i=0,\ i=n-1$) zu Null, so erhält man ein lineares Gleichungssystem, mit dem sich die zweiten Ableitungen ermitteln lassen ($i=1,\ldots,n-2$). Die zur Lösung erforderliche Matrix ist symmetrisch, tridiagonal und diagonal-dominant. Das Gleichungssystems hat deshalb immer eine eindeutige Lösung, das heißt auch die Splinefunktion ist eindeutig /100/.

$$h_{i-1}pkt''_{i-1.} + 2(h_{i-1}+h_i)pkt''_{i.} + h_i pkt''_{i+1.} =$$

$$\frac{6}{h_i}(pkt_{i+1.} - pkt_{i.}) - \frac{6}{h_{i-1.}}(pkt_{i.} - pkt_{i-1.})$$

Das Gleichungssystem kann mit den Regeln von Sarrus (Determinante einer Matrix) und Cramer (Lösung eines Gleichungssystems mit Haupt- und Nebendeterminanten) bestimmt werden. Diese Vorgehensweise ist für den konkreten Fall effizienter als der allgemeine Lösungsalgorithmus nach Gauß. Es ergeben sich die folgenden Gleichungen:

$$pkt''_{1.} = \frac{det_{1.}}{det} = \frac{4(h_1+h_2)(h_2+h_3)b_{1.}+h_1h_2b_{3.}-h_2^2b_{1.}-2h_1(h_2+h_3)b_{2.}}{8(h_0+h_1)(h_1+h_2)(h_2+h_3)-2h_2^2(h_0+h_1)-2h_1^2(h_2+h_3)}$$

$$pkt''_{2.} = \frac{det_{2.}}{det} = \frac{4(h_0+h_1)(h_2+h_3)b_{2.}-2h_2(h_0+h_1)b_{3.}-2h_1(h_2+h_3)b_{1.}}{8(h_0+h_1)(h_1+h_2)(h_2+h_3)-2h_2^2(h_0+h_1)-2h_1^2(h_2+h_3)}$$

$$pkt''_{3.} = \frac{det_{3.}}{det} = \frac{4(h_0+h_1)(h_1+h_2)b_{3.}+h_1h_2b_{1.}-2h_2(h_0+h_1)b_{2.}-h_1^2b_{3.}}{8(h_0+h_1)(h_1+h_2)(h_2+h_3)-2h_2^2(h_0+h_1)-2h_1^2(h_2+h_3)}$$

Um eine fortlaufende Berechnung über eine beliebige Anzahl von Stützpunkten zu ermöglichen, ist eine dynamische Fortschaltung des Berechnungsfensters notwendig (Bild 8.17).

Im Übergang von einem Bereich zum nächsten können durch Rundungsfehler Sprünge im Geschwindigkeitsverlauf /55/ resultieren. Diese Fehler lassen sich vermeiden, wenn beim Übergang auf den nächsten Polynom-Abschnitt die aktuelle Geschwindigkeit und Position sowie die Krümmung berücksichtigt werden. Restwege (=Wegdifferenzen), die durch das getaktete Abarbeiten zwangsläufig entstehen, lassen sich bei Verwendung von Gleitkomma-Arithmetik und entsprechender Verrechnung vollständig ausschließen.

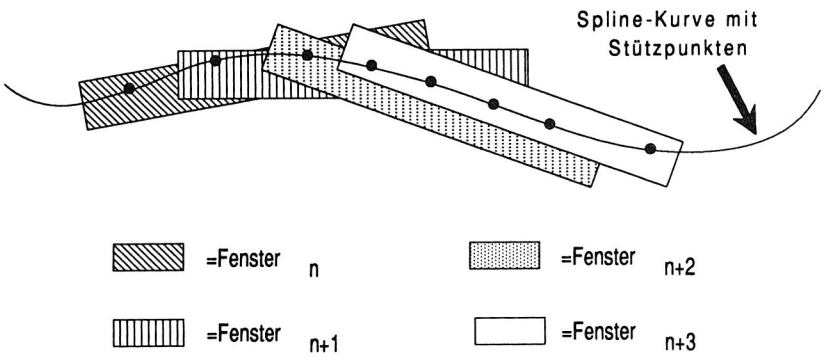



Bild 8.17: Fenstertechnik zur Bahnberechnung

Berücksichtigt man auch noch den Geschwindigkeitsfehler, der durch die unterschiedliche Krümmung der Segmente entsteht, so erhält man auch einen glatten Verlauf der Geschwindigkeit an den Übergangsstellen. Die Geschwindigkeit auf der Bahn hängt von den Inkrementen ($\Delta t$) des Parameters in der Polynomdarstellung ab. Für die Berechnung der Geschwindigkeit kann primär der Abstand zweier Stützpunkte verwendet werden. Da sich bei einer starken Krümmung der Bahn (Bild 8.16) aber ein größerer Weg ergibt als jener der Verbindungsstrecke würde eine höhere Geschwindigkeit als die gewünschte entstehen. Diesen unerwünschten Effekt kann man durch eine Korrekturrechnung, welche die mittlere Krümmung im Segment verwendet, stark reduzieren.

Durch eine Regelung der wahren Geschwindigkeit während der Fahrt läßt sich der Restfehler auch noch beseitigen. Für diese Kontrollrechnungen ist jedoch eine hohe Rechengenauigkeit erforderlich, da die Geschwindigkeit aus den sehr kleinen Wegstückchen ($\Delta s$), die pro Interpolationstakt zurückgelgt werden, ermittelt werden muß. Dies ist umso wichtiger, als man den aufwendig berechneten 'glatten' Verfahrweg sonst durch Rucke in der Mechanik wieder zunichte macht.

Unstetigkeiten in der Geschwindigkeitsvorgabe wirken sich sehr stark auf die gesamte Laufruhe und Genauigkeit des Roboters aus. Ganz abgesehen vom erhöhten Verschleiß durch die stärkere Beanspruchung des gesamten Antriebsstrangs.

Im Bild 8.18 ist die Struktur des Spline-Interpolators dargestellt. Wesentliches Merkmal ist die Realisierung als **einstufige Interpolation**. Die Umsetzung auf dem Multiprozessorsystem erlaubte den Verzicht auf die Ebene der Feininterpolation /91/ und ermöglichte dennoch einen Interpolationstakt von nur 5 Millisekunden. Die Auslastung einer Rechnerkarte liegt während der Fahrt mit Splineinterpolation bei etwa 30-40 Prozent der verfügbaren Zeit. Ein Teil dieser Reserven wird aufgebraucht, wenn das Interpolationsverfahren um eine Orientierungsnachstellung erweitert wird. Für das Programm zur Koeffizientenberechnung (POLY-Task) wurde Ebene 3, Tasknummer 10 gewählt. Die Berechnung der Führungsgrößen (SPLINE-Task) befindet sich in Ebene 4 mit Tasknummer 10. Für die Ermittlung von neuen Parametern des Polynoms für den nächsten Bahnabschnitt (neuer Stützpunkt) wird die freie Rechenkapazität von etwa zwei Interpolationstakten benötigt.

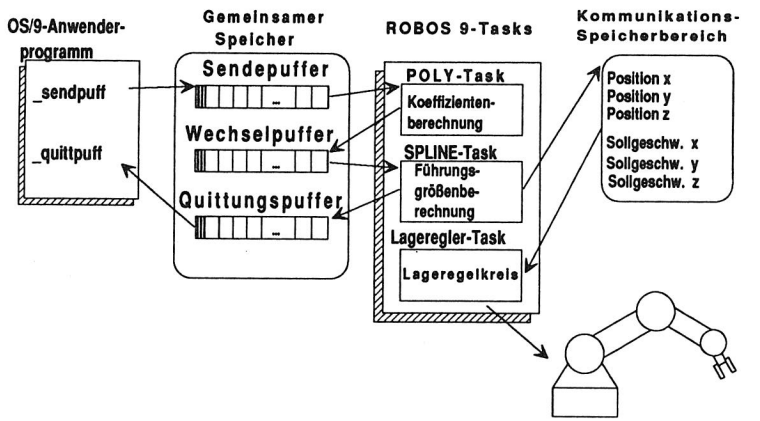



Bild 8.18: Struktur des Spline-Interpolators

Legt man einen Stützpunktabstand von 1mm zugrunde, so kann eine Geschwindigkeit von 100 mm/s gefahren werden, bevor eine Überlastung des Rechners eintritt. Dieser Grenzwert ist nur von theoretischer Bedeutung, denn eine Splineinterpolation bei engen Abständen der Stützpunkte und hohen Geschwindigkeiten (nur zwei bis drei berechnete Zwischenwerte) kommt ohnehin einer Linearinter-

polation gleich. Um die hohen möglichen Geschwindigkeiten nicht zu gefährden, muß eine Entkopplung der drei Komponenten Interpreter, Koeffizientenberechnung und Interpolation vorgenommen werden.

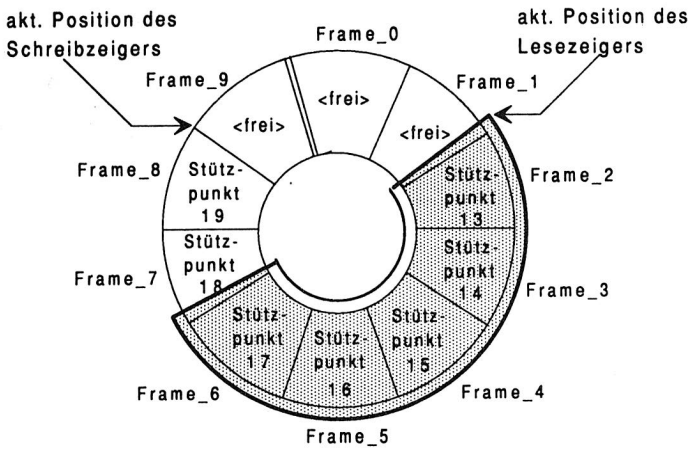



Bild 8.19: Organisation der Stützpunktdaten

Zwischen Interpolation und Vorbereitungsrechnung reicht zum Betrieb die Nutzung eines Wechselpuffers aus. Ein Speicherbereich hält die aktuellen Parameter, und in einem anderen können schon die Werte für den nächsten Abschnitt eingetragen werden.

Die Interpretation des Anwendungsprogramms geschieht auf dem OS9™-Rechner und kann zusätzliche Berechnungen für die programmierten Punkte enthalten. Deshalb ist zwischen Interpreter und Koeffizientenberechnung ein größerer Puffer in Form eines Ringpuffers (FIFO) zu realisieren. Die Speichermöglichkeit von 10 Stützpunkten (Bild 8.19) bringt eine ausreichende Entkopplung, und zudem müssen die Informationen nicht umkopiert werden. Durch Weiterschaltung der Zeiger stehen immer fünf Stützpunkte für die Berechnung zur Verfügung. Die Quittierung von erreichten Stützstellen erfolgt vom Interpolator direkt an den Interpreter.

Die praktischen Versuche haben ein glattes und ruhiges Fahrverhalten der beiden Roboter gezeigt, auch wenn engere Radien mit großer Geschwindigkeit durchfahren wurden.

## 8.4 Multisensoreinsatz zur Werkstückidentifikation

Ein mögliches anderes Einsatzgebiet ist die Erkennung von Fertigungsteilen. Bei der Identifikation von Werkstücken besteht häufig die Situation, daß Eigenschaften, welche sich durch einen einzigen Sensor ermitteln lassen, nicht ausreichen, um eine Erkennung zu gewährleisten. Deshalb ist eine gezielte Anwendung mehrerer Sensoren für eine solche Aufgabe sinnvoll. Ein möglicher Ansatz zur Realisierung einer flexiblen Werkstückidentifikation wird in diesem Kapitel vorgestellt.

### 8.4.1 Systemaufbau

Als Basis dient wiederum die Modellanlage mit den Doppelrobotern. Für die Identifikationsaufgabe wurde der logische Aufbau des Systems geändert. Während der Identifikation entspricht die Aufgabenverteilung der in Bild 8.20 gezeigten Struktur. Die Systemkontrolle wird dem Bildverarbeitungsrechner übertragen, der auch die gesamten Informationen über Teile und Sensoren besitzt.

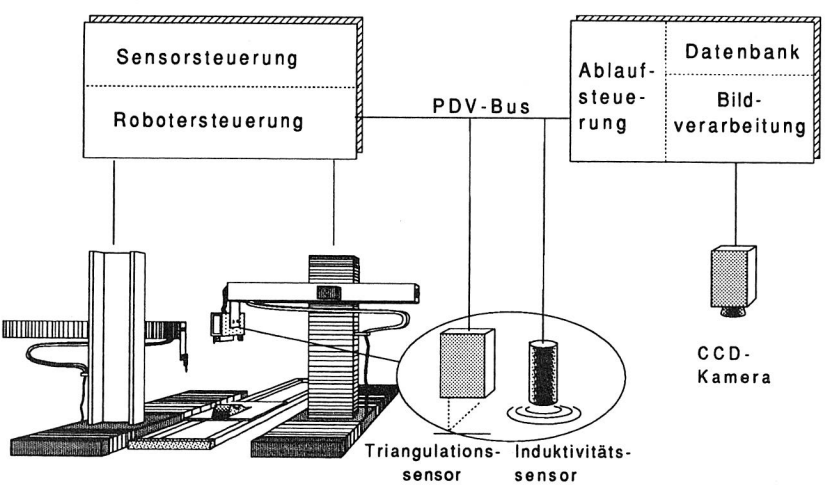



Bild 8.20: Struktur des Identifikationssystems

Es handelt sich um ein Erkennungssystem, das auf Bildverarbeitung basiert und die gewonnenen Informationen durch den Einsatz weiterer Sensoren präzisiert. In der Realisierung standen ein Triangulationssensor für Höhenmessungen und ein binärer Induktivitätssensor zur Verfügung. Grundlage der Identifikation ist eine Datenbank, die (konstante) Informationen über das System (Werkstücke, Sensoren, Geräte, ..) zum Abruf bereithält.

### 8.4.2 Werkstück- und Sensor-Datenbank

Die Datenbank enthält hauptsächlich Angaben zu den einzelnen Werkstücken. Alle relevanten Merkmale eines Teils, die sensorisch erfaßt werden können, sind hier hinterlegt. Aber auch Informationen über die vorhandenen Sensoren und Geräte (Roboter) sind gespeichert. Für den Betrieb und die Wartung wurden zwei Schnittstellen zur Datenbank realisiert (Bild 8.21).

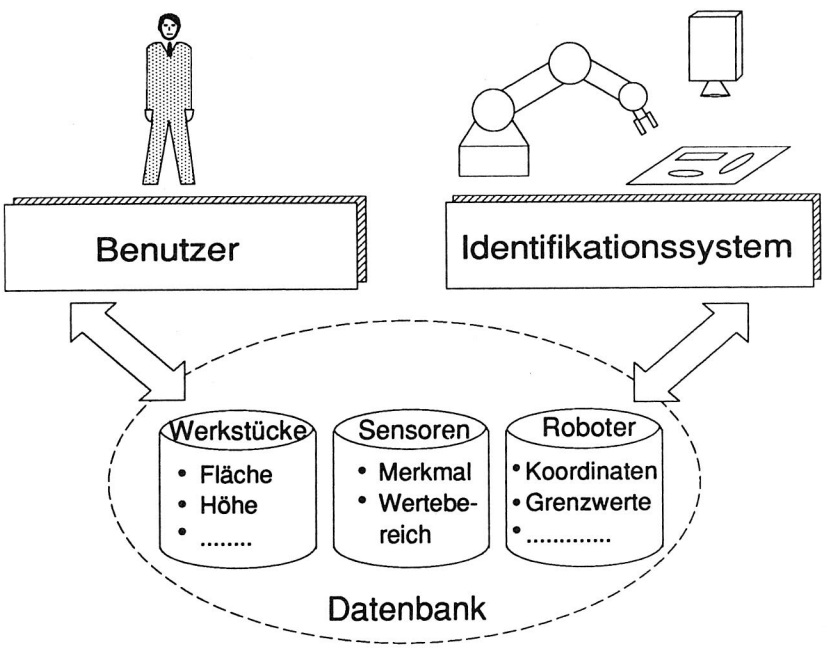



Bild 8.21: Datenbank des Identifikationssystems

Eine Bedienoberfläche erlaubt Benutzerinteraktion zur Eingabe und Modifikation von beliebigen Daten. Für das eigentliche Identifikationssystem steht eine Programmschnittstelle mit über 30 leistungsfähigen Funktionsaufrufen zur Verfügung /59/.

Neben den reinen Datenabfragefunktionen stehen auch Aufrufe zur Verwaltung von aktuellen Werkstücklisten und Unterstützungsroutinen bei der Auswahl des nächsten Analyseschritts zur Verfügung. Die gesamte Datenbanksoftware wurde unter dem Betriebssystem OS9™ mit kommerziellen Datenbankwerkzeugen /70, 71, 72/ erstellt. Als Eingabemedium war ein Standard-Terminal (VT220) vorhanden. Das Bild 8.22 zeigt ein Beispiel zur Selektion von vorhandenen Werkstücken.

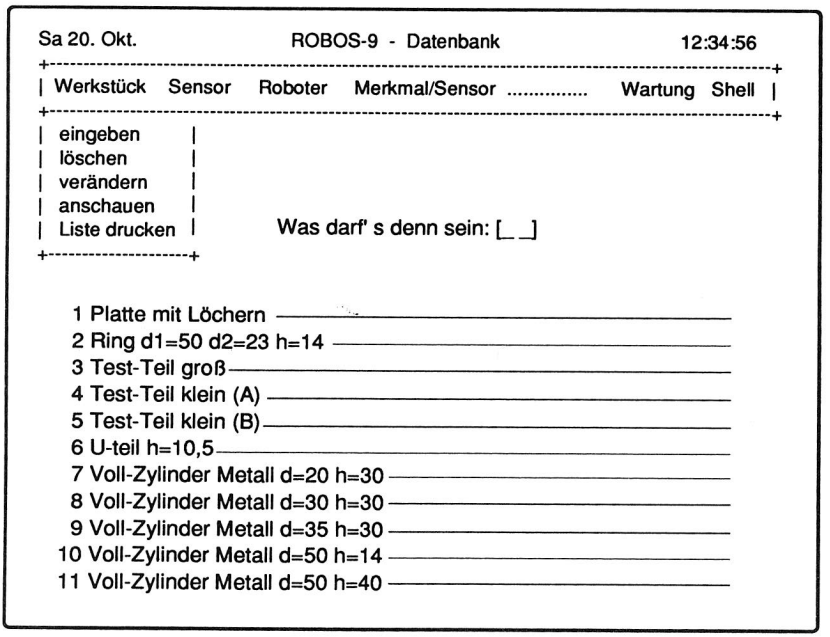



Bild 8.22: Datenbankoberfläche zur Werkstückauswahl

### 8.4.3 Ablaufsteuerung

Der Ablauf des Identifikationssystems basiert primär auf der Analyse der Teile auf einer Palette durch das Bildverarbeitungssystem.

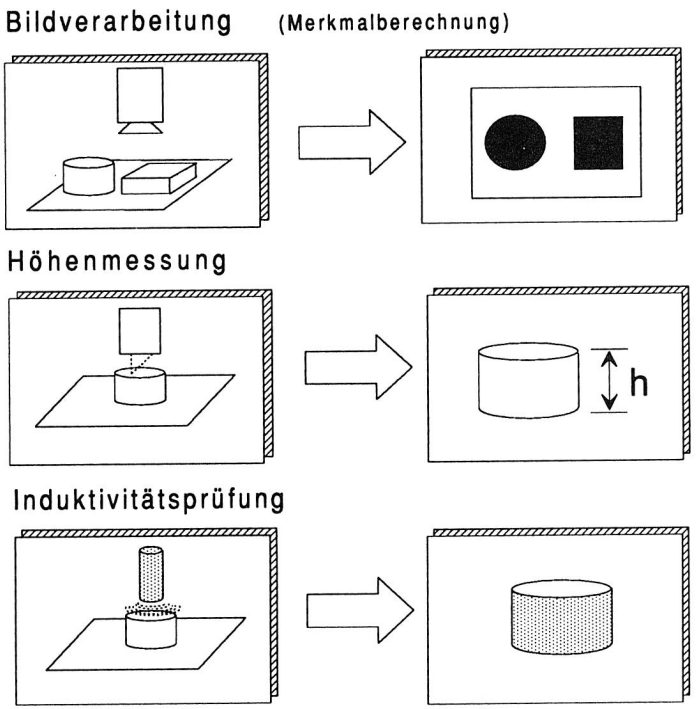



Bild 8.23: Ablauf des Sensoreinsatzes

Die bekannten Randbedingungen für optische Erfassung, vor allem der Kontrast von Werkstück und Hintergrund, müssen dabei eingehalten werden. Es lassen sich dann die Anzahl und Lage der Werkstücke ermitteln /48/.

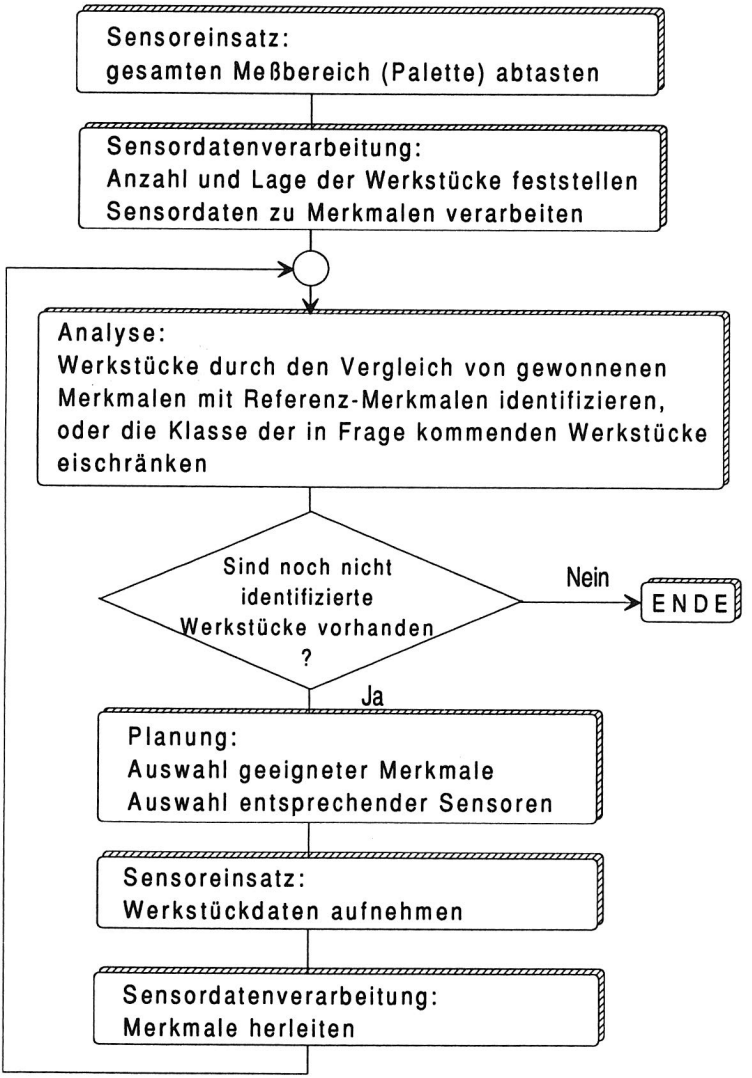



Bild 8.24: Ablaufsteuerung der Identifikation

Die Berechnung von Merkmalen aus dem Kamerabild (zum Beispiel Fläche, Anzahl der Löcher, Momente) erlaubt einen Vergleich mit Referenzmerkmalen in

der Datenbank. Damit läßt sich die Klasse der möglichen Werkstücke einschränken oder eventuell auch das Teil direkt bestimmen.

Liefert die Datenbankabfrage kein eindeutiges Ergebnis ('Werkstücke sehen gleich aus'), so sind zusätzliche Merkmale notwendig. Es gilt, in einer online-Planung ein geeignetes Merkmal und den richtigen Sensor zu ermitteln. Im konkreten Fall besteht hier kein Freiheitsgrad, da bedingt durch die Art der zwei vorhandenen Sensoren die Reihenfolge und die möglichen Merkmale fest vorgegeben sind (zuerst Höhenmessung = Triangulationssensor, dann Induktivitätsprüfung = Induktiv-Sensor). Da die beiden Sensoren jedoch an der Roboterhand angebracht sind, kann der Einsatzort frei bestimmt werden.

In der Datenbank sind eine Merkmal-Sensorrelation sowie Restriktionen in der Anwendungsreihenfolge hinterlegt. So kann die Auswahl auch bei einer größeren Anzahl von Sensoren unterstützt werden.

### 8.4.4 Identifikationsbeispiel

Aus der Bildverarbeitung werden die Fläche, die Anzahl der Löcher und sogenannte Momente berechnet. Mit den gewonnenen Werten muß nun zunächst eine Lagenormierung durchgeführt werden. Es handelt sich um einen flachen Quader, welcher zwei Zapfen gleicher Höhe aus unterschiedlichen Materialien (Metall, Kunststoff) besitzt. Für das Beispiel (Bild 8.25) ergeben sich dabei vier Möglichkeiten der Drehlage. Die Berechnung der Vorzugs-Orientierung kann nur für asymmetrische Teile oder maximal noch für Vierecke direkt aus den Momenten erfolgen. Hat ein Werkstück mehr als 4 symmetrische Winkelstellungen muß die Ermittlung über aufwendigere Verfahren erfolgen. Dazu werden die Konturlinien in eine Polardarstellung bezüglich des Schwerpunktes der Fläche transformiert. Aus dem Abstandsverlauf Außenkontur/Schwerpunkt lassen sich auch Vielecke und sogar Kreise erkennen. Im Falle eines Kreises kann keine Vorzugslage bestimmt werden.
Im aktuellen Beispiel (4 Symmetrien) muß also die Höhenmessung an mindestens zwei Positionen erfolgen. Die relative Lage der Zapfen auf dem Werkstück ist dabei vorher bekannt (Datenbank). Die Ablaufsteuerung übergibt die Meßaufträge mit den Ortsparametern an die Robotersteuerung. Der Roboter fährt die entsprechenden Positionen an und initiert die punktförmige Höhenmessung durch den Lasertriangulationssensor.

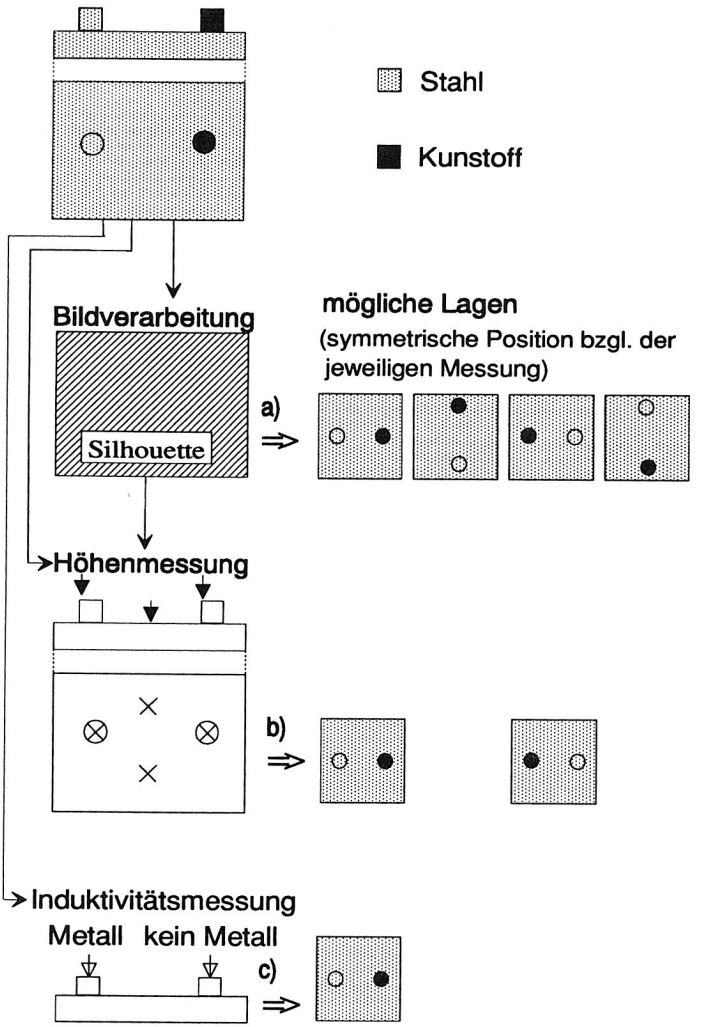



Bild 8.25: Identifikationsbeispiel

Jetzt reduzieren sich die möglichen Symmetrielagen auf zwei Positionen. Durch die abschließende Induktivitätsprüfung eines Zapfens kann die exakte Lage bestimmt werden. Hierzu ist neben der Meßposition in der Ebene auch noch das Ergebnis der vorangegangenen Höhenmessung nötig, denn der Sensor hat nur einen sehr kleinen Erfassungsbereich (etwa 2mm). Gibt es nur das eine Werk-

stück im Teilespektrum, so reicht die Bestimmung der Induktivität eines Zapfens aus, um die Lage des Teils endgültig festzulegen.

Das vorgestellte Sytem läßt sich einfach um weitere Sensoren ergänzen und sich somit an andere Aufgaben anpassen. Die Ablaufsteuerung und die Datenbank sind für Erweiterungen vorbereitet.

## 8.5 Visualisierungskomponente

Bei der Darstellung von Informationen gilt es, zwei große Bereiche zu unterscheiden. Ein Großteil der internen Programme muß alphanumerische Ausgaben tätigen. Ein Teil der Informationen kann graphisch dargestellt und deshalb durch den Bediener einfacher erfaßt werden.

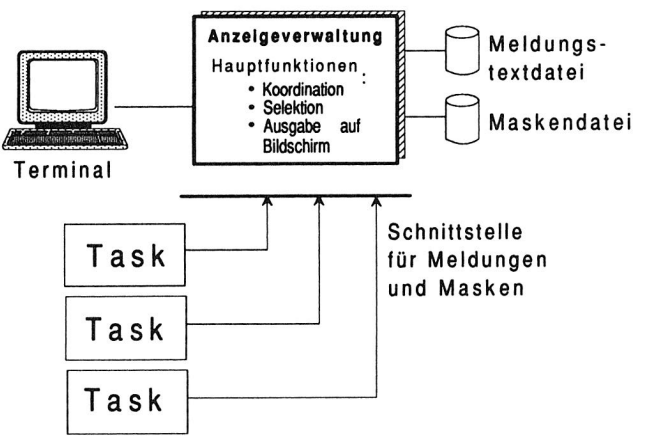



Bild 8.26: Grobstruktur des Anzeigesystems

Eine Hauptaufgabe der Anzeigeverwaltung ist die Koordinierung der Zugriffe auf das Ausgabemedium Bildschirm. Damit nicht alle Tasks wahllos Textausgaben machen können, muß ein eigenes Programm zur Kontrolle der Abläufe eingesetzt werden. Es ist sinnvoll, den Bildschirm in einzelne Bereiche zu gliedern (LE = Logische Einheiten), um die gleichzeitige Darstellung verschiedenartiger Daten zu ermöglichen. Eine bewährte Strukturierung ist die Trennung von Kopf/Menue-Leiste, Text - (Grafik -) Fenster, Kommandozeile und Meldungsbereich. Diese Ab-

schnitte werden üblicherweise in der genannten Reihenfolge von oben nach unten auf dem Bildschirm angeordnet. Der Einsatz in einer Gerätesteuerung erfordert noch zusätzliche Funktionalität, was die Pufferung und Priorisierung der Ausgaben angeht. Es gibt eine Unterscheidung in der Dringlichkeit der Meldungen.

Die Meldungen lassen sich in drei Klassen unterteilen.

- Meldung ohne Quittung durch Bediener,
- Meldung mit Quittung durch Bediener,
- Meldung mit Quittung durch Bediener und Rückmeldung an das beauftragende Programm.

Die letzte Klasse kann dabei nochmals unterteilt werden, je nach Dringlichkeit der Meldung. Diese Aufteilung in verschiedene Meldungsklassen spiegelt sich in der internen Struktur der Anzeigeverwaltung wider (Bild 8.27). Da keine Meldungen verloren gehen dürfen, müssen Puffer für jeden Bereich vorgesehen sein.
Eine hochpriore Meldung muß zudem eine bereits auf dem Bildschirm stehende Meldung verdrängen können. Nach Quittierung der aktuellen Anzeige wird dann die nächstwichtige Meldung wieder zur Anzeige gebracht. Dieses Vorgehen ist deshalb notwendig, da durch einen Fehler mehrere Reaktionen gleichzeitig in unterschiedlichen Tasks entstehen können und keine Meldung verloren gehen soll.

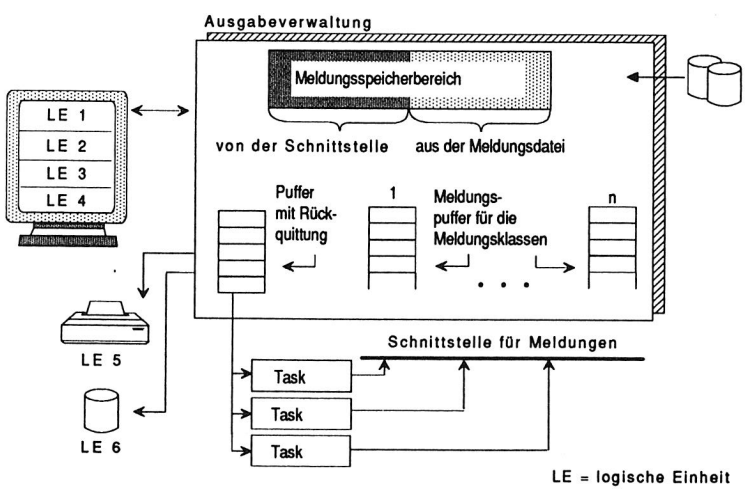



Bild 8.27: Interner Aufbau Monitorverwaltung

In der Realisierung auf einem Alpha-Terminal erscheint die Oberfläche in der angesprochenen Strukturierung (Bild 8.28). Das Programm *Monitorverwaltung* besitzt eine niedrige Priorität und ist deshalb als Ebene 0 Tasknummer 28 eingeordnet. Neben der Ausgabefunktionalität ist auch eine einfache Bedienoberfläche realisiert.

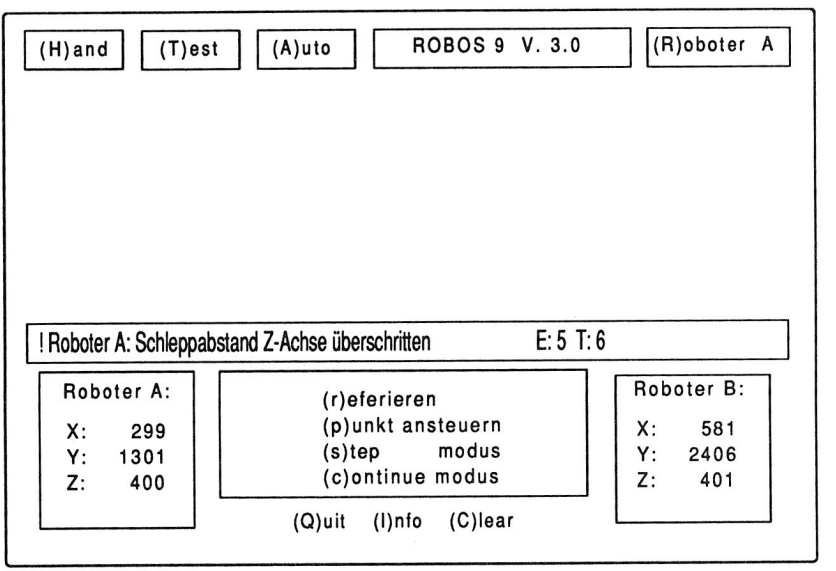



Bild 8.28: YARC - Bedienoberfläche

Mit einer Überwachungs- und einer Aufzeichnungskomponente kann die aktuelle Auslastung der Echtzeitrechner beobachtet werden. Die Ausgabe erfolgt in Echtzeit auf einem Graphikschirm, gegliedert nach Prioritätsebenen. Außerdem sind die gerade aktiven Tasks markiert (Bild 8.29).

Analog dazu können alle Informationen dargestellt werden, die sich im globalen Speicher des Steuerungssystems befinden. Istposition, Geschwindigkeit oder auch Schleppabstand können erfaßt und visualisiert werden. Somit läßt sich zum Beispiel die Adaption von Reglerparametern durchführen. Um eine starke Beeinflussung des Steuersystems durch die Visualisierung zu vermeiden, wurde der globale Speicher als Zweitor-Speicher realisiert, der auf der einen Seite über den VME-Bus (=Steuerungssystem) und von der anderen Seite durch einen Personalcomputer zu erreichen ist. Die Darstellung der Daten erfordert deshalb im Steuer-

rechner keine Rechenleistung. Ist ausreichende Kapazität in der Steuerung vorhanden, kann die graphische Ausgabe natürlich auch durch einen Rechnerknoten des Steuerungsclusters erfolgen.

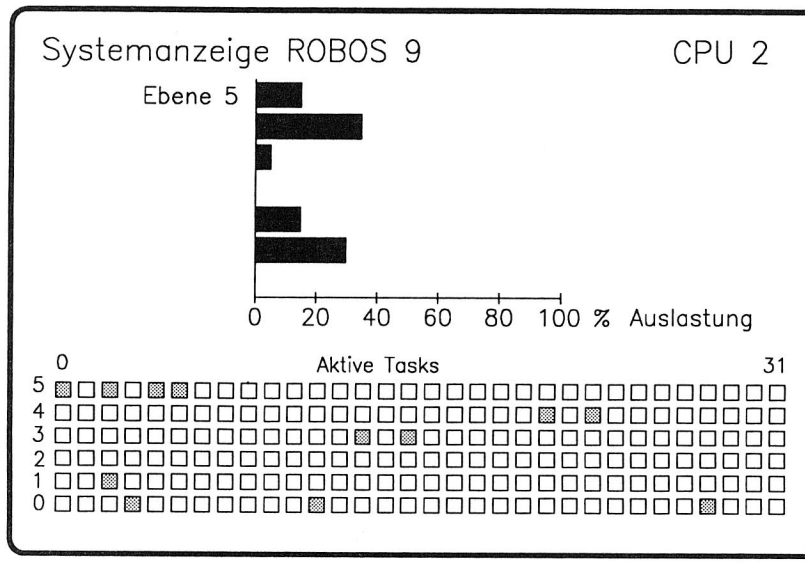



Bild 8.29: Anzeige der Auslastung einer Rechnerkarte

Die Grafikfunktion online alle Systemvariablen anzuzeigen, gestattet zum Beispiel auch die Darstellung des Schleppfehlers im laufenden Betrieb. Die sofortige Diagnose des aktuellen Zustands der Steuerung und des Roboters wird so ermöglicht.

# 9 Zusammenfassung

Die geforderte Flexibilität moderner Fertigungsanlagen läßt sich nur durch umfassenden und durchgängigen Rechnereinsatz realisieren. Entscheidend für den Erfolg sind die Leistungsfähigkeit der eingesetzten Geräte - respektive der Steuerungen - und ebenso die Möglichkeiten der Sensorintegration. Eine lückenlose informationstechnische Verknüpfung aller Elemente einer Montagezelle ist zudem auch Voraussetzung für eine fertigungsbegleitende Qualitätssicherung. Deshalb ist die Möglichkeit der Behandlung einer Zelle als ein verteiltes Steuerungssystem und die damit verbundene Informationstransparenz von großer Bedeutung.

Für eine erfolgreiche Modularisierung des Gesamtkomplexes `Steuerungsaufgaben in Montagezellen` müssen Werkzeuge aus drei verschiedenen Bereichen kombiniert werden. Auf der Seite der Formulierung von Montageaufgaben war der Entwurf einer abstrakten Funktionsbeschreibung notwendig. Das in der konkreten Umsetzung entstehende Softwaresystem sollte sich durch ein leistungsfähiges Betriebssystem, selbst noch in der Gerätesteuerung, in überschaubare Abschnitte gliedern lassen. Und nicht zuletzt ließ sich die Variabiltät im Ausbau der Komponenten eines Steuerungssystems durch den Einsatz von entsprechenden Bussystemen und Multiprozessorrechnern erreichen.

Vorangestellt waren Betrachtungen über Echtzeit-Betriebssysteme. Diese bilden die Ansätze zur Behandlung von prinzipiellen Fragen der Verwaltung von Mehrprogramm-Systemen ( *Multitasking* ). Es galt insbesondere Fragen der Vergabe von Prozessorleistung und die Möglichkeiten der Kommunikation zwischen Programmen zu analysieren. Die Präsentation von aktuellen Entwicklungen auf diesem Gebiet runden die Darstellung ab.

In einem zweiten Schritt wurde die systematische Gliederung von Montageaufgaben durchgeführt. Die vorgestellte Funktionshierarchie beschreibt die Trennung von montagerelevanter Information in unterschiedliche Abstraktionsebenen. Es resultiert eine Separation der Daten nach Produkt-, Anlagen- und Montageprozeß-Information und bildet die Basis für eine erhöhte Universalität des Steuerungssystems einer Zelle. Die Aufgabenfelder für Gerätesteuerungen in organisatorisch-dispositiver Richtung können direkt abgeleitet werden.

Bei der Erstellung einer Anforderungsliste für künftige Gerätesteuerungen wurden Aspekte der Planbarkeit integriert. Es mußte deshalb eine hohe Modularität in

Hard- und Software verlangt werden. Aus der Anwendung heraus, beziehungsweise der geforderten Integration in die Fertigungszelle, sind Fragen der Sensorintegration oder auch der Koordination und Kooperation von mehreren Geräten wesentliche Gesichtspunkte.

Die Überlegungen münden in ein Konzept für eine flexible und universelle Gerätesteuerung. Kernpunkt ist der Entwurf eines hybriden Rechnerkonzepts zur Verbindung von Standardfunktionen mit Echtzeitfähigkeit. Die Skalierbarkeit des Systems wird durch modularen Aufbau des Grundsystems und Multiprozessorunterstützung gesichert.

Zur besseren Handhabung eines feingranularen Softwaresystems für Gerätesteuerungen wurde ein objektorientierter Ansatz zur Verwaltung der einzelnen Programme konzipiert. Dieser ist speziell auf die Belange innerhalb einer Steuerung optimiert und erlaubt zudem die Modellierung von anwendungsabhängigen Systemzuständen. In Verbindung mit einem Multiprozessorsystem lassen sich auch Fehlertoleranzaspekte realisieren.

Die erweiterten Steuerungskonzepte erlauben den Entwurf von neuen Aufgabenfeldern, wie zum Beispiel die dynamische Kooperation von zwei Robotern. Eine entsprechende Struktur der Steuerungen gestattet nicht nur die Bearbeitung von gemeinsamen Montageaufgaben, sondern zudem die Herstellung der Synchronisation bei fahrenden Robotern.

Eine Umsetzung der Konzepte erfolgte in Form einer Multiprozessorsteuerung für die Doppelroboter einer Modellanlage. Die für aufwendige Anwendungen notwendige Rechenleistung läßt sich durch die Mehrprozessoranordnung in Verbindung mit der Verteilung der Intelligenz in der Zelle erreichen. Neue Formen der Koordination und komplizierte Formen der Bewegungsführung lassen sich sogar bei Einsatz einer Hochsprache realisieren. Weitere Anwendungen zeigen die Möglichkeiten der Sensorintegration. Es wurde eine schnelle Datenerfassung mittels Feldbus zur Konturerfassung realisiert, ebenso eine Werkstückidentifikation mit einer Multisensoranordnung.

Die klare Strukturierung der Steuerungselemente und das resultierende Gesamtsystem einer Montagezelle macht es möglich, die Forderung nach Leistungsfähigkeit, Flexibilität, Transparenz und Portabilität zu erfüllen. Gerade vor dem Hintergrund ständig steigender Softwarekosten von Automatisierungslösungen ist der Einsatz von Hochsprachen und eine weitgehende Anlagenunabhängigkeit der

Software bis hinab in die Gerätesteuerungen ein bedeutender Wirtschaftlichkeitsfaktor. Entscheidender Vorteil der vorgestellten Konzepte ist zudem noch die erreichte Modularität des Gesamtsystems, die eine flexible Anpassung von Geräten und Zelle an wechselnde Anforderungen gestattet. Die Leistungsfähigkeit des Multiprozessorsystems, in Verbindung mit den erstellten Softwarekomponenten, erlaubte eine praktische Anwendung von neuen Steuerungsverfahren für Doppelroboter und auch die Realisierung von aufwendigen Interpolationsverfahren (Kubische Splinefunktion) mit hohen Taktraten. Durch den Einsatz eines spezifischen Betriebssystems können sogar in direkter Prozeßnähe Modularität und Universalität garantiert werden.

Das realisierte System erbringt den Nachweis, daß die Strukturierung eines in einer Hochsprache erstellten Softwaresystems für Montageaufgaben und die Anwendung auch in zeitkritischen Applikationen (Robotersteuerung) mit einem Multiprozessorsystem keine Gegensätze mehr sein müssen.

# 10 Literatur


1. Ahmad, S.; Li, B.:
   Optimal Design of Multiple Arithmetic Processor-Based Robot Controllers.
   Proceedings of IEEE International Conference on Robotics and Automation (1987), Vol. 2, S.660-663
2. Alford, C.O.; Belyeu, S.M.:
   Coordinated Control of Two Robot Arms.
   International Conference on Robotics, Atlanta, Georgia (1984), S. 468-473
3. Almes, G.T.;Black, A.P.; Lazowska, E.D.; Noe, J.D.:
   The Eden System: A technical rewiew.
   IEEE Transactions on Software Engineering (1985), Vol. SE-11
4. Asada, H.;Slotine, J.-J.E.:
   Robot Analysis and Control.
   John Wiley and Sons, New York 1986
5. Bärnreuther, B. u.a.:
   Die Standardisierung von Datenschnittstellen in der Montageautomatisierung.
   Informatik-Spektrum 12 (1989), H. 6, S.312-320
6. Bärnreuther, B.; Dietsch, H.:
   Offene Kommunikation in der Fertigung.
   Workshop Grundlagen verteilter und paralleler Systeme, Arbeitsberichte Informatik, Universität Erlangen/Nürnberg, Band 22 (1989), Nummer 13, S.1-16
7. Ballig, G.; Fuehrer, D.:
   Einfache Programmerstellung für Roboter durch sensorgesteuerte Raumpunktgenerierung.
   Energie & Automation 9 (1987), Heft 4, S.12-14
8. Bauder, M; Pritschow, G.:
   Steuerungsstruktur und Programmierkonzept zur On-Line-Bewegungskoordinierung zweier Roboter in einer flexiblen Montagezelle.
   Robotersysteme 6 (1990), S.211-217
9. Binder, M.:
   Entwurf einer kartesischen Kopplung zweier Roboter.
   Studienarbeit, Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik, Universität Erlangen/Nürnberg, 1989

10. Blume, C.; Jacob, W.:
Programmiersprachen für Industrieroboter.
Vogel Verlag, Würzburg, 1983

11. Butner, S.; et al:
Design and Simulation of RIPS: An advanced robot control system.
Proceedings of IEEE International Conference on Robotics and Automation (1988), Vol. 1, S. 470-474

12. Carlow, G.D.:
Architecture of the Space Shuttle Primary Avionics Software System.
Communications of the ACM (1984), 27(9)

13. Cheng, S-C., Stankovic, J.A.:
Scheduling Algorithms for Hard Real-Time Systems - A Brief Survey.
IEEE, 6/1988.

14. Chien, Y.P.; Koivo,A.J.; Lee, B.H.:
On-line Generation of Collision Free Trajectories for Multiple Robots.
Proceedings of IEEE International Conference on Robotics and Automation (1988), Vol. 1, S. 209-214

15. Chochoiek, R.:
Die 4. Generation von VME-Bus-basierten Echtzeitrechnern.
Kongreß-Vortrags-Band, Echtzeit '90 (1990), S.307-315

16. Classe, D:
Beitrag zur Steigerung der Flexibilität automatisierter Montagesysteme durch Sensorintegration und erweiterte Steuerungskonzepte.
Carl Hanser Verlag, München, 1988

17. Classe, D.; Scholz, W.:
Verfahrensbeschreibende Parameterschlüssel zur systematischen Montageautomatisierung.
wt Werkstattstechnik 77 (1987), S.684-689

18. Dahlberg, S.:
Verwaltung von Funktionsmodulen einer Robotersteuerung.
Studienarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik Universität Erlangen/Nürnberg, 1990.

19. Dasarathy, B.:
Timing Containts of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them.
IEEE Transactions on Software Engineering 11(1), 1/1985.

20. Davies, C.:
New Frontiers for Real-Time Architectures.
Kongreß-Vortrags-Band, Echtzeit '90 (1990), S.289-305

21. Dietsch, H.; Bärnreuther, B. u.a.:
Eine MMS-Implementierung in einer Feldbusumgebung.
VDI-Berichte 723, VDI-Verlag, Düsseldorf, 1989

22. Dirauf, F.; Maier, R.:
Positionierung und Bahnsteuerung.
Design & Elektronik (1991), Heft 7, S.79-80

23. Duelen, G; Kirchhoff, U; Held, J; Münch, M:
Automatische Bewegungssynthese für bahnbezoge kooperierende Industrieroboter.
Springer-Verlag, Robotersysteme 3 (1987), S.107-113.

24. Dungern, O.v.; Freyberger, F.; Schmidt, G.:
Eine modulare Grundsteuerung für flexible Montagezellen.
Automatisierungstechnische Praxis 32 (1990), Heft 1, S.22-29

25. Dungern, O.v.; Schmidt, G.:
Vorbereitete und begleitende Ablaufplanung für flexible Montagezellen in industrieller Umgebung.
Robotersysteme 6 (1990), S.225 -235

26. Elnakhal, A.E.; Kulka, I; Rzehak, H.
Die Eignung von "Manufacturing Message Specification" (MMS) als Anwender-Schnittstelle von MAP-Netzen für Realzeitumgebungen.
Kongreß-Vortrags-Band, Echtzeit 90 (1990), S.65 - 74

27. Faust, D.:
Sensorgestützte Programmierung von Roboterbewegungen.
Studienarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 1990

28. Feldmann, K.; Kudorfer, A.:
Automatisierte Montagesysteme am Beispiel elektromechanischer und elektronischer Produkte.
VDI-Berichte Nr. 479, S.37-44, 1983

29. Feldmann, K.; Classe, D.:
Sensor aided robot-programming.
Proc. of the 5th Int. Conference on Robot Vision and Sensory Controls, Amsterdam (1985), S.369-382

30. Feldmann, K.; Classe. D.:
A dual armed Robot System for Assembly Tasks.
16 th International Symposium on Industrial Robots & 8 th Conference on Industrial Robot Technology, Brüssel, Springer-Verlag, Berlin u.a., S.731-741, 1986

31. Feldmann, K.:
Entwicklung und Einsatz rechnerintegrierter Produktionssysteme.
ZwF 83 (1988), H. 6, S. 290-295

32. Feldmann, K.; Flohr, R.:
Rechnerintegration in der Elektronikproduktion.
atp 32 (1990), Heft 9

33. Furth, B.; et al:
Issues in the Design of an Industry Standard Operating System for Time-Critical Applications.
Kongreß-Vortrags-Band, Echtzeit '90 (1990), S.483-492

34. Geus, L.:
Konzepte für Robotersprachen - Ein Sprachentwurf.
Dissertation an der Universität Erlangen/Nürnberg, Band 22 Nr. 6, 6/1989.

35. Göhringer, J.:
VMEexec - Verbindung von UNIX und Echtzeit.
Kongreß-Vortrags-Band, Echtzeit '90 (1990), S.341-353

36. Gramatke, H.-P.; Senft, F. R.:
Echtzeitverarbeitung in UNIX.
Automatisierungstechnische Praxis 30 (1990), Heft 2, S.89-95

37. Groha, A.:
Universelles Zellenrechnerkonzept für flexible Fertigungssysteme.
Springer Verlag, Berlin u.a., 1988

38. Grummel, M.:
Entwurf und Implementierung einer Führungsgrößenberechnung für Roboter zum Erzeugen glatter Kurven im Raum.
Studienarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Universität Erlangen/Nürnberg, 1990

39. Gruver, W. A.; et al.:
Evaluation of commercially available Robot Programming Languages.
Proc. of the 13th Int. Symp. on Industrial Robots, Chicago 1983, S.58-68

40. Hashimoto, K.; Ohashi, K.; Kumura, H.:
An Implementation of a Parallel Algorithm for Real-Time Model-Based Control on a Network of Microproccesors.
The International Journal of Robotics Research, Vol. 9 (1990), No. 6, S. 37-47

41. Herrscher, A.:
Flexible Fertigungssysteme - Entwurf und Realisierung prozeßnaher Steuerungsfunktionen.
ISW 39, Springer Verlag, Berlin u.a., 1981

42. Hirzinger, G.:
Adaptiv sensorgeführter Roboter mit besonderer Berücksichtigung der Kraft-Momenten-Rückkopplung.
Robotersysteme 1 (1985), S.161-171

43. Hörmann, A.; Hugel, T.; Meier, W.:
Ein Ansatz zur Realisierung intelligenter fehlertoleranter Robotersysteme.
Robotersysteme 4 (1988), H. 4, S. 223-231

44. Hofmann, F.:
Betriebssysteme: Grundkonzepte und Modellvorstellungen.
Teubner Verlag, Stuttgart, 1984

45. Hofmann, W.:
Koordinierungsprobleme und ihre Implementierung auf aktuellen Multiprozessorarchitekturen.
Workshop: Grundlagen verteilter und paralleler Systeme, Arbeitsberichte Informatik, Universität Erlangen/Nürnberg, Band 22, Nummer 13, S. 29-49

46. Hopfengärtner, H.:
Modellbildung und Regelung elektrischer Servoantriebe am Beispiel eines Industrieroboters.
Dissertation, Universität Erlangen, 1980

47. Hucks, A.:
Einsatz eines Bildverarbeitungssystems zur Konturverfolgung mit einem Roboter.
Studienarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 1989

48. Hucks, A.:
Multisensorsystem zur Teileerkennung in der flexiblen Fertigung.
Diplomarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 1990

49. Ishide, T.:
Force control in coordination of two arms.
Proc. of the Int. Joint Conf. Artificial Intelligence, Aug. 1977, S.717-722

50. Ish-Shalom, J.; Kazanzides, P.:
SPARTA: Multiple signal processors for high- performance robotic control.
Proceedings of 1988 IEEE International Conference on Robotics and Automation (1988), Vol. 1, S. 284-290

51. Jordan, D; Godbout, L.F.:
On the effect of sampling interval variance on the response of a digital control system.
10th World Congress on Automatic Control, IFAC Munich 1987, Vol. 4, S. 119-124

52. Kaminski, M.A.; Dwivedi, P.:
General Motors - Manufacturing Automation Protocol - An Overview.
General Motors Technical Center, Warren, 1987

53. Kaneff, S.:
EUROS: Enhanced Universal Real-Time Operating System.
Kongreß-Vortrags-Band, Echtzeit '90 (1990), S.97-106

54. Kasahara, H.; Narita, S.:
Parallel Processing of Robot-Arm Control Computation on a Multi-microprocessor System.
IEEE Journal of Robotics and Automation 1(2), 6/1985

55. Keppler, M.:
Führungsgrößenerzeugung für numerisch bahngesteuerte Industrie-roboter.
Springer Verlag, Berlin, 1984

56. Klink, M:
Kommunikation in einer Montagezelle mittels Feldbussystem.
Diplomarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 10/1990.

57. Kohno, M.; Miyakawa, A.
Real Time Synchronization of two Robots for coordinated Assembly.
16 th International Symposium on Industrial Robots & 8 th Conference on Industrial Robot Technology, Brüssel, Springer-Verlag, Berlin/Heidelberg/New York/Tokyo, 1986, p. 219-228

58. Kopetz,.H.; Ochsenreiter, W.:
Clock Synchronization in Distributed Real-Time Systems.
IEEE Transaction on Computers 36(8), 8/1987

59. Lammermann, D.:
Einsatz einer Datenbank zur Verwaltung von Merkmalen und Sensoren für ein Werkstückerkennungssystem.
Studienarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 1990

60. Lauer, P.:
Neues Offline-Programmiersystem für Robotersteuerung.
ZwF 81 (1986), Nr. 5, S.253-255

61. Le Blanc, T.J.; Friedberg, S.A.:
HPC: A Model of Structure and Change in Distributed Systems.
IEEE Transactions on Computers (1985), Vol. C-34

62. Lin, S.-K.:
Aufbau von Modellen zur Lageregelung von Industrierobotern.
Hanser Verlag, München, Wien,1989

63. Liu, C.L.;Layland, J.W.:
Scheduling Algorithms for Multiprogramming in a Hard-Real- Time Environment.
JACM 20(1), 1973

64. Ma,P-Y. R.; Lee, E.Y.S.; Tsuchiya, M.:
A Task Allocation Model for Distibuted Computing Systems.
IEEE Transactions on Computers 31(1), 1982

65. Milberg, J.; Groha, A.:
Der Zellengedanke als Strukturierungsprinzip im Informations- und Materialfluß flexibler Fertigungssysteme.
ZwF CIM 81 (1986), H. 12, S.682-686

66. Milberg, J.; Wrba, P.:
Roboter-Einsatzplanung und Offline-Programmierung mit USIS.
ZwF 81 (1986) Nr.9, S.484 - 488

67. N.N.:
Manufactoring Message Specification - Protocol Specification Draft International Standard ISO/DIS 9506.
Interantional Organization for Standardization, 1985

68. N.N.:
Bitserielles Prozeßbusschnittstellensystem.
DIN 19241

69. N.N.:
Mehrprozessor-Steuersystem für Arbeitsmaschinen (MPST) - Regeln für den Informationsaustausch.
Entwurf DIN 66264, Teil 2, Beuth Verlag, Berlin, 1984

70. N.N.:
d-tree Development Toolbox - Programmer's Reference Guide.
Second Edition, Version 3.1 Release F, Faircom Corporation, Columbia (Montana), 1989

71. N.N.:
r-tree Report Generator - Programmer's Reference Guide.
Second Edition, Version 1.1 Release E, Faircom Corporation, Columbia (Montana), 1988

72. N.N.:
c-tree File Handler - Programmer's Reference Guide.
Fifth Edition, Version 4.3 Release C, Faircom Corporation, Columbia (Montana), 1989

73. N.N.:
OS9 Advanced Systems Software Language Manual.
Microware Systems Corporation

74. N.N.:
OS9 Advanced Systems Software System Manual.
Microware Systems Corporation

75. Paul, R.P.:
Robot Manipulators: Mathematics, Programming and Control.
MIT Press, 1981

76. Pischinger, T.:
Systematische Auftragsabwicklung in rechnergeführten Montagezellen und Ableitung von Softwarestrukturen für eine konkrete Anwendung.
Diplomarbeit am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 1990

77. Pompe, E.:
Real-Time UNIX für schnelle Echtzeitanwendungen.
Kongreß-Vortrags-Band, Echtzeit '90 (1990), S.493-507

78. Poppelstone, R.J; Ambler, A.P; Bellos, J.:
RAPT: A language for describing assemblies.
The Industrial Robot, Sept. 1978, S.131-134

79. Preece, C.:
Fault Tolerant Microprocessor Systems for Industrial Control.
in NATO ASI Series, Vol.F64, 'Sensory Robotics for the Handling of Limp Materials, Springer Verlag, Berlin/Heidelberg, 1990, S. 327 - 333

80. Pritschow, G.; Spur, G.; Weck, M. (Hrsg.):
Sensordatenverarbeitung in der Fertigungstechnik.
Hanser Verlag, München/Wien, 1987

81. Rathgeber, K.:
Echtzeitregelung von Industrierobotern.
Elektronik 23 (1989), S. 112 - 120

82. Rojek, P.:
Bahnführung eines Industrieroboters mit Multiprozessorsystem.
Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden, 1989

83. Schäfer, G.:
Integrierte Informationsverarbeitung bei der Montageplanung.
Carl Hanser Verlag, München,1991

84. Scheller, J.; Sommer, E.:
Hierarchisches Steuerungskonzept für flexible Montagezellen.
ATP (1989), H. 4, S.166-173

85. Scheller, J.:
Modellierung und Einsatz von Softwaresystemen für rechnergeführte Montagezellen.
Carl Hanser Verlag, München,1991

86. Schill, A.:
Verteilte objektorientierte Systeme: Grundlagen und Erweiterungen.
Informatik Forschung und Entwicklung (1991), Heft 6, S.14-27

87. Schönhoff, B.:
Strategien zur Prozessorzuteilung.
Design & Elektonik, 12/1989.

88. Schulz, M.:
Sensorunterstützte Programmierung von Raumkurven für das Entgraten mit Industrierobotern.
ZwF 81 (1986) Nr. 7, S.378-379

89. Schwan, K.; Gopinath, P.; Bo, W.:
CHAOS-Kernel Support for Objects in the Real-Time Domain.
IEEE Transactions on Computers (1987),36(8), S.904-916

90. Schwan, K.:
CHAOS: Why One Cannot Have Only An Operating System for Real-Time Applications.
ACM Operating Systems Rewiew (1989), Vol. 23

91. Schwarz, W. u.a.:
Industrierobotersteuerungen.
Hüthig Verlag, Heidelberg, 1985

92. Sommer, E.:
Anforderungen an die Parallelverarbeitung bei Steuerung von Montagegeräten.
Workshop: Grundlagen verteilter und paralleler Systeme, Arbeitsberichte Informatik, Universität Erlangen/Nürnberg, Band 23, Nummer 3, S. 131-141

93. Späth, H.:
Spline-Algorithmen zur Konstruktion glatter Kurven und Flächen.
R. Oldenburg Verlag, 1973

94. Sperber, M.:
Steuerungssystem für Handhabungs- und Fügevorgänge in einer flexiblen Montagezelle.
Diplomarbeit, Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik an der Universität Erlangen/Nürnberg, 1987

95. Spur, G.; Stöferle, T. (Hrsg.):
Handbuch der Fertigungstechnik, Band 5: Fügen, Handhaben, Montieren.
Hanser Verlag, München/Wien, 1986

96. Spur, G.:
Entwicklungslinien der Fertigungsautomatisierung.
ATP 29(1987), Sonderheft Fertigungsautomatisierung, S.4-10

97. Stankovic, J.A.; Ramamritham, K.; Cheng, S.:
Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems.
IEEE Transactions on Computers (1985), 34(12)

98. Stankovic, J.A.; Ramamritham, K.:
Hard Real-Time Systems.
Computer Society Press of the IEEE, 1988.

99. Stankovic, J.A.; Ramamritham, K.:
The Spring Kernel: A New Paradigm for Real-Time Operating Systems.
ACM Operating Systems Rewiew (1989), Vol. 23

100. Stoer, J.:
Einführung in die numerische Mathematik I.
Springer Verlag, Berlin/Heidelberg/New York, 1983

101. Swaczina, K.:
Sensordatenverarbeitung für bahngesteuerte Handhabungsautomaten.
Hanser Verlag, München/Wien, 1983

102. Tarn, T.I.; Bejczy, A.K.; Yun, X.:
Coordinated Control of Two Robot Arms.
Proc. of the Int. Conf. on Robots and Automation (1986), San Francisco, S.1192 - 1203

103. Walter, C.J.; Kieckhafer, R.M.; Finn, A.M.:
MAFT: A Multicomputer Architecture For Fault-Tolerance in Real- Time Control Systems.
IEEE Proceedings of the Real-Time Systems Symposium (1985).

104. Wensley, J.H.; Lamport, L.; Goldberg, J.:
SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control.
Proceedings of the IEEE (1978), 66(11)

105. Zörntlein, G.:
Flexible Fertigungssysteme.
Carl Hanser Verlag, München/Wien, 1988

# LEBENSLAUF

| | |
|---|---|
| Persönliches: | Egon Sommer, geboren am 10. November 1957 in Sulzbach am Main, verheiratet mit Claudia Sommer, geb. Denkes<br>Eltern: Waldemar Sommer, Katharina Sommer, geb. Helfrich |
| Schulbildung: | |
| 1964 - 1968 | Volksschule Sulzbach am Main |
| 1968 - 1977 | Friedrich-Dessauer-Gymnasium Aschaffenburg |
| Juni 1977 | Abitur |
| Wehrdienst: | |
| 1977 - 1978 | Grundwehrdienst im Luftwaffen-Fernmelderegiment 72 |
| Studium: | |
| 1978 - 1984 | Friedrich-Alexander-Universität Erlangen/Nürnberg<br>Studienfach: Elektrotechnik<br>Schwerpunkte: Regelungstechnik, Nachrichtentechnik, Digitale Signalverarbeitung |
| Juni 1984 | Diplomhauptprüfung |
| Berufstätigkeiten: | |
| 1978 - 1983 | Praktikantentätigkeit bei verschiedenen Industrieunternehmen und studentische Hilfskraft am Lehrstuhl für Regelungstechnik |
| 1984 - 1985 | Mitarbeiter der Firma Reis Maschinenbau und Elektronik GmbH, Abteilung: Entwicklung Industrieroboter-Steuerungen |
| 1985 - 1991 | Wissenschaftlicher Mitarbeiter am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Universität Erlangen/Nürnberg |

# Reihe Fertigungstechnik Erlangen

Band 1
Andreas Hemberger
**Innovationspotentiale in der rechnerintegrierten Produktion durch wissensbasierte Systeme**
208 Seiten, 107 Bilder. 1988. Kartoniert.

Band 2
Detlef Classe
**Beitrag zur Steigerung der Flexibilität automatisierter Montagesysteme durch Sensorintegration und erweiterte Steuerungskonzepte**
194 Seiten, 70 Bilder. 1988. Kartoniert.

Band 3
Friedrich-Wilhelm Nolting
**Projektierung von Montagesystemen**
201 Seiten, 107 Bilder, 1 Tabelle. 1989.
Kartoniert.

Band 4
Karsten Schlüter
**Nutzungsgradsteigerung von Montagesystemen durch den Einsatz der Simulationstechnik**
177 Seiten, 97 Bilder. 1989. Kartoniert.

Band 5
Shir-Kuan Lin
**Aufbau von Modellen zur Lageregelung von Industrierobotern**
168 Seiten, 46 Bilder. 1989. Kartoniert.

Band 6
Rudolf Nuss
**Untersuchungen zur Bearbeitungsqualität im Fertigungssystem Laserstrahlschneiden**
206 Seiten, 115 Bilder, 6 Tabellen. 1989. Kartoniert.

Band 7
Wolfgang Scholz
**Modell zur datenbankgestützten Planung automatisierter Montageanlagen**
194 Seiten, 89 Bilder. 1989. Kartoniert.

Band 8
Hans-Jürgen Wißmeier
**Beitrag zur Beurteilung des Bruchverhaltens von Hartmetall-Fließpreßmatrizen**
179 Seiten, 99 Bilder, 9 Tabellen. 1989. Kartoniert.

Band 9
Rainer Eisele
**Konzeption und Wirtschaftlichkeit von Planungssystemen in der Produktion**
183 Seiten, 86 Bilder. 1990. Kartoniert.

Band 10
Rolf Pfeiffer
**Technologisch orientierte Montageplanung am Beispiel der Schraubtechnik**
216 Seiten, 102 Bilder, 16 Tabellen. 1990. Kartoniert.

Band 11
Herbert Fischer
**Verteilte Planungssysteme zur Flexibilitätssteigerung der rechnerintegrierten Teilefertigung**
201 Seiten, 82 Bilder. 1990. Kartoniert.

Band 12
Gerhard Kleineidam
**CAD/CAP : Rechnergestützte Montagefeinplanung**
203 Seiten, 107 Bilder. 1990. Kartoniert.

Band 13
Frank Vollertsen
**Pulvermetallurgische Verarbeitung eines übereutektoiden verschleißfesten Stahls**
XIII + 217 Seiten, 67 Bilder, 34 Tabellen. 1990. Kartoniert.

Band 14
Stephan Biermann
**Untersuchungen zur Anlagen- und Prozeßdiagnostik für das Schneiden mit $CO_2$ - Hochleistungslasern**
VIII + 170 Seiten, 93 Bilder, 4 Tabellen. 1991. Kartoniert.

Band 15
Uwe Geißler
**Material- und Datenfluß in einer flexiblen Blechbearbeitungszelle**
124 Seiten, 41 Bilder, 7 Tabellen. 1991. Kartoniert.

Band 16
Frank Oswald Hake
**Entwicklung eines rechnergestützten Diagnosesystems für automatisierte Montagezellen**
XIV + 166 Seiten, 77 Bilder. 1991. Kartoniert.

Band 17
Herbert Reichel
**Optimierung der Werkzeugbereitstellung durch rechnergestützte Arbeitsfolgenbestimmung**
198 Seiten, 73 Bilder, 2 Tabellen. 1991. Kartoniert.

Band 18
Josef Scheller
**Modellierung und Einsatz von Softwaresystemen für rechnergeführte Montagezellen**
198 Seiten, 65 Bilder. 1991. Kartoniert.

Band 19
Arnold vom Ende
**Untersuchungen zum Biegeumformen mit elastischer Matrize**
166 Seiten, 55 Bilder, 13 Tabellen. 1991. Kartoniert.

Band 20
Joachim Schmid
**Beitrag zum automatisierten Bearbeiten von Keramikguß mit Industrierobotern**
XIV + 176 Seiten, 111 Bilder, 6 Tabellen. 1991. Kartoniert.

Band 21
Egon Sommer
**Multiprozessorsteuerung für kooperierende Industrieroboter in Montagezellen**
188 Seiten, 102 Bilder. 1991. Kartoniert.

Band 22
Georg Geyer
**Entwicklung problemspezifischer Verfahrensketten in der Montage**
192 Seiten, 112 Bilder. 1991. Kartoniert.

Band 23
Rainer Flohr
**Beitrag zur optimalen Verbindungstechnik in der Oberflächenmontage (SMT)**
186 Seiten, 79 Bilder. 1991. Kartoniert.

Band 24
Alfons Rief
**Untersuchungen zur Verfahrensfolge Laserstrahlschneiden und -schweißen in der Rohkarosseriefertigung**
VI + 145 Seiten, 58 Bilder, 5 Tabellen. 1991. Kartoniert.