PEARL

ASSOCIATION
VEREIN E.V.

Vorträge zum Workshop über Realzeitsysteme

PEARL 88

am 1. und 2. Dezember 1988
in Boppard am Rhein

Herausgegeben von
D. Sauter und K. Stieger

Veranstaltet vom PEARL-Verein e.V.

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

PEARL-Verein e.V. - Geschäftsstelle München - 8014 Neubiberg

PEARL 88

Workshop über Realzeitsysteme

Herausgegeben von
D. Sauter
K. Stieger

PEARL 88

Vorträge zum Workshop über Realzeitsysteme
am 1. und 2. Dezember 1988
in Boppard am Rhein

Veranstaltet vom PEARL-Verein e.V.

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

PEARL-Verein e.V. - Geschäftsstelle München

Tagungsleitung:

Dipl.-Ing. D. Sauter	Institut für Rundfunktechnik GmbH Floriansmühlstr. 60 D-8000 München 45

Programmkomitee:

R. Bodem	BWB, Koblenz
D. Eberitzsch	Krupp Atlas Elektronik, Bremen
Dr. R. Henn	GPP, Oberhaching
A. Küchle	Dornier-System, Friedrichshafen
Prof. Dr. H. Rzehak	UniBw München
D. Sauter	IRT, München

Tagungsorganisation:

Dipl.-Inform. Klaus Stieger	PEARL-Verein e.V. Geschäftsstelle München Werner-Heisenberg-Weg 39 D-8014 Neubiberg Tel. (089) 6004-2254/2542

Inhaltsverzeichnis

Vorwort

Der Workshop PEARL '88 folgt einer guten Tradition und bietet auch dieses Mal den Freunden der Programmiersprache PEARL und den Anwendern von Realzeitsystemen ein Forum zur Information und Diskussion.

Die Vorträge, die das Programmkomitee ausgesucht hat, haben die Themen:

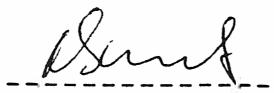
1. Software-Engineering in Realzeitprojekten unter der Berücksichtigung von standardisierten Vorgängen bei der Softwareerstellung und unter Anwendung von modernen Ansätzen in der objektorientierten Programmierung
2. Algorithmen und Konzepte
   Mit einer Reihe von Erfahrungen bei der Programmierung von PEARL
3. Aktuelle Anwendungen mit Berichten über PEARL-programmierte Systeme
4. CIM und Prozeßrechentechnik

Es wird natürlich auch, wie immer, aus der Arbeit des PEARL-Vereins berichtet.

Das Programmkomitee dankt den Vortragenden für ihre Bereitschaft, die PEARL-Tagung mitzugestalten und der Geschäftsstelle, Herrn Stieger, für die Organisation und die viele Zeit, die er in die Vorbereitungen gesteckt hat.

Den Mitgliedern des Programmkomitees möchte ich für die konstruktive Arbeit bei der Zusammenstellung des Programms danken und den Besuchern wünsche ich eine rege Diskussion in Boppard.

München, den 12. Oktober 1988
gez. Sauter



Standardisiertes Vorgehen im Rahmen einer Softwareentwicklungsumgebung für Waffen- und Waffeneinsatzsysteme

Wolfgang Dröschel
Bundesamt für Wehrtechnik und Beschaffung
Postfach 7360; 5400 Koblenz

Zusammenfassung

Aus den bisherigen Erfahrungen bei der Entwicklung von Software für Waffen- und Waffeneinsatzsysteme ergab sich ein zweifacher Ansatz für die Festlegung einer standardisierten Softwareentwicklungsumgebung. Im Vorgehensmodell wird der gesamte Entwicklungsprozeß mit seinen Aktivitäten und notwendigen Ergebnissen beschrieben. Parallel dazu werden im Rahmen einer experimentellen Umgebung Werkzeuge untersucht, die Bestandteil einer Softwareentwicklungsumgebung sein können.

Der Schwerpunkt der Arbeiten liegt derzeit auf dem Vorgehensmodell (V-Modell) [1.]. Ziel des Vorgehensmodells ist es, die Aktivitäten und Ergebnisse des Entwicklungsprozesses so zu beschreiben, daß die speziellen Anforderungen an Waffen- und Waffeneinsatzsystemen berücksichtigt werden können.

Das V-Modell beschreibt die Aktivitäten und Produkte des Softwareentwicklungsprozesses auf unterschiedlichen Detaillierungsstufen. Es werden Produktzustände und logische Abhängigkeiten zwischen Aktivitäten und Produkten beschrieben. Neben den reinen Entwicklungstätigkeiten enthält das V-Modell auch die Aktivitäten für die Qualitätssicherung (QS), das Konfigurationsmanagement (KM) und das Projektmanagement (PM). Dazu ist das V-Modell ist in Submodelle gegliedert.

Das V-Modell regelt ausschließlich den funktionalen Aspekt der SW-Entwicklung, d.h. es werden keine Festlegungen bzgl. der Aufbau- und Ablauforganisation eines Entwicklungsprojekts getroffen. Die Anpassung des V-Modells an verschiedenste Randbedingungen (organisatorische Anpassung; Tailoring) erfolgt durch den Auftraggeber bzw. Auftragnehmer zu Beginn eines Projektes.

Um eine bessere Akzeptanz des Vorgehenssmodells im Amtsbereich und in der Industrie zu erreichen, wurden Fachgruppen (FG SEU-WS) gebildet, die das Projekt begleiten. Von industrieller Seite sind der BDI mit seinen Fachverbänden sowie der BDU in der FG vertreten. Das V-Modell liegt Ende 1988 in einer Version 1.0 vor.

1. Ausgangssituation

In zunehmendem Maße werden wichtige Funktionen in Waffen- und Waffeneinsatzsysteme per Software realisiert. Damit soll die Reaktionsfähigkeit und Einsatzfähigkeit der Systeme verbessert sowie eine bessere Anpassung an die sich ändernde Bedrohungslage erreicht werden. Der Softwareanteil nimmt zu und damit steigen die Aufwendungen für die Software, ohne daß bisher, wie bei der Hardware, eine relative Qualitätssteigerung und Kostensenkung erzielt werden konnte.

Der Verteidigungsbereich wird bei der Softwareentwicklung und bei der Softwarenutzung künftig mit erheblichen Kostenproblemen bei steigenden Qualitätsanforderungen konfrontiert werden. Eine Standardisierung des Entwicklungsprozesses kann hier eine Basis schaffen für eine Verbesserung der Qualität des Produktes im Sinne einer konstruktiven Qualitätssicherung. Allerdings ist diese Standardisierung nicht als Produktstandardisierung möglich. Dafür sind die Umgebungsbedingungen und Strukturen für die Entwicklung eines Softwareanteils zu verschieden. Waffensysteme werden als Auftragsfertigung von unterschiedlichen Firmen ggf. mit Unterauftragsvergabe hergestellt. Daraus ergeben sich schon große Differenzen hinsichtlich der beim Entwickler vorhandenen SW-technologischen Kenntnisse und der benutzten Entwicklungs-Umgebung. Eine Standardisierung des Softwareentwicklungsprozesses ist nur auf logischer Ebene sinnvoll.

Bei Waffensystemen kommt hinzu, daß es viele betroffene Personen und Bereiche gibt, die Einfluß auf die angestrebte Standardisierung nehmen wollen. Diese Interessen müssen aufgefangen bzw. entsprechende Einflußfaktoren müssen berücksichtigt werden. So sind zunächst einmal die Nutzer der Waffensysteme, der Bedarfsdecker als Verantwortlicher für die Entwicklung und die Industrie an einer Standardisierung interessiert bzw. von ihr betroffen. Da aber in zunehmendem Maße Waffensysteme nur gemeinsam mit anderen NATO-Partnern entwickelt werden, müssen NATO-Standards bzw. Standards anderer NATO-Staaten ebenfalls Beachtung finden.

Der BMVg hat das BWB angewiesen, Enwicklungsarbeiten durchführen zu lassen, die zu einer technischen Standardisierung einer Softwareentwicklungsumgebung (SEU) führen sollen. Der Schwerpunkt der Arbeiten lag bisher in der Beschreibung der Abläufe des Entwicklungsprozesses im Rahmen des Vorgehensmodells. Dieses Vorgehensmodell ist nur ein Aspekt, der im Rahmen einer SEU betrachtet werden muß. Parallel zu diesen Arbeiten wird deshalb die Struktur einer SEU selbst festgeschrieben und einzelne Bausteine und ihre Schnittstellen innerhalb der SEU beschrieben.

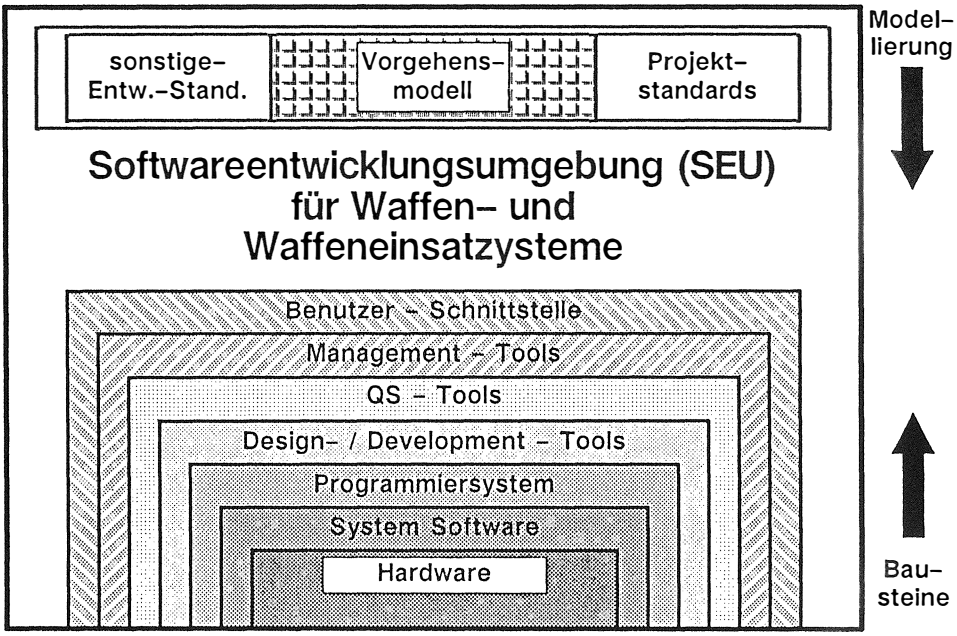


Bild 1: Komponenten einer SEU

2. Zielsetzung des Vorgehensmodells als SW-Entwicklungsstandard

Es gibt bereits eine Reihe von Standards zur Regelung des Softwareentwicklungsprozesses die im nationalen und im internationalen Bereich angewandt werden. Diese Standards decken im allgemeinen nur Teilbereiche ab bzw. betrachten den Softwareentwicklungsprozeß isoliert, so daß die Entwicklung des SW-Anteils als Teil der Waffensystementwicklung nur unzureichend abgebildet wird. So beschreibt die jetzt im nationalen Bereich geltende "Richtlinie für die Softwaredokumentation von DV-Anteilen in Wehrmaterial" [2.] nur die Ergebnisse bzw. Dokumente, die im Entwicklungsprozeß anfallen. Diese Ergebnisse setzen gewisse Aktivitäten in einer Reihenfolge voraus, die jedoch nicht explizit beschrieben sind. Darüberhinaus konnte sich diese Richtlinie im internationalen Bereich nicht durchsetzen, weil sie zu viele nationale Besonderheiten enthält und auf die internationale Begriffsnorm nicht leicht zu übertragen war.

Im internationalen Bereich hat sich der amerikanische Standard DoD STD 2167A [3.] durchgesetzt, der jedoch ebenfalls die speziellen Aspekte von Waffen- und Waffeneinsatzsytemen nur unzureichend abdeckt (vergleiche auch [4.]):

* Der Softwareentwicklungsprozeß wird als isolierter Prozeß betrachtet, losgelöst von den parallel verlaufenden Spezifikations- und Entwicklungstätigkeiten auf Systemebene bzw. für die Geräte- und Hardwareanteile. Die

Wechselwirkungen zwischen den Aktivitäten für die verschiedenen Anteile werden nicht beschrieben.

* Die Aktivitäten des Projektmanagements, der Konfigurationsverwaltung und der Qualitätssicherung für den Softwareanteil werden nicht beschrieben.
* Der DoD STD 2167A referenziert eine ganze Reihe von mitgeltenden Standards, ohne die sich der DoD STD 2167A nicht sinnvoll anwenden läßt. Daraus ergeben sich Inkompatibilitäten zwischen einzelnen Standards.
* Der DoD STD 2167A beschreibt im wesentlichen das AG/AN-Verhältnis. Er eignet sich nur bedingt als Anleitung für die Erstellung selbst. Für den Entwickler ist der Standard auf einem zu groben Abstraktionslevel.

BWB FE VI 2 hat sich deshalb entschlossen, in Zusammenarbeit mit der deutschen Industrie einen neuen Standard zu entwerfen. Dabei wurden folgende Ziele verfolgt:

2.1 Standardisierung des Softwareentwicklungsprozesses

Da unterschiedliche Gruppen auf der Amtsseite (Bedarfsdecker, Nutzer, etc.) und der Industrieseite (unterschiedliche Firmen als GU bzw. Unterauftragnehmer etc.) bei der Entwicklung bzw. später in der Nutzungsphase bei Softwarepflege- und Änderungsmaßnahmen beteiligt sind, hat die Bundeswehr ein Interesse, die Produkte, die dabei entstehen, gewissen Standards zu unterwerfen. Dadurch soll eine Mindestqualifikation der Ergebnisse sichergestellt werden, die Dokumente sollen besser lesbar und damit der Weg des Produktes von den Anforderungen über den Entwurf bis hin zum fertigen Code besser nachvollziehbar sein. Eine präzise Festlegung der notwendigen Zwischenergebnisse und Endergebnisse erleichtert die Kommunikation zwischen den Beteiligten und macht das Projekt für das Management besser steuerbar, da die Zuordnung zu Meilensteinen definiert ist.

Das Vorgehensmodell beschreibt nun die Tätigkeiten und die Ergebnisse, die im Erstellungsprozeß anfallen, sowie die logischen Abhängigkeiten zwischen ihnen als Netz von Aktivitäten und Produkten. Dabei wird nicht nur der Erstellungsprozeß im engeren Sinne beschrieben, sondern Qualitätssicherung, Konfigurationsverwaltung und Projektmanagement werden mit betrachtet. Bild 2 soll das verdeutlichen, wobei die Aktivitäten als Rechtecke, die Ergebnisse aber als Kreise dargestellt sind.

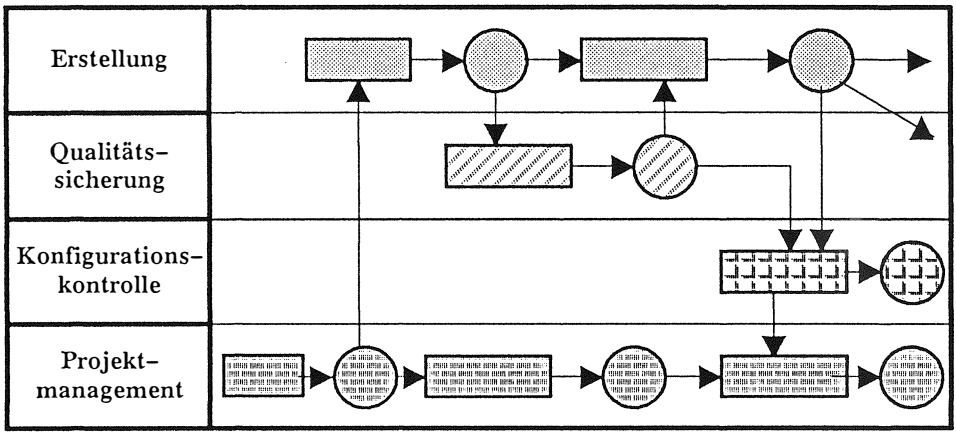


Bild 2: Tätigkeiten und Ergebnisse des Vorgehensmodells als Netz

Als **Aktivität** wird eine Tätigkeit im Rahmen des Softwareentwicklungsprozesses bezeichnet, die hinsichtlich ihres Ergebnisses und ihrer Durchführung genau beschrieben werden kann. Aktivitäten können aus einer Reihe festgelegter "Teilaktivitäten" bestehen, wenn jede dieser Teilaktivitäten ihrerseits definierte "Zwischenergebnisse" aufweist. Die Aktivitäten der obersten Detaillierungsebene werden "Hauptaktivitäten" genannt.

Als **Produkt** wird der Bearbeitungsgegenstand bzw. das Ergebnis einer Aktivität bezeichnet. Analog der Zerlegung von Aktivitäten in Teilaktivitäten kann sich die Zerlegung von Produkten in "Teilprodukte" ergeben.

Eine Aktivität kann
- die Erstellung eines Produkts,
- eine Zustandsänderung eines Produkts oder
- die inhaltliche Änderung eines Produkts

zum Gegenstand haben.

Zu jeder **Aktivität** existiert eine Aktivitätenbeschreibung, dem **Aktivitätenschema,** als Arbeitsanleitung, der bei der Ausführung der Aktivität zu folgen ist. Falls eine Aktivität in Teilaktivitäten zerlegt wird und logische Abhängigkeiten zwischen Teilaktivitäten und Produkten herausgestellt werden müssen, enthält das Aktivitätenschema eine **Aktivitätenzerlegung.**

Zu jedem **Produkt** existiert eine Produktbeschreibung, dem **Produktschema,** welche die Inhalte des Produkts definiert. Im Rahmen des Produktschemas wird für jedes Produkt eine kurze Zusammenfassung des Produktinhaltes und i. a. eine Auflistung der inhaltlichen Gliederungspunkte des Produktes angegeben. Formale Aspekte und Anforderungen werden separat festgelegt. Bild 3 gibt einen Überblick über die Haupktivitäten und die Produkte des Softwareerstellungsprozesses:

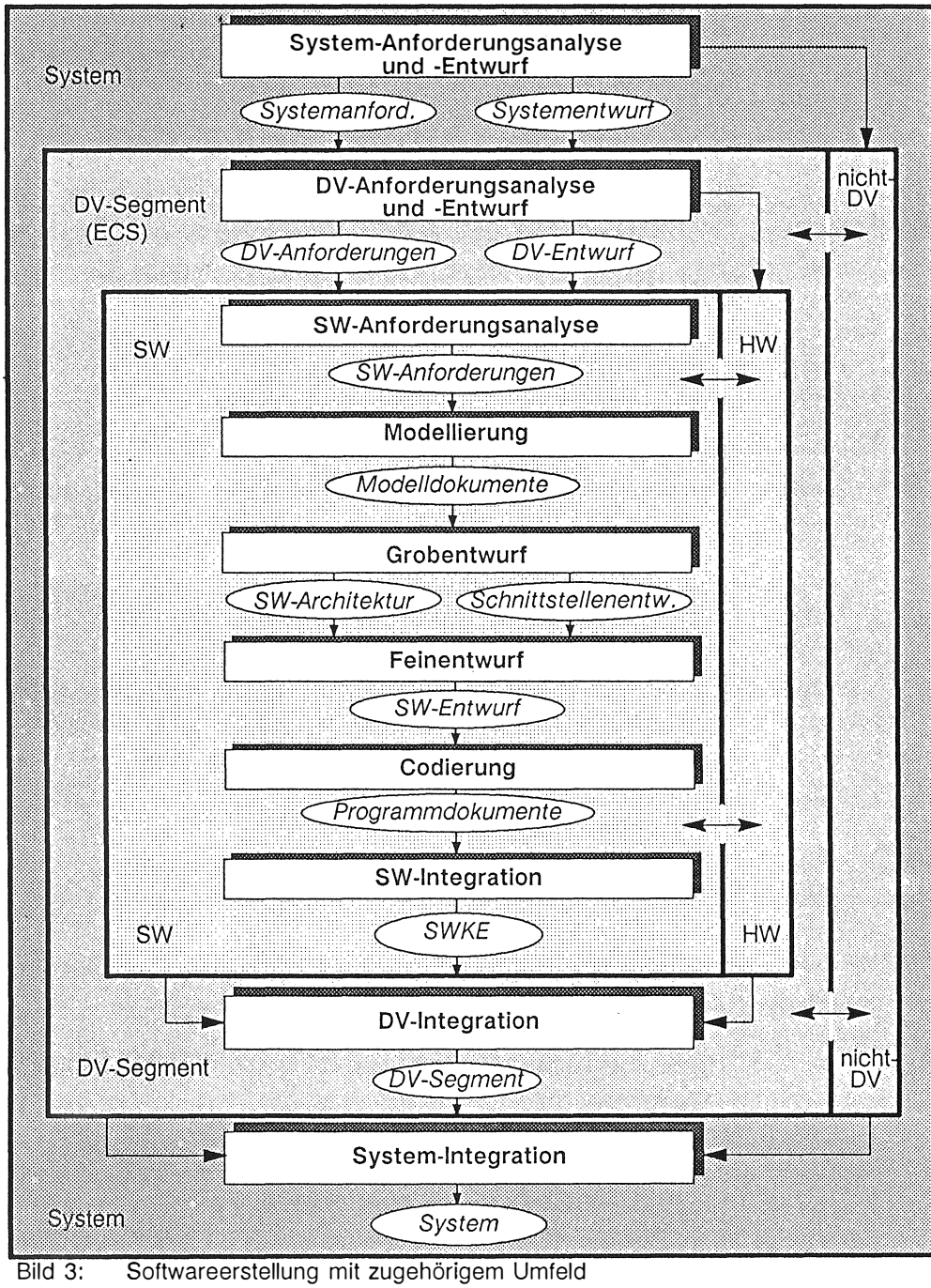


Bild 3: Softwareerstellung mit zugehörigem Umfeld

Bei der Definition der einzelnen Aktivitäten und Produkte soweit möglich auch die international akzeptierte Begriffswelt berücksichtigt. Im wesentlichen ist das Vorgehensmodell diesbezüglich kompatibel mit:

- DoD STD 2167 A
- IEEE-Standards zur Qualitätssicherung und zum Konfigurationsmanagement
- AQAP's
- RTCA / DO 178 (Luftfahrtindustrie)

Das Vorgehensmodell bildet nicht den organisatorischen Rahmen ab, in dem die Entwicklungsvorhaben für Waffen- und Waffeneinsatzsysteme durchgeführt werden. Es ist eine technische Richtlinie, die die Abläufe aus funktionaler Sicht beschreibt. Die Abbildung auf die Organisationsstrukturen muß zu Beginn des Projektes vom Management vorgenommen werden. Dadurch ist eine hohe Flexibilität gegeben, die es erlaubt, dieses Modell auch bei internationalen Vorhaben bzw. als Firmenstandard an die jeweiligen organisatorischen Besonderheiten anzupassen.

2.2 Spezielle Eignung für Waffen- und Waffeneinsatzsysteme

Die Besonderheiten der Software in Waffen- und Waffeneinsatzsystemen, im amerikanischen als Embedded Computer Systems (ECS) bezeichnet, ergeben sich einmal aus der Struktur des Systems, da die Software hier nur ein Teil eines komplexen Ganzen ist, und der Struktur des Entwicklungsvorhabens selbst, da hier für das Management auf der oberen Ebene die softwarespezifischen Aspekte nicht transparent werden. Eine weitere Klasse von Besonderheiten läßt sich aus den hohen qualitativen Anforderungen an die Software ableiten, die sich hinsichtlich Reaktionszeiten, Parallelität der Abläufe und Einpassung in das System wesentlich von anderen Informationssystemen unterscheiden.

Der Softwareentwicklungsprozeß für Embedded Computer Systems kann nicht losgelöst von der Hardware betrachtet werden. Er ist vielmehr in die Entwicklung des gesamten Waffen- und Waffeneinsatzsystems, in dem automatisierbare Funktionen durch Software realisiert werden sollen, zu integrieren.

Dabei muß die Softwareentwicklung in enger Verzahnung mit der Hardwareentwicklung vorgenommen werden, oder die Software muß unter ständiger Berücksichtigung der ausgewählten Hardware und ihrer Eigenschaften realisiert werden. Nur unter dieser Voraussetzung kann die Erfüllbarkeit der geforderten System-, DV- und SW-Funktionen über den gesamten Entwicklungsprozeß hinweg kontrolliert und gewährleistet werden.

Im folgenden Bild 4 wird beschrieben, wie der Begriff "System" im V-Modell zu verstehen ist:

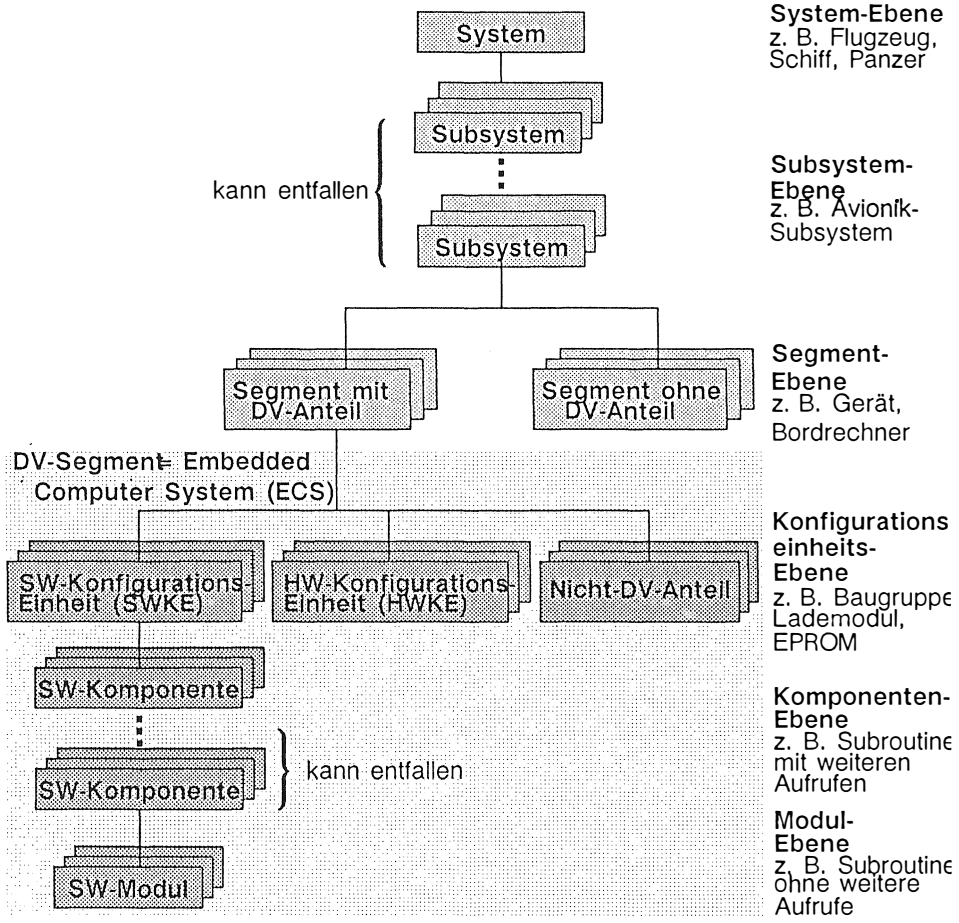


Bild 4 Systemstruktur

Ein Projekt kann ein (eigenständiges) **System** (z. B. Flugzeug) oder ein **Subsyste** (z. B. Avionik-Subsystem) eines übergeordneten Systems oder Subsystems zum Zi haben.

Ein System / Subsystem gliedert sich in **Segmente**, die unterschieden werden solche mit DV-Anteilen (DV-Segmente) und solche ohne DV-Anteile (Nicht-DV-Se mente). **Embedded Computer Systems** (ECS) wie z. B. Bordrechner entsprech in diesem Gliederungsschema den DV-Segmenten.

DV-Segmente bzw. ECS' werden weiter in **Software-Konfigurationseinheite** (SWKE), **Hardware-Konfigurationseinheiten** (HWKE) und **Nicht-DV-Anteile** gegli

dert. Die Nicht-DV-Anteile werden vom V-Modell nicht erfaßt und werden deshalb im folgenden nicht weiter betrachtet.

SWKEs (z. B. Lademodule) und HWKEs (z. B. Baugruppen) stellen aus der Sicht des System- / Subsystementwurfs elementare Objekte dar. Die Regelung der Entwicklung von SWKEs wird durch das V-Modell abgedeckt, wobei relevante Eigenschaften des System-/Subsystementwurfs als Vorgabe erwartet werden und zu berücksichtigen sind.

SWKEs bestehen aus **Softwarekomponenten** (SW-Komponenten), wobei SW-Komponenten entweder selbst wieder aus SW-Komponenten oder aus **Softwaremodulen** (SW-Modulen) zusammengesetzt sein können. Komponenten sind abgeschlossene Softwareeinheiten einer höheren Ebene. Module werden in der Regel die kleinsten getrennt übersetzbaren Programmbausteine sein .

Hinsichtlich der hohen qualitativen Anforderungen an ECS – Software sind zwei Ansätze notwendig. Zunächst muß der Modellierungsprozeß methodisch unterstützt werden, so daß es möglich ist, die verschiedenen Aspekte vor allem auch bei den kritischen Realzeitanwendungen zu modellieren. So müssen die Informationsflüsse, die Kontrollflüsse und die Ereignisflüsse modellierbar sein und in einen gemeinsamen Entwurf integriert werden können. Hierzu werden methodische Ansätze untersucht. Sie sind im Vorgehensmodell nicht enthalten, sondern sollen in einem eigenen Methodenhandbuch beschrieben werden.

Bei besonders kritischen Elementen ist es wichtig, daß man vom Ablauf her Aktivitäten initiiert, mit denen es möglich ist, die DV-Anforderungsanalyse zu verifizieren und zu detaillieren, die Auswirkungen und Konsequenzen von zu erfüllenden Lastenheftforderungen aufzuzeigen bzw.frühzeitig die Realisierbarkeit der entsprechenden Komponente zu untersuchen. Solche Aktivitäten werden als "Prototyping" im Vorgehensmodell beschrieben.

Auswirkungen, die im Rahmen des Prototypings untersucht werden, können z. B. sein das Überschreiten der verfügbaren Rechenleistung oder des vorhandenen Speicherbereichs, was dann zur Konsequenz hat, daß die Hardware in Teilbereichen anders ausgelegt werden muß. Ferner können z.B. kritische Teile des Systems im Hinblick auf Reaktionszeiten von Benutzer- und Prozeßschnittstellen, Rechnerkern- und Massenspeicherbelastung, Einschwingverhalten, Genauigkeit und Rechenzeit von Algorithmen, Kapazität von Warteschlangen, etc. untersucht werden.

Prototyping umfaßt nicht nur die Erstellung der Prototyp-Software, sondern auch die Erstellung oder Bereitstellung geeigneter Hardware- und/oder Software-Monitore und -Meßwerkzeuge. Um Prototyping mit einem vertretbaren Aufwand durchzuführen, müssen unterstützende Werkzeuge zur Verfügung stehen.

2.3 Flexibilität zu unterschiedlichen Projektbedingungen

Bei den bisherigen Entwicklungsstandards wurde bemängelt, daß diese Standards zu wenig flexibel seien, weil sie den SW–Entwicklungsprozeß zu starr als linearen Ablauf von Aktivitäten beschreiben bzw. die Beschreibung der Abläufe zu sehr von organisatorischen Rahmenbedingungen geprägt sind. Das Vorgehensmodell versucht hier, in hohem Maße flexibel zu sein, so daß eine Anpassung an unterschiedliche Rahmenbedingungen möglich ist.

2.3.1 Einbettung in den Systementwicklungsprozeß

In Bild 3 wurden die Aktivitäten der SW–Erstellung auf dem obersten Level der Aktivitätenhierarchie dargestellt. Diese Abbildung veranschaulicht die Einbettung des SW–Entwicklungsprozesses in den System–Entwicklungsprozeß und den DV–Anteil–Entwicklungsprozeß. Wechselwirkungen zwischen SW und Hardware, Nicht–DV–Anteil und DV–Anteil werden dargestellt.

Die Reihenfolge der Aktivitäten in Bild 3 erscheint sequentiell. Dies ist eine idealisierte Darstellung der Softwareerstellung. Es entspricht der Vorstellung vom Softwareentwicklungsprozeß als einem strengen Top-down-Vorgehen. Wegen der engen Verzahnung bei ECS-Entwicklungen sind jedoch Iterationen im Entwicklungsprozeß üblich.

Die System-Anforderungsanalyse kann in ihren Teilaktivitäten beeinflußt werden durch die Realisierbarkeitsuntersuchungen auf DV-Segmentebene. Die Untersuchungen bei der SW-Anforderungsanalyse können Rückwirkungen auf die Ergebnisse der DV-Anforderungsanalyse haben. Insbesondere bei innovativen und komplexen Problemlösungen werden die Anforderungen auf den drei verschiedenen Ebenen in mehreren Zyklen erstellt werden müssen, um Konsistenz, Vollständigkeit und erforderliche Genauigkeit zu erreichen.

Ähnliche Beziehungen bestehen zwischen der Funktions- und Datenmodellierung und dem Grobentwurf. Beide Aktivitäten sind zumindest teilweise überlappend durchzuführen. Rückwirkungen von der technischen Realisierung auf Funktionsabläufe und Datendesign müssen planerisch einkalkuliert werden.

Eine weitere Ursache für die Wiederaufnahme zurückliegender Aktivitäten sind Fehlerbehebung und die Berücksichtigung von Änderungsvorschlägen. Korrekturen können zu Revisionen der Ergebnisse früherer Aktivitäten bis hin zu Entscheidungen führen, die im Rahmen der Anforderungsanalysen getroffen wurden. Um die Wartbarkeit des Systems zu gewährleisten, sind derartige Revisionen unter Berücksichtigung der Regeln des Konfigurationsmanagements in den betroffenen Produkten zu dokumentieren.

2.3.2 Tailoring

Das Vorgehensmodell zeichnet sich durch Allgemeingültigkeit, durch Firmen- und Projektunabhängigkeit aus. Es ist ein generisches Modell, das vor einer konkreten Anwendung einer produkt- und projektspezifischen Anpassung bedarf.

Als "Tailoring" wird die Streichung nicht relevanter Aktivitäten und Produkte und die Festlegung des Detaillierungsgrades von Aktivitäten/Produkten bezeichnet. Die hierbei entstehende Teilmenge des V-Modells wird **Projekthandbuch** genannt.

Tailoring ist in folgenden Schritten durchzuführen:

- *Produktspezifische* Anpassung (Anpassung an die Art des Entwicklungsproduktes; Produkte sind hier z. B. Systeme, Subsysteme, Geräte)
- *Projektspezifische* Anpassung (Anpassung an ein konkretes Projekt; Projekte sind hier z. B. Entwicklungs- oder SWPÄ-Projekt).

Tailoring liegt primär in der Verantwortung des Auftraggebers, der diese Verantwortung aber auch ganz oder teilweise an das Projektmanagement des Auftragnehmers übertragen kann. So ist es z. B. durchaus denkbar, daß der AG das V-Modell hinsichtlich der Unternehmens- und Produktausrichtung einem Tailoring unterzieht, dem AN jedoch Freiheit in projektspezifischen Regelungen läßt.Dadurch wird dem Auftragnehmer ermöglicht, sein Know-how, seinen technologischen Stand, seine Präferenzen, Charakteristika seines Unternehmens, etc. einzubringen. Der Entscheidungsspielraum des ANs ist durch den Tailoringgrad des AGs vorgegeben.

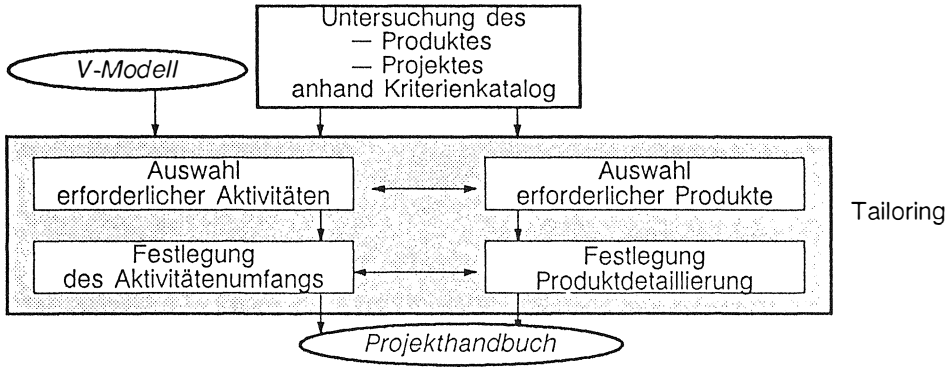


Bild 5 Tailoring-Vorgehen

Beim Tailoring des V-Modells ist zunächst das Produkt bzw. das Projekt anhand eines Kriterienkatalogs zu analysieren. Auf der Basis dieser Untersuchungsergebnisse kann dann eine Auswahl an Aktivitäten/Produkten und die Bestimmung des Detaillierungsgrades von Aktivitäten/Produkten erfolgen.

2.3.3 Organisatorische Anpassung

Unter 2.1 wurde bereits darauf hingewiesen, daß das Vorgehensmodell technische Abläufe funktional beschreibt. Dadurch ist eine hohe Flexibilität hinsichtlich unterschiedlicher organisatorische Rahmenbedingungen gegeben. Es ist Aufgabe des Projektmanagements, zu Beginn des Projektes das V – Modell auf die Aufbau– und Ablaufstrukturen der beteiligten Organisationen abzubilden. Das V – Modell legt verschiedene Rollen fest, die jedoch nur funktional definiert sind. Zu jeder Rolle wird festgestellt, welche Erfahrungen, Kenntnisse und Fähigkeiten notwendig sind. Bild 6 gibt die Rollen wieder, die zu einzelnen Teilmodellen definiert sind:

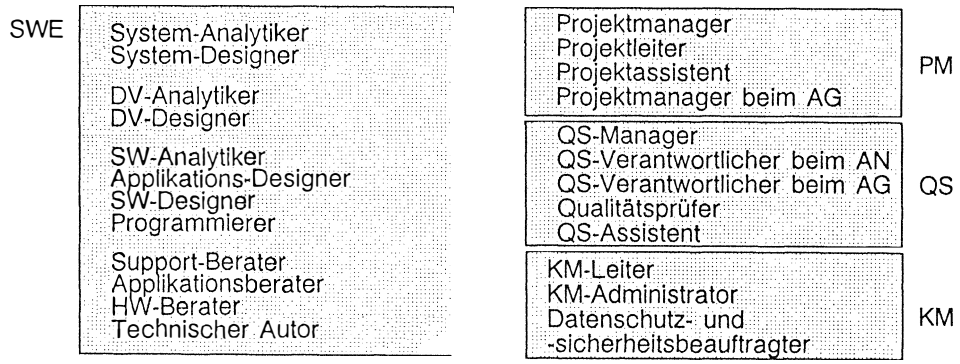


Bild 6 Rollen im V-Modell

Diese Rollen sind den wirklichen Aufgabenträgern bei der Festlegung der Projektstruktur zuzuordnen. Hierzu gehört auch die Festlegung der Schnittstellen zwischen Auftraggeber und Auftragnehmer. Die Festlegung einer solchen Schnittstelle ist sehr stark geprägt durch die Projektorganisation auf AG- und AN-Seite. Weiter wird eine solche Schnittstelle auch durch unterschiedliche Projektgegebenheiten mitbe–stimmt.

Aus diesen Gründen kann das V-Modell für die Schnittstelle Auftraggeber — Auftragnehmer keine vollständigen Regelungen vorgeben. Da jedoch die Festlegung der Rechte und Einflußmöglichkeiten des AGs gegenüber einem AN eine wichtige Basis für die Projektabwicklung ist, ist eine Zuordnung zwischen Aktivitäten des V-Modells und möglichen Rechten (z. B. Entscheidungsbeteiligung, Kontrollmöglichkeiten) des AGs beispielhaft angegeben, die als Leitlinie für die Festlegung der Schnittstelle Auftraggeber — Auftragnehmer dienen kann.

3. Projektorganisation und weiteres Vorgehen

Um eine bessere Akzeptanz des Vorgehensmodells im Amtsbereich und in der Industrie zu erreichen, wurden Fachgruppen (FG SEU-WS) gebildet, die das Projekt begleiten. Von industrieller Seite sind der BDI mit seinen Fachverbänden sowie der BDU in der FG vertreten. Die Projektstruktur ist aus Bild 7 ersichtlich.

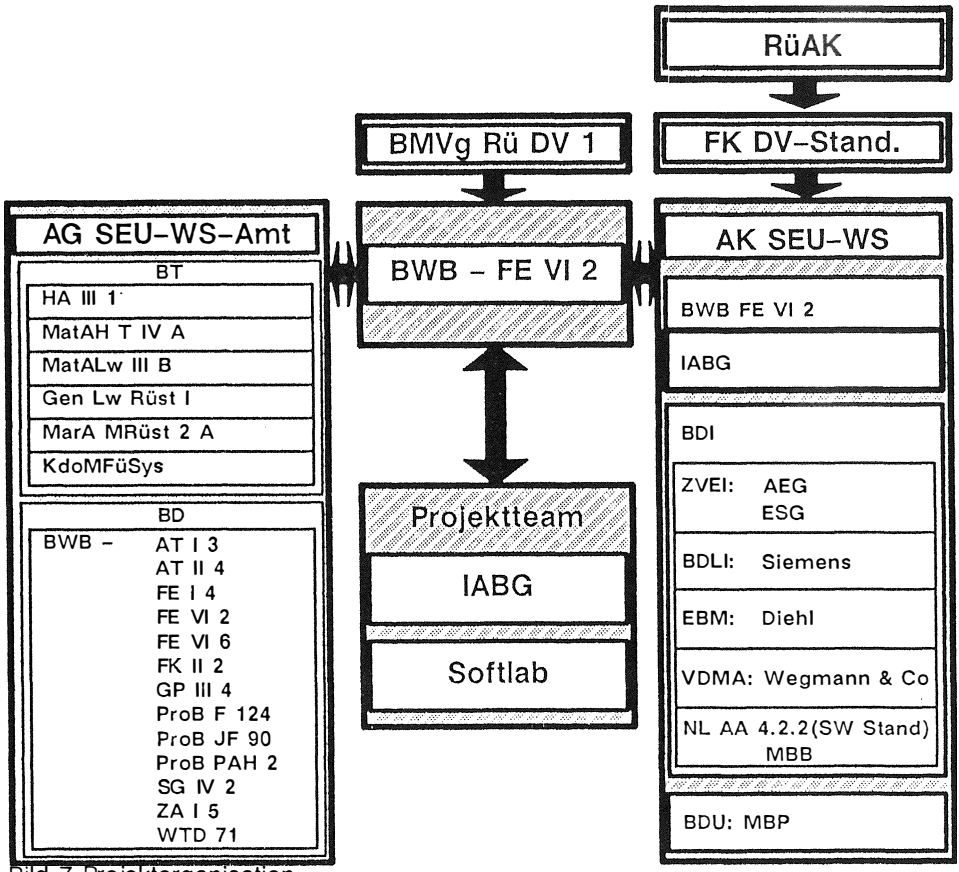
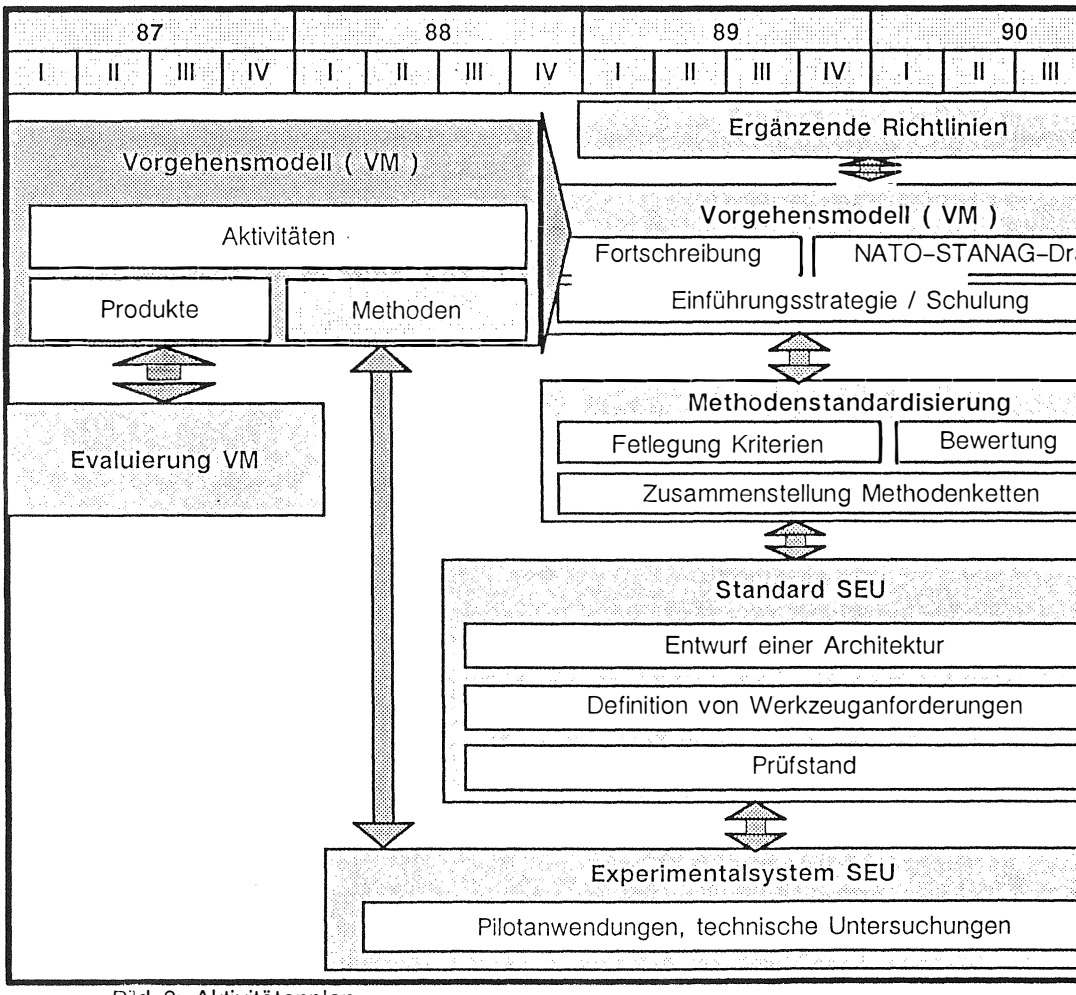


Bild 7 Projektorganisation

Das V-Modell liegt Ende 1988 in einer Prototyp-Version vor. Bis Ende 1988 erfolgt eine erste Festschreibung von Standardmethoden. Darauf aufbauend sollen im Rahmen eines Experimentalsystems einzelne Werkzeuge bzw. Werkzeugkombinationen erprobt werden. Ein grober Aktivitätenplan ist aus Bild 8 ersichtlich.



Biid 8: Aktivitätenplan

4. Literaturverzeichnis

[1.] BWB FE VI 2-62-08-00 vom 30.9.1988: "Vorgehensmodell einer Softwareentwicklungsumgebung für Embbedded Computer Systems"

[2.] BMVg Org 1-62-01 vom 18.4.1983: Richtlinie für die Software-Dokumentation von DV-Anteilen in Wehrmaterial (SWDokWM)

[3.] DoD-STD-2167A vom 29. Februar 1988: Military Standard Defense System Software Development.

[4.] September 1987: Report of the Defense Science Board Task Force on Military Software

Interaktion der SW-Erstellung, Qualitätssicherung, Konfigurationsmanagement und Projektmanagement im Rahmen eines Vorgehensmodells

Klaus P. Plögert
INDUSTRIEANLAGENBETRIEBSGESELLSCHAFT mbH
Einsteinstr. 20 in D-8012 Ottobrunn

Zusammenfassung

Das im Vortrag aufgezeigte Vorgehensmodell regelt aus funktionaler Sicht die Gesamtheit aller im Rahmen der Softwareentwicklung für Embedded Computer Systems anfallenden Aktivitäten und Produkte. Festlegungen bezüglich Aufbau- und Ablauforganisation werden nicht getroffen, sondern sind firmen- oder amtsspezifisch zu regeln. Das Vorgehensmodell ist in vier sog. Submodelle gegliedert: Softwareerstellung (SWE), Qualitätssicherung (QS), Konfigurationsmanagement (KM) und Projektmanagement (PM). Jedes dieser Submodelle regelt den entsprechenden Bearbeitungsablauf und die daraus entstehenden Produkte und gibt praktische Hilfestellung für die Durchführung der einzelnen Aktivitäten.

1 Zielsetzung des Vorgehensmodells

Ziel des Vorgehensmodells ist die Standardisierung aller Aktivitäten und Produkte, die bei der Softwareerstellung für Embedded Computer Systems (ECS) durchzuführen sind bzw. entstehen. Die Standardisierung soll nicht Selbstzweck sein. Sie ist für große und häufig firmenübergreifende Projekte unentbehrlich, da hierdurch Transparenz im Vorgehen einer großen Zahl von Mitarbeitern geschaffen wird. Dies gilt sowohl zwischen Auftraggeber und Generalunternehmer (GU) als auch zwischen GU und seinen Unterauftragnehmern. Wesentliches Ziel ist es, Reibungsverluste zu vermeiden und damit auch Geld einzusparen.

Mit diesem Vorgehensmodell wird der Softwareentwicklungsprozeß aus funktionaler Sicht mit den begleitenden Tätigkeiten für Qualitätssicherung (QS), Konfigurationsmanagement (KM) und Projektmanagement (PM) für ECS in der Bundeswehr vereinheitlicht und in Zukunft verbindlich vorgeschrieben.

Es findet Anwendung bei Vertragsgestaltung, Spezifikation von Nutzeranforderungen, Softwareerstellung, Projektsteuerung/-begleitung, Abnahme/Nachweisführung und Software-Pflege/-Änderung in der Nutzungsphase.

Anwender des Vorgehensmodells sind das BWB als Auftraggeber für Waffen- und Waffeneinsatzsysteme, Ämter der Teilstreitkräfte als Vertreter der Nutzer, andere

Einrichtungen des Amtsbereiches, die bei der Entwicklung oder Pflege/Änderung der Software beteiligt, und Industriefirmen, die mit der Entwicklung oder Pflege/Änderung von Software beauftragt sind.

Das Vorgehensmodell beinhaltet eine Menge von Regelungen, die für jedes Projekt konkret anzupassen ist. Diese Anpassung für ein Projekt ("Tailoring" genannt) wird durch Streichung von nichtanwendbaren Regelungen oder Produkten und Festlegung des Detaillierungsgrades durchgeführt. Im Anschluß daran müssen die organisatorischen Zuordnungen der so gefundenen Aktivitäten erfolgen.

2 Einführung in das Vorgehensmodell (V-Modell)

Das V-Modell **regelt** die Gesamtheit aller

- Aktivitäten und Produkte

sowie

- Produktzustände und logischen Abhängigkeiten zwischen Aktivitäten und Produkten

während des Softwareentwicklungsprozesses und der Software-Pflege/-Änderung.

Das V-Modell **beschreibt** Aktivitäten und Produkte auf unterschiedlichen Detaillierungsebenen.

Eine Aktivität kann

- die Erstellung eines Produkts,
- eine Zustandsänderung eines Produkts oder
- die inhaltliche Änderung eines Produkts

zum Gegenstand haben.

Diese Grundelemente "Aktivität" und "Produkt" werden durch spezielle Symbole repräsentiert (siehe Abb. 2.1).

Symbol für Aktivität Symbol für Produkt

Abb. 2.1 Symbole für Aktivitäten und Produkte

2.1 Beschreibungsmuster

Aktivitäten, Aktivitätenzerlegungen und Produkte werden im V-Modell durch verschiedene festgelegte Muster beschrieben.

Eine Aktivität wird nach folgendem Muster beschrieben:

Name der Aktivität
1 Voraussetzungen

2 Produktfluß
3 Abwicklung

Eine besondere Bedeutung für die Dynamik des Modells hat der Produktfluß. Hier werden alle Produkte oder Teilprodukte, die in die (Teil-) Aktivität eingehen oder von ihr modifiziert oder neu erstellt werden, aufgelistet und ihre Behandlung beschrieben. Das geschieht in Form einer Tabelle (siehe Abb. 2.3), die entsprechend ausgefüllt wird.

von		Produkt	nach		QS	KM
Aktivität	Zustand		Aktivität	Zustand		
SWE 3.4	akzeptiert	SW-Anforderungen	—	—		
—	—	Datenmodell	SWE 4.4 SWE 4.5	In Bearb.		

Abb. 2.3 Muster eines Produktflusses

Zu jedem (Teil-) Produkt (Spalte 3), das durch die zu beschreibende Aktivität betroffen ist, wird der Zustand zu Beginn der Aktivität (Spalte 2) und der Zustand nach Beendigung der Aktivität (Spalte 5) angegeben. Hat die Aktivität keinen Einfluß auf den Produktzustand oder existiert kein Produktzustand, so ist dies in der Tabelle durch einen Strich angegeben.

2.2 Submodelle

Innerhalb des Vorgehensmodells werden gewisse Aktivitäten zusammengefaßt, weil sie jeweils aus einer bestimmten Sicht ein in sich abgeschlossenes Modell repräsentieren. Ein derartiges "Modell" wird als **Submodell** bezeichnet.

Folgende vier Submodelle bilden das V-Modell:

— Softwareerstellung *(SWE)*
— Qualitätssicherung *(QS)*
— Konfigurationsmanagement *(KM)*
— Projektmanagement *(PM)*

Dabei ist zu beachten, daß sich diese Submodelle nur auf die *Softwareentwicklung* eines Systems, nicht jedoch auf das gesamte System beziehen. In bezug auf das gesamte System würden sich sehr ähnliche Submodelle ergeben, jedoch mit erheblich anderen Inhalten.

3 Übersicht über Aktivitäten und Produkte

Im folgenden werden die vier Submodelle SWE, QS, KM und PM auf der Ebene der Hauptaktivitäten im Funktionsüberblick dargestellt. Dabei sind auch die wichtigsten Produkte, die in den Submodellen entstehen, mit angegeben.

Funktionsüberblick Submodell SWE

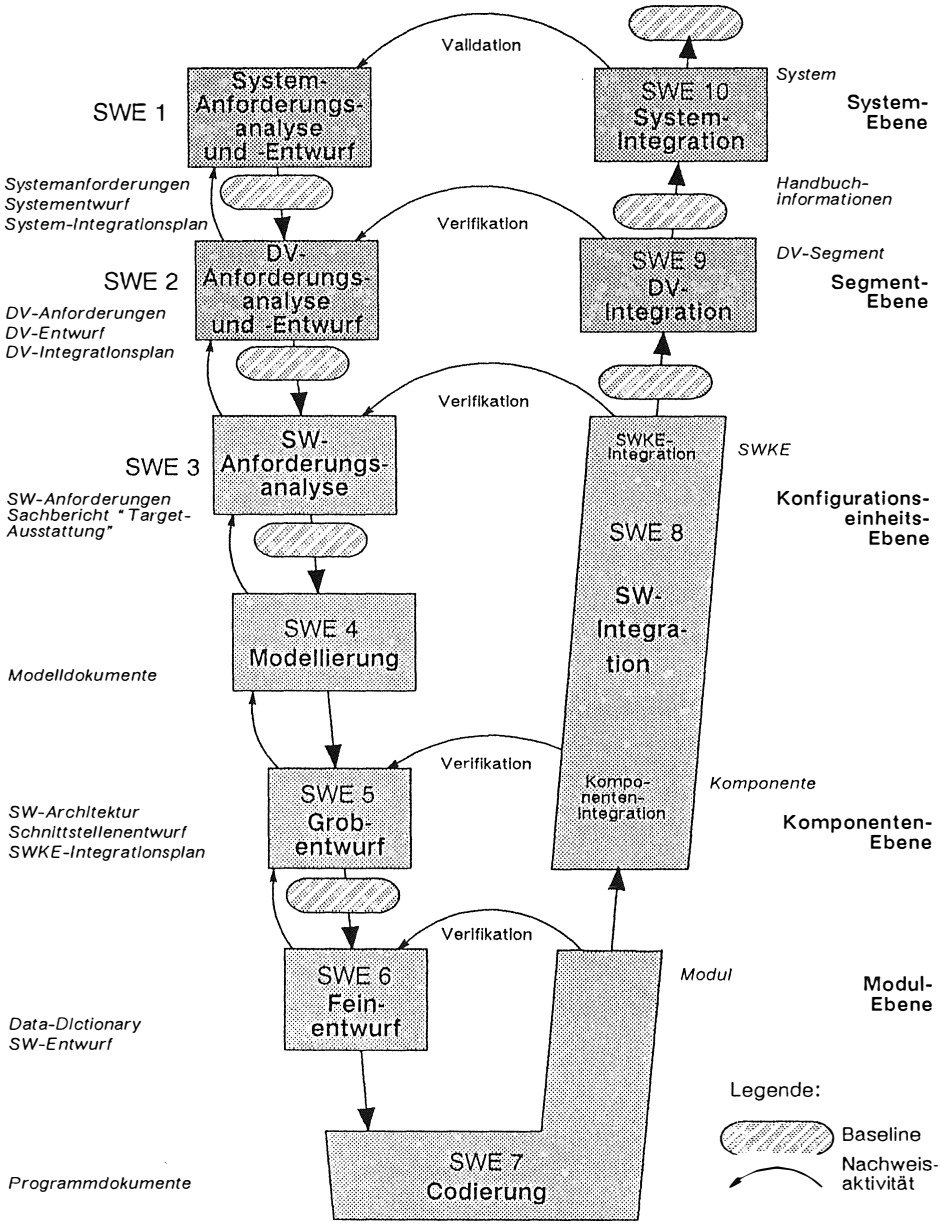


Abb. 3.1 Funktionsüberblick Submodell SWE

Funktionsüberblick Submodell QS

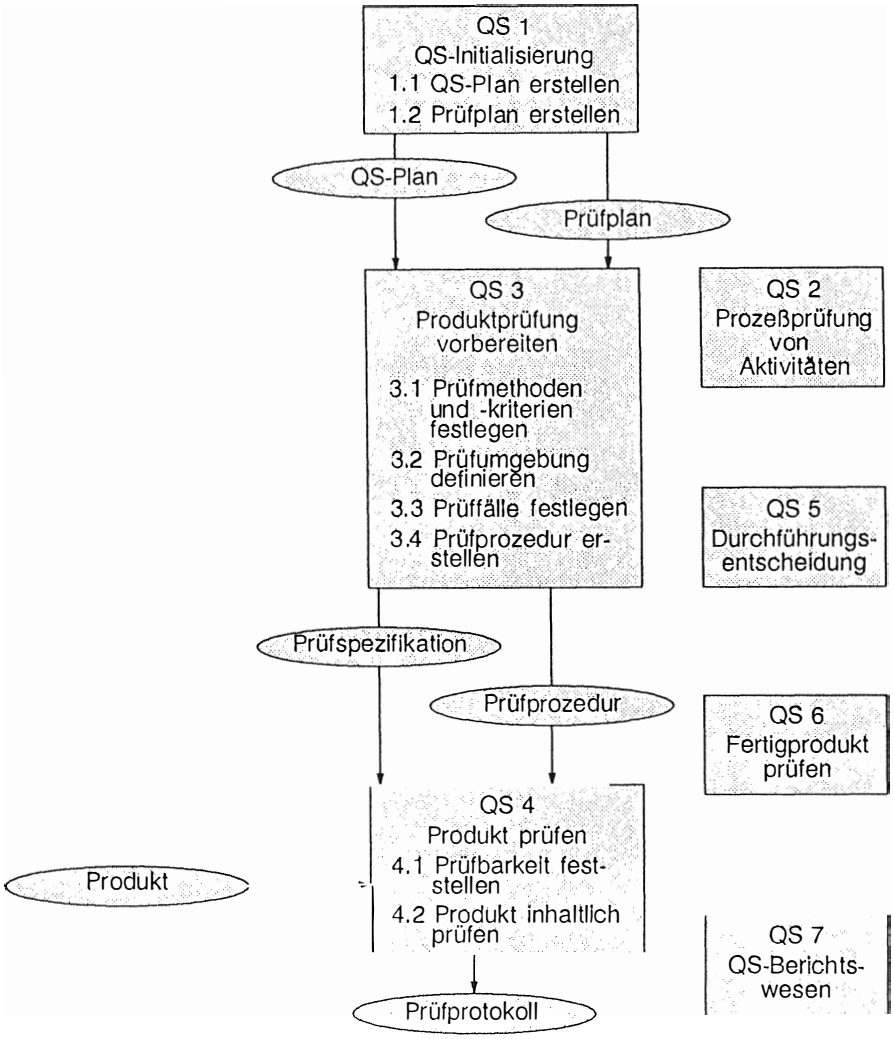


Abb. 3.2 Funktionsüberblick Submodell QS

Funktionsüberblick Submodell KM

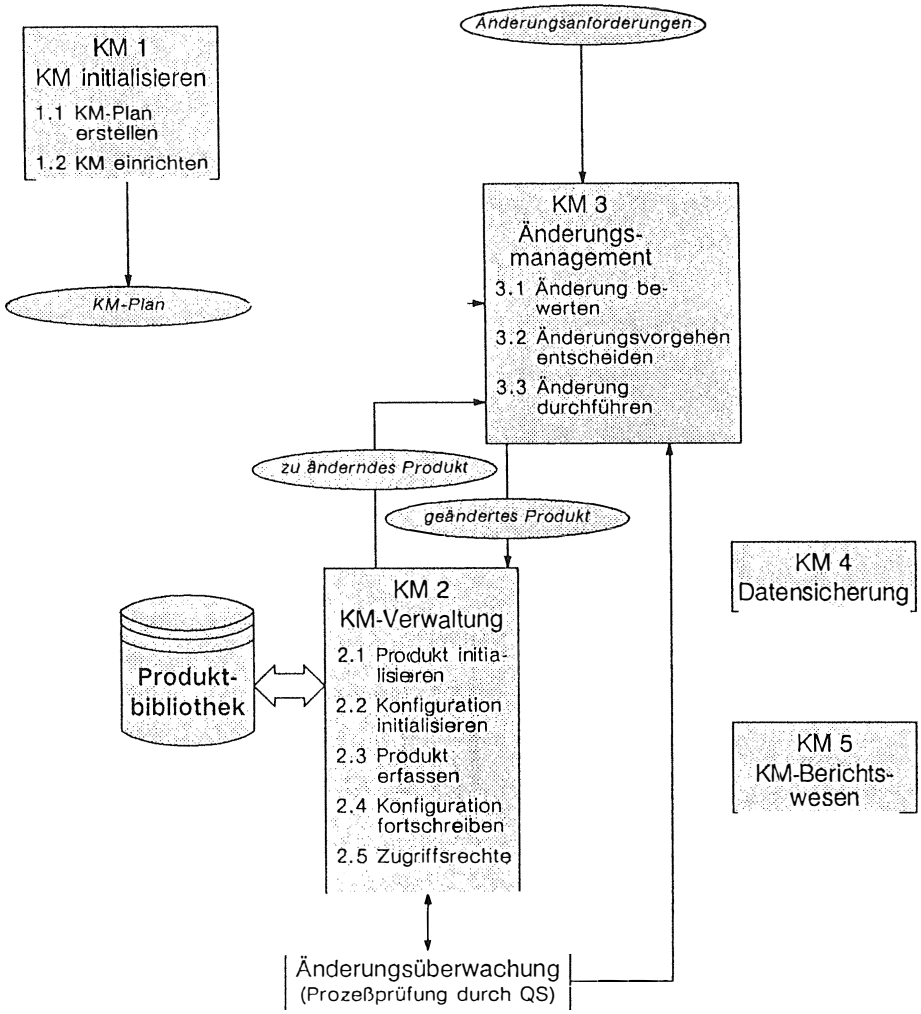
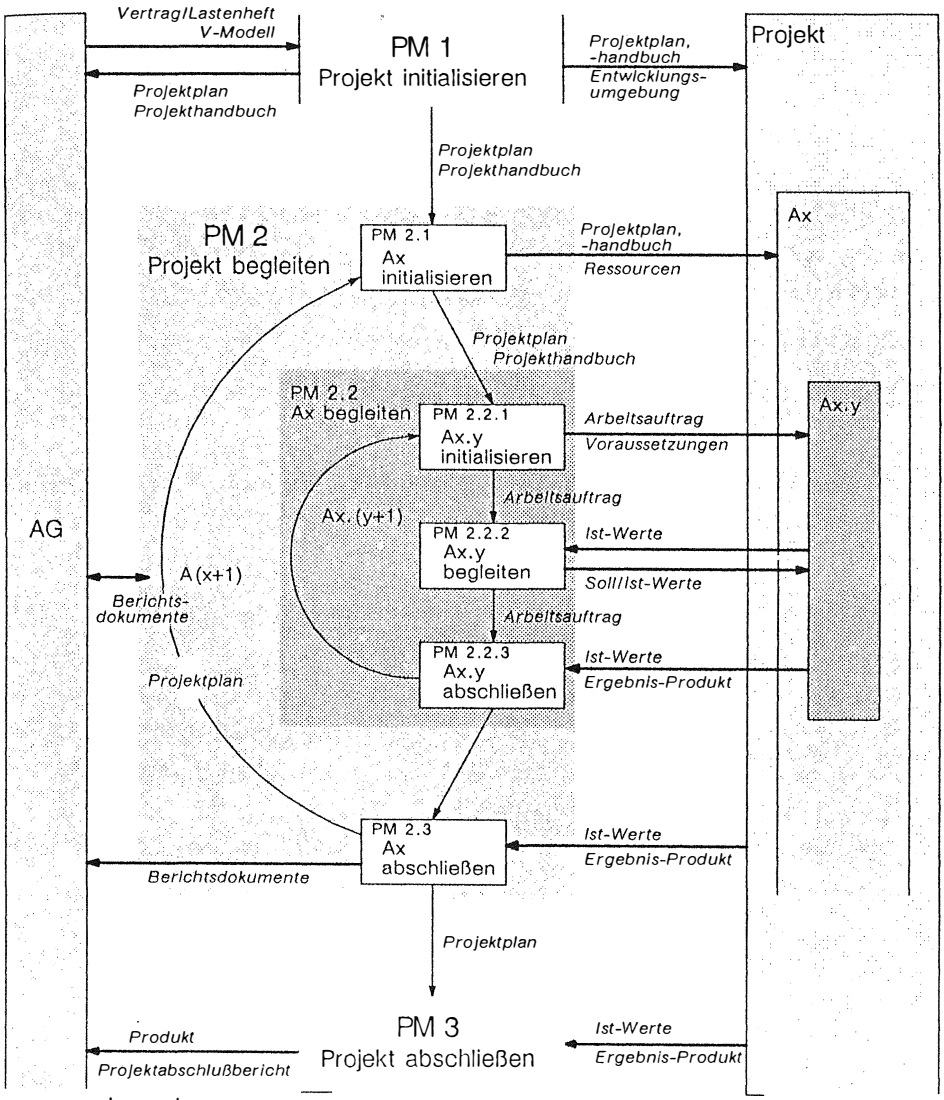


Abb. 3.3 Funktionsüberblick Submodell KM

Funktionsüberblick Submodell PM



Legende:

Ax: Aktivität x
Ax.y: Teilaktivität y der Aktivität x

Abb. 3.4 Funktionsüberblick Submodell PM

4 Interaktion der Submodelle

Die Interaktion der vier Submodelle ist in vergröberter Darstellung aus Abb. 4.1 zu entnehmen. Einige Details der Modelldynamik werden in den folgenden Abbildungen 4.2 und 4.3 gezeigt.

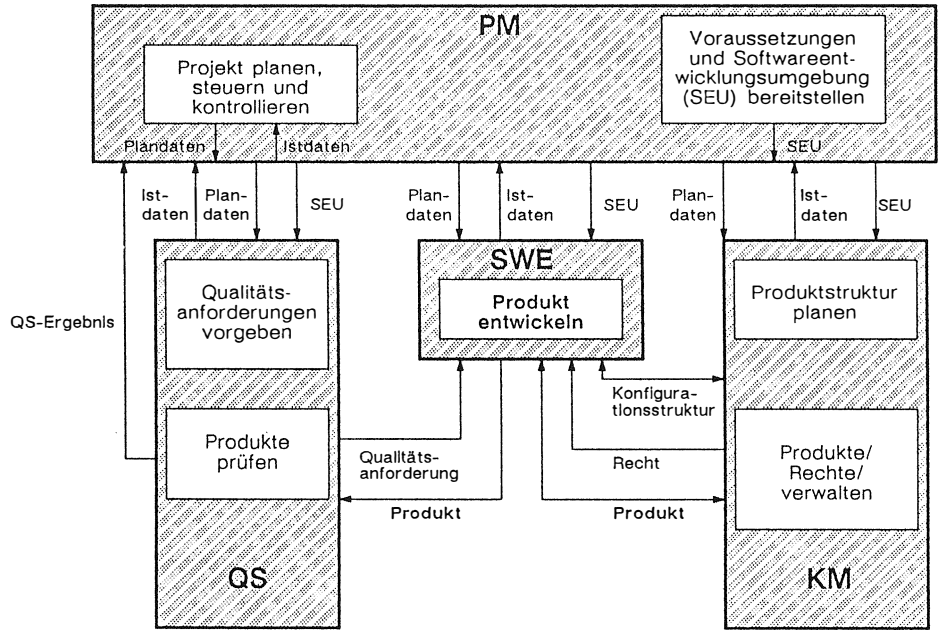


Abb. 4.1 Interaktion der Submodelle SWE, QS, KM und PM

Das komplizierte Netz an Aktivitäten und Produkten, das für den Ablauf der SWE-Aktivität "Modul implementieren" erforderlich ist, wird in Abb. 4.2 dargestellt. Hier soll deutlich werden, daß die Planung aller SWE-Aktivitäten für den Feinentwurf abgeschlossen sein muß, um die Durchführungsentscheidung herbeiführen zu können. D. h. mit PM 2.3 wird SWE 6 (Feinentwurf) abgeschlossen. PM überprüft den Projektfortschritt, erstellt die erforderlichen Berichtsdokumente, schreibt die Projektstatistik und -historie fort. Alle im Feinentwurf identifizierten Module und Komponenten werden durch KM 2.1 in der Produktbibliothek mit dem Namen erfaßt, und die Attribute (Zustand etc.) werden initialisiert.

In der Durchführungsentscheidung (QS 5) wird in Form eines Management-Reviews überprüft, ob alle für diesen Meilenstein geplanten Produkte in einer von QS akzeptierten Form vorliegen und ob die nächsten Aktivitäten (hier im Beispiel SWE 5 "Codierung") ordnungsgemäß geplant sind.

Mit dem positiven Abschluß der Durchführungsentscheidung kann die Codierung beginnen. Sie setzt sich aus mehreren Teilaktivitäten zusammen:

- Implementierung der Module und Komponenten
- Erstellung der zugehörigen Prozeduren
- Debugging
- informellen Prüfungen.

In Abb. 4.2 wird lediglich die erste Codierungsaktivität dargestellt.

Initiiert wird der Bearbeitungszyklus durch einen Arbeitsauftrag aus der Aktivität PM 2.2.1 an SWE, QS und KM. Dieser bewirkt zunächst, daß in KM 2.5 die erforderlichen Zugriffsrechte auf die Ressourcen in der Produktbibliothek hergestellt werden.

Mit der Information des Projekthandbuchs, des Projektplans, der Modulbeschreibung, dem Pseudocode und der Kritikalitätseinstufung, die auch die konstruktiven QS-Maßnahmen festlegt, kann die Modul-Implementierung durchgeführt werden.

Parallel hierzu wird in QS 3 die Produktprüfung vorbereitet. Es werden die der Kritikalität des Moduls oder der Komponente entsprechenden Prüffälle und Prozeduren erstellt.

Mittels dieser Prüfprodukte kann das von SWE implementierte Modul in QS 4 dem formellen Qualitätsnachweis unterzogen werden. In Abhängigkeit vom Ergebnis der Produktprüfung muß SWE entweder nachbessern oder das Produkt ist von QS akzeptiert, womit gleichzeitig auch ein Zustandswechsel verbunden ist. Der Aktivitätenzyklus wird durch die Produkterfassung (Modul und Prüfprotokoll) durch KM und durch PM 2.2.3 "Teilaktivität abschließen" abgeschlossen.

Sind alle für die Codierungsaktivität geplanten Produkte erstellt, von QS akzeptiert und von KM erfaßt, so kann die nächste Durchführungsentscheidung abgehalten werden, an die sich dann (vgl. Abb. 3.1) die SW-Integration von Modulen zu Komponenten und von Komponenten zur SW-Konfigurationseinheit anschließt.

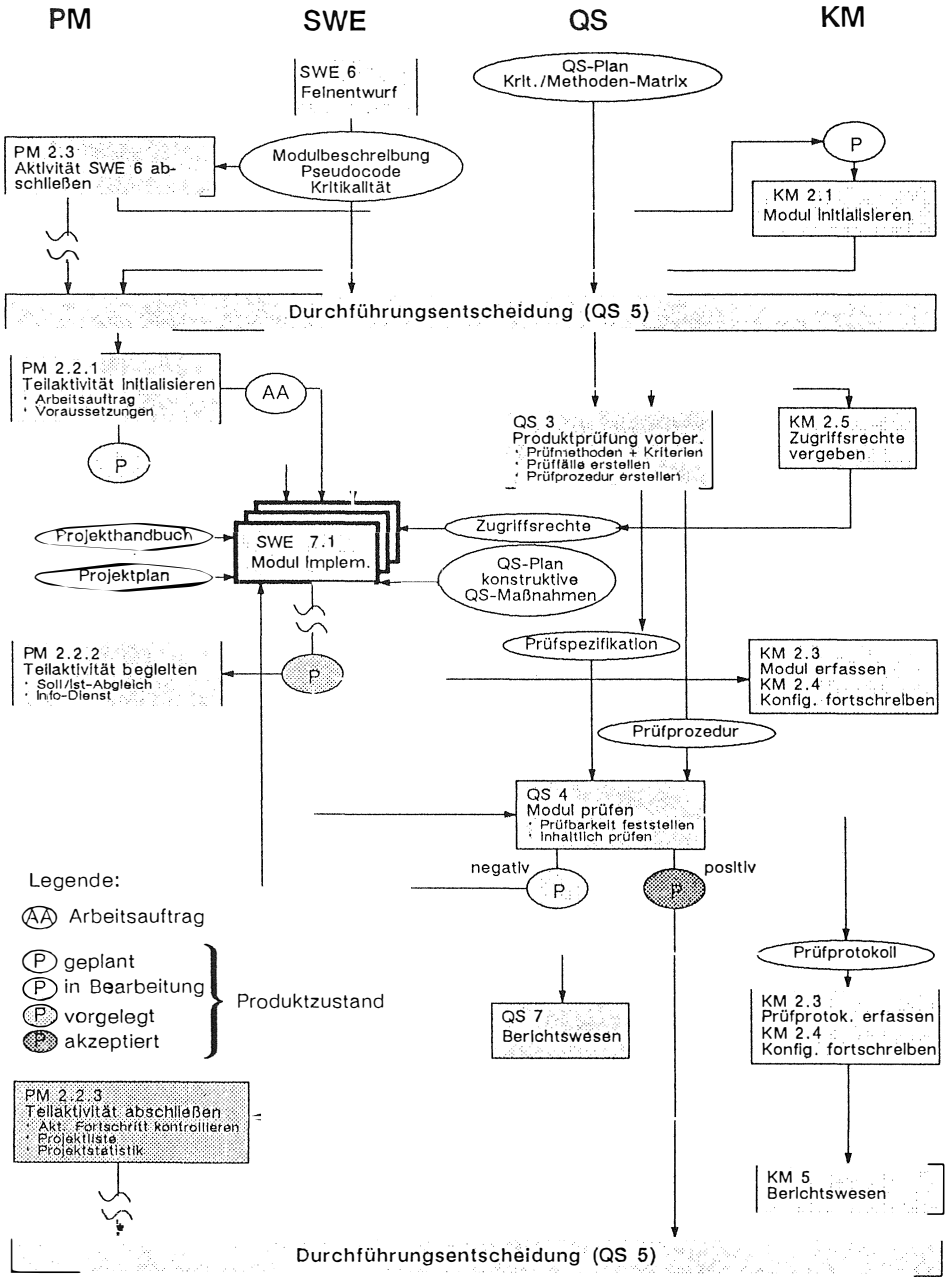


Abb. 4.2 Beispielhafte Darstellung des Aktivitäten und Produktflusses bei Codierungsaktivitäten

Die Vorgehensweise und das Zusammenspiel zwischen SWE und QS skizziert Abb. 4.3. In den Aktivitäten SWE 1 bis SWE 5 wird die Kritikalität der Software-Funktionseinheiten bestimmt, verfeinert und in Kritikalitäten/Funktionen-Matrizen dargestellt. Unter Verwendung der Kritikalitäten/Methoden-Matrix im QS-Plan (vgl. Abb.3.2) werden in Aktivität QS 3 die Prüfprodukte (Prüfspezifikation, Prüfprozedur) erstellt und die Prüfumgebung definiert. Die Prüfung der Prüfprodukte in QS 4 darf nur durch Personen erfolgen, welche nicht an der Erstellung dieser Produkte beteiligt waren. Mittels der geprüften Prüfprodukte werden zunächst in QS 4 die Produkte aus den Aktivitäten SWE 1 bis SWE 7 geprüft. Nach Integration der Produkte auf der Prüf- und Integrationsumgebung in SWE erfolgt dann wiederum in QS 4 die Prüfung der Integrations-Produkte aus den Aktivitäten SWE 8 bis SWE 10.

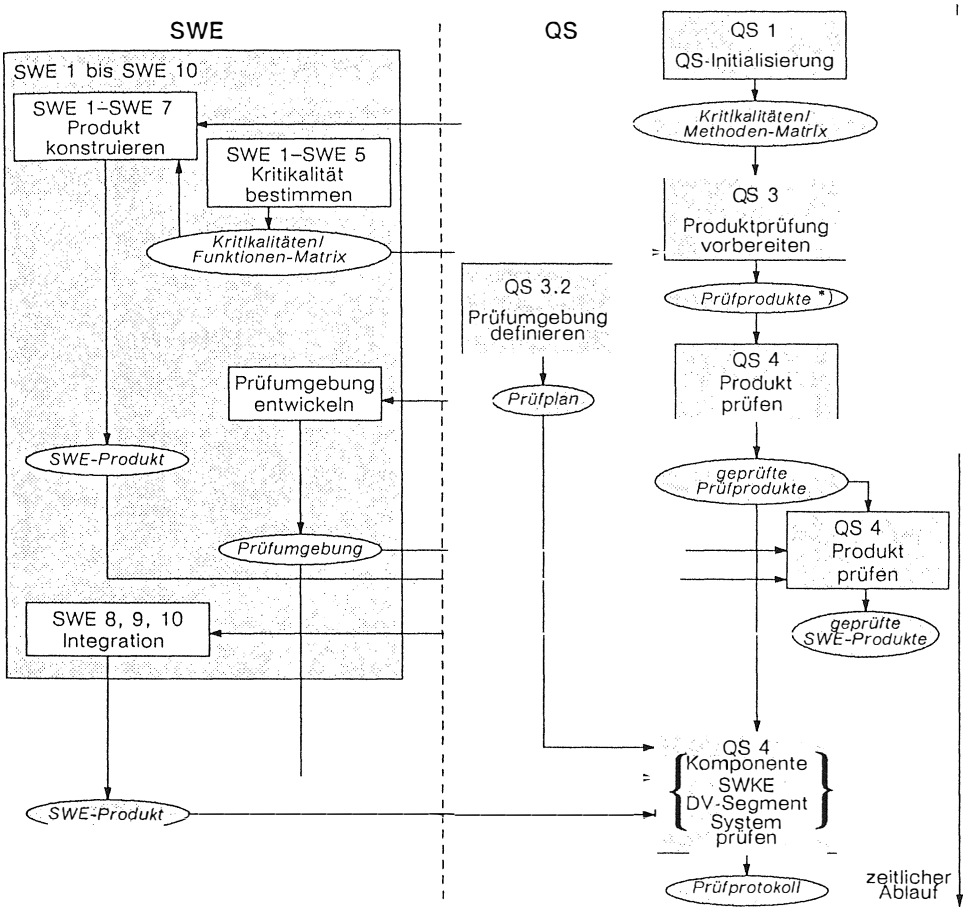


Abb. 4.3 SWE- und QS-Zusammenarbeit bzgl. Prüfaktivitäten

*) Prüfspezifikation und Prüfprozedur

5 Ausblick

Das in groben Umrissen skizzierte Vorgehensmodell ist lediglich Teil eines Projekts, dessen Ziel es ist, die Standardisierung der DV-Entwicklung für ECS im Verteidigungsbereich herbeizuführen. Insgesamt werden folgende Schwerpunkte untersucht:

1) Standardisierung der Aktivitäten und Produkte im Rahmen eines Vorgehensmodells.

2) Auswahl von zu verwendenden Methoden, mit denen die unter 1) genannten Aktivitäten durchgeführt werden (Methodenstandard).

3) Spezifikation der Anforderungen an Werkzeuge und damit Vereinheitlichung der verwendeten Werkzeuge (Werkzeugstandardisierung).

Die Arbeiten zu Punkt 1) sind unter starker Einbeziehung der Industrie nahezu abgeschlossen, während die Arbeiten zu 2) schwerpunktmäßig 1989 und zu 3) schwerpunktmäßig 1990 durchgeführt werden.

SPRAM
Ein Werkzeug zur Unterstützung des Störmeldeverfahrens im Rahmen des Konfigurationsmanagements

Dipl. Math. Roland J. Frank
Dipl. Inf. (FH) Kathrin Wiedmann
Dornier-System GmbH
Postfach 13 60
7990 Friedrichshafen

Zusammenfassung

In der Gewährleistungs- und Wartungsphase in EDV-Projekten fallen Störungen an, deren organisatorische Verwaltung und technische Behebung umfangreiche Tätigkeiten sowie Kontakte zwischen mehreren Beteiligten erfordert. Zur rechnergestützten Erfassung der Störungen über Störmeldungen und deren Bearbeitung über Änderungsanträge, Stellungnahmen etc., wurde in PEARL ein Programm (SPRAM) unter Verwendung einer Datenbank entwickelt. Es dient gleichzeitig als Werkzeug für das Konfigurationsmanagement, die Überwachung und Verfolgung der ergriffenen Maßnahmen und Änderungen bis zu deren Inbetriebnahme.

1. Einleitung

Um insbesondere größere Projekte, bei denen u.U. mehrere Firmen/Personen an verschiedenen Orten beteiligt sind, während der Gewährleistungs- und Wartungsphase zu unterstützen, wurde dieses Programm realisiert.

Bisher erfolgten Störmeldungen (wenn überhaupt) nur mittels Papierformularen. Bei deren Bearbeitung ließen sich langwieriges Aktendurchsuchen und manuell erstellte Listen (zum Zuordnen von Störmeldungen zu den zugehörigenn Änderungsanträgen u.a. Dokumente) nicht vermeiden.

Zudem verging bei diesem papiergestützten Verfahren erhebliche Zeit, bis eine Störmeldung beim betroffenen Entwickler eintraf.

Derart den aktuellen Bearbeitungszustand zu erkennen, und auch noch alle beteiligten Personen/Firmen zu informieren und auf dem laufenden zu halten, war äußerst schwierig.

SPRAM (Störreport und Aenderungsmaßnahmen bzw. Systemsoftware Performance Report and Maintenance) dient zur rechnergestützten Erfassung von Störmeldungen, Mängelberichten, Änderungs- und Modifikationswünschen sowie zur Überwachung und Verfolgung der daraufhin ergriffenen Maßnehmen und Änderungen bis zu ihrer Inbetriebnahme.

Beim Bedienungsvorgang wird gewährleistet, daß nicht mehrere Benutzer gleichzeitig ein und dasselbe Dokument ändern.

2. Störmelde- und Änderungsverfahren

Das Störmelde- und Änderungsverfahren ist ein Verfahren des Konfigurations-Managements. Es wird eingesetzt zur Erfassung und Bearbeitung von Störungen an Konfigurationseinheiten.

'Störung' wird hier als Sammelbegriff für folgende Anlässe verstanden:

- Einbringen von neuen, nicht im Pflichtenheft definierten Funktionen.

- Gewünschte Änderungen aus Gründen der Vereinheitlichung bzw. Optimierung.

- Beseitigung von erkannten Fehlern, die eine Beeinträchtigung der im Pflichtenheft definierten Funktionen zur Folge haben.

Alle 'Störungen' können nur über das Störmeldeverfahren bearbeitet werden. Das Störmeldeverfahren wird durch das Stellen einer Störmeldung eingeleitet. Innerhalb des Störmeldeverfahrens wird dann das Änderungsverfahren angestoßen.

Prinzipiell wird mittels folgender Aktivitäten das Störmelde- und Änderungsverfahren durchgeführt:

- Störmeldung
- Änderungsantrag
- Stellungnahme
- Maßnahmenmeldung und
- Änderungsmeldung

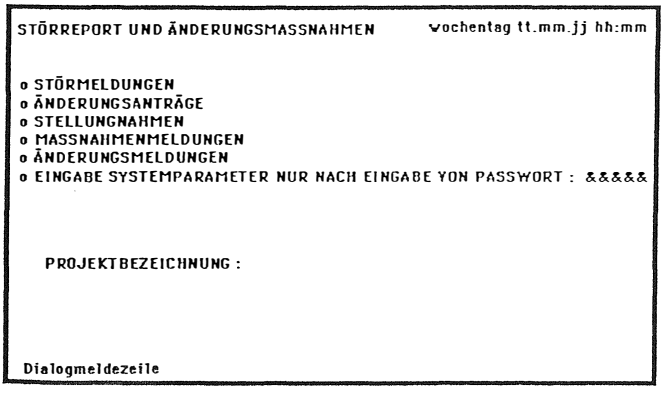


Bild 2 : Einstiegsbild von SPRAM

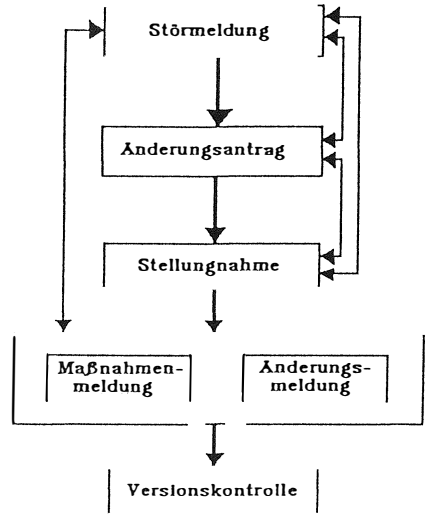


Bild 1: Ablauf einer Systemänderung von der Störmeldung bis zur Inbetriebnahme der neuen Version

Der Ablauf des Störmelde- und Änderungsverfahrens ist in Bild 2 dargestellt.

Störmeldung

Sobald eine Störung oder ein Modifikationswunsch in einem in Betrieb befindlichen System auftritt wird eine Störmeldung (SM) erstellt.

Sie wird üblicherweise von demjenigen erstellt, der diese Störung (SM) bzw. Mangel oder Modifikationswunsch festgestellt hat.

Da eine Störmeldung noch keine Hinweise auf den Grund der Störung enthalten muß, sind zur Erstellung keinerlei spezielle Kenntnisse des Systems erforderlich.

Trotzdem soll eine SM möglichst genaue Angaben enthalten über:

- o den Zeitpunkt der Störung (Datum und Uhrzeit)
- o die Kategorie der Störung (Absturz, Mangel,...)
- o die Beschreibung des Symptoms
- o Firmen/Personen, die voraussichtlich betroffen sind
- o Firmen/Personen, die informiert werden sollen (Verteiler)
- o ein Stichwort als Titel und Suchbegriff der SM
- o Datum, bis wann die Störung bearbeitet sein soll (Termin)

In der SM kann auf hierzu korrespondierend angelegte SM's verwiesen werden. Verweise auf Änderungsanträge, Stellungnahmen, Maßnahmenmeldungen und/oder Änderungsmeldungen werden automatisch erstellt.

Mit Hilfe dieser Verweise kann dann nach zusammengehörenden Dokumenten gesucht und eine gegenseitige Statuskontrolle durchgeführt werden.

Beim Erstellen einer SM wird der Status automatisch (siehe Kap. 2.2.) auf 'erstellt' gesetzt. Erst wenn ein Prüfer eingetragen ist, oder der Termin überschritten wurde, ist der Status 'geprüft' bzw. 'überfällig'. Eine explizite Statusänderung (abschließen) durch einen Berechtigten ist möglich.

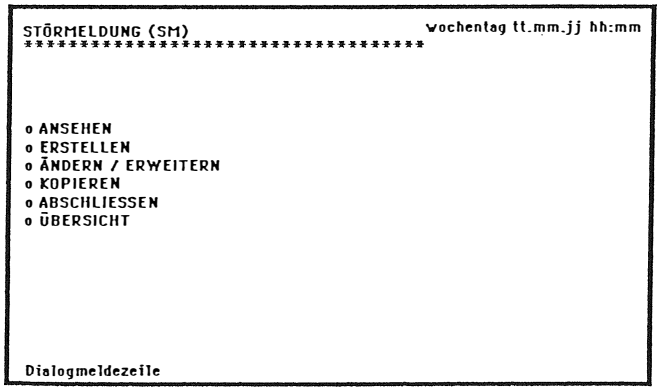


Bild 3 : Unterfunktionen der Störmeldung

Änderungsantrag

ist eine der (von der SM) betroffenen Personen/Firmen, der Ansicht, den Fehler in seinem Teilsystem/Programm gefunden zu haben, erstellt er einen Änderungsantrag (AA) als Vorschlag für eine Softwareänderung oder eine Maßnahme.

Dieses Dokument enthält Angaben über:

- o den Antragsteller
- o betroffene Firmen, Module und/oder Quellfiles
- o den Termin
- o Beschreibung und Begründung der vorgesehenen Änderung
- o voraussichtliche Auswirkungen der vorgesehenen Änderung
- o Verweise auf SM's, auf die sich dieser AA bezieht
- o Verweise auf andere AA's,
- o Stichwort als Titel dieses AA's

Beim Erstellen eines AA's ist der Status automatisch auf 'gestellt'. Erst wenn hierzu ein Vollzugsdokument (Maßnahmenmeldung oder Änderungsmeldung) erstellt ist, wird der Status auf 'in Bearbeitung' gesetzt. Explizite Statusänderungen (freigeben) ist möglich.

Stellungnahme

Eine Stellungnahme (ST) kann von jedem erstellt werden, der sich durch eine SM oder AA betroffen, fühlt. Dieses Dokument enthält die Daten, die üblicherweise nur aus einem Kommentar also einer Stellungnahme zur Störmeldung besteht.

Im SPRAM enthält eine Stellungnahme :

- o Bezüge, auf welche SM und/oder AA sich diese ST bezieht

- Bezüge auf betroffene (Teil-) Systeme und Module
- Bezüge auf betroffene Firmen/Personen
- Text
- Verteiler
- Verfasser

Der Status wird beim Erstellen auf 'abgegeben' gesetzt, kann aber explizit geändert werden (akzeptieren).

Maßnahmenmeldung

Wurde in einem Änderungsantrag eine Maßnahme vorgeschlagen später bewilligt (freigegeben) und durchgeführt, so muß das Dokument Maßnahmenmeldung (MM) als Vollzugsdokument erstellt werden.

Eine Maßnahme besteht aus:

- Verfasser
- u.U. modifizierten Parametereinstellungen
- Dokumentenänderung
- betrieblichen Maßnahmen etc.

Sie enthält im Prinzip ähnliche Angaben wie eine Stellungnahme. Der Status ist beim Erstellen auf 'abgegeben' gesetzt und kann explizit auf 'in Betrieb' gesetzt werden.

Änderungsmeldung

Wurde ein Änderungsantrag von einer verantwortlichen Person/ Firma freigegeben und die Änderung durchgeführt, so muß der Ändernde ein Vollzugsdokument erstellen, in dem er genau beschreibt, was er geändert hat.

Die Änderungsmeldung (AM) beinhaltet vor allem:

- o Verfasser
- o Bezüge auf (Teil-) Systeme und/oder Modulpakete
- o Kurztitel
- o ausführliche Beschreibung der Tests und ähnliches
- o Verweise auf andere Dokumente, insbesondere den AA

Der Status wird beim Erstellen auf 'abgegeben' gesetzt, und kann explizit auf 'codiert', 'für Test frei', 'im Test', 'getestet', 'betriebsbereit' oder 'in Betrieb' gesetzt werden.

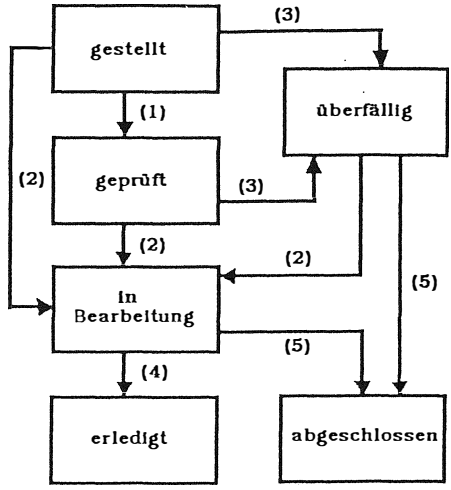
2.2 Die Stati der Dokumente

Eine Störmeldung (SM) ist im Status:

- o 'gestellt' — wenn eine SM interaktiv erstellt worden ist.
- o 'geprüft' — wenn sich im Dialog Prüfer (evtl. nachträglich beim Ändern/ Erweitern) eingetragen hat. Wer als Prüfer befugt ist, wird für das jeweilige Projekt (System) vereinbart - z.B. Systembeauftragter des Betriebs, SW-Wartungspersonal etc.
- o 'überfällig' — wenn zur SM noch keine Stellungnahme und/oder Änderungsantrag existiert, die SM also den Status 'gestellt' oder 'geprüft' hat, der Termin aber überschritten wurde.
- o 'in Bearbeitung' — wenn eine Stellungnahme und/oder ein Änderungsantrag erstellt wurde, die auf diese SM verweisen.

- 'abgeschlossen' wenn die SM durch einen Berechtigten explizit als unerledigt (oder nicht zu erledigen) gesetzt wurde, die Störung also nicht weiterbearbeitet werden soll (oder kann).

- 'erledigt' wenn alle Änderungen und/oder Maßnahmen in Betrieb sind, also alle Änderungsund Maßnahmenmeldungen den Status 'in Betrieb' haben.



(1) Prüfung und evtl. Ergänzung
(2) Stellungnahme (2.1) oder Änderungsantrag (2.2)
(3) Keine Stellungnahme / Bearbeitungsbeginn bis zu vorgesehenen Termin
(4) Maßnahme / Änderung in Betrieb Stellungnahme akzeptiert
(5) Unerledigt abgeschlossen

Bild 4 : Zustände und Übergänge einer Störmmeldung

Ein Änderungsantrag (AA) ist im Status:

'gestellt' 'freigegeben', 'in Bearbeitung' und 'in Betrieb'.

- o 'freigegeben' wenn der Auftraggeber oder der Pro-Projektleiter diesen AA explizit freigibt, so daß nun die Maßnahme bzw. Änderung durchgeführt werden kann.

- o 'in Betrieb' wenn die beantragte Maßnahme/Änderung in Betrieb ist, d.h. alle Maßnahmen- und Änderungsmeldungen, die auf diesen AA verweisen, den Status 'in Betrieb' haben. Hiermit ist die SM erledigt.

Eine Stellungnahme (ST) ist im Status:

- o 'abgegeben' wenn diese erstellt worden ist. Sie bezieht sich dabei auf eine SM und/oder Änderungsantrag , deren Status sich auf 'in Bearbeitung' ändert, wenn er es nicht vorher schon war.

- o 'akzeptiert' wenn sie vom Auftraggeber oder Projektleiter explizit auf nicht weiter zu bearbeiten gesetzt wurde und die SM damit erledigt ist.

Eine Maßnahmenmeldung (MM) ist im Status:

'abgegeben' und 'in Betrieb'.

Eine Änderungsmeldung (AM) ist im Status:

'abgegeben', 'codiert', 'für den Test frei', 'in Test', 'getestet', 'betriebsbereit' und 'in Betrieb'.

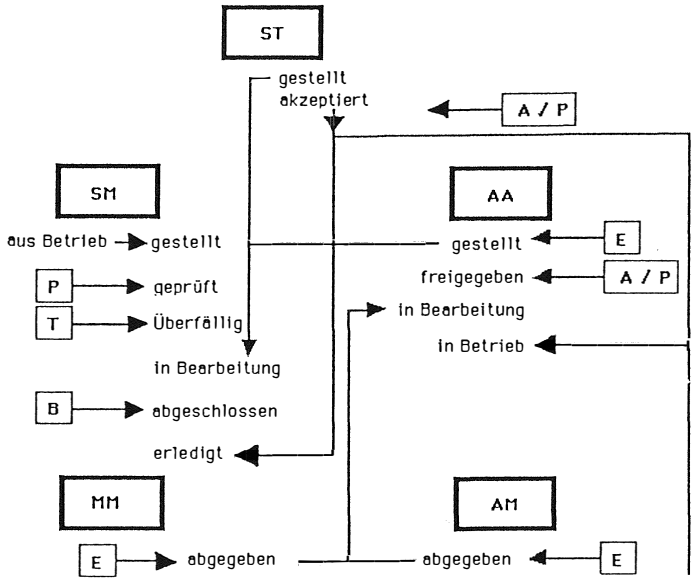
- 'codiert' wenn der Softwareentwickler mit der Codierung fertig ist und die AM vom Auftraggeber oder Projektleiter explizit auf 'codiert' gesetzt hat.

- 'für Test frei' wenn der Auftraggeber oder der Projektleiter die Aenderung explizit für den Test freigegeben hat.

- 'in Test' wenn der Auftraggeber oder der Projektleiter die Aenderung explizit auf 'in Test' gesetzt hat.

- 'getestet' wenn der Auftraggeber oder der Projektleiter die Aenderung explizit als getestet anerkannt hat.

- 'betriebsbereit' wenn der Auftraggeber oder der Projektleiter die Änderung explizit als betriebsbereit bereit erklärt hat.



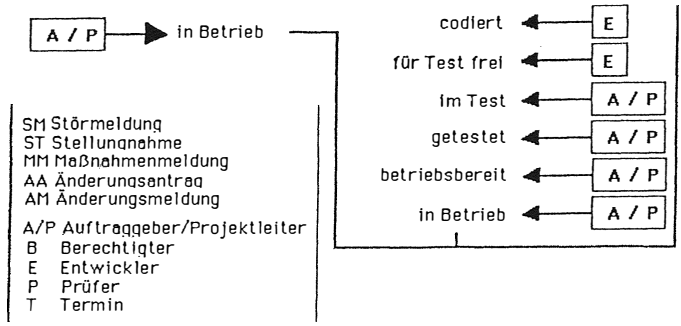


Bild 5 : Zusammenhang zwischen den Statusänderungen

3.2 Aufbau und Realisierung

Das Programm wurde in einer ersten Version in PASCAL für DEC-Rechner unter dem Betriebssystem VMS entwickelt. Bei der Portierung auf einen UNIX-Rechner wurden insbesondere die Gesichtspunkte einer standardisierten Datenhaltung zusätzlich mitaufgenommen, Daraus ergab sich die Notwendigkeit des Einsatzes einer Datenbank. Als durchgängige Basissoftwarelösung boten sich auf dem UNIX-Rechner CADMUS von PCS die Produkte der Fa. WERUM/Lüneburg an:

- Masken- und Dialogsystem MMC (2)
- offenes Echtzeit-Datenbanksystem BAPAS-DB (3)
- Programmschnittstelle zur Datenbank DBV (4)

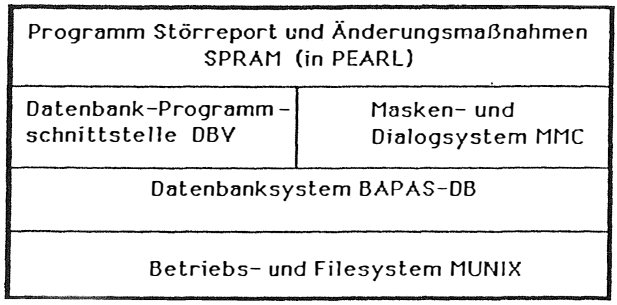


Bild 6 : Softwareaufbau SPRAM

Als Programmiersprachen kamen für die o.a. Basiskomponenten C und PEARL in Frage. Die Entwicklung für PEARL basierte auf den guten Erfahrungen im Rahmen des Großprojektes ARD--Hörunkstern, wo sich insbesondere die klaren und einfachen Schnittstellen sowie das Zusammenwirken zwischen PEARL und den o.a. Basissoftwarekomponenten bewährte.

SPRAM wurde über ein hausinternes Projekt in drei Phasen realisiert. Auf Basis eines detaillierten Lastenheftes wurde eine VAX-Version mit einem Aufwand von ca. 1 Fraujahr realisiert. Die UNIX- und PEARL-Version, die wegen des Einbezuges einer Datenbank ein Redesign erforderte, wurde im Rahmen eines Praxisjahres eines Praktikanten erstellt.

4. Ausblick

Im ursprünglichen Konzept war auch der Einbezug des Managements der Softwarekonfigurationen mitangedacht worden. Da es aber hierfür auf dem Markt bereits für alle gängien Betriebssysteme eingeführte und bewährte Produkte gibt, wurde dieser Ansatz durch den Einsatz eines entsprechenden Produktes (VICO, Fa. Werum) abgedeckt. Eine Verbindung der beiden Werkzeuge SPRAM und VICO erscheint sinnvoll und insofern aussichtsreich, als beide dasselbe Datenbanksystem BAPAS-DB verwenden.

Literatur

(1) Müller, Rainer; Wiedmann, Kathrin:
Bedienungsanleitung für SPRAM, DORNIER-SYSTEM GmbH, Oktober 1987

(2) WERUM Datenverarbeitungssysteme GmbH:

BAPAS-MMC, Ein Werkzeug zur Erstellung von Dialogen und Reporten, Leistungsbeschreibung.

(3) WERUM Datenverarbeitungssysteme GmbH:
BAPAS-DB, Das offene Echtzeit-Datenbanksystem für die Prozeßautomatisierung. Nr. 3.1.3/8805/Ex

(4) WERUM Datenverarbeitungssysteme GmbH:
BAPAS-DB, Echtzeitdatenbanksystem BAPAS-DB, Benutzerhandbuch Release 4.15.02, Kap. 4, Die Programmschnittstelle DBV, Reg. 3.2.1/8802/FB

(5) WERUM Datenverarbeitungssysteme GmbH:
VICO, Version Interface and Configuration Control, 8709.

Ein objekt-orientierter Ansatz für die Echtzeitdatenverarbeitung

Lambis Tassakos, Klaus W. Pleßmann

Lehr- und Forschungsgebiet für Verfahren der
Prozeßdatenverarbeitung und Prozeßführung
(pdv)
RWTH Aachen, Steinbachstr. 54a, D-5100 Aachen

Zusammenfassung

Der objekt-orientierte Ansatz stellt heute eine echte Alternative zur Entwicklung von großen Software-Systemen dar. Gleichzeitig mit der noch andauernden intensiven Grundlagenforschung auf diesem Gebiet werden die ersten kommerziellen Systeme angeboten und in verschiedenen Projekten mit der Ambition eingesetzt, die auch heute noch vorhandene Softwarekrise in den Griff zu bekommen. Unsere Beitrag beschäftigt sich mit der Frage, ob und mit welchen Voraussetzungen sich diese Entwicklungen auch im Bereich der Echtzeitsysteme in der Automatisierungstechnik einsetzen lassen. In diesem Zusammenhang stellen wir die Konzepte eines parallelen, objekt-orientierten Modells vor, das im Hinblick auf seinen Einsatz in komplexen Anwendungen in der Prozeßdatenverarbeitung entwickelt worden ist. Dabei werden sowohl verschiedene Entwurfsentscheidungen erläutert als auch die wichtigsten Aspekte vorgestellt und diskutiert. Zuvor wird zum allgemeinen Verständnis und zu einer klaren Definition und Abgrenzung der entsprechenden Begriffe kurz auf die grundlegenden Prinzipien des objekt-orientierten Ansatzes eingegangen, wie er heute allgemein auf internationaller Ebene verstanden wird.

1. Einleitung

Die objekt-orientierte Programmierung wird heute als eine praktische und nützliche Methode angesehen, große und komplexe Programme herzustellen /1,2/. Mit diesem Begriff wird nicht nur eine bestimmte Programmiersprache sondern auch die allgemeinere Programmierumgebung und -technik der Softwareproduktion mit aufgefaßt /3/. Diese innovative Programmtechnik unterstützt insbesondere einen modularen Entwurf in Zusammenhang mit einer inkrementellen Realisierung des Systems und bringt gleichzeitig die Voraussetzungen dafür, dem Traumziel einer allgemeinen, ausgedehnten Wiederverwendbarkeit von Software näher zu kommen. Gerade wegen dieser Eigenschaft verspricht die objekt-orientierte Methodik, ein wirksames Mittel gegen den größten Feind eines großen, komplexen Software-Systems zu sein: die ständig auftretenden Änderungen der Anwendungsgegebenheiten während des gesamten Software-Zyklus.

Die Motivation für die Anwendung einer objekt-orientierten Programmierung in der Prozeßdatenverarbeitung ist die gleiche, wie für andere Software-Systeme. Die mit der Entwicklung und vor allem mit der Wartung von Software zusammenhängenden Probleme sind in diesem Feld auch vorhanden. Gerade aus der Tatsache der ständigen Änderungen in allen Stadien des Lebens eines großen Software-Systems resultiert die heutige unbefriedigende Situation im Bereich der Software-Entwicklung, die mit dem Schlagwort Software-Krise bezeichnet wird. Während der Hardware-Ingenieur in den letzen Jahren gelernt hat, in Termen von wiederverwendbaren Modulen zu denken, was akkumulativ einen exponentiellen Zuwachs seiner Produktivität erlaubte, realisiert der Programmierer immer weiter seine Programme durch Hintereinanderschreiben von einmal verwendbaren Programmzeilen.

Durch die spezifischen Eigenschaften des objekt-orientierten Ansatzes, nämlich ausgedehnte Wiederverwendbarkeit, erhöhte Lesbarkeit, inhärente Flexibilität und Modularität, erhofft man sich auch im Bereich der Softwareproduktion einen ähnlichen exponenziellen Zuwachs, wie es für den Hardware-Entwurf heute der Fall ist, zu erhalten. Dementsprechend ist es nicht von ungefähr, daß man in diesem Zusammenhang nicht nur von Programmpaketen sondern auch gern von "Software-ICs" spricht /4/.

Mit einem gezielten Einsatz eines objekt-orientierten Modells für die Modellierung und Programmierung von Realzeitsystemen kann eine Verbesserung der Software-

Produktivität und eine Herabsetzung der Entwicklungs- und Wartungskosten dieser Systeme herbeigeführt werden. Echtzeitsysteme haben oft ein relativ langes Systemleben, so daß die Verminderung der Wartungskosten einen deutlichen Beitrag zur Verminderung der Gesamtkosten bedeutet. Die gestiegenen Anforderungen an die Flexibilität solcher Systemen können nämlich nur mit einer entsprechenden Methodik bzgl. der Wartung begegnet werden, was für moderne, mit lokaler "Intelligenz" ausgestatteten Systeme zum größten Teil mit der Wartung der Software sehr eng zusammenhängt.

2. Der objekt-orientierte Ansatz

Die Entstehung und die Evolution des objekt-orientierten Ansatzes hat sich in zwei Bereichen abgespielt, die ähnliche Ziele verfolgt haben. Der erste Bereich, innerhalb dessen die Ideen des objekt-orientierten Ansatzes entwickelt worden sind, ist der Bereich der Systemmodellierung und der Programmiersprachen /5/. Der andere Bereich liegt im Gebiet des Operationsprinzips der Rechnersysteme und ist gleichzeitig aus der Organisation der Systemzuverlässigkeit und der Protektion von Rechnerarchitekturen geprägt /6/. In beiden Fällen stand die Intention im Vordergrund, durch Erhöhung der Abstraktionsebene aus der Sicht des Benutzers den Einsatz von Rechnern plausibler, überschaubarer, effizienter, zuverlässiger und wirtschaftlicher zu verwirklichen.

Die Erhöhung der Abstraktion basiert im Gegensatz zum aktions-orientierten konventionellen Ansatz auf dem Objekt-Modell. Dabei wird die Ansicht hervorgehoben, daß ein Programm im Grunde genommen die Definition, Kreation, Manipulation und Wechselwirkung zwischen Mengen von autonomen Komponenten beschreibt, die Objekte genannt werden /7/. Dieser Ansicht sollten die entwickelten Operationsprinzipien der Rechner Rechnung tragen und dementsprechend ein objekt-orientiertes Berechnungsmodell unterstützen. Damit könnten diese Prinzipien dazu beitragen, die große semantische Lücke zwischen Hardware und Software zu schließen.

Bei der traditionellen Sicht werden die Softwaresysteme als eine Sammlung von Daten und eine Menge von Prozeduren angesehen. Die Daten präsentieren die Informationen im System, und die Prozeduren die Manipulation der Information. Dabei werden Daten und Prozeduren so behandelt, als ob sie voneinander unabhängig wären, was natürlich nicht der Fall ist. Beim objekt-orientierten Ansatz hat

man dagegen nur einen grundlegenden Begriff, der sowohl Daten als auch Prozeduren repräsentiert: das Objekt. Die Objekte besitzen in dieser Hinsicht einen dualen Charakter: Sie können als Daten angesehen werden, da sie manipulierbare Information darstellen. Sie verhalten sich aber auch als Prozeduren, da sie auch die Manipulationen der Daten beschreiben. Die Objekte können also sowohl passive als auch aktive Rollen übernehmen. Dies stellt einen Dualismus von passiven Daten und aktiven Prozeduren dar. Durch diese Integration (encapsulation) von Daten und ihren assoziierten Aktionen innerhalb der Objekte wird das Prinzip der Informationsabkapselung (information hiding) direkt unterstützt.

Objekt-orientierte Systeme betrachten die Welt als eine Ansammlung von Objekten, wobei sich die Welt dadurch verändert, indem ihre Objekte ihren inneren Zustand verändern und indem sie durch das Senden und Empfangen von Nachrichten in Wechselwirkungen mit den anderen Objekten treten /8/. Die Evolution des Informationsprozesses findet durch die Zustandstransitionen der Objekte statt.

Obwohl es sehr schwierig ist, den Begriff "Objekt" exakt zu definieren, kann man im allgemeinen ein Objekt folgendermaßen beschreiben:

> Ein Objekt ist eine abgekapselte Einheit, die sowohl aus einer Informationsmenge (lokalen Daten) als auch aus den diese Informationen manipulierenden Operationen (Prozeduren) besteht.

Die Objekte beinhalten also die Informationen, die nur ihnen gehören. Damit verkörpern die objekt-orientierten Systeme das Prinzip der Datenabkapselung auf eine natürliche Weise. Information wird durch das Senden einer Nachricht an das Objekt, das diese Information enthält, manipuliert. Diese Nachricht benennt die gewünschte Manipulation aus der Menge der im Objekt vorhandenen Manipulationsmöglichkeiten (Operationen). Für diesen Zweck enthalten die Nachrichten einen sogenannten Selektor, der vom empfangenden Objekt dazu benutzt wird, die entsprechende Operation aus den ihm zur Verfügung stehenden Operationen herauszufinden. Diese Operationen werden im objekt-orientierten Jargon Methoden genannt.

Durch den Begriff der Nachricht werden gleichzeitig zwei Dinge realisiert. Zum einem werden durch das Senden von Nachrichten die Objekte zu Verarbeitungsaktivitäten aufgefordert, wobei diese Aktivitäten innerhalb der Objekte stattfinden. Bei einer solchen "Aktivierung" führt das Objekt eine seiner Manipula-

tionsroutinen auf seine eigenen Daten aus und kann währenddessen auch weitere Nachrichten an andere Objekte senden, um weitere Aktivitäten anzufordern. Zum anderen stellen die Nachrichten die Basis zur Kommunikation zwischen den Objekten dar, weil eine Nachricht außer als Ankündigung einer Anforderung zusätzlich als Träger beliebiger Informationen (damit auch ganzer Objekte !) benutzt werden kann.

3. Echtzeitverarbeitung und der objekt-orientierte Ansatz

Entsprechend dem Prinzip der Objektabkapselung können die Teile eines Echtzeitsystems als Objekte mit einer konkreten Funktionalität und mit bestimmten Eigenschaften herausgehoben und spezifiziert werden. Der Entwerfer eines Objektes gestaltet es entsprechend seiner spezifizierten Funktionalität und bietet den potentiellen Benutzern dieses Objektes eine abstrakte Schnittstelle für seine Interaktion mit der übrigen Welt an. Dies geschieht durch die Angabe seines Protokolls (d.h. die Menge aller seiner aufrufbaren Methoden). Damit verbirgt er gleichzeitig alle intern programmierten, für den Benutzer unnötigen Einzelheiten des Objektes. Auf diese Art kann in einem logischen Sinn sowohl eine Zerlegung des gesamten Systems in seine natürlichen Teile, als auch seine Synthese aus seinen Bestandteilen erreicht werden, wobei diese beiden Möglichkeiten normalerweise kombiniert verwendet werden.

Als Ausgangsbasis für den Aufbau der Objektwelt einer Applikation sollte das Ergebnis der Anforderungsphase und der Systemanalyse genommen werden. Dadurch kann man danach in die Entwurfsphase des Systems in einer homogenen Weise ohne aufwendige und fehleranfällige semantische Transformationen zwischen Anforderungen und Entwürfen übergehen. Eins der wesentlichen Merkmale ist hierbei, daß man keine starren und abgekapselten Projektphasen mehr hat, da der oben erwähnte fließende Übergang solche Grenzen aufhebt und eine flexible Rückkopplung und offene Wechselwirkung zwischen diesen Phasen ermöglicht /9/.

Durch die Benutzung eines entsprechenden objekt-orientierten Systems, welches das grundlegende Modell in der Software und eventuell sogar mit seiner Hardware unterstützt, kann man den oben geschilderten natürlichen Phasenübergang bzw. die traditionelle Phasenauflösung bis hin zu der Realisierung und sogar Wartung des Systems ausdehnen. Die daraus entstehende Homogenität der angewandten Methodiken und der verwendeten Werkzeuge während des gesamten Zyklus eines Pro-

jektes stellt ein sehr wirksames Mittel zur Bekämpfung und Reduzierung der Komplexität und der Kosten eines Projekts dar.

Um einen glatten Übergang (smooth transition) zwischen den einzelnen Projektphasen zu ermöglichen, muß allerdings der zugrundeliegende objekt-orientierte Ansatz mit seiner konzeptionellen Mächtigkeit und seinen Spezifikationsmöglichkeiten die entsprechenden Bedürfnisse des jeweiligen Einsatzgebietes und in unserem Fall die der Echtzeitverarbeitung abdecken. Dabei ist die Hervorhebung eines aktiven Charakters der Bestandteile des Echtzeitsystems von elementarer Bedeutung. In diesem Umfeld sind viele Systemkomponenten ständig aktiv und befinden sich in einer kontinuierlichen Wechselwirkung miteinander. Der mit dem objekt-orientierten Ansatz verbundene Dualismus der Objekte bezüglich Information und Manipulation bzw. Aktivität bietet hierfür eine hervorragende Grundlage. Dafür reicht allerdings nicht das übliche, nach Smalltalk-Art gestaltete Modell aus, wo in jedem Zeitpunkt nur ein Objekt aktiv ist und wo ein Objekt erst nach Beendigung seiner aktuellen Methode in der Lage ist, weitere an ihm gerichtete Nachrichten zu empfangen. Damit wird angedeutet, daß für den Einsatz des objekt-orientierten Ansatzes in der Prozeßdatenverarbeitung eine Variante benötigt wird, die eine selbständige und parallele Aktivitätsentfaltung der Objekte in ihrem Grundkonzept beinhaltet.

Aufgrund des interpretativen Charakters des Selektionsmechanismus, der mit der Assoziierung einer Objektmethode zu einer angekommenen Nachricht verbunden ist, bietet sich der objekt-orientierte Ansatz im Hinblick auf die Laufzeiteffizienz der Software auf konventionellen Prozeßrechner eher für lose-gekoppelte Strukturen. Bei Echtzeitsystemen können die Vorteile des dynamischen Bindens voll ausgenutzt werden, wo keine strengen Anforderungen an eine 100% Einhaltung von Zeitlimits vorliegen. Moderne, komplexe Echtzeitsysteme enthalten allerdings eine Mischung von eng und lose gekoppelten Strukturen, so daß ein objekt-orientiertes Modell hybrider Natur (d.h., es muß keine reine objekt-orientierte Welt aufgebaut werden, sondern es wird innerhalb von Objekten auch eine effiziente, auf konventionelle Art durchzuführende Programmausführung ermöglicht) für die Prozeßdatenverarbeitung vom besonderen Interesse zu sein scheint. Die für die Berechnung der Bahn eines Roboterarmes benötigten Operationen und die Natur ihrer Operanden ist ein Beispiel von einer eng gekoppelten Struktur, wo alles schon bei der Compilierzeit bekannt ist und harte zeitliche Anforderungen gesetzt werden. Im Gegensatz dazu präsentiert eine flexible Produktionsinsel ein lose gekoppeltes System, wo

einerseits die zeitlichen Anforderungen anderen Charakters sind und anderseits eine hohe Flexibilität bzgl. Modifikationen und Erweiterbarkeit gefordert wird.

Die Ausnahme- und Unterbrechungsbehandlung gehören zu den grundlegenden Aspekten in Echtzeitsystemen. Im Gegensatz zum normalen objekt-orientierten Modell müssen die Objekte hier u.U auch in der Lage sein, ihre aktuellen Aktivitäten zu unterbrechen und sie entsprechend den ständig verändernden Gegebenheiten des technischen Prozesses anzupassen. Eine besondere Anforderung an die Spezifikationsmöglichkeiten des Modells stellt die Notwendigkeit dar, sogenannte Unterbrechungsobjekte als solche zu spezifizieren und in eine einfache und effiziente Art an das Unterbrechungssystem des Rechners anzubinden. Effizient bedeutet hierbei meistens eine feste und hardwaremäßig unterstützte Verbindung, was wiederum die Notwendigkeit einer harmonischen Koexistenz einer statischen mit einer dynamischen Struktur in großen, komplexen Echtzeitsystemen zum Ausdruck bringt.

Einer der Vorteile des objekt-orientierten Ansatzes für die Spezifikation und Programmierung von technischen Systemen ist, daß er ein natürliches Modell für die Abbildung der realen Welt des technischen Prozesses auf die Software anbietet. Mit seiner Hilfe kann man reale Objekte aus der Prozeßumgebung direkt in entsprechende Software-Objekte in das Rechnersystem abbilden: Sensor A sendet der Maschine B die Nachricht "Überschreitung der Temperatur" und löst damit eine entsprechende Reaktion der Maschine B aus, deren Natur und Einzelheiten nur die Maschine B selbst kennt und aufgrund der übertragenen Parameter und ihres internen Zustandes entscheidet (design decision and information hiding). Auf diese Weise hat man zweierlei Gewinn: einerseits einen adäquaten und logischen Systemaufbau und anderseits einen modularen und hochqualitativen Entwurf, der einfach zu warten ist.

4. Ein paralleles, objekt-orientiertes Modell für die Echtzeitverarbeitung

4.1 Grundlagen

Als Grundelemente in diesem Modell stellen die Objekte aktive, autonome Aktoren dar, die Fähigkeiten zur eigenen Aktivitätsentfaltung in der Objektwelt und ein eigenes Gedächnis zum Festhalten von Daten, Informationen und sogar Wissen inhärent besitzen können. Durch die Entfaltung eigener Aktivitäten demon-

striert jedes Objekt ein individuelles Verhalten zu seiner Umwelt, wobei es durch den Kontakt mit seiner unmittelbaren Umwelt in Wechselwirkungen mit der gesamten übrigen Objektwelt kommt. Dieser Kontakt äußert sich insbesondere durch eine auf Nachrichtenaustausch basierte Kommunikation mit den übrigen Objekten. Jedes Objekt entspricht einer logischen oder physikalischen Einheit im Problembereich. Damit kann die konzeptionelle Lösung des Problems in einer unifizierten und natürlichen Weise stattfinden, wobei das gleiche auch für die Implementation in Software und Hardware gilt.

Um gefährliche interne Interferenzen vorzubeugen, wird jedem elementaren Objekt erlaubt, zu jedem Zeitpunkt nur eine Aktion durchzuführen. Es kann z.B. nicht gleichzeitig mehrere seiner Manipulationsroutinen auf seinen inneren (permanenten und/oder temporären) Zustandsraum anwenden, oder gleichzeitig senden und empfangen. Die Parallelität kommt erst durch das Zusammenspiel mehrerer Objekte zustande, wie es im nächsten Unterkapitel erläutert wird. Zusammengesetzte Objekte bieten auch die Möglichkeit, mehrere parallele Aktivitäten innerhalb eines einzigen (nicht mehr elementaren) Objektes zu modellieren.

Der wesentliche Unterschied zu anderen objekt-orientierten Modellen besteht allerdings darin, daß ein Objekt parallel zu seiner operationellen Tätigkeit in der Lage ist, ankommende Nachrichten wahrzunehmen. Ein solches Objekt kann dann gegebenenfalls die aktuell ausgeführte Routine (Methode) abbrechen und mit der Ausführung einer anderen Routine beginnen. Entsprechend der Semantik und der Dringlichkeit der angekommenen Nachricht und in Abhängigkeit seines internen Zustandes kann nämlich das Objekt selbständig entscheiden, ob es die aktuelle Tätigkeit fortsetzt oder sie abbricht und mit der Ausführung der mit der angekommenen Nachricht assoziierten Methode beginnen soll. Damit sind insbesondere Aspekte der Behandlung von Ausnahme- und Fehlersituationen einerseits und von Unterbrechungen andererseits verbunden. Diese inhärente Objektfähigkeit eröffnet erst den Weg zu einem echtzeitkonformen Operationsmodell.

4.2 Parallelität

Wir betrachten die Parallelität als eine grundlegende Eigenschaft unseres Modells, die von Anfang an zusammen mit den übrigen Grundkonzepten entwickelt und nicht wie bei anderen Modellen erst später als eine zusätzliche Erweiterung addiert wurde (z.B. /Concurrent Smalltalk/). Ein paralleles Operationsprinzip ist nämlich die intuitive und natürliche Art, auf die die vielen eigenständigen Objekte eines

technischen Prozesses bzw. eines großen, verteilten Systems ihre Aktivitäten entfalten /10/. Motiviert auch von dem Wunsch, das hier diskutierte Modell für Anwendungen der Prozeßdatenverarbeitung benutzen zu wollen, wird hier die Parallelität durch das Hinzufügen eines jeweils eigenen Rumpfes (Body) in die Objekte gestaltet. Das Objekt ist dann in der Lage, diesen Body unabhängig vom Eintreffen von Nachrichten auszuführen, und damit als eine unabhängige, aktive Einheit eigenständig zu fungieren. Auf diese Weise können die ständig laufenden Prozesse und die kontinuierlichen Aktivitäten, wie sie in der technischen Umgebung vorkommen, auf eine natürliche Art modelliert werden, womit eine wesentliche Forderung der Prozeßdatenverarbeitung auf eine für den Benutzer intuitive Weise erfüllt wird.

Die Aufname eines Body in ein Objekt ist optional. Nach seiner Erzeugung beginnt jedes Objekt zunächst mit der Ausführung seines Body. In diesem Fall gibt es durch den "accept"-Befehl explizite Stellen in den Body-Routinen, wo die Bedienung von externen, angekommenen Nachrichten erlaubt ist. Dies läßt die Konstruktion und Einhaltung von internen Invarianten zu. Nur Ausnahme-Nachrichten (wie sie im folgenden Unterkapitel erläutert werden) können die normale Programmausführung auch in anderen Stellen unterbrechen und eine Ausnahmebehandlung auslösen.

Die eventuell notwendige Garantie der internen Variableninvarianten innerhalb eines Objektes und/oder die Koordination und Synchronisation seiner Aktivitäten obliegt der alleinigen Verantwortung des Objektes selbst. Das bedeutet, daß hier keine Semaphoren-Objekte für die Koordinierung des Zugriffes auf gemeinsam bekannte Objekte verwendet werden, wie es z.B. in parallelen Versionen von Smalltalk der Fall ist. Wenn ein Objekt ein eingeschränktes Betriebsmittel darstellt (z.B. ein eingeschränkter Puffer), muß es für die benötigte Koordination mit Hilfe seines Body selbst sorgen.

4.3 Kommunikation und Synchronisation

Das Kommunikationsmodell ist für einen parallelen, objekt-orientierten Ansatz von zentraler Bedeutung, da damit nicht nur ein flexibler Aufruf- und Kommunikationsmechanismus sondern insbesondere die Synchronisation und die Koordination der kooperativen Aktivitäten der parallelen Objekte verbunden ist. In unserem Modell unterscheiden wir bewußt zwischen der Wahrnehmung über das Ankommen einer Nachricht und dem Beginn der Ausführung der assoziierten Methode. Weil die Objekte als eigenständige Aktoren fungieren, sollen sie in der Lage sein, ihre Aktivi-

täten zu entfalten und gleichzeitig das Eintreffen einer Nachricht zu registrieren. Dies erlaubt den Objekten selektive Aktivitäten auf der Basis von vorprogrammierten Schemata zu verwirklichen.

Es gibt sowohl eine synchrone als auch eine asynchrone Kommunikationsart, die allerdings auf den gleichen semantischen Prizipien aufbauen. Damit kann der Benutzer auf eine einfache Weise die für ihn jeweils geeignete Art auswählen. Vier Kommunikationsprimitiven bilden die Basis für unser Kommunikationsmodell:

- **send** (destination object, message)
- **reliableSend** (destination object, message)
- **ask** (destination object, message) und
- **reply** (destination object, expression)

Mit Hilfe des **send**-Befehls findet eine asynchrone Kommunikation statt. Nachdem die Nachricht vom Sender (Objekt A) konstruiert und gesendet wurde, kann er seine weitere Aktivitäten fortsetzen (Bild 1). Die Nachricht wird nach einer gewissen Zeit an ihrem Empfangsort ankommen (tB1), vom Empfänger (Objekt B) wahrgenommen (tB2), und noch später (vielleicht auch direkt, wenn das empfangende Objekt auf Nachrichten gewartet hat und keine andere Nachricht vorliegt) die Selektion und Ausführung einer Methode des Empfängers auslösen (tB3).

Obwohl durch den send-Befehl der maximale Parallelitätsgrad erreicht wird, wird die Gültigkeit zweier Annahmen bzgl. der Nachrichtenübertragung nicht garantiert,

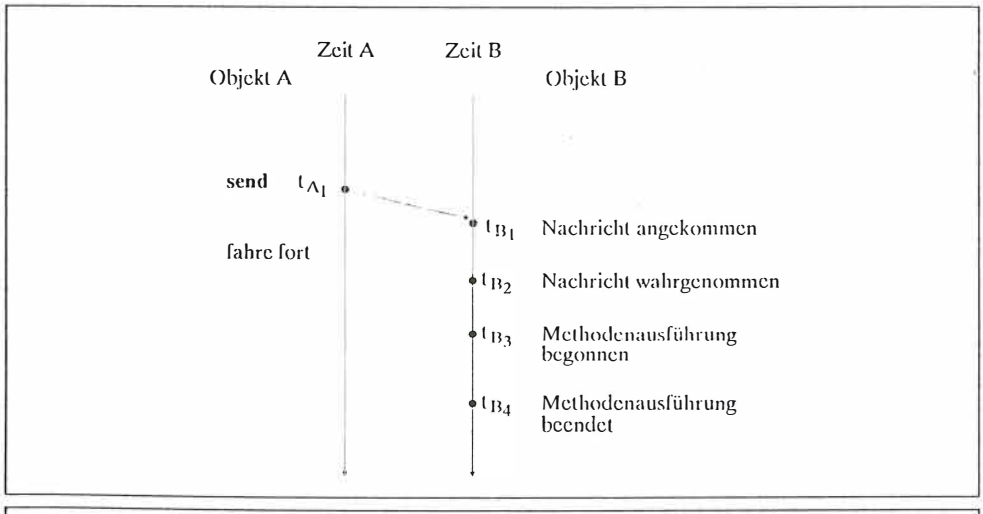


Bild 1. Der send-Befehl

nämlich:

- die tatsächliche Lieferung der Nachricht an den Empfänger und
- die Einhaltung der zeitlichen Ordnung zweier Nachrichten bei ihrer Übertragung /11/

Der **reliableSend**-Befehl realisiert inhärent beide Annahmen. Das sendende Objekt A wartet mit der Entfaltung weiterer Aktivitäten, bis es eine Bestätigung für die Wahrnehmung der gesendeten Nachricht von Seite des Empfängers erhalten hat (Bild 2). Damit kann der Sender sicher sein, daß der Empfänger seine Nachricht erhalten hat. Darüber hinaus ist die zweite obige Forderung auch erfüllt, weil bei einer wiederholten Ausführung solcher reliableSend-Befehle vom Objekt A jeweils auf die entsprechende Bestätigung gewartet werden muß, bevor der eine Befehl abgeschlossen und der nächste begonnen werden kann.

Es gibt eine spezielle Form des reliableSend-Befehls, die sich von der normalen dadurch unterscheidet, daß das empfangende Objekt nach der Registrierung der Nachricht seine bisherige Aktivität abbricht und sofort mit der Ausführung der mit der Nachricht assoziierten Methode anfängt (Bild 3, tB2 = tB3):

- **exceptionalSend** (destination object, message)

Die durch diesen Befehl erzeugten Nachrichten heißen Ausnahmenachrichten (ex-

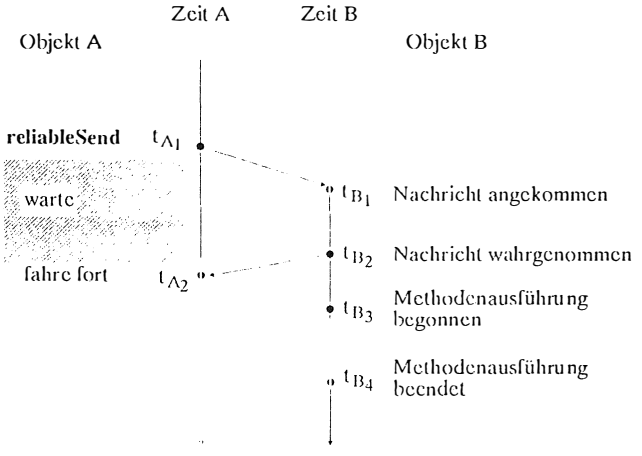


Bild 2. Der reliableSend-Befehl

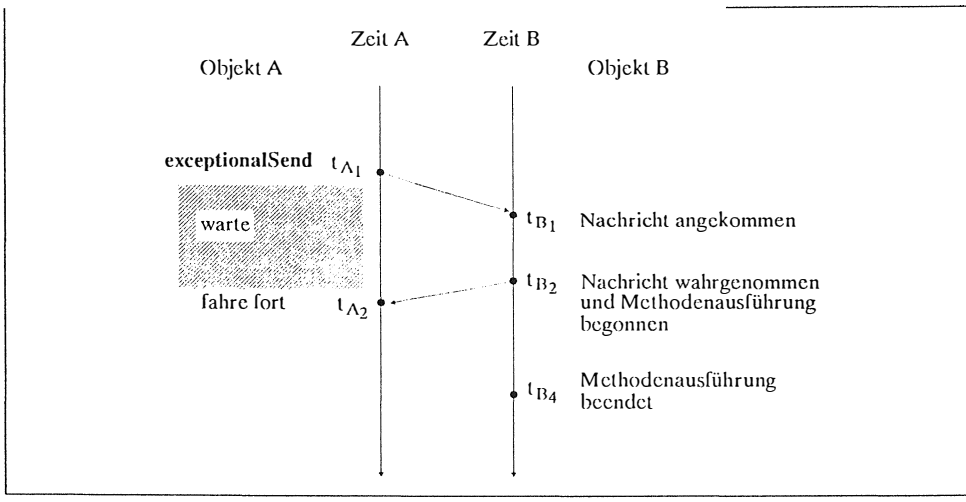


Bild 3. Der exceptionalSend-Befehl

ceptional messages). Es kann auch vereinbart werden, daß diese Nachrichten die Stimulierung von nur einer bestimmten Sorte von "Ausnahme-Methoden" zur Folge hat, welche den Aspekten von Ausnahme- und Unterbrecherbehandlung dienen. Diese Ausnahmemethoden können durch das Ankommen weiterer Nachrichten nicht mehr unterbrochen werden; das Objekt kapselt sich von der Außenwelt ab, um sich auf die Ausnahmebehandlung zu "konzentrieren".

Wenn das sendende Objekt die Ergebnisse seiner Nachrichtentransaktion benötigt, um die Ausführung seiner aktuellen Routine fortzusetzen, dann benutzt es den **ask**-Befehl. Dabei wartet der Sender nicht nur, bis seine Nachricht an den Empfänger angekommen ist, sondern solange, bis dieser seine Anforderung bearbeitet hat und die Resultate an ihn durch den **reply**-Befehl zurückgesendet hat (Bild 4). Damit verkörpert das Befehlspaar ask/reply die Funktionalität eines ("remote") "procedure call"-Befehls. Die durch den reply-Befehl erzeugte Nachricht unterscheidet sich dadurch von einer normalen Nachricht, daß sie bei ihrem Ankommen an das anfordernde Objekt keine Methodenselektion und -ausführung auslöst, sondern einfach das Resultat der gestellten Anforderung überträgt. Damit sieht man, daß es eigentlich zwei Stellen gibt, wo Nachrichten an ein Objekt eintreffen können.

Die an den Empfänger einer Nachricht gestellte Dienstleistungsanforderung muß er nicht unbedingt selbst weiter bearbeiten. Wenn er ein anderes Objekt kennt, das diese Dienstleistung (besser) erbringen kann, kann er durch eine weitere entsprechende Nachricht die Anforderung an dieses Objekt **weiterleiten**. Durch gleichzeiti-

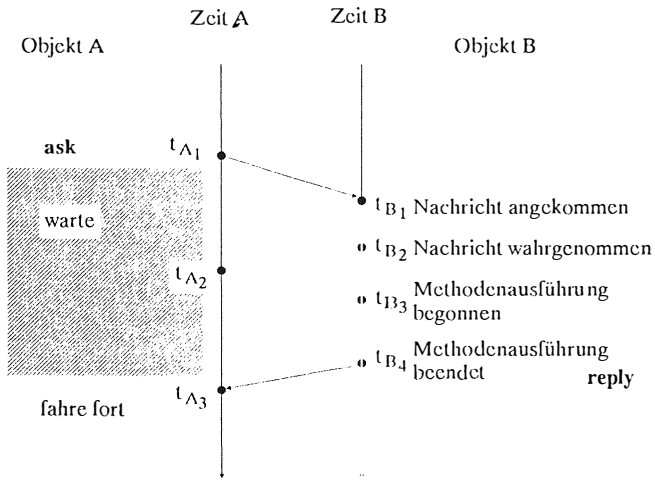


Bild 4. Das Befehlspaar ask/reply

ge Bekanntmachung des Namens des ursprünglich anfordernden Objektes kann er sogar erreichen, daß er sich von jeglicher Weiterverarbeitung der Anforderung befreit, da jetzt das dritte Objekt die Antwort direkt an das erste Objekt schicken kann. Damit entsteht eine hervorragende Grundlage für die Einführung globaler Server in das System, die mit Hilfe paralleler "Helfer-Objekte" sehr schnell mehrere Anfragen gleichzeitig bearbeiten können.

4.4 Zusammengesetzte Objekte

In vielen Fällen besteht der Wunsch und z.T. auch die Notwendigkeit, eine bestimmte Anzahl bzw. Gruppe von Objekten als eine einzige und logisch zusammenhängende Einheit zu betrachten und anzusprechen /12/. Man könnte so auf eine natürliche Art eine Maschine modellieren, bei der die einzelnen Teile trotz der Zusammenfassung zu einem zusammengesetzten Objekt immer noch als eigenständige (und kooperierende) Objekte definiert werden können. Die meisten objekt-orientierten Systeme unterstützen heute keine Spezifikationen und Wechselwirkungen solcher Art, sondern sie erlauben lediglich den Ausdruck von Kollektionen von Objekten, wobei die Zusammengehörigkeit nur über lose Verweise zum Erscheinen kommt.

Ein komplexes Objekt kann so angesehen werden, daß es aus einer strukturellen Hierarchie von Komponentenobjekten besteht /13/. Im Gegensatz zu der "ist-ein"-

Relation, die man in der Klassenhierarchie findet, handelt es sich hier um eine "ist-Teil-von"-Relation. Es ist wichtig dabei anzumerken, daß alle Komponentenobjekte eines zusammengesetzten Objektes in der vorgesehenen Weise vom Eintreffen einer an das zusammengesetzte Objekt gesandten Nachricht beeinflußt werden. Der Begriff des zusammengesetzten Objektes unterstützt einen parallelen Programmentwurf in unserem Modell in einer zuverlässigen Weise. Er trägt eine wesentliche Rolle zur Erhaltung einer ausgedehnten Parallelität trotz der Definition und Einführung von komplexen Objekten bei.

Ein solches Objekt könnte zwar durch eine ausgiebige und strukturierte Ausnutzung der Vererbungsmechanismen eine komplizierte Funktionalität auf modulare und zuverlässige Weise realisieren, könnte aber jederzeit nur eine einzige Aktivität entfalten, womit es in einer eher "verklemmten" Weise in Wechselwirkungen mit seiner Umwelt treten würde. Da es jederzeit z.B. eine einzige Nachricht empfangen bzw. bearbeiten kann, könnte es oft nur nach einer gewissen "Hysterese" auf das Eintreffen weiterer Nachrichten reagieren. Durch seine Aufteilung in eine Anzahl von kleineren, eigenständigen und doch zusammenhängenden Objekten kann es mehrere Aktivitäten gleichzeitig entfalten und dies vor allem auf eine direktere Art, was für die Belange der Prozeßdatenverarbeitung ja von außerordentlicher Bedeutung ist.

Nicht nur das Stammobjekt sondern auch die Komponentenobjekte eines zusammengesetzten Objektes können nämlich für die übrigen Objekten einer Applikation sichtbar sein. Die Interaktion zwischen einem zusammengesetzten Objekt und einem anderen Objekt oder sogar zwischen zwei aus mehreren Teilobjekten bestehenden Objekten kann auf eine natürliche und hoch parallele Weise stattfinden, ohne daß die jeweiligen Stammobjekte (an der Spitze des jeweiligen Abstraktionsbaums) für alle Interaktionen der Objekte ihrer abhängigen Objektbäume immer intervenieren müssen (Bild 5). Dies resultiert in einer Leistungssteigerung des Systems, weil die Nachrichten, die zwischen zwei Objekten unterschiedlicher Teilobjektbäume ausgetauscht werden sollen, direkt gesendet werden können und nicht durch alle Abstraktionsschichten laufen müssen. Die Konfiguration der Nachrichtenwege findet zwar auf der Basis der Funktionalität der gesamten zusammengesetzten Objekte statt, die Nachrichtentransaktionen aber werden danach nur in der alleinigen Verantwortung der betreffenden Teilobjekte durchgeführt.

Ein wichtiger Aspekt ist hierbei sicherlich die Spezifikation einer solchen Konfiguration. Die Konfiguration ist sogar in der Lage, aufgrund des dynamischen

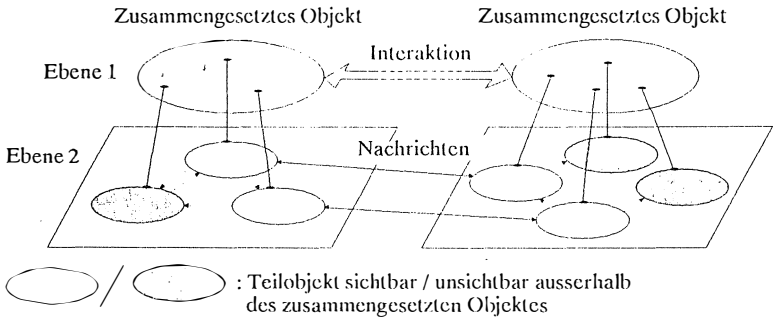


Bild 5. Interaktion zwischen zwei zusammengesetzten Objekten

Charakters des objekt-orientierten Ansatzes sich im Laufe der Zeit zu verändern. Auf diese leichte Änderbarkeit werden u.a. die Weichen zur Konstruktion flexibler Systeme gestellt. Dafür müssen wir aber die Übersicht behalten und uns mit der erhöhten Komplexität auseinandersetzen. Dazu werden strukturierte Mittel und Methoden gebraucht; das Konzept der zusammengesetzten Objekte soll ein solches Mittel darstellen. Die parallelen, echtzeitkonformen, vom Benutzer des zusammengesetzten Objektes zum großen Teil abgekapselten Aktivitätsentfaltungen, die gleichzeitig ein hohes Maß an Objektintegrität besitzen, waren die größte Motivation für die Einführung und die Entwicklung des Begriffes der zusammengesetzten Objekte in diesem Modell.

5. Schlußbemerkungen

Es ist sicherlich sehr interessant, ein komplettes Echtzeitsystem in Termen von aktiven, autonomen und parallel auf das gesamte verteilte System wirkende Einheiten zu modellieren, zu programmieren und zu realisieren. Der objekt-orientierte Ansatz bietet vielversprechende Grundlagen zur Realisierung einer solchen Vorgehensweise. In unseren Beitrag haben wir unsere Vorstellungen darüber erläutert, in welcher Art sich ein objekt-orientiertes Modell in der Prozeßdatenverarbeitung einsetzen ließe. Dazu reichen die heute übliche Modelle nicht aus, die eher für den Bereich der symbolischen Verarbeitung ausgelegt sind. Wichtig für die erfolgreiche Einführung eines Ansatzes in die Praxis ist allerdings u.a. auch seine Effizienz sowohl bzgl. der Modell- und Programmentwicklung aber auch bzgl. der Programmausführung. Für die erste Komponente bieten die spezifischen Eigenschaften des objekt-orientierten Ansatzes eine hervorragende Grundlage. Für die zweite

Komponente ist es vor allem wichtig, daß es eine effiziente Kooperation zwischen der Software und der zugrundeliegenden Hardware existiert, da nur auf diese Weise besonders elegante und effiziente Systeme entstehen können. In diesem Sinn arbeiten wir gleichzeitig an einer objekt-orientierten Architektur, die den grundlegenden Rahmen für die effiziente Ausführung unseren Modells zur Verfügung stellen soll /14/.

6. Literaturverzeichnis

1. Wegner, P., Shriver, B. (Eds.): Object-Oriented Programming Workshop, IBM Yorktown Heigts, 9-13 June 1986, SIGPLAN Notices, Vol. 21, No. 10, Oct. 1986
2. Meyrowitz, N. (Ed.): Proc. ACM Conf. on Object-Oriented Systems, Languages, and Applications. Orlando, Florida, 1987. Special Issue of SIGPLAN Notices, Vol. 22, No. 12, Dec. 1987
3. Cook, S.: Languages and object-oriented programming. Software Engineering Journal, March 1986, 73-80
4. Cox, B.J.: Object-Oriented Programming: An Evolutionary Approach. Reading, Mass. Addison-Wesley Publishing Company, 1986
5. Nygaard, K.: Basic Concepts in Object Oriented Programming. SIGPLAN Notices, Vol. 21, No. 10, Oct. 1986, 128-132
6. Levy, H.M.: Capability-Based Computer Systems. Bedford, Mass.: Digital Equipment Corp., 1984
7. Organick, E.I.: A Programmer's View of the Intel 432 System. New York, NY, McGraw-Hill Book Company, 1983
8. Goldberg, A., Robson, D.: Smalltalk 80: The Language and its Implementation. Reading, MA: Addison-Wesley, 1983
9. Matsumoto, Y.: Requirements Engineering and Software Development. In: Güth, R. (Ed.): Computer Systems for Process Control, New York: Plenum Press, 1986, 241-264
10. Pleßmann, K.W., Tassakos, L.: Concurrent, object-oriented program design in real-time systems. Proc. EUROMICRO 88, Microprocessing and Microprogramming, Vol. 24, Nrs. 1-5, 1988, 257-265
11. Yonezawa, A., Shibayama, E., Takada, T., Honda, Y.: Modelling and Programmming in an Object-Oriented Concurrent Language ABCL/1. In: Object-Oriented Concurrent Programming, Yonezawa, A., Tokoro, M. (Eds.), Cambridge, Mass., London, The MIT Press (Series in Computer Sciences), 1987, 55-89
12. Stefik, M., Bobrow, D.G.: Object-Oriented Programming: Themes and Variations. AI Magazin, Jan. 1986, 40-62
13. Banerjee, J. et al.: Data Model Issues for Object-Oriented Applications. ACM Trans. on Office Information Systems, Vol. 5, No. 1, Jan. 1987, 3-26
14. Tassakos, L., Pleßmann K.W.: pdvPool: A real-time object-oriented multiprocessor system. Presented in Short Notes of EUROMICRO 88, will published in a special issue of "Microprocessing and Microprogramming", the Euromicro Journal, in spring 89.

Deadline-Scheduling in PEARL

Christine Hilbert
GPP mbH
Kolpingring 18a
8024 Oberhaching
Tel.: 089/61 30 4-2 29

Zusammenfassung:
In der Programmiersprache PEARL ist bislang nur eine prioritätsgesteuerte Prozessorvergabe möglich. Es gibt nun viele Problemstellungen in der Praxis, bei denen sich dieses Verfahren als sehr umständlich erweist, nämlich immer dann, wenn ein Prozeß bis zu einem bestimmten Zeitpunkt beendet sein muß. In diesen Fällen stellt sich das Deadline-Scheduling als wesentlich besser und praxisorientierter heraus. Jeder Task wird bei der Deklaration ihre Antwortzeit, also diejenige Zeitspanne innerhalb der sie abgearbeitet sein muß, mitgegeben. Das Betriebssystem vergibt den Prozessor an diejenige Task, deren Soll-Endzeitpunkt am nächsten liegt. Dadurch wird sichergestellt, daß jede Task - soweit dies im Gesamtsystem überhaupt möglich ist - innerhalb der geforderten Zeit beendet wird. Dem Programmierer wird gleichzeitig ein langwieriges Abstimmen von Taskprioritäten im System abgenommen.
In erweiterter Form bietet das Deadline-Scheduling Möglichkeiten für das Handling von Überlastsituationen an.

1. Einleitung

Ein Kennzeichen einer Realzeit-Programmiersprache ist, daß sie nicht nur einen linearen Programmaufbau mit hierarchisch organisierten Unterprogrammaufrufen gestattet. Sie stellt auch Sprachelemente zur Verfügung, die es dem Programmierer ermöglichen, Programmteile zu erstellen, sogenannte Tasks, die unabhängig voneinander, "parallel" ablaufen können. Ob dieses parallel gleich echt parallel zu setzen ist, hängt davon ab, ob es sich um Ein- oder Mehrprozessorsysteme handelt. Nur bei einem Mehrprozessorsystem ist es überhaupt möglich, mehrere Tasks echt zur selben Zeit ablaufen zu lassen. Bei einem Einprozessorsystem entscheidet in jedem Fall das Betriebssystem, an welche aktivierte Task der Prozessor und die Betriebsmittel vergeben werden. Kriterium für diese Vergabe ist die Priorität einer Task, also eine Zahl, die im Task-Kontrollblock vermerkt ist und die angibt, an welcher Stelle eine zu aktivierende Task in das Ranggefüge aller zu einem bestimmten Zeitpunkt aktivierten Tasks eingefügt wird.

Die Priorität wird dabei bisher vom Benutzer bei der Deklaration einer Task in PEARL festgelegt. Die entsprechende Anweisung lautet:

<taskname> :TASK PRIO <prioritätskennzahl>;.

In PEARL bedeutet dabei eine hohe Prioritätskennzahl eine niedere Priorität und umgekehrt, d.h. das Betriebssystem wird diejenige Task starten, die die niedrigste Prioritätskennzahl besitzt.

Die Rangfolge bei dieser Form der Prozessorsteuerung ist dabei von Anfang an statisch vorgegeben und unabhängig vom Programmverlauf.

Der prioritätsgesteuerten Prozessorvergabe soll in diesem Vortrag das Deadline-Scheduling gegenübergestellt werden.

2. Prinzip des Deadline-Scheduling

Beim Deadline-Scheduling oder der antwortzeitgesteuerten Prozessorvergabe wird vom Programmierer keine Priorität sondern eine Antwortzeit angegeben. Die Antwortzeit ist dabei das Zeitintervall, innerhalb dessen die Task abgearbeitet sein muß. Die Antwortzeit besitzt zum Zeitpunkt der Taskaktivierung ihren Maximalwert, nämlich die vom Programmierer bei der Deklaration angegebene Zeitspanne. Sie legt zusammen mit dem Zeitpunkt der Aktivierung den konstant bleibenden Soll-Endzeitpunkt t_e fest, den spätesten Zeitpunkt, zu dem eine Task noch termingerecht beendet ist.
Im Gegensatz zu dieser Konstanten verringert sich die Antwortzeit einer Task stetig, um schließlich bei t_e den Wert 0 zu erreichen. Anders ausgedrückt ist die Antwortzeit die Differenz zwischen Soll-Endzeitpunkt t_e und der momentanen Zeit t.

Die Vergabe des Prozessors erfolgt an diejenige ablauffähige also aktivierte Task, die die niedrigste aktuelle Antwortzeit besitzt oder anders ausgedrückt, deren Soll-Endzeitpunkt am nächsten liegt. Die Soll-Endzeitpunkte die bei der Taskaktivierung berechnet werden, legen eine konstant bleibende Task-Rangfolge fest. Für die nicht aktivierten (ruhenden) Tasks liegen dagegen keine Soll-Endzeitpunkte vor; ihre Positionen in der Rangfolge sind noch undefiniert.
Betriebssystemintern läßt sich die Tabelle mit den Soll-Endzeitpunkten 1:1 in eine Prioritätsskala umrechnen. Je früher der Soll-Endzeitpunkt einer Task liegt, desto höher ist ihre Priorität. Der wesentliche Unterschied zur echten prioritätsgesteuerten Prozessorvergabe liegt darin, daß diese "Priorität" erst zum Zeitpunkt der Aktivierung mit der damit verbundenen Berechnung des Soll-Endzeitpunktes bestimmt werden kann.
Ein einfaches Beispiel mag dies verdeutlichen.
Gegeben seien drei Tasks mit einer Laufzeit von 10s und den folgenden Antwortzeiten:

Task	Antwortzeit
T1	10 s
T2	20 s
T3	30 s

Bild 1: System mit 3 Tasks

Werden sie alle zum selben Zeitpunkt T_o=12.°° gestartet, ergibt sich folgende Prozessorbelegung:

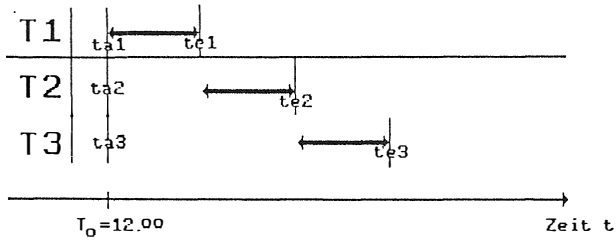


Bild 2: Prozessorbelegung bei simultaner Taskaktivierung

Dabei bedeutet t_a : Aktivierungszeitpunkt
t_e : Soll-Endzeitpunkt
<->: Task belegt den Prozessor

Werden die Tasks T_2 und T_3 zum Zeitpunkt T_o=12.°°, T_1 aber erst um T_1=12.15 aktiviert, erhält man folgende Prozessorbelegung:

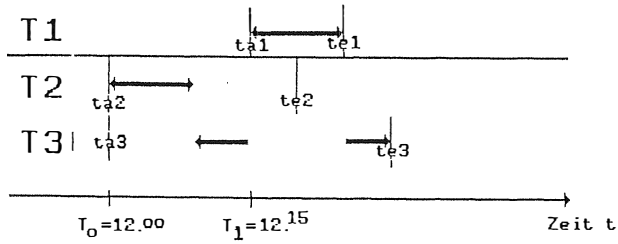


Bild 3: Prozessorbelegung bei gestaffelter Taskaktivierung

Im Unterschied zur prioritätsgesteuerten Prozessorvergabe bedeutet dies, daß die Rangfolge der Echtzeittasks nicht von Anfang an vorgegeben ist, sondern vom Zeitpunkt der Aktivierung abhängt.
Bei Eingliederung einer neuen Task in das Ranggefüge durch ihre Aktivierung, bleibt die Reihenfolge der bereits aktivierten Tasks untereinander natürlich erhalten, da wie gesagt die Soll-Endzeitpunkte konstant bleiben.
Das Deadline-Scheduling erfordert neue Sprachelemente in PEARL. Für die Angabe der Antwortzeit bei der Task-Deklaration wird folgende Syntax vorgeschlagen /1/:
<taskname> :TASK DUE AFTER <duration-expression>;

3. Vergleich prioritäts- und antwortzeitgesteuerter Prozessorzuteilung

Anhand eines einfachen Beispiels sei nun die Prozessorvergabe bei prioritäts- bzw. antwortzeitgesteuerter Zuteilung gegenübergestellt.

Beispiel:
Bei der Steuerung eines technischen Prozesses werden 3 Meßfühler benötigt, die den Wert verschiedener Zustandsgrößen wie Temperatur, Druck oder Stoffkonzentrationen in bestimmten Zeitintervallen ermitteln. Aus diesen Daten werden Steuergrößen, die zur Regelung benötigt werden bestimmt. Im Programm übernehmen 3 Tasks das Ablesen der Zustandsgrößen sowie die Berechnung und Weitergabe der Steuergrößen. Die Tasks werden zyklisch aktiviert und sollen jeweils innerhalb ihrer Zykluszeit beendet werden. Die Zykluszeit ist umso kürzer, je kritischer die Änderung einer bestimmten Zustandsgröße für den Prozeß ist. Sie ist somit vom technischen Prozeß vorgegeben.
Dem Programmierer fällt nun die Aufgabe zu, die statischen Prioritäten für die einzelnen Tasks so zu wählen, daß die vorgegebene Randbedingung, also die Beendigung jeder Task

innerhalb ihrer Zykluszeit, einhalten werden kann. Da die Tasks um so dringender abgearbeitet werden müssen, je kürzer ihre Zykluszeiten sind, ergibt sich die in der folgenden Tabelle aufgelistete Task-Rangfolge.

	Zykluszeit	PEARL Priorität	Laufzeit
Task 1	0,3 s	10	100 ms
Task 2	0,5 s	20	150 ms
Task 3	1,0 s	30	200 ms

Bild 4: System mit 3 Tasks

Alle 3 Tasks werden zyklisch aktiviert, beginnend zum Zeitpunkt $t_a = 0$.

Die Belegung des Prozessors, die sich aus dieser Prioritätenwahl ergibt, ist in Bild 5 dargestellt.

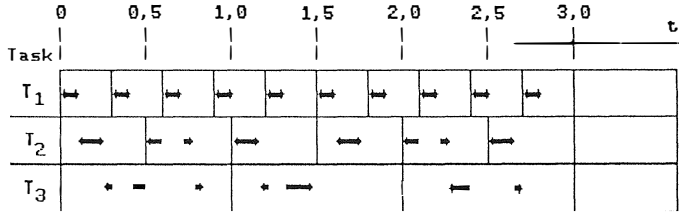


Bild 5: Prozessorbelegung bei prioritätsgesteuerter Prozessorvergabe

Für jede Task sind die Zyklen eingezeichnet und die Zeit, in der sie tatsächlich den Prozessor besitzt.
Man erkennt, daß die Forderung, jede Task muß innerhalb ihrer Zykluszeit beendet sein, erfüllt ist. Hier führt die prioritätsgesteuerte Prozessorvergabe zum Erfolg.
Probleme treten auf, wenn nachträglich eine weitere Task in

das System aufgenommen werden muß. So könnte sich in unserem Beispiel die Notwendigkeit ergeben, um eine feinere Regelung des Prozesses zu ermöglichen, eine weitere Zustandsgröße zu messen und zu verarbeiten. Die Zeitbedingungen für die drei vorhandenen Tasks sollen sich dabei nicht ändern.
Um eine geeignete Priorität für die neue Task bestimmen zu können, müssen dem Programmierer die Prioritäten aller anderen Tasks bekannt sein.
Die Zykluszeit von Task Nr. 4 sei 1,5 s. Da dies die längste der 4 Zykluszeiten ist, legt sie die Vermutung nahe, daß die zugehörige Priorität am geringsten sein muß, die entsprechende Prioritätskennzahl in PEARL also am höchsten. Denkbar ist, z.B.:

	Zykluszeit	PEARL Priorität	Laufzeit
Task 4	1,5 s	40	250 ms

Bild 6: nachträglich ins System aufgenommene Task

Für die Task-Rangfolge spielen dabei natürlich nicht die absoluten Werte, sondern nur die relative Reihenfolge eine Rolle.
Mit dieser Priorität ergibt sich folgende Prozessorbelegung:

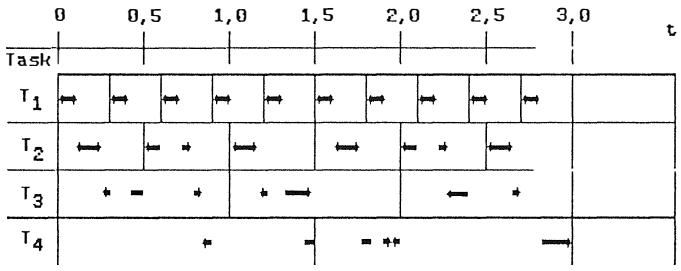


Bild 7: Prozessorbelegung bei prioritätsgesteuerter Prozessorvergabe

Wie man sieht, ist es nicht möglich, die Zeitbedingung für die Task Nr. 4 einzuhalten. Sie kann innerhalb ihres ersten Zyklusses nicht beendet werden.
Dabei tritt hier nicht etwa eine System-Überlastung auf. Die Summe aller Task-Laufzeiten ist gleich der Gesamtzyklus-Zeit von 3 s, d.h. der Prozessor ist maximal ausgelastet. (Die Gesamtzykluszeit ist dabei dadurch definiert, daß nach ihrem Ablauf dieselbe Situation wie zum Ausgangspunkt $t_a = 0$ herrscht.)
Gesamtzykluszeit = 3000 ms
= 10 x 100 ms + 6 x 150 ms + 3 x 200 ms + 2 x 250 ms

Es läßt sich beweisen, daß auch mit einer anderen (statischen) Verteilung der Prioritäten die Zeitbedingung für Task 4 nur auf Kosten einer anderen Task eingehalten werden kann /2/. Hier versagt also eine prioritätsgesteuerte Prozessorvergabe.
Anders sieht es dagegen aus, wenn das Deadline-Scheduling eingesetzt wird. Die Suche nach einer geeigneten Prioritätsverteilung entfällt, das Kriterium für die Prozessorvergabe, die Deadline, ist vom technischen Prozeß bereits vorgegeben.
In PEARL könnte sich z. B. folgende Formulierung des Problems ergeben:

```
GPP M I D S I Z E P E A R L COMPILER VERSION 1.0 MODULE:EXPAMPLE

DATE: 20.10.88 TIME:  8:49:53    PAGE:  1


      1       MODULE (EXPAMPLE);

      3   2   SYSTEM;

      5   4   PROBLEM;

      7   6     MAIN:TASK DUE AFTER 0.1 SEC GLOBAL;
      8           ALL 0.3 SEC ACTIVATE T1
      9           ALL 0.5 SEC ACTIVATE T2 ;
```

```
10            ALL 1.0 SEC ACTIVATE T3 ;
11            ALL 1.5 SEC ACTIVATE T4 ;
12          END;
13
14          T1:TASK DUE AFTER 0.3 SEC GLOBAL;
15          /* Task1 */
16          END;
17
18          T2:TASK DUE AFTER 0.5 SEC GLOBAL;
19          /* Task2 */
20          END;
21
22          T3:TASK DUE AFTER 1.0 SEC GLOBAL;
23          /* Task3 */
24          END;
25
26          T4:TASK DUE AFTER 1.5 SEC GLOBAL;
27          /* Task4 */
28          END;
29
30
31        MODEND;


    KEINE FEHLER

    ENDE DER UEBERSETZUNG
```

Es kommt immer diejenige Task zum Ablauf, deren Deadline am nächsten ist. Damit erhält man folgende Prozessorvergabe.

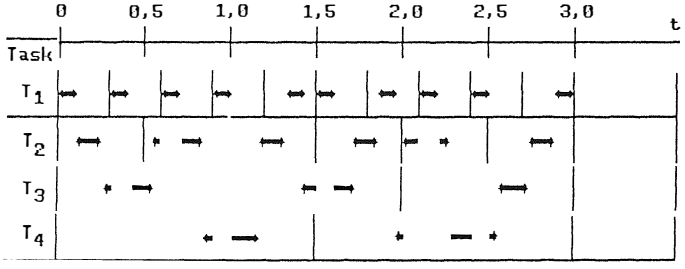


Bild 8: Prozessorbelegung bei antwortzeitgesteuerter Prozessorvergabe

Wie man leicht überprüfen kann, können die Zeitbedingungen sämtlicher Tasks eingehalten werden, jede ist innerhalb ihres Zyklusses beendet. Damit erweist sich hier die antwortzeitgesteuerte Prozessorvergabe der prioritätsgesteuerten überlegen.
Zusammengefaßt ergeben sich folgende Vorteile des Deadline-Schedulings.

4. Vorteile des Deadline-Schedulings

- Problemnähe
  In vielen technischen Prozessen treten Randbedingungen bezüglich der Zeit auf. Die Ergebnisse eines Prozesses, der im Programm durch eine Task repräsentiert wird, sind nur dann noch brauchbar, wenn sie innerhalb eines vorgegebenen Zeitrahmens vorliegen.
  Während bisher diese Zeitschranke in PEARL erst in eine Priorität umgerechnet werden mußte, bietet das neue Sprachelement eine unkomplizierte, direkte Umsetzung der Randbedingung ins Programm an.

- unabhängige Erstellung von Systemkomponenten
  Der Programmierer muß nicht den gesamten Programmkomplex kennen, wie es im Fall der prioritätsgesteuerten Prozessorvergabe der Fall ist. Dort benötigt er die Prioritäten sämtlicher Tasks, um sicherzustellen, daß seine Task höherprior ist als alle Tasks, die weniger dringend abgearbeitet werden müssen und umgekehrt. Im Fall des Deadline-Schedulings wird eine System-Integration bei vorausgehender, unabhängiger Erstellung der Komponenten wesentlich erleichtert.

- Erweiterungs- und Änderungsfreundlichkeit
  Was in der Entwicklungsphase für ein neues System gilt, trifft natürlich auch auf eine spätere System-Erweiterung zu. Will man nachträglich neue Tasks einfügen, so ist bei

einer prioritätsgesteuerten Prozessorvergabe die genaue Kenntnis der Prioritäten aller bereits bestehenden Tasks und ein langwieriges Abstimmen des erweiterten Systems nötig. Dieser Aufwand steigt exponentiell mit der Anzahl der beteiligten Tasks. Je mehr Information für eine Änderung erforderlich ist, desto größer ist natürlich auch das Fehlerrisiko.

Testhilfe durch das Betriebssystem

Im Fall des Deadline-Schedulings kann das Betriebssystem die Einhaltung der Antwortzeiten überwachen. So bietet das Betriebssystem PORTOS der GPP dem Programmierer Informationen über eine eventuelle System-Überlastung an und stellt damit ein wertvolles Hilfsmittel zur Verfügung.

Grundsätzliche Überlegenheit in bestimmten Systemen

Wie im vorausgehenden Beispiel gezeigt wurde, gibt es Systeme, deren Zeitbedingungen mit Hilfe von statischen Prioritäten nicht mehr eingehalten werden können.
Mittels einer dynamischen Prioritätsänderung kann das Ziel zwar erreicht werden, aber auch hier gelten die aufgeführten Nachteile. Das Abstimmen eines Systems über dynamische Prioritätsvergabe erfordert ebenso eine genaue Kenntnis des Gesamtsystems und ist wesentlich komplizierter zu programmieren und zu testen, als es bei antwortzeitgesteuerter Prozessorvergabe erforderlich wäre.

5. Erweiterte Tasksteuerung in PEARL

Das Konzept des Deadline-Schedulings in PEARL schließt nicht aus, daß der Anwender parallel dazu nicht auch die Möglichkeit hat, Tasks mit Prioritäten zu versehen.
Neben Echtzeit-Aufgaben gibt es in Prozeßrechensystemen in der Regel auch Aufgaben, für deren Bearbeitung keine oder nur geringe Zeitbedingungen einzuhalten sind. Als

Beispiele hierfür seien die Ausgabe von Protokollen und die Dateneingabe über eine Tastatur genannt. Für Tasks, die derartige Aufgaben durchführen, kann eine sinnvolle Antwortzeit nur schwer festgelegt werden.
Eine denkbare Lösung, die im Betriebssystem PORTOS der Firma GPP gewählt wurde, ist es, verschiedene Prioritätsklassen einzurichten, nämlich die Klasse der Tasks, die über Prioritäten gesteuert werden, die Klasse jener Tasks, die über Antwortzeiten gesteuert werden und die Klasse der Systemtasks. Dabei haben Systemtasks Vorrang gegenüber antwortzeitgesteuerten Tasks und diese wiederum gegenüber prioritätsgesteuerten Tasks.
Hier besteht z. B. auch die Möglichkeit, eine Anwendertask kurzfristig aus einer der beiden niederen Klassen in die Klasse der Systemtasks hochzuschalten.

- In einer zweiten Ausbaustufe bietet das Deadline-Scheduling dem Anwender noch weitere Möglichkeiten.
So ist es z. B. denkbar, Überlastsituationen mit geeigneten Maßnahmen zu begegnen. Dazu wird jeder Task ein weiterer Parameter, die Laufzeit bzw. eine obere Schranke für die Laufzeit mitgegeben. Nun hat das System schon im voraus die Möglichkeit, Situationen zu erkennen, in denen eine vollständige Abarbeitung aller Tasks bis zu ihrer Deadline nicht mehr möglich ist. Für diesem Fall wurden vom Anwender bestimmte Tasks mit dem Attribut KEEP versehen /1/. Nur derartig gekennzeichnete Tasks laufen in einer Überlastsituation weiter, allen anderen wird der Prozessor entzogen. So wird sichergestellt, daß die wichtigsten Prozesse auch in kritischen Situationen fortgesetzt werden können. Das Deadline-Scheduling bietet somit nicht nur in der Entwicklungsphase durch eine einfache Programmierung von zeitkritischen Tasks Vorteile, sondern auch in der Betriebsphase, wenn z. B. auf Grund von Fehlerfällen eine Überlastsituation auftritt.

Literaturverzeichnis

/1/ Additional PEARL Language Structures for the Implementation of Reliable and Inherently Safe Real-Time Systems
W. A. Halang, R. Henn
Draft for 15th IFAC WORKSHOP ON REAL TIME PROGRAMMING 1988

/2/ Zeigerechte Prozessorzuteilung in einer harten Realzeit-Umgebung
R. Henn
GI - 6. Jahrestagung
Berlin, Heidelberg
Springer-Verlag

/3/ Nutzerhandbuch der antwortzeitgesteuerten Prozessorzuteilung GPP/442/88-2.8
Ausgabe A
H. Piche, GPP
29.02.1988

/4/ PEARL-Übersetzungs- und Ausführungssystem IBM PC-AT, GPP/462/88
C. Hilbert, GPP
Mai 1988

Erzeugung laufzeitoptimaler PEARL-Programme

Karlotto Mangold
ATM Computer GmbH
Konstanz

Zusammenfassung:

An Hand von Programmbeispielen wird dargestellt, wie sowohl durch geeignete Programmierstrategien, als auch durch Implementierung ausgewählter Optimierungsverfahren laufzeitoptimaler Code erzeugt werden kann. Dabei werden folgende Optimierungsmöglichkeiten erläutert:

- Explizite und implizite Identitäts-Spezifikationen
- Vorziehen schleifeninvarianter Ausdrücke
- Erkennen nicht reenterbarer Prozeduren
- Unvollständige Berechnung logischer Ausdrücke
- Optimierung von Teilausdrücken
- Schleifenindizierung durch voreinstellbare und fortschaltbare Anteile
- Rechnerabhängige Optimierungen

An einem konkreten Beispiel und entsprechenden Laufzeitmessungen werden die Effizienzsteigerungen dargestellt, die mit diesen Maßnahmen erreicht werden konnten.

Optimierungsmöglichkeiten

Zunächst sollen hier einige Optimierungsmöglichkeiten in ihrer Anwendung und Wirkung erläutert werden.

1. Verwendung von Identitäets-Spezifikationen

In PEARL werden bei der Deklaration von Objekten Zugriffsfunktionen auf diese Objekte erzeugt. Diese Zugrifsfunktionen sind ihrer Art nach dynamisch, das heißt, bei jedem Zugriff wird der Wert der Zugriffsfunktion neu berechnet und dann mit diesem Wert auf das entsprechende Objekt zugegriffen. Die Berechnung einer solchen Zugriffsfunktion kann recht aufwendig werden, besonders bei Indizierung mehrdimensionaler Felder oder bei Strukturselektionen.

Wird eine solche Zugriffsfunktion mehrfach benötigt und ist bekannt, daß alle "Parameter" dieser Zugriffsfunktion dabei denselben Wert haben, so empfiehlt es, sich auf Quellebene eine Identitätsspezifikation einzuführen. Dabei wird der Wert der Zugriffsfunktion nur einmal berechnet und dann bei der Verwendung der Indentitäts-Spezifikation direkt der bereits berechnete Wert zum Zugriff verwendet.

2. Vorziehen schleifeninvarianter Ausdrücke

Betrachtet man die Ausführungszeit eines Programms und untersucht dabei, in welchen Programmteilen die meiste Zeit verbraucht wird, so zeigen sich die Schleifenrümpfe als besonders rechenintensiv Deshalb sind Optimierun-

gen, die in Schleifenrümpfen ansetzen, besonders wirkungsvoll. Es ist deshalb besonders lohnend, alle Ausdrücke in Schleifenrümpfen darauf zu untersuchen, ob sie von der Schleifenvariable unabhängig sind. Solche unabhängigen Ausdrücke oder Teilausdrücke können aus dem Schleifenrumpf herausgezogen werden und müssen dann nur noch einmal vor Betreten der Schleife statt n-mal in der Schleife berechnet werden. Falls ein Compiler diese Optimierung nicht durchführt, so kann dieser Effekt durch "Umsortieren" der Quellzeilen erzielt werden.

3. Erkennen nicht-reenterbarer Prozeduren

In Realzeitsystemen mit parallelen Aktivitäten kann jede außerhalb einer Task deklarierte Prozedur von mehreren Tasks quasiparallel aufgerufen werden. Dabei können natürlich innerhalb des Prozedurcodes Überholvorgänge nicht ausgeschlossen werden. In der Praxis bedeutet dies, daß der Prozedurcode zwar sharebar nur einmal vorhanden sein muß, daß aber die lokalen Daten der Prozedur für jeden Prozeduraufruf neu angelegt werden müssen. Dies erfolgt am zweckmäßigsten im dynamischen Datenbereich der aufrufenden Task. Leider führt diese Datenorganisation aber dazu, daß alle diese Daten nicht direkt adressierbar sind, sondern über ein taskspezifisch gesetztes Basisregister indirekt adressiert werden. Kann nun für eine Prozedur, die außerhalb einer Task deklariert ist, sichergestellt werden,daß sie weder quasiparallel von verschiedenen Tasks noch von sich selbst direkt oder indirekt rekursiv aufgerufen wird, so brauchen die prozedurlokalen Daten nur einmal prozedurspezifisch angelegt zu werden. Diese Daten können dann von der Prozedur direkt adressiert werden, so daß bei jedem derartigen Datenzugriff eine indirekte Adressierungsstufe entfallen kann.

4. Unvollständige Berechnung logischer Ausdrücke

Komplexe logische Ausdrücke lassen sich mit Methoden elementarer Logik in sogenannnte Normalformen umwandeln. Dabei entstehen Folgen von Teilausdrücken, die alle entweder mit AND oder OR verknüpft werden. Solche mit AND verknüpften Folgen von Teilausdrücken müssen nur so weit ausgewertet werden, bis ein Teilausdruck den Wert "false" liefert. Damit ist der Wert des gesamten Ausdrucks "false". Die Berechnung weiterer Teilausdrücke ändert an diesem Wert nichts. Mit OR verknüpfte Folgen von Teilausdrücken müssen stattdessen nur so weit ausgewertet werden, bis ein Teilausdruck den Wert "true" liefert. Damit ist der Wert des gesamten Ausdrucks "true". Nutzt man bei der Umformung noch die Kommutativität so aus, daß die Teilausdrücke von einfachen zu komplizierten (rechenintensiven) sortiert sind, so können mit dieser Optimierungsstrategie z.T. erhebliche Laufzeitgewinne erzielt werden. Werden logische Ausdrücke nur in Bedingungen verwendet und nicht logischen Variablen zugewiesen, so kann dieser Effekt unabhängig von der Implementierung in einem Compiler auch durch geeignete IF-Kaskaden anstelle eines IFs mit komplizierter Bedingung auf Quellebene erzielt werden.

5. Teilausdrücke wiederverwenden

Durch Einführen von Hilfsvariablen, denen der Wert eines (Teil-) Ausdrucks einmal zugewiesen wird, und deren Wert mehrfach weiterverwendet wird, kann die Häufigkeit der Berechnung dieser Teilausdrücke reduziert werden. Dies führt besonders dann zu deutlichen Laufzeiteinsparungen, wenn in der Berechnung des Teilausdrucks Funktionsaufrufe ausgeführt werden müssen.

6. Indizierung in Schleifen durch voreinstellbare und fortschaltbare Anteile

Betrachtet man die Adreßberechnung eines mehrdimensionalen arrays, so ergibt sich die Adresse eines Feldelements aus der Adresse des ersten Elements (Feldanfangsadresse) und einem offset, der von den aktuellen Werten der Indices und den Kantenlängen des arrays in den einzelnen Dimensionen abhängt. Dieser offset berechnet sich im allgemeinem Fall nach einem Hornerschema. Bei einem n-dimensionalen array sind das bei jeder Adreßberechnung n Multiplikationen und n Additionen. Die Berechnung erfolgt nach der Formel: $\sum_i$ (Index$_i$ - untere Grenze$_i$)*Stride$_{i-1}$, wobei Stride$_i$ die Elementanzahl der i-ten Dimension und Stride$_0$ die Länge eines Elementes in Bytes ist. Feldelemente werden meist in Schleifen bearbeitet, dabei sind die Indices oft nur linear von der Schleifenvariablen abhängig sind. Das heißt, zwei Feldelemente in aufeinanderfolgenden Schleifendurchläufen haben eine konstante Adreßdifferenz. Aus den deklarierten Dimensionen des arrays läßt sich diese Adreßdifferenz zur Compilezeit ermitteln, oder bei dynamischen arrays zumindest eine Codefolge erzeugen, die vor dem Eintritt in die Schleife diese Fortschaltgröße berechnet. Wird nun vor der Schleife auch noch eine geeignete virtuelle Anfangsadresse, die sogenannte Voreinstellung, bestimmt, so kann die bei jedem Schleifendurchlauf mindestens einmal erforderliche Adreßberechnung statt mit Hilfe des Hornerschemas durch eine einmalige Addition der Fortschaltgröße ersetzt werden.

die Variablen im Speicher verändert werden.

7. Rechnerabhängige Optimierungen

Abhängig von der jeweiligen Hardware-Struktur (verfügbare Register, vorhandene Befehle) können vom jeweiligen Code-Generator maschinenabhängige Optimierungen durchgeführt werden. Falls beispielsweise hardwaremäßig "Kurzsprungbefehle" vorhanden sind, so kann bei oder nach der Code-Generierung geprüft werden, welche (bedingten) Sprungbefehle durch die entsprechenden Kurzsprünge ersetzt werden können, weil das Sprungziel innerhalb der Reichweite des Kurzsprungs liegt. Damit wird sowohl der Code, als auch die Ausführungszeit verkürzt. Obwohl dies bei jedem Sprungbefehl nur zwei Byte Codeverkürzung und wenige Mikrosekunden Laufzeitverbesserung bringt, lohnt sich diese Maßnahme, da sie häufig bei implizit generierten Sprungbefehlen wirkt. Während die hoffentlich seltenen GOTOs meist Sprungziele haben, die außerhalb der Reichweite von Kurzsprüngen liegen, lohnt sich die Optimierung bei Sprungbefehlen, die vom Compiler zur Schleifenorganisation, zur Realisierung von IF- und CASE-Konstrukten oder bei bedingten Anweisungen generiert werden.

Eine weitere maschinenabhängige Optimierung ist die Identifizierung von Registerinhalten und deren Wiederverwendung. Das Hauptproblem liegt hierbei in der Identifizierung von Ausdrücken und dem Erkennen, wie lange die Faktoren dieser Ausdrücke unverändert bleiben. Eine solche Registeroptimierung kann zum Beispiel dazu führen, daß in "kleinen" Schleifen Variablen nur in Registern geführt werden und erst nach Verlassen der Schleife abgespeichert werden. Selbst bei globalen Variablen, auf die möglicherweise quasiparallel zugegriffen wird, ist dies möglich, da undefiniert ist, wann die Variablen im Speicher verändert werden.

Konsequenzen dieser Optimierungsmöglichkeiten

Wie können nun diese Möglichkeiten zur Einsparung von Laufzeit genutzt werden?
Zumindest die ersten fünf der oben vorgestellten Möglichkeiten können von einem gut ausgebildeten Programmierer bei Beachtung entsprechender Programmierkonventionen direkt genutzt werden, unabhängig davon, wie gut ein spezieller Compiler optimiert.
Leider zeigt es sich aber in der Praxis, daß die dafür notwendigen Quelländerungen oft die Lesbarkeit der Programme vermindern oder auf Grund von Terminzwängen die Optimierungen zunächst gar nicht implementiert werden. Eine spätere Analyse und Quelländerung eines dann bereits getesteten und integrierten Programms wird häufig aus Risikogründen gescheut.
Es kommt hinzu, daß einige Optimierungen (z.B. Kurzsprungoptimierung) gar nicht auf Quellebene eingebaut werden können. Deshalb ist es aus Projektsicht erforderlich, maschinelle Hilfsmittel zur Optimierung zu haben.
An einem Beispielprogramm, das nur Demonstrationszwecken dient, und hoffentlich keine Ähnlichkeit mit einem realen Programm hat, sollen die Effekte einiger dieser Optimierungen gezeigt werden.

Demonstrations-Programm

Im beiliegenden Demonstrationsprogramm werden drei der obengenannten Optimierungsmöglichkeiten dargestellt. Die Zugriffsoptimierung durch Identitätsspezifikationen, die Optimierung von Teilausdrücken und die Optimie-

rung des indizierten Zugriffs innerhalb einer Schleife gezeigt.

In den Zeilen 31 bis 43 (Programmteil I) wird ein zweidimensionales Feld initialisiert und anschließend der jeweilige Initialwert quadriert. Derselbe Vorgang wird in den Zeilen 50 bis 63 (Programmteil II) wiederholt. Hier tritt jedoch das in Zeile 60 durch Identitätsspezifikation einmal pro Schleifendurchlauf definierte Objekt AR2IJ an die Stelle des viermal benötigten Feldelements AR2(I,J).

Im Programmteil III wird in den Zeilen 71 bis 89 der Teilausdruck A+1 insgesamt siebenmal verwendet. Dabei ist zu berücksichtigen, daß bei Programmverzweigungen (IF- oder CASE-Anweisungen) dynamisch immer nur ein Zweig durchlaufen wird und deshalb nach Ende einer solchen Verzweigung der aktuelle Wert des Ausdrucks von der durchlaufenen Alternative abhängen kann.

Da die Anweisungen in der Meßschleife (Zeile 73 bis 88) nicht vom Schleifenindex I abhängen, sind diese Anweisungen im Programmteil IV (Zeile 96 bis 114) vor die Schleife gezogen. Damit ist diese Schleife leer (Zeile 112 bis 116) und muß eigentlich gar nicht mehr ausgeführt werden. Um diesen Effekt bei einem entsprechend optimierenden Compiler auszuschließen, ist im Programmteil V (Zeile 121 bis 140) der Teilausdruck A+1 durch den von der Laufvariablen abhängigen Ausdruck A+K (mit K= I REM 10) ersetzt worden.

Im Programmteil VI (Zeile 147 bis 154) werden in einer IF-Anweisung zwei logische Ausdrücke mit AND verknüpft, während dies im Programmteil VII (Zeile 160 bis 170) in zwei geschachtelte, logisch äquivalente IF-Anweisungen aufgeteilt ist.

Laufzeitmessungen

Das Demonstrationsprogramm wurde einmal mit und einmal ohne Optimierung übersetzt. Die so erzeugten Objekte wurden gebunden und ausgeführt. Die beiden Ergebnisausdrucke befinden sich im Anhang, hinter dem Programmlisting. Bei den ermittelten Meßwerten ist zu berücksichtigen, daß die Meßgenauigkeit eine Viertelsekunde beträgt.
Stellt man lediglich die Laufzeiten der optimierten und der nicht optimierten Fassung gegenüber, so ergibt sich folgende Tabelle:

Programmteil	nicht optimiert	optimiert	prozentualer Gewinn	Beschleunigungs-Faktor
I	32.0 sec	5.75 sec	82,0 %	5,5
II	11.0 sec	5.25 sec	52,2 %	2,1
III	4.5 sec	2.75 sec	38,9 %	1,6
IV	0.25 sec	0.25 sec	0 %	1,0
V	5.0 sec	3.00 sec	40,0 %	1,6
VI	4.25 sec	0.75 sec	82,3 %	5,6
VII	0.5 sec	0.50 sec	0 %	1,0

Betrachtet man nicht nur die im Compiler implementierten Optimierungsstrategien, sondern bezieht man zusätzlich die Optimierungen auf Quellebene mit ein, so ergibt sich bei der Indizierung in der Schleife, daß die Verwendung der Identitätsspezifikation die Laufzeit der Schleife auf ein Drittel

verkürzt. Hat man jedoch einen entsprechend optimierenden Compiler, so erzeugt dieser aus beiden Programmteilen praktisch gleich guten Objekt-Code. Diese Feststellung wird durch Vergleich der Laufzeiten in den Programmteilen VI und VII praktisch bestätigt. Umgekehrt bedeutet dies aber, daß die Optimierung um so mehr bringt, je schlechter die Ausgangasbasis ist.
Die Leerschleife, die im Programmteil IV entsteht, ist im Rahmen der Meßgenauigkeit des Demonstrationsprogramms praktisch nicht meßbar

Schlußfolgerungen

Es wurde gezeigt, daß ein optimierender Compiler durch die damit erzielbaren Laufzeitgewinne erheblich zur Erzeugung laufzeitoptimalen Codes beitragen kann. Andererseits wurde aber auch dargestellt, daß durch entsprechende Programmiererausbildung und Programmierdisziplin ein beträchtlicher Anteil dieser Laufzeitgewinne erreicht werden kann.
Da zur Software-Entwicklung in immer größerem Maße Tools eingesetzt werden, ist ein optimierender Compiler ein wichtiges Tool, dessen Einsatz sich im Hinblick auf die Objektqualität auszahlt.

Anschrift des Verfassers:
K. Mangold, ATM Computer GmbH, Bücklestraße 1-5, 7750 Konstanz

Anhang: Programmlisting und Ergebnisse

```
iTM 8030-PEARL   (5.54)          05.10.88  DEM000     S.    1


    1  MODULE;
    2  /*$$ZONE,ZONET,2*/
    3  SYSTEM;
    4  DIAL:SPL001;
    5  PROBLEM;
    6    SPC DIAL DATION INOUT ALPHIC DIM (,) TFU MAX CONTROL(ALL) GLOBAL;
    7
    8
    9  T:TASK;
   10    /* Demonstrationsprogramm fuer PEARL-Optimierungen */
   11
   12    DCL (TIMA,TIME) CLOCK; /* zur Zeiterfassung */
   13    DCL ANZS FIXED INIT(1000), /* Anzahl Schleifendurchlaeufe */
   14        ANZT FIXED INIT(20000),
   15        ANZIF FIXED INIT (32000);
   16    DCL (A,B) FIXED INIT(0,1);
   17    DCL (N,M) FIXED INIT(9), K FIXED;
   18    DCL (FIX1,FIX2) INV FIXED INIT(2,1);
   19    DCL AR1(0:N) FIXED;
   20    DCL AR2(0:N,0:M) FIXED;
   21
   22    PROT: PROC(TEXT INV CHAR() IDENT);
   23      /* Ausgabe der gemessenen Laufzeit */
   24      PUT TEXT,(TIME-TIMA) TO DIAL BY SKIP,A,D(24,2),SKIP;
   25    END /* PROT */;
   26
   27    OPEN DIAL;
   28    PUT 'Start Testprogramm  ****' TO DIAL BY SKIP,A,SKIP,SKIP;
   29    /* Schleifen-Optimierung */
   30    TIMA := NOW;
   31    FOR I TO ANZS
   32    REPEAT
   33      /* Initialisierung einer n*m Matrix
   34         und Quadrierung der Matrix-Elemente */
   35      FOR I FROM 0 TO N
   36      REPEAT
   37        FOR J FROM 0 TO M
   38        REPEAT
   39          AR2(I,J):= 10*I+J;
   40          AR2(I,J):= AR2(I,J) * AR2(I,J);
   41        END; /* J */
   42      END; /* I */
   43    END    /* ANZS */;
   44    TIME := NOW;
   45    PUT AR2 TO DIAL BY SKIP,(10)((10)F(5),SKIP);
   46
   47    PROT('Schleife:          ');
   48    /* Schleifen-Optimierung mit Identitaetsspezifikation */
   49    TIMA := NOW;
   50    FOR I TO ANZS
   51    REPEAT
   52      /* Initialisierung einer n*m Matrix
   53         und Quadrierung der Matrix-Elemente */
   54      FOR I FROM 0 TO N
```

I

```
 55       REPEAT
 56         FOR J FROM 0 TO M
 57         REPEAT
 58           SPC AR2IJ FIXED IDENT (AR2(I,J));
 59           AR2IJ:= 10*I+J;
 60           AR2IJ:= AR2IJ * AR2IJ;
 61         END; /* J */
 62       END; /* I */
 63     END     /* ANZS */;
 64     TIME := NOW;
 65     PROT('Schleife  mit Ident-Spezifikation:');
 66
 67
 68
 69     /* Teilausdrucksoptimierung */
 70     TIMA := NOW;
 71     FOR I TO ANZT
 72     REPEAT
 73       AR1(A+1) := AR1(A+1)+1;
 74       IF A < B
 75       THEN
 76         AR1(A+1) := 1;
 77       ELSE
 78         AR1(A+1) := 2;
 79       FIN /* A < B */;
 80
 81       CASE A
 82       ALT
 83         AR1(A+1) := 1;
 84       ALT
 85         AR1(A+1) := 2;
 86       FIN;
 87       AR1(A+1) := 3;
 88       A := 0;
 89     END  /*ANZT */;
 90     TIME := NOW;
 91     PROT('Teilausdruck:     ');
 92
 93
 94     /* Vorziehen konstanter Teile aus Schleife */;
 95     TIMA:= NOW;
 96     AR1(A+1) := AR1(A+1)+1;
 97     IF A < B
 98     THEN
 99       AR1(A+1) := 1;
100     ELSE
101       AR1(A+1) := 2;
102     FIN /* A < B */;
103
104     CASE A
105     ALT
106       AR1(A+1) := 1;
107     ALT
108       AR1(A+1) := 2;
109     FIN;
```

```
110     AR1(A+1) := 3;
111     A := 0;
112     FOR I TO ANZT
113     REPEAT
114     END  /*ANZT */;
115     TIME := NOW;
116     PROT('vorgezogene Konstanten:');
117
118
119     /* variable Anteile im Teilausdruck */;
120     TIMA := NOW;
121     FOR I TO ANZT
122     REPEAT
123       K := I REM 10;
124       AR1(A+K) := AR1(A+K)+1;
125       IF A < B
126       THEN
127         AR1(A+K) := 1;
128       ELSE
129         AR1(A+K) := 2;
130       FIN /* A < B */;
131
132       CASE A
133       ALT
134         AR1(A+K) := 1;
135       ALT
136         AR1(A+K) := 2;
137       FIN;
138       AR1(A+K) := 3;
139       A := 0;
140     END  /*ANZT */;
141     TIME := NOW;
142
143     PROT('Teilausdruck  schleifenabh.:');
144
145     /* Optimierung von IF-Kaskaden */;
146     TIMA:= NOW;
147     FOR I TO ANZIF
148     REPEAT
149
150       IF A > B AND AR1(A) < AR1(B)
151       THEN
152         AR1(A) := AR1(B);
153       FIN;
154     END  /* ANZIF */;
155     TIME := NOW;
156     PROT('IF-Kaskade:  ')
157
158     /* Optimierung von IF-Kaskaden auf Quellebene */;
159     TIMA:= NOW;
160     FOR I TO ANZIF
161     REPEAT
162
163       IF A > B
164       THEN
```

```
165          IF  AR1(A) < AR1(B)
166          THEN
167            AR1(A) := AR1(B);
168          FIN;
169        FIN;
170      END  /* ANZIF */;
171      TIME := NOW;
172      PROT('IF-Kaskade (quellopt):');
173      PUT ' Ende Testprogramm  ****' TO DIAL BY SKIP,A,SKIP;
174
175      END /* T */;
176    MODEND;
```

VII

Bindemodul-Liste (ATM 8030) :

Name	Bin-Mo	Datei	Code	Daten(stat/dyn)			LBmin
Systemteil	DEM001	DEM001		0004	0004	0000	
Moduldaten	DEM002	DEM002		0026	0026	0000	
T	DEM052	DEM052	0A80			00A0	0C00
Modultask	DEM051	DEM051	00A6			001A	00A6

Zuordnung Zeile/rel. Codeadresse fuer Modul T - DEM052

9 0000	13 0016	14 001E	15 0026	16 002E
17 003A	19 0044	20 0064	22 0090	24 00A4
25 012C	27 0132	28 013C	30 01A2	31 01AA
35 01CA	37 01EA	39 020A	40 0226	41 0266
42 026A	43 026E	44 0272	45 027A	47 02FA
49 0316	50 031E	54 033E	56 035E	58 037E
59 0390	60 03A2	61 03AA	62 03AE	63 03B2
64 03B6	65 03BE	70 03DA	71 03E2	73 0402
74 042E	76 0438	77 044E	78 0452	81 0468
83 0480	85 0498	87 04AE	88 04C4	89 04CA
90 04CE	91 04D6	95 04F2	96 04FA	97 0526
99 0530	100 0546	101 0548	104 055E	106 0576
108 058E	110 05A4	111 05BA	112 05C0	114 05E0
115 05E2	116 05EA	120 0606	121 060E	123 062E
124 0642	125 0672	127 067C	128 0694	129 0696
132 06AE	134 06C6	136 06E0	138 06F8	139 0710
140 0716	141 071A	143 0722	146 073E	147 0746
150 0766	152 07AC	154 07CE	155 07D2	156 07DA
159 07F6	160 07FE	163 081E	165 082A	167 0850
170 0872	171 0876	172 087E	173 089A	175 08F2

Zuordnung Zeile/rel. Codeadresse fuer Modul MTASK - DEM051

5 0004 9 0026 176 0038

Uebersetzungszeit= 0 Min 58 Sek

Ben. Parameter
ZF: 30 BL: 25 TY: 44 AS: 11 SR: 0 PR: 6 DT: 1 FL: 18
FKT:BXP

nicht optimiert

```
 Start Testprogramm  ****


    0    1    4    9   16   25   36   49   64   81
  100  121  144  169  196  225  256  289  324  361
  400  441  484  529  576  625  676  729  784  841
  900  961 1024 1089 1156 1225 1296 1369 1444 1521
 1600 1681 1764 1849 1936 2025 2116 2209 2304 2401
 2500 2601 2704 2809 2916 3025 3136 3249 3364 3481
 3600 3721 3844 3969 4096 4225 4356 4489 4624 4761
 4900 5041 5184 5329 5476 5625 5776 5929 6084 6241
 6400 6561 6724 6889 7056 7225 7396 7569 7744 7921
 8100 8281 8464 8649 8836 9025 9216 9409 9604 9801

 Schleife:            0 HRS 00 MIN 32.00 SEC

 Schleife  mit Ident-Spezifikation:  0 HRS 00 MIN 11.00 SEC

 Teilausdruck:       0 HRS 00 MIN 04.50 SEC

 vorgezogene Konstanten:  0 HRS 00 MIN 00.25 SEC

 Teilausdruck  schleifenabh.:  0 HRS 00 MIN 05.00 SEC

 IF-Kaskade:     0 HRS 00 MIN 04.25 SEC

 IF-Kaskade (quellopt):  0 HRS 00 MIN 00.50 SEC

  Ende Testprogramm  ****
```

optimiert

```
Start Testprogramm  ****


    0    1    4    9   16   25   36   49   64   81
  100  121  144  169  196  225  256  289  324  361
  400  441  484  529  576  625  676  729  784  841
  900  961 1024 1089 1156 1225 1296 1369 1444 1521
 1600 1681 1764 1849 1936 2025 2116 2209 2304 2401
 2500 2601 2704 2809 2916 3025 3136 3249 3364 3481
 3600 3721 3844 3969 4096 4225 4356 4489 4624 4761
 4900 5041 5184 5329 5476 5625 5776 5929 6084 6241
 6400 6561 6724 6889 7056 7225 7396 7569 7744 7921
 8100 8281 8464 8649 8836 9025 9216 9409 9604 9801

Schleife:            0 HRS 00 MIN 05.75 SEC

Schleife  mit Ident-Spezifikation:  0 HRS 00 MIN 05.25 SEC

Teilausdruck:       0 HRS 00 MIN 02.75 SEC

vorgezogene Konstanten:  0 HRS 00 MIN 00.25 SEC

Teilausdruck  schleifenabh.:  0 HRS 00 MIN 03.00 SEC

IF-Kaskade:     0 HRS 00 MIN 00.75 SEC

IF-Kaskade (quellopt):  0 HRS 00 MIN 00.50 SEC

 Ende Testprogramm  ****
```

Dynamische Generierung von parallelen Ablaufstrukturen

Holger Herzog
Siemens AG
Zentrale Forschung und Entwicklung
ZTI SOF 41
Otto-Hahn-Ring 6, 8000 München 83

Verteilte Systeme (VS) haben eine weite Verbreitung in Realzeitanwendungen gefunden. Gegenüber zentral organisierten Rechnerarchitekturen bieten sie eine Vielzahl von Vorteilen, die sich aus den in VS möglichen Transparenzaspekten wie z.B. der Leistungs-, der Fehler- und der Ortstransparenz herleiten. Zur Nutzung einer Reihe dieser Transparenzaspekte ist es notwendig Aktionen parallel auszuführen. Deshalb wird für VS und damit implizit auch für Realzeitanwendungen eine Entwicklungsunterstützung von parallelen Ablaufstrukturen benötigt. In Kapitel 2 dieser Arbeit werden Ansätze zur Entwurfsunterstützung von Softwaresystemen auf der Basis von Graphen kurz vorgestellt. Diese Ansätze werden in Kapitel 3 mit der Theorie der Graph-Grammatiken zu einem Konzept zur Entwicklungsunterstützung von parallelen Ablaufstrukturen verknüpft. An einem Anwendungsbeispiel wird demonstriert, daß dadurch eine nahezu durchgängige Entwicklungsunterstützung möglich wird.

1. Motivation

Parallele Ablaufstrukturen sind im Vergleich zu sequentiellen Ablaufstrukturen weitaus schwerer zu verstehen, zu verifizieren, zu testen und zu warten. Obwohl eine Reihe von Werkzeugen zur Entwurfsunterstützung /Dä 87/, zur Testunterstützung /Ga 85/ und zur Untersuchung der Korrektheit und Lebendigkeit von parallelen Systemen /Rö 86/ vorgeschlagen wurden, ist der erreichte Status unzureichend /Ka 87/. Die Komplexität des Entwicklungsprozesses paralleler Ablaufstrukturen verlangt nach einer möglichst durchgängigen Unterstützung, wobei aufgrund des hohen Schwierigkeitsgrades die Systementwicklung auf formalen Konzepten und Methoden aufsetzen sollte /Sch 86/.
Graphbasierte Modelle haben zur Darstellung von parallelen Abläufen eine weite Verbreitung gefunden. Insbesondere aufbauend auf Petri-Netzen wurden darüber hinaus Werkzeuge zur Entwurfsunterstützung entwickelt. Kapitel 2 stellt kurz einige Ansätze vor. In Kapitel 3 wird ein Konzept zur Unterstützung der Entwicklung von parallelen Ablaufstrukturen beschrieben, das die Ansätze aus Kapitel 2 mit der Theorie der Graph-Grammatiken verbindet. Die Umsetzung dieses Konzeptes in Werkzeuge demonstriert Abschnitt 3.3.

2. Systementwicklungsunterstützung durch Graphen

Die Entwicklung der Theorie der Parallelverarbeitung wurde begleitet durch die Entwicklung von graphbasierten Modellen zur Repräsentation von asynchronen nebenläufigen Aktionen. Die Knoten der Graphen stellen dabei einen Zustand, ein Ereignis oder die Ausführbarkeit einer Operation dar. Die Kanten modellieren Präzedenz- oder andere Abhängigkeitsbeziehungen zwischen den Knoten, d.h. sie können u.a. die Abarbeitungsreihenfolge in einem System aufzeigen. Der Gedanke - Graphen zur Entwicklungsunterstützung einzusetzen - lag deshalb nahe.
Es sind zwei Einsatzarten von Graphen zu unterscheiden:

- Graphen als direkte Benutzerschnittstelle zur visiuellen Darstellung komplexer Abläufe und Strukturen und
- Graphen als Datenstruktur zur internen Repräsentation komplexer Abläufe und Strukturen.

Die ersten Ansätze der Anwendung von Graphen beschränkten sich auf die Unterstützung eines einzigen Spezialgebietes (Entwurf, Beschreibung, Simulation) und hatten kaum eine formale Fundierung /Go 83/. Insofern war beim Systementwicklungsprozeß ein Wechsel zwischen den Modellierungskonzepten notwendig, und es

konnte kaum eine Gültigkeitsprüfung durchgeführt werden. Beide Faktoren stellen mögliche Fehlerquellen dar. Neuere Konzepte und Werkzeuge bauen auf Petri–Netzen /Dä 87//Rö 86//Re 83/ oder der allg. Netztheorie /Wi 86/ auf, um so die durch die formalen Grundlagen gegebenen Analysemöglichkeiten ausnutzen zu können. Auch sie sind allerdings häufig nur für einen Teil des Softwareentwicklungsprozesses konzipiert.

Die Methode ISAC /Wi 86/ unterstützt die Beschreibung und Analyse von informationsverarbeitenden Systemen. Ihr primäres Einsatzgebiet liegt somit in der ersten Phase der Softwareentwicklung – der Problemanalyse. Entsprechend einem top–down–Vorgehen beschreibt der Entwickler für jede Abstraktionsebene die relevanten Aspekte durch zweisortige Netze. Die Netze bestehen aus Aktivitäten und Informationen, die durch gerichtete Kanten verbunden werden. Beide Netzelemente können verfeinert werden, so daß sich ein gesamtes ISAC–Dokument als hierachische Struktur von zweisortigen Netzen darstellt. Insofern weist ISAC eine gewisse Ähnlichkeit mit der in /Re 83/ entwicklelten Methode zur Anforderungsdefinition und zum Systementwurf durch Petri–Netze auf.

Funktionsnetze /Go 83/ versuchen beide Konzepte miteinander zu verbinden, um so die Vorteile beider Modelle – die Modellierung dynamischer Abläufe bei Petri–Netzen und die Möglichkeit auf einer Menge von Abstraktionsebenen zu arbeiten bei ISAC – miteinander zu verbinden. Funktionsnetze erweitern die Kanal/Instanzen–Interpretation im Rahmen von Petri–Netzen u.a. indem Instanzen, Kanäle und Systembeziehungen in Klassen eingeteilt werden, denen je eine spezielle Bedeutung zugeordnet wird. Beispielsweise wird neben dem in Petri–Netzen gegebenen 'und'–Schaltverhalten ein partielles Schalten eingeführt. Im Gegensatz zu den o.a. Methoden wurden aufbauend auf Funktionsnetze auch Werkzeuge vorgestellt, die die Softwareentwicklungsphasen Implementierung und Test unterstützen /GT 86/.

3. Entwicklungsunterstützung von parallelen Ablaufstrukturen

3.1 Parallele Ablaufstrukturen

Bei der Implementierung der Steuerung paralleler Abläufe sind prinzipiell drei Möglichkeiten zu unterscheiden:

- die Realisierung als ein kommunizierendes Mehrprozeßsystem
- die Realisierung als prozeßinterne Aktivitätsverwaltung, wie sie aus dem Transaktionssystem CICS von IBM und dem Datenbanksystem SESAM von Siemens bekannt ist, und

- die Realisierung als Mehrprozeßsystem, bei dem jeder Prozeß intern eine Menge von Aktivitäten verwaltet.

Die Entscheidung zugunsten einer der Realisierungsmöglichkeiten wird durch eine Reihe von Faktoren beeinflußt. Je nach Betriebssystem gibt es eine unterschiedliche Obergrenze für gleichzeitig aktive Prozesse. Ist der Grad der möglichen Nebenläufigkeit größer als diese Obergrenze, so ist entweder die zweite oder dritte Alternative zu wählen oder der parallel mögliche Ablauf muß sequentialisiert werden, wobei auch eine prozeßinterne Aktivitätsverwaltung auf einem Einprozessorsystem eine Sequentialisierung darstellt. In Abschnitt 3.3 wird an einem Beispiel gezeigt, daß dies für ein Client/Server–System nicht in gleichem Sinne gelten muß. Ein weiterer durch das zugrundeliegende Betriebssystem bestimmter Faktor sind die Kosten für einen Prozeßwechsel, die äquivalent der Ausführungszeit von 5000 Befehlen bei Großrechenanlagen wie IBM/370 und mit 10–20 Befehlen beim System Embos von ELXSI /Ol 85/ angegeben werden. Ein dritter Faktor ist die Komplexität der Fehlerbehandlung, d.h. die Maßnahmen eine aktuelle Bearbeitung abzubrechen und auf einem definierten Sicherungszustand zurückzukehren. Insbesondere bei Mehrprozeßsystemen bedarf es zusätzlicher Strukturierungskonzepte, um eine adäquate Fehlerbehandlung zu erreichen /CR 86/.

3.2 Entwicklungsunterstützung durch Graph–Grammatiken

Die Entwurfsunterstützung der parallelen Ablaufstrukturen durch graphbasierte Werkzeuge liegt aufgrund von Kapitel 2 nahe. Im Gegensatz zu den sehr allgemein gehaltenen Konzepten aus Kap. 2 weisen parallele Ablaufstrukturen innerhalb festgelegter Anwendungsklassen wiederkehrende Strukturen auf, d.h. daß sich die Abhängigkeitsbeziehungen zwischen den parallel ausführbaren Aktivitäten durch eine Menge von Regeln beschreiben lassen.
Mit Hilfe von Graph–Grammatiken /Na 78/ ist es möglich, Strukturen in einem Graphen aufzubauen und dann im weiteren diese Strukturen als ein Objekt zu modifizieren. Die einzelnen Graphproduktionen modellieren dabei je ein bestimmtes Strukturmerkmal, das durch die die Abhängigkeitsbeziehungen darstellenden Kanten und durch eine mit einer Bedeutung versehenen Knotenmarkierung modelliert wird.

In dieser Arbeit werden Graphen als zentrale Datenstruktur zur Repräsentation der internen Ablaufstruktur bei einer prozeßinternen Aktivitätsverwaltung und als Basis einer automatischen Generierung eines kooperierenden Prozeßsystems vorgeschlagen. Dadurch wird es möglich durch Spezifikation einer Menge von Graphproduktionen eine Datenstruktur zu generieren, aus der parallele Ablaufstrukturen

abgeleitet werden können. Eine solche Eigenschaftsspezifikation paralleler Strukturen abstrahiert von der algorithmischen Beschreibung der Ablaufstruktur und von den benötigten Synchronisationsmechanismen. Dem Benutzer kann – entsprechend den Ansätzen in Kap. 2 – zur Unterstützung des interaktiven Entwurfs eine graphische Benutzeroberfläche angeboten werden. In Abschnitt 3.3 wird gezeigt, daß es Anwendungsfälle gibt, für die eine sprachliche Spezifikation geeigneter ist. Beide Beschreibungsformen können in das Konzept integriert werden.
Die Graphproduktionen werden aus den Abhängigkeitsbeziehungen der parallelen Operationen der speziellen Anwendung – also informell formulierten Regeln – abgeleitet. Bei den Knotenmarkierungen sind prinzipiell folgende Bedeutungen zu unterscheiden:

- Knotenmarkierungen, die auf die statische Struktur des Graphen Einfluß nehmen, indem von ihnen die Anwendbarkeit einer Graphproduktion abhängt
- Knotenmarkierungen, die den dynamischen Ablauf beeinflußen (Beispiele sind die Instanzen in Funktionsnetzen, die ein partielles Schalten ermöglichen, sowie die in Abschnitt 3.3 vorgestellten Knotenmarkierungen), und
- Knotenmarkierungen, denen eine auszuführende Aktion zugeordnet wird.

Durch Definition von Zuständen auf Knotenmarkierungen wird es möglich durch eine Folge von Zustandswechseln den dynamischen Ablauf zu beschreiben. Zusammen mit den durch die Generierung mittels Graphproduktionen festgelegten statischen Struktureigenschaften erreicht man ein deterministisches Schaltverhalten. Diese Eigenschaft läßt sich aufgrund der theoretischen Fundierung formal beweisen /Her 88c/. Dadurch kann eine prozeßinterne Aktivitätsverwaltung ohne Verklemmungsprobleme erzeugt werden. Wie am Beispiel in Abschnitt 3.3 erläutert wird, ist dies auch für kooperierende Prozeßsysteme möglich.
Die Graphstruktur (und damit implizit die Ablaufstruktur) kann durch Einfügen und Löschen von Graphproduktionen auf einfache Weise geändert werden, so daß die Wartung von Ablaufstrukturen sehr vereinfacht wird.

3.3 Ein Anwendungsbeispiel

In /Her 88a/ wird die automatische Parallelisierung von Aktivitäten in einem auf dem Client/Server–Konzept aufbauenden Datenbanksystem für nicht–konventionelle Anwendungen (NDBS) vorgestellt. Die Serverschnittstelle ist dabei als eine mengenorientierte Operationsschnittstelle realisiert. Der Client sendet eine Menge von Operationen an den Server, die dieser unabhängig von der Reihenfolge der Ankunft

der Aufträge bearbeiten kann. Auf Fertigmeldungen vom Server für einzelne Operationen reagiert der Client mit dem Senden von nun entsprechend der Abhängigkeitsbeziehungen ausführbaren Operationen.

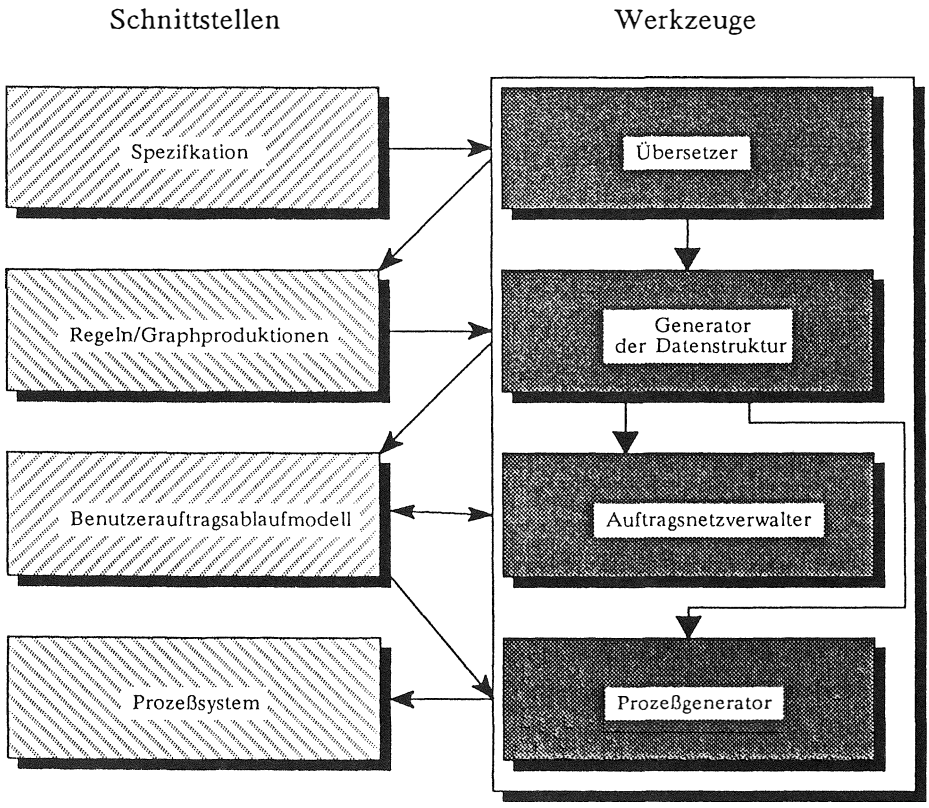


Bild 1: Entwicklungsunterstützung von Ablaufstrukturen

Der Entwerfer der zur Synchronisation der parallel zu bearbeitenden Serveraufträge notwendigen Ablaufstruktur wird durch vier aufeinander aufbauende Werkzeuge unterstützt:

- einem Übersetzer, der die Spezifikation der parallel ausführbaren Serveraufträge in eine Menge von Regeln überführt,
- einem regelbasierten Graphgenerator zur Erzeugung der zentralen Datenstruktur

- einem Generator eines entsprechend den spezifischen Abhängigkeitsbeziehungen kommunizierenden Prozeßsystems und
- alternativ zum Prozeßsystem der sog. Auftragsnetzverwalter, der eine prozeßinterne Aktivitätsverwaltung realisiert.

Durch anwendungsbezogene Sprachkonstrukte spezifiziert der Entwerfer, ob zwischen den Operationen Abhängigkeitsbeziehungen bestehen. Ist beim Entwurf schon sicher, daß solche Beziehungen nicht existieren (d.h. daß die Operationen parallel ausgeführt werden können), so wird dies durch die Schlüsselwörter (**OL ... OL**) und (**DL ... DL**) gekennzeichnet. Soll die Abhängigkeitsstruktur vom Übersetzer aus der Spezifikation abgeleitet werden, so ist dies durch (**OPAR .. OPAR**) zu spezifizieren. Der Übersetzer nutzt dazu Informationen über Eingangs- und Ausgangsgrößen der Serveroperationen und der spezifizierten Prozeduren aus. Das Prozedurkonzept erlaubt es, Abhängigkeitsbeziehungen zwischen Teilablaufstrukturen zu beschreiben. In Bild 2 ist ein fiktiver Rahmen eines Quellprogramms des Übersetzers dargestellt.

```
PROC MAIN ( in .... out ....)
          (OPAR     OpCALL  Suche ( in ... out ...),
                    OpCALL  Suche ( in ... out ...),
                    OpCALL  Prüfe  ( in ... out ...),
                    OpCALL  Lösche ( in ... out ...)
          OPAR)
ENDPROC

OpPROC Suche ( in ... out ...)
          (DL       RSC Suche_X1 ( in... out...)
                    RSC Suche_X2 ( in ... out ...)
          DL)
ENDPROC

OpPROC Prüfe ( in ... out ... )
          (OSEQ     RSC Prüfe_X1 ( in ... out ...),
                    RSC Prüfe_X2 ( in ... out ...)
          OSEQ)
ENDPROC

OpPROC Lösche ( in ... out ... )
          (OL       RSC Lösche_X ( in ... out ...),
                    RSC Lösche_Y ( in ... out ...)
          OL)
ENDPROC
```

Bild 2: Rahmen eines Quellprogramms des Übersetzers

Der Übersetzer bildet die Spezifikation auf eine Menge von Graphproduktionen ab. Durch Anwendung des Graphgenerators entsteht aus den Graphproduktionen eine Datenstruktur, die auf dem Benutzerauftragsablaufmodell (BA2M) aufbaut. Das BA2M wurde in /Her 88c/ zur Darstellung der Abhängigkeitsbeziehungen zwischen parallelen Aktivitäten in einem NDBS entwickelt. Ein Graph heißt BA2M, wenn er durch Anwendung der Produktionen der Graph-Grammatik für BA2M sequentiell aus dem Startgraphen abgeleitet werden kann und nur Knoten mit Markierungen aus dem terminalen Knotenmarkierungsalphabet enthält (Bild 3). Bis auf die terminale Knotenmarkierung der auszuführenden Serveroperation haben die Markierungen sowohl Einfluß auf die statische Struktur als auch – im Zusammenhang mit den weiter unten beschriebenen Auftragsnetzen – auf den dynamischen Ablauf. Bild 4 zeigt Beispiele für Graphproduktionen des BA2M, die die Erzeugung einer bestimmten Teilstruktur (1), einer bestimmten Form von Parallelität (2) und die die Generierung einer zusätzlichen Abhängigkeitsbeziehung (3) ermöglichen. Bild 5 zeigt ein vollständiges BA2M, das die im VLSI-Entwurf zur Darstellung und zur interaktiven Manipulation von Layouts benötigte Fensteroperation in einer möglichen Realisierungsform graphisch repräsentiert.

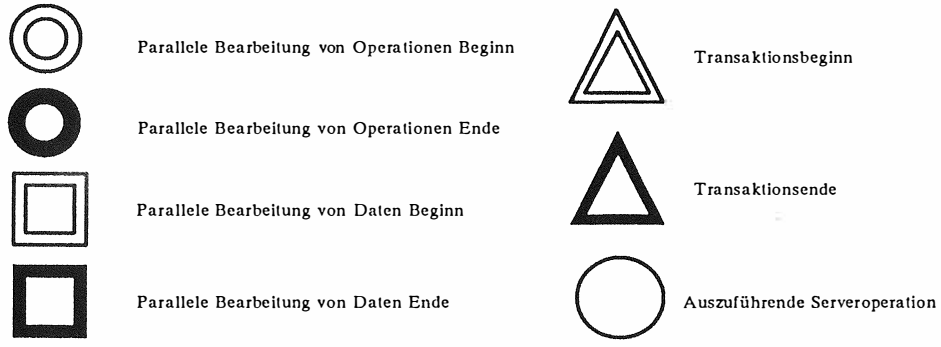


Bild 3: Darstellung des terminalen Knotenmarkierungsalphabets

Durch den Prozeßsystemgenerator wird ein BA2M auf zwei Arten von Prozessen abgebildet:

- Prozesse, die Operationsausführungen durch den Server kontrollieren und die nur eine aktive Phase haben, und
- Prozesse, die den parallelen Ablauf einzelner Teile des gesamten Auftrages initieren und kontrollieren.

Die Prozesse erster Art entsprechen im BA2M den Knoten mit der Markierung der auszuführenden Serveroperation. Jede 'Beginn'-'Ende'-Klammer in einem BA2M wird auf einen Prozeß zweiter Art abgebildet. Der Prozeßgenerator übersetzt die interne Darstellung eines BA2M in ein PEARL-Modul sowie in Kommandodateien zum Übersetzen und Laden dieses Moduls. Aus dem speziellen Sprachumfang von PEARL werden die Semaphoren als Synchronisationsmechanismus sowie Sprachkonstrukte zur Prozeßdeklaration und -verwaltung benutzt.

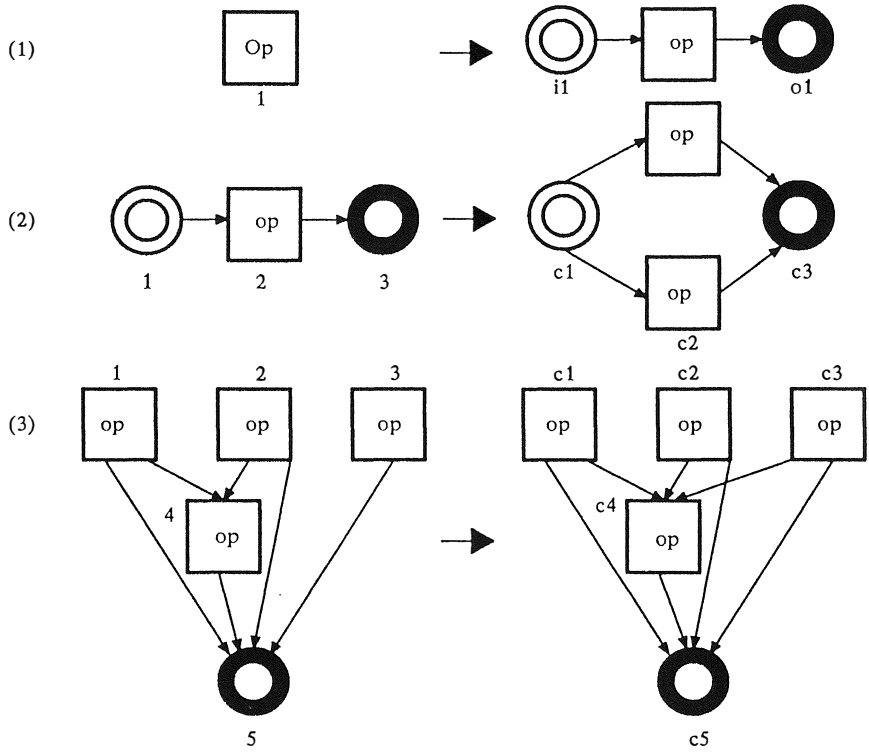


Bild 4: Beispiele für Graphproduktionen

Die Realisierung der Steuerung paralleler Abläufe durch eine prozeßinterne Aktivitätsverwaltung wird durch das Konzept der Auftragsnetze ermöglicht. Dazu werden auf den Knoten eines BA2M Zustände definiert. Die in Abhängigkeit von den Knotenmarkierungen festgelegten Zustandswechsel leisten die Synchronisation des Verarbeitungsvorgangs, wobei man sich prinzipiell ein Schaltverhalten wie bei Petri-Netzen vorstellen kann. Nimmt ein mit der Markierung der auszuführenden Serveroperation versehener Knoten den Zustand **gesendet** ein, so wird dem Server ein

Auftrag zur Ausführung übergeben. Anschließend sucht der Auftragsnetzverwalter nach weiteren Knoten, bei denen ein Zustandswechsel erfolgen kann. Durch eine Auftragsfertigmeldung vom Server wird in jedem Fall ein Fortschalten der Zustände ausgelöst. Auftragsnetze haben sich als geeignetes Konzept zur Erreichung einer adäquaten Fehlerbehandlung, die u.a. den Abbruch und die anschließende Wiederholung einer semantisch verbunden Menge von Operationen ermöglicht, erwiesen.

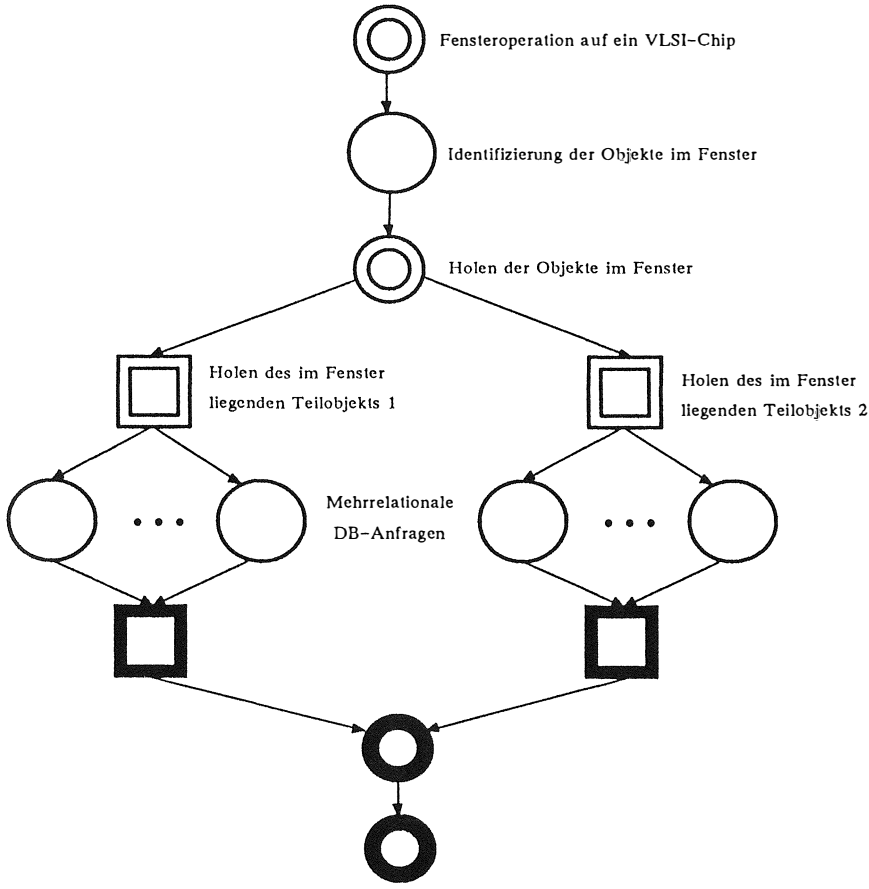


Bild 5: Beispiel für ein Benutzerauftragsablaufmodell

Die dritte Realisierungsform von parallelen Ablaufstrukturen kann durch eine Mischung zwischen Prozeßsystem und Auftragsnetzen erreicht werden. Es werden lediglich noch Prozesse der o.a. zweiten Art generiert, wobei die Prozeßanzahl

flexibel von der Schachtelungstiefe der Daten- und Operationsparallelität abhängig gemacht werden kann. Innerhalb dieser Prozesse verwalten Auftragsnetze den parallelen Verarbeitungsfluß.

3.4. Dynamische Generierung von parallelen Ablaufstrukturen

Der Softwareentwicklungsprozeß wird in eine Reihe von Phasen zerlegt:

Problemanalyse -> Entwurf -> Implementierung -> Test -> Installation -> Wartung

Das in dieser Arbeit vorgestellte Konzept zur Entwicklungsunterstützung von parallelen Ablaufstrukturen auf der Basis von Graphen als zentrale Datenstruktur sowie einer anwendungsbezogenen Graph-Grammatik zur Generierung dieser Datenstruktur unterstützt vier der genannten Phasen:

- *den Entwurf:* durch die Möglichkeit der Spezifikation auf der Basis von Eigenschaften
- *die Implementierung:* durch die Möglichkeit der automatischen Generierung von Ablaufstrukturen
- *den Test:* durch das durch die formale Fundierung und durch die automatische Generierung der Ablaufstrukturen garantierte "Schaltverhalten" und
- *die Wartung:* durch einfache Änderbarkeit der Graphstruktur und damit impliziert durch die vorgestellten Werkzeuge auch der Ablaufstruktur.

Somit bietet das vorgestellte Konzept eine gute Basis für die in Kap. 1 geforderte durchgängige Entwicklungsunterstützung. Da alle der in Abschnitt 3.1 dargestellten Möglichkeiten zur Implementierung der Steuerung von parallelen Abläufen unterstützt werden, ergibt sich ein gewisses Maß an Unabhängigkeit von den Einflußfaktoren bei der Entscheidung bzgl. der Implementierungsart.

4. Literatur

/CR 86/ Campbell, R.H.; Randell, B.: *"Error Recovery in synchronous Systems"*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 8, August, 1986

/Dä 87/ Dähler, J. et al.: *"A Graphical Tool for the Design and Prototyping of Distributed Systems"*, ACM SIGSOFT, Software Engineering Notes, Vol. 12, No. 3, July, 1987

/Ga 85/ Gait, J.: *"A Debugger for Concurrent Programs"*, Software – Practice and Experience, Vol. 15, No. 6, June, 1985

/Go 83/ Godbersen, H.P.: *"Funktionsnetze – Eine Modellierungkonzeption zur Entwurfs- und Entscheidungsunterstützung"*, Ladewig Verlag, Birkbach, 1983

/GT 86/ Godbersen, H.P.; Trümner, H.: "*Ein graphisch orientierter Ansatz zur Bildung operationaler Modelle*", 16. GI–Jahrestagung, IFB 127, Springer–Verlag, Berlin, 1986

/Her 88a/ Herzog, H.: "*Automatische Parallelisierung von Aktivitäten in einem verteilten System*", Proc. 18.GI–Jahrestagung, Hamburg, Springer–Verlag, Berlin, 1988

/Her 88b/ Herzog, H.; Hildebrandt, F.: "*Überlegungen zur Entwicklung von Datenbankmaschinen für nicht konventionelle Anwendungen*", Informatik–Bericht 88–08, TU Braunschweig, 1988

/Her 88c/ Herzog, H.: "*Synchronisation kooperierender Aktivitäten in Nicht–Standard–Datenbanksystemen*", Dissertation, TU Braunschweig, 1988

/Ka 87/ Karp, A.H.: "*Programming for Parallelism*", IEEE Computer, Vol.20, No. 5, May, 1987

/Na 78/ Nagl, M.: "*Graph–Grammatiken. Theorie. Anwendungen. Implementierung*", Vieweg, Braunschweig, 1978

/Ol 85/ Olson, R.: "*Parallel Processing in a Message–Based Operating System*", IEEE Software, Vol.2, No. 4, July, 1985

/Re 83/ Reisig, W.: "*System Design using Petri Nets*", in: Hommel,G.; Krönig, D. (Hrsg.): "Rrequirements Engineering", Arbeitstagung der GI, IFB 74, Springer–Verlag, Berlin, 1983

/Rö 86/ Röhrich, J.: "*Parallele Systeme*", Informatik–Fachbericht 117, Springer–Verlag, Berlin, 1986

/Sch 86/ Schneider, H.–J.: "*Formale Gestaltungsaspekte in der Systementwicklung*", Handbuch der modernen Datenverarbeitung, Heft 130, 1986

/Wi 86/ Winter, D. et al.: "*ISAC – ein System zur Unterstützung der Systembeschreibung und Systemanalyse*", Handbuch der modernen Datenverarbeitung, Heft 130, 1986

Erfahrungen mit der Portierung eines Prozeßleitsystems

Dipl.-Inform. Erwin Kneuer
Werum GmbH, 2120 Lüneburg

Zusammenfassung

Für die Überwachung und Steuerung von Fertigungsabläufen und Produktionsprozessen auf Basis von industriefähigen Arbeitsplatzrechnern hat die Werum GmbH das Prozeßleitsystem PAS-PLS konzipiert und realisiert.

Das aus mehr als 30 Tasks bestehende System wurde 1986/87 rechnerunabhängig entworfen und zunächst in Pascal auf und für Sicomp PC 16-11 unter Concurrent CPM realisiert. Pascal wurde verwendet, weil auf diesem Rechner PEARL nicht verfügbar war. Nach Erscheinen des AT-kompatiblen Sicomp PC 16-20 wurde versucht, die Pascal-Version von PAS-PLS auf diesen Rechner zu portieren. Dabei zeigte sich rasch, daß der Zeitaufwand für die Anpassung der Taskstruktur, der Task-Anweisungen und der E/A-Anschlüsse größer geworden wäre als die Zeit, die für die Realisierung eines bereits laufenden Projekts zur Verfügung stand. Deshalb entschloß sich Werum, PAS-PLS nach PEARL umzuschreiben und parallel seinen portablen PEARL-Compiler auf Sicomp PC 16-20 unter FlexOS zu implementieren. Beides konnte rechtzeitig abgewickelt werden, wobei insbesondere deutlich wurde, daß die daran beteiligten Ingenieure wesentlich weniger Aufwand mit den rechnerunabhängigen problemorientierten Echtzeit- und E/A-Anweisungen von PEARL als mit den rechnerabhängigen Pascal-Calls für Concurrent CPM oder FlexOS hatten.

Dieser große Vorteil von PEARL - seine einheitliche, genormte und ingenieur-gerechte Echtzeit-Betriebssystemschnittstelle - zeigte sich inzwischen bei den weiteren Portierungen von PAS-PLS, die alle erstaunlich *wenig Ingenieursaufwand* erforderten. Wie bei der Schaffung von PEARL beabsichtigt, liegt der Portierungsaufwand vor allem bei den *Informatikern*, die den PEARL-Compiler und die PEARL-Betriebssystemschnittstelle implementieren. So konnte PAS-PLS mittlerweile von Ingenieuren auf folgenden Rechenanlagen installiert werden, für die jeweils das PEARL-System von Werum verfügbar ist:

DEC VAX mit VMS
IBM PS/2 mit OS/2
PCS PEARL Engine 68000 mit UNIX / BAPAS®-K
Sicomp PC 16-20 mit FlexOS
Sicomp M mit ORG M

1. Charakterisierung des Prozeßleitsystems PAS-PLS

Computergesteuerte Automatisierungssysteme können für verschiedene Anwendungsbereiche dann kostengünstig eingesetzt werden, wenn gemeinsame Problemstellungen durch ein konfigurierbares und parametrierbares Softwaresystem gelöst werden.

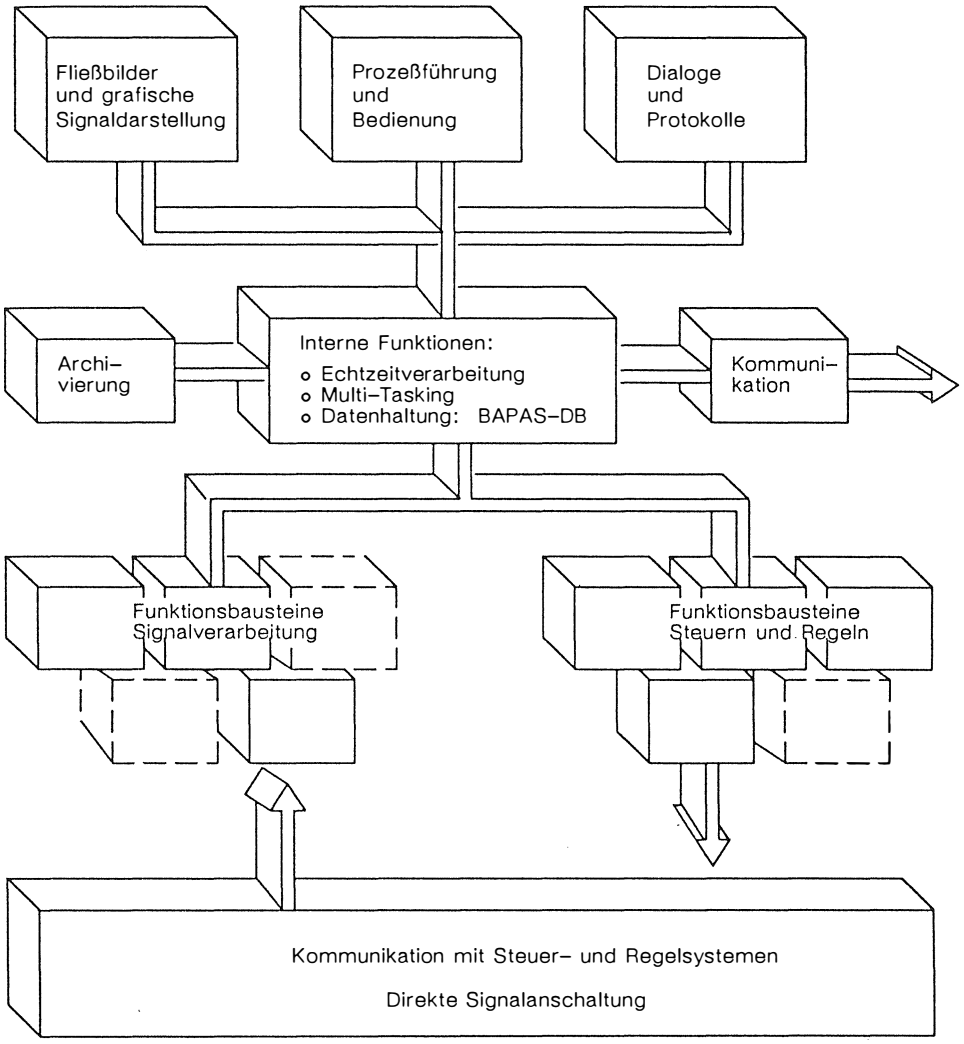
Für die Überwachung und Steuerung von Fertigungsabläufen und Produktionsprozessen auf Basis von industriefähigen Arbeitsplatzrechnern hat die Werum GmbH das Prozeßleitsystem PAS-PLS konzipiert und realisiert. Typische Einsatzgebiete für PAS-PLS sind:

- Überwachung und Steuerung verfahrenstechnischer Prozesse
- Meßwerterfassung und -verarbeitung für Prüfstände und in Labors
- Erfassung und Verarbeitung von Klima- und Umweltdaten
- Überwachung von Produktionseinheiten in der Grundstoffindustrie
- Betriebsdatenerfassung
- Online-Management-Informationssystem für Produktionsdaten

PAS-PLS verbindet dabei die Standardfunktionen von Prozeßleitsystemen - das aktuelle Anzeigen und Bedienen - mit Archivierungs- und Auswertungsfunktionen.

1.1 Funktionalität

PAS-PLS ermöglicht individuelle Funktionskonfigurationen auf Basis folgender Standardkomponenten:



Funktionsbausteine von PAS-PLS

Zur Datenhaltung wird als integraler Bestandteil das Echtzeit-Datenbanksystem BAPAS®-DB von Werum eingesetzt.

Folgende Funktionen sind Bestandteil von PAS-PLS:

- **Kommunikation**
  - mit übergeordneten oder parallelen Systemen über serielle Kopplung oder ein LAN
  - mit unterlagerten Systemen wie SPS, BDE-Terminalstationen, Prozeß-einheiten oder klassischen PLS: seriell oder über Bussysteme

- **Signalverarbeitung**
  - Digitale Signale
  - Analoge Signale
  - Impulszähler
  - Berechnete Werte

  Typischerweise beinhaltet die Signalverarbeitung
  - Filterung
  - Grenzwertüberwachung
  - Gradientenprüfung
  - Statistische Funktionen wie Mittelwertbildung und Standardabweichung

  Dabei können die Signalparameter abhängig von Herstellanweisungen (Rezepten) für die Sollwerte, Grenzwerte und den Umfang der Auswertung der zu überwachenden Signale sein. Neue Rezeptdaten werden bei Produktwechsel gültig.

- **Anlagenbilder und grafische Signaldarstellung**
  Sämtliche Signale sind ständig aktuell in einem oder mehreren Anlagenbild(ern) anzeigbar. Unter Zuhilfenahme des Grafik-Editors PAS-GRAPH können Bilder online erstellt oder verändert werden. Die Darstellung ist als numerischer Wert/Kurve/Balken oder durch Symbole möglich. Koordinatensysteme, Layout und Aktualisierungszyklen sind wählbar. Bei Zugriff auf archivierte Daten sind Koordinatensysteme eines Bildes verschiebbar und zoombar. Ein Lineal ermöglicht das Ablesen von Werten.

- **Prozeßführung und Bedienung**
  Funktionsbausteine Steuern und Regeln. Die Ausgabe von Befehlen und Sollwerten ist parametrierbar abhängig von
  - Zeit
  - Ereignissen
  - Rezepten.

- **Dialoge und Protokolle**
  Die Mensch-Maschine-Kommunikation ist über verschiedene Terminaltypen wie Schaltpult oder BDE-Terminals durch Dialogmasken möglich.
  Die Gestaltung des Layouts für Dialoge und Protokolle erfolgt interaktiv mit dem Maskeneditor BAPAS-MMC von Werum.

- **Interne Funktionen** beinhalten
  - Multitasking
  - Echtzeitverarbeitung
  - Systemüberwachung (Ausfallstrategien).

o **Archivierung**
Die Archivierung erfolgt nach kundenspezifischen Kriterien. Standardmäßig sind möglich:

- Zeitorientierte Archivierung (Stunde, Tag, Woche, Monat)
- Schichtorientierte Archivierung
- Auftragsorientierte Archivierung, wobei Aufträge in Chargen und Chargen in Produktionseinheiten unterteilbar sind. Aufträge sind unterbrechbar
- Produktionsanlagenorientierte Archivierung

Archiviert werden können sämtliche Signalarten. Zusätzlich ist die Archivierung von im Dialog eingegebenen Werten möglich.

1.2 Mengengerüst

Folgende generierbaren Parameter sind in der Standardversion PAS-PLS je nach Hardwaresystem einstellbar, **wobei diese Zahlen als Anhaltspunkt dienen sollen.** Grenzen setzt hier nur die Hardwareausstattung bzgl. Verarbeitungsgeschwindigkeit, Haupt- und Plattenspeicher sowie die Möglichkeit, zusätzliche Peripherie anzuschließen.

	Leitrechner
Anzahl Meldungen	≥ 2.000
Anzahl Meßwerte	≥ 1.000
Anzahl Zählwerte	≥ 100
Anzahl Befehle	≥ 100
Anzahl Sollwerte	≥ 100
Grundzyklus	1 s – 3 s
Verarbeitungsrate bei voller Verarbeitung (Meßwertverarbeitungen pro Minute)	≥ 500 – 1.000
Anzahl der Grenzwertverletzungen (pro Tag)	≥ 600 – 1.000
Änderungshäufigkeit der Meldungen (pro Tag)	≥ 10.000
Anzahl Grafik-Bilder	≥ 100
Anzahl unterlagerter SPS-Systeme	1 bis 20
Anzahl Grafik-Terminals	1 bis 8
Anzahl Drucker	1 bis 10

Eine Änderung ist projektspezifisch und abhängig von der Ausbaustufe der Hardware möglich.

PAS-PLS hat folgenden Umfang:

- 70 Moduln
- 32 Tasks
- 40 Synchronisiervariablen
- 50 - 70.000 Quellzeilen (konfigurationsabhängig)
- Ca. 1 MB Objektcode und -daten (konfigurationsabhängig)

1.3 Wesentliche Schnittstellen

Entsprechend seinem Funktionsumfang besitzt PAS-PLS verschiedene Schnittstellen:

- o Prozeß-E/A
  - Digitale Ein- und Ausgänge
  - Analoge Ein- und Ausgänge
  - Serielle Schnittstellen zu Steuer- und Regelsystemen
- o Standard-E/A
  - Druckerausgabe für Protokolle und Fehlermeldungen
  - Platten-E/A für das Grafik-System
  - Terminal-E/A
- o Datenhaltung
  - DBV-Schnittstelle von BAPAS-DB
- o Grafik
  - VDI-Schnittstelle für geometrische Grundfunktionen und zum Setzen von Attributen
- o Multi-Tasking
  - Definition und Aktivieren von Tasks
  - Datenaustausch zwischen Tasks
  - Verzögerung von Tasks
  - Synchronisation von Tasks
  - Koordinierung der Zugriffe von Tasks auf gemeinsam benutzte Betriebsmittel

2. Erstimplementierung in Pascal

PAS-PLS wurde 1986/87 rechnerunabhängig entworfen und zunächst in Pascal auf und für Sicomp PC16-11 realisiert. Pascal wurde verwendet, weil auf diesem Rechner PEARL nicht verfügbar war.

2.1 Hardware- und Softwareumgebung

Der Rechner besaß folgende Charakteristika:

Siemens Sicomp PC16-11

Hardware

- 8086-Prozessor
- 8087-Coprozessor
- 1 MB Hauptspeicher

Software

- CCPM/86-Betriebssystem
- Pascal MT+ - Compiler
- LINK MT - Linker
- DK3964R - Treiber für Simatic S5
- GSX86 - Grafiksystem

2.2 Implementierungserfahrungen

Tasks wurden als CCPM/86-Programme implementiert. Aufgrund der wenig problemorientierten Tasking-Schnittstelle von CCPM/86 und der langjährigen positiven Erfahrungen mit dem PEARL-Tasking-Modell wurden die Tasking-Anweisungen für die o.g. Schnittstellen als PEARL-ähnliche Prozeduraufrufe

- call ACTIVATE (parameter, ...)
- call SUSPEND (parameter, ...)
- call CONTINUE (parameter, ...)
- usw.

mittels einer Assembler-Schnittstelle zwischen Anwendung und Betriebssystem realisiert. Für die Kommunikation und Synchronisation zwischen Programmen wurden Queues (ggf. mit shared Memory) eingesetzt.

Diese erste Implementierung (Version 1) erforderte ca. 2 Mannjahre Aufwand. Sie stellte ein reines Überwachungssystem mit Polling-Modus für eine zu überwachende Anlage dar. Ihr Umfang betrug ca. 30% des heute verfügbaren Funktionsumfangs.

3. Reimplementierung in PEARL

3.1 Gründe für die Reimplementierung

Nach Erscheinen des AT-kompatiblen Rechners Siemens Sicomp PC16-20 sollte PAS-PLS auch auf diesem Rechner eingesetzt werden. Doch bereits bei dieser ersten Portierung innerhalb einer Rechnerfamilie traten erhebliche Probleme auf:

- CDOS ließ sich wegen seines beschränkten Adreßraums und wegen ungenügender Grafik-Möglichkeiten nicht benutzen.
- Unter dem erklärten Nachfolger von CCPM/86, dem Multitasking-Betriebssystem FlexOS (beide von Digital Research) war das unter CCPM/86 verwendete Pascal-System nicht verfügbar, sondern nur ein anderes, keine Programm-Portabilität bietendes Pascal-System.
- Die Betriebssystem-Schnittstellen zwischen CCPM/86 und FlexOS waren sehr unterschiedlich. Eine Anpassung der Pascal-Quellen hätte mehr Zeit gebraucht, als wegen eines bereits laufenden Projekts zur Verfügung stand.

Außerdem waren zusätzliche Anforderungen zu berücksichtigen:

- Weitere Portierungen auf sehr verschiedene Hardware- und Softwareumgebungen waren bereits absehbar (Siemens Sicomp M unter ORG M, IBM PS/2 unter OS/2).
- Die Wartung und Weiterentwicklung von PAS-PLS sollte für alle künftigen Zielsysteme in *einer* Mutterversion erfolgen, die sich möglichst wenig von Einsatzversionen unterscheidet.
- Die Weiterentwicklung sollte auf *einem* Hostsystem, aber *unabhängig* von einer bestimmten Systemumgebung erfolgen können.
- Die Applikationsingenieure, die PAS-PLS einsetzen, warten und weiterentwickeln, sollten sich auf die Anforderungen der technischen Aufgabe konzentrieren können und nicht mit Betriebssystem-Spezifika verschiedener Rechner belastet werden.

Diese Gründe führten dazu, daß Werum PAS-PLS nach PEARL umschrieb und parallel seinen portablen PEARL-Compiler auf Sicomp PC16-20 unter FlexOS implementierte.

3.2 Hardware- und Softwareumgebung

Siemens Sicomp PC16-20 (AT-kompatibel)

Hardware
- 80286-Prozessor
- 80287-Coprozessor
- 2 MB Hauptspeicher

Software

- FlexOS286 – Betriebssystem
- PEARL–Programmiersystem
- Standard–Linker
- DK3964R–Treiber für Simatic S5
- VDI/EGA–Grafiksystem, integriert in FlexOS.

3.3 Implementierungserfahrungen

Die Multitasking–Funktionen von PAS–PLS konnten direkt auf PEARL–Sprachmittel abgebildet werden (ihre Prozeduraufrufe waren ja schon entsprechend definiert). Die Umschreibung von Pascal nach PEARL erforderte ca. 3 Mannmonate Aufwand. Danach wurde festgestellt, daß man 30 – 40% des Aufwands zur Erstimplementierung hätte sparen können, wenn diese gleich in PEARL durchgeführt worden wäre.

Mittlerweile wurden weitere 12 Mannmonate für die Weiterentwicklung von PAS–PLS in PEARL aufgewendet. Die aktuelle Version 2.2 kann für mehrere Anlagen mit Regelparameter–Vorgaben eingesetzt werden; neben Polling ist nun auch eine event–gesteuerte Verarbeitung möglich.

4. Portierungen auf weitere Zielsysteme

Die Verfügbarkeit in PEARL erlaubte 1988 bereits vier weitere Portierungen auf vier sehr unterschiedliche Zielsysteme:

Computersystem	Betriebssystem	Charakterisierung
Siemens Sicomp M	ORG M	Multitasking, Single–user Siemens–Prozessor 16 Bit
IBM PS/2	OS/2	Multitasking, Single–user Intel 286–, 386–Prozessoren
PCS PEARL Engine 68000	MUNIX / BAPAS–K	Doppelprozessor–Maschine mit MUNIX: Multi–user, Multitasking BAPAS–K: Multitasking, Single–user Motorola 68020 – Prozessor
DEC VAX	VMS	Multitasking, Multi–user VAX–Prozessor 32 Bit

Dabei stellten sich unterschiedliche Portierungsaufwände für die Komponenten von PAS-PLS heraus:

(1)	Portierung der Mutter-Version (PEARL-Quellen)	ca. 1 Mannwoche
(2)	Anschluß an ein vorhandenes Grafiksystem	1 – 3 Mannwochen
(3)	Anschluß der Dialogsteuerung an die eingesetzten Terminals	1 – 2 Mannwochen
(4)	Kommunikation mit SPS (speicherprogrammierbaren Steuerungen), abhängig davon, ob im Betriebssystem bereits Treiber für das SPS-Protokoll existieren und nur noch in das PEARL-System integriert werden müssen oder ob Treiber für ein SPS-Protokoll neu erstellt werden müssen.	1 – 4 Mannwochen

Wesentlich höher ist natürlich der Aufwand für die Implementierung des PEARL-Programmiersystems, falls PEARL auf dem Zielrechner noch nicht verfügbar ist. Aber dies war ja gerade eines der großen Ziele der Schaffung von PEARL: Der Portierungsaufwand für PEARL-Programme fällt einmal in größerem Umfang bei den Informatikern an, die das PEARL-Programmiersystem implementieren und nicht jedesmal bei den portierenden Ingenieuren.

Automatische Funknetzüberwachung - Multitasking mit PEARL

Dipl.-Ing.(FH) Albert Langer
AEG Ulm, Abt. A12E322
Sedanstr. 10
7900 Ulm/Donau
0731/3923688

Zusammenfassung:

Anhand eines Systems zur automatischen Funknetzüberwachung wird ein Verfahren zur Synchronisation und Kommunikation von Prozessen in einem Realzeit-/Multitsking-System vorgestellt.

Das System hat die Aufgabe ein vorgegebenes Frequenzband über maximal 5 Empfänger, die ständig je einen Teilbereich des Frequenzbandes nach Sendern absuchen, zu überwachen. Gefundene Sender werden online über der Frequenz auf einem Grafikbildschirm dargestellt. Es werden die Aktivitäten der letzten maximal 90 Minuten komplett gespeichert und auch gleichzeitig in einer Übersichtsdarstellung in komprimierter Form dargestellt. Außerdem kann ein Teil des Frequenzbandes in einer gespreizten Form dargestellt werden.

1. Einleitung

Für die Entwicklung des Systems zur automatischen Funknetzüberwachung (AFÜ) gab es mehrere Gründe:

- einen bisher manuellen Vorgang weitestgehend zu automatisieren

  den Vorgang der Frequenzbandüberwachung um eine Faktor von ca. 1000 zu beschleunigen

- die Entdeckungswahrscheinlichkeit kurzer Sendungen zu erhöhen

- eine zentrale Steuerung aller vorhandener Empfänger vom leitenden Operator aus

  Überwachung eines großen Frequenzbandes durch eine Aktivitätsanzeige

  der Operator kann eine sich ändernde Senderaktivität sofort erkennen und schnell reagieren

2. Systembeschreibung

Bei dem System AFÜ handelt es sich um ein mobiles und auf mehrere Kabinen aufgeteiltes Konzept (Bild 1). Zu einem kompletten System gehören

- eine Zentrale mit einem Rechner und zwei Empfängern (leitender Operator)

  bis zu 4 Erfasserkabinen mit je 4 Empfängern und zwei Operatorarbeitsplätzen pro Kabine

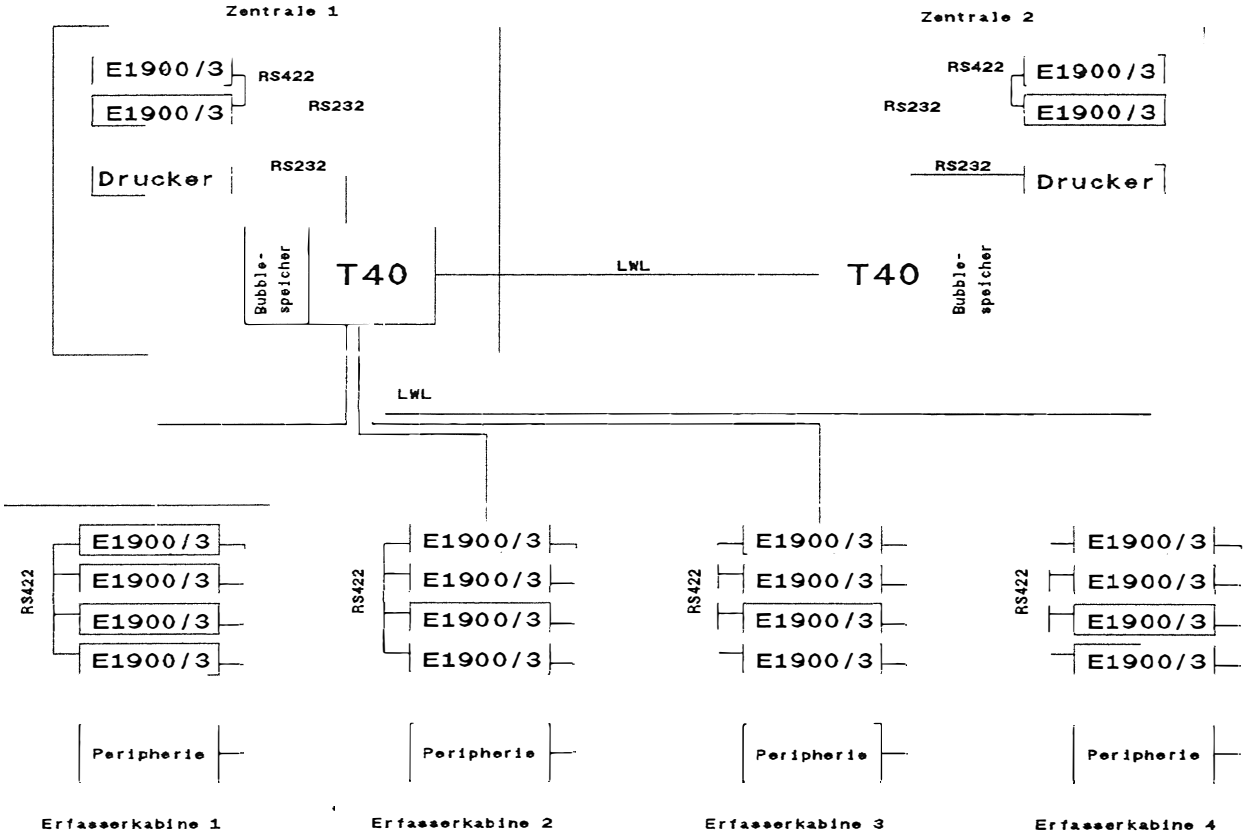


Bild 1: Standardkonfiguration

2.1 Hardwarekonfiguration

Das Herz der Zentrale ist der Arbeitsplatz-Terminal-Rechner T40. Der T40-Rechner besteht aus folgenden Komponenten:

- Host-Computer: Prozessrechner der Firma MODCOMP in Konstanz (ATM 80-16) mit 1MByte CMOS-Ram Hauptspeicher
- Grafikprozessorkarte (Entwicklung der Abt. A12E3)
- monochromer Bildschirm mit der Auflösung 720 auf 340
- Tastatur mit 108 Tasten
- Magnetblasenspeicher mit 1MByte Speicherkapazität
- Rollkugeleinheit

Alle Komponenten bis auf die Rollkugel sind ein einem Gehäuse untergebracht und gegen elektromagnetische Abstrahlung gesichert.

Folgende Peripherieanschlüsse stehen zur Verfügung:

- ein Lichtwellenleiter(LWL)-Anschluß zur Kopplung mit einer weiteren Zentrale
- 4 LWL-Anschlüsse für die 4 Erfasserkabinen
- 1 Druckeranschluß (RS232)
- 1 Anschluß für Zusatzempfänger (RS232)

Als exterener Datenspeicher wird der wechselbare Magnetblasenspeicher mit 1MByte Speicherkapazität verwendet. Für den Ausdruck von Nachrichten, Listen und Ortungsergebnissen steht ein Matrixdrucker mit 80 Zeichen pro Zeile zur Verfügung.

Über einen LWL-Anschluß kann die Zentrale mit einer weiteren Zentrale zum Zwecke einer Aufgabenentlastung gekoppelt werden.

Über 4 weitere LWL-Anschlüsse können Erfasserkabinen angeschlossen werden. Die 4 Empfänger einer solchen Kabine sind somit voll von der Zentrale aus steuerbar, können aber für bestimmte Aufgaben auch lokal bedient werden.

2.2_Funktionsbeschreibung

Das System AFÜ kennt zwei Betriebsarten, nämlich den Aufnahmebetrieb und den Ortungsbetrieb.

2.2.1_Aufnahmebetrieb

In dieser Betriebsart kann ein vorgegebenes Frequenzband zu Erhhöhung der Suchgeschwindigkeit über maximal 5 Empfänger überwacht werden. Das gesammte Band wird mittels einer Liste, in der auch die Empfänger für den Suchbetrieb ausgewählt werden, in bis zu 5 Teilbänder zerlegt. Die einzelnen Teilbänder werden dann von den Empfängern parallel nach belegten Frequenzen abgesucht. Außerdem muß in der oben erwähnten Liste die Betriebsart, Bandbreite und die Schrittweite für den Suchbetrieb definiert werden. Findet einer der Empfänger eine belegte Frequenz, so meldet er sie an den T40-Rechner. Der Rechner aktualisiert daruf hin die Aktivitätsanzeige online.

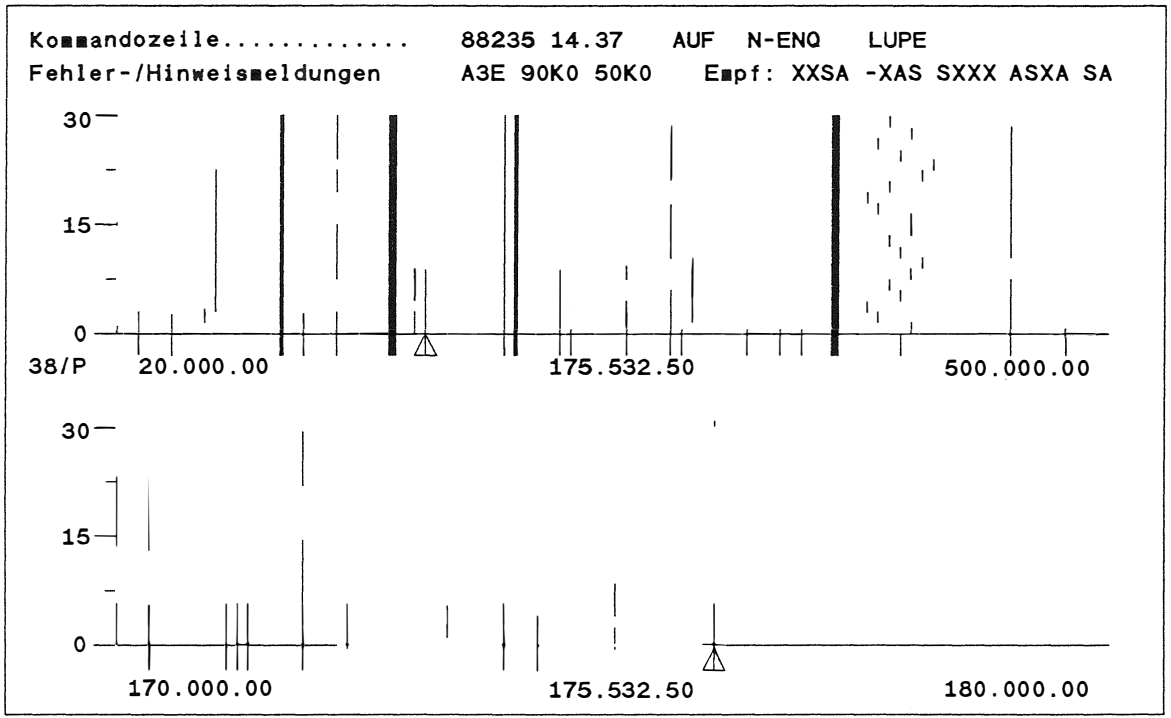


Bild 2: Bildschirmdarstellung im Aufnahmebetrieb

Die Aktivitätsanzeige (Bild 2) stellt die belegten Frequenzen über einen vorgegeben Zeitraum und über ein eingegebenes Frequenzband dar. Bei dieser Darstellung handelt es sich also um eine zweidimensionale Darstellung mit der Frequenz in x-Richtung und der Zeit in y-Richtung. Die Zeitachse (Histogrammzeit) kann im Bereich zwischen 10 und 90 Minuten frei gewählt werden. Zur Abbildung der Histogrammzeit auf dem Bildschirm stehen 90 Pixelzeilen zur Verfügung, daß bedeutet das bei einer eingestellten Histogrammzeit von z. B. 15 Minuten alle 10 Sekunden der Bildschirmberich um eine Pixelzeile nach oben gerollt werden muß und sich somit das Histogramm langsam von unten nach oben aufbaut.

Aufgrund der Größe des möglichen Frequenzbandes (20 bis 500MHz) ergeben sich bei einem Frequenzraster von 25kHz 19200 Kanäle. Aus Abbildungsgründen mußte eine komprimierte Darstellung gewählt werden, um das gesammte Frequenzband abbilden zu können. D. h. es werden mehrere Kanäle zusammengefaßt um sie gemeinsam in einer Pixelspalte auf dem Bildschirm darstellen zu können.

Um nun nicht eine erheblichen Informationsverlust zu bekommen, ist es möglich sich innerhalb der komprimierten Darstellung mit dem Cursor ein Kanalpaket anzuwählen und dieses mit einer Art Lupe zu betrachten. Dabei wird um die Mittenfrequenz des Kanalpaketes eine gespreitzte Darstellung von +/- 120 Kanälen aufgebaut. Die Übersichtsdarstellung, sowie auch die Lupendarstellung, wird weiterhin ständig online aktualisiert.

Innerhalb der Lupendarstellung kann nun mit dem Cursor gezielt eine Frequenz angewählt werden und mit dieser Frequenz ein freier Empfänger automatisch eingestellt werden. Diese Empfänger kann dann der jeweilige lokale Operator gezielt feinabstimmen und den Funkverkehr aufzeichnen.

Weiter hat der Operator am T40-Rechner die Möglichkeit eine Liste mit 96 diskreten Frequenzen zu erstellen und diese Liste an einen freien Empfänger zu senden und anschließend eine Scan über diese Frequenzen zu starten. Findet der Empfänger auf einer dieser Frequenzen einen Sender so stoppt er seinen Suchbetrieb und der lokale Operator kann den Funkverkehr aufzeichnen oder den Suchbetrieb weiter fortsetzen.

2.2.2 Ortungsbetrieb

Während der T40-Rechner als Suchempfänger arbeitet kann man per Tastendruck in den Ortungsbetrieb umschalten. Die Grafikdarstellung wird dadurch in den Hintergrund verdrängt, sie wird jedoch ständig weiter aktualisiert.

Im Ortungsbetrieb können Senderstandorte auf Grund von manuellen Peilungen berechnet und ausgedruckt werden.

Bei der Rückschaltung in den Aufnahmebetrieb erscheint wieder die aktuelle Grafikdarstellung.

2.2.3 Betrieb mit zwei Zentralen

Im Fall, daß zwei Zentralen über den dafür vorgesehenen LWL-Kanal gekoppelt sind, wird eine Zentrale in den Aufnahmebetrieb und die andere in den Ortungsbetrieb geschaltet. Ist dies erfolgt so werden die Daten des anderen jeweils auf dem neueste Stand gehalten, so daß nach erfolgter Trennung die beiden Zentralen jeweils auf dem neuesten Datenstand sind. Außerdem können die Bediener über den LWL-Kanal auch Nachrichten austauschen.

2.3 Datentechnische Probleme

Da im Suchbetrieb bis zu fünf Empfänger parallel arbeiten ist es im Extremfall möglich, daß auf allen fünf Kanälen Frequenzmeldungen mit der vollen Datenrate von 4800 Baud pro Kanal empfangen und verarbeitet werden müssen. Das bedeutet eine Summenbaudrate von 24000 Baud. Bei 10 Bit pro Byte Nutzdaten und 6 Byte pro Frequenz ergeben sich 400 Belegungsmeldungen pro Sekunde. D. h. um eine Frequenzmeldung abzuarbeiten bleiben dem Rechner ca. 2.5 ms.

In dieser Zeit muß die Frequenz von BCD-Code auf FIXED(31) umgerechnet werden, in eine Kanalnummer transformiert werden und für die Ausgabe auf dem Bildschirm sowohl in der komprimierten Darstellung als auch in der Lupendarstellung aufbereitet werden. Außerdem muß die Frequenz noch in den Rohdatenpuffer, der für den Lupenaufbau notwendig ist, eingetragen werden. Nebenbei wird auch noch Zeit für andere Tätigkeiten wie z. B. die Ortungsberechnug benötigt. Der größte Teil des Programms wurde in PEARL geschrieben, lediglich extrem zeitkritische Teile mußten in ASSEMBLER realisiert werden.

Ein weiteres Problem bestand darin, daß für den Histogrammaufbau ein pixelweises Rollen eines Windows auf dem Bildschirm implementiert werden mußte. Da dies auf keinen Fall vom Rechner (Grund siehe oben) in einer brauchbaren Zeit zu realisieren war, wurde diese Funktion auf der Grafik-prozessorkarte integriert.

3. Softwaredesign und Softwareentwicklung

Das Softwaredesign und auch die Software wurde mit Hilfe des Softwareentwicklungswerkzeuges PROMOD entwickelt.

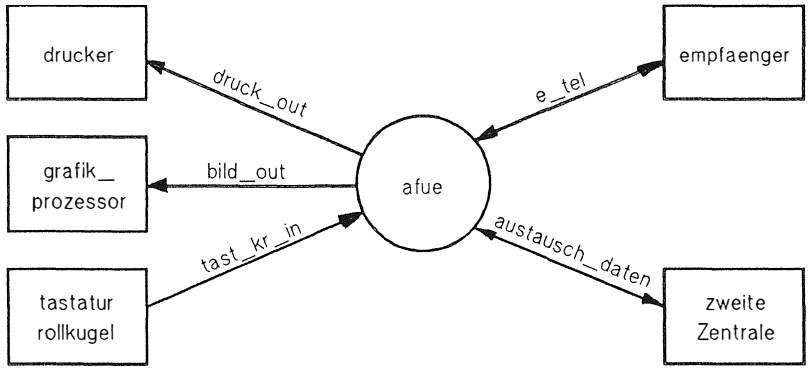


Bild 3: Systemübersicht (SA-Phase)

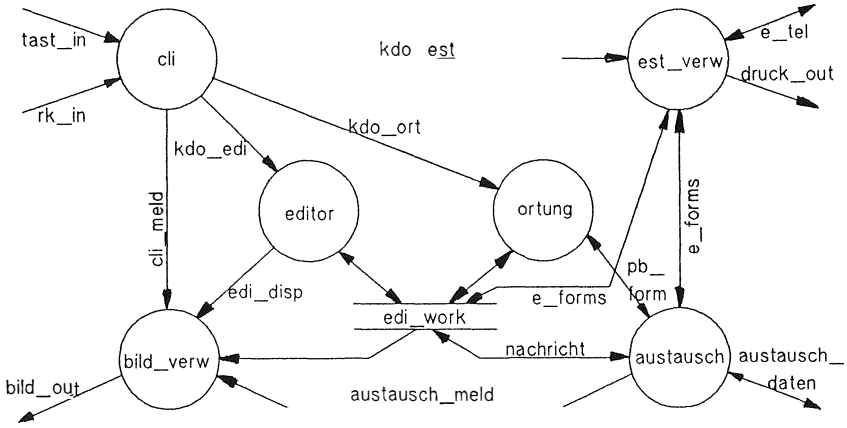


Bild 4: Verfeinerung von afue

3.1_Softwaredesign

PROMOD unterstützt die Entwicklung des Systems von der Definition und Anforderungsanalyse (SA-Phase) über das Systemdesign (MD-Phase) bis hin zum Programmdesign (PC-Phase).

In der SA-Phase (Srtuctured Analysis) wurde ein strukturiertes Systemmodell entwickelt, daß allen Anforderungen des Anwenders gerecht wurde. Bild 3 zeigt eine Übersicht über das System. Hier ist das Softwarepaket als Bubble afue dargestellt. Außerdem zeigt dieses Bild die Schnittstellen zur Peripherie. Bild 4 stellt nun die erste Verfeinerung des Bubbles afue dar. Die weitergehenden Verfeinerungen sind nicht mehr dargestellt.

In der MD-Phase (Modulare Design) sollte eine Systemarchitektur entwickelt werden, die folgenden Ansprüchen gerecht wird:

- Unabhängigkeit vom Zielrechner
- Portierbarkeit des Systems und der einzelnen Module
- Optimierung in Bezug auf Änderungsfreundlichkeit

Das Ziel der MD-Phase ist ein hierarchisches, modulares System unter Berücksichtigung von Information Hiding zu bekommen.

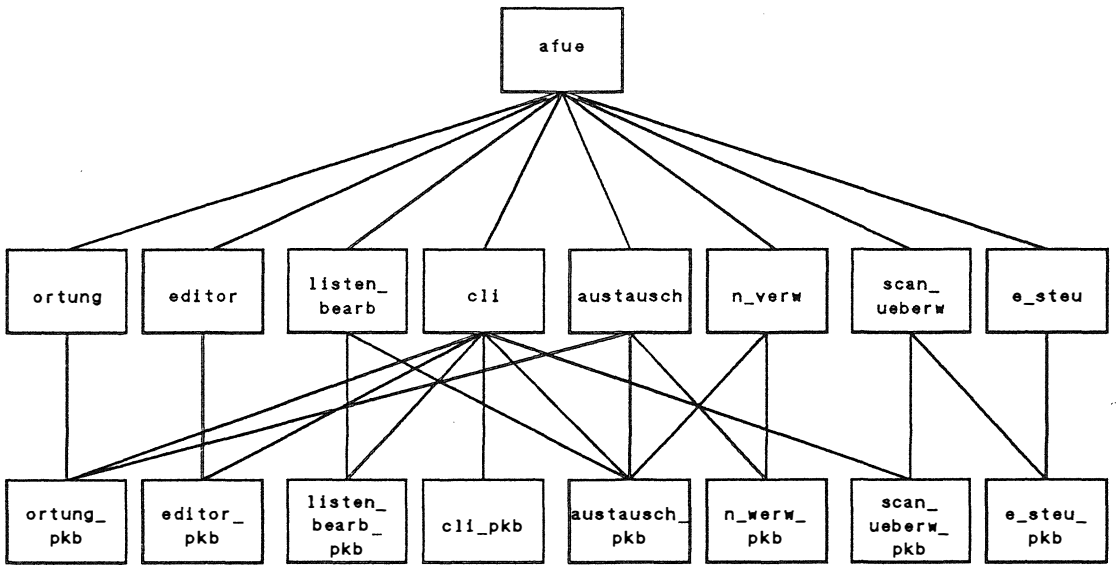


Bild 5: Systemübersicht (MD-Phase)

Aus diesen Gründen wurden nach der Transformation von der SA-Phase in die MD-Phase einige Änderungen vorgenommen:

- Module (Bubbles), die in der SA-Pase nur wegen einer besseren Übersichtlichkeit eingeführt wurden, entfallen wieder

- durch die Transformation entstandene Funktiunktionen wurden wegen ihrer Wichtigkeit und ihres Funktionsumfanges zu Modulen umgewandelt

- zur Verdeutlichung (und auch Realisierung) der Prozeßkommunikation wurden die sogenannten Prozeßkommunikationsblöcke (PKB) eingefügt

Bild 5 zeigt nun die dadurch entstandene Struktur. Man kann daraus die Zugriffs- (Aktivierungs-) Hierarchie der Module untereinander ableiten. Weiter bedeuten die senkrechten Linien zwischen der zweiten und dritten Ebene die Entgegennahme von Aufträgen und die schrägen Linien die Erteilung von Aufträgen. Somit wurde eine Kommunikation von Modulen der gleichen Ebene erreicht.

Die Aufgabe der PKB-Moduole ist die Übergabe von Aufträgen von einem Prozeß (Modul) an einen anderen. Außerdem haben die PKBs die Fähigkeit Aufträge nach dem FIFO-Prinzip zu puffern.

3.2_Realisierung_des_Systems

Der Modul (Task) afue in der obersten Ebene hat lediglich die Aufgabe alle Module in der unmittelbar unter ihm liegenden Ebene zu starten. Dies geschieht in einem Art Handshake-Verfahren (ACTIVATE - SUSPEND - CONTINUE). Dadurch ist gleichzeitig das Hochlaufen des Systems beschrieben.

Prinzipieller Aufbau der Task afue:

```
AFUE:  TASK    GLOBAL;

       ACTIVATE EDITOR;
       SUSPEND;
       ACTIVATE SCAN_ÜBERW;
       SUSPEND;
       ACTIVATE E_STEU;
       SUSPEND;
          .
          .
          .
       ACTIVATE CLI;
       SUSPEND;
       END;
```

Die PROMOD-Module der zweiten Ebene werden alle mit der gleichen Struktur realisiert. Diese Struktur sicht, gezeigt am Modul (Task) e_steu, wie folgt aus:

```
E_STEU:    TASK    GLOBAL;
      /* Initialisierung von globalen und lokalen Daten */
      /* Initialisierung von Schnittstellen             */
      /* Initialisierung des PKBs                       */
           CONTINUE AFUE;
           REPEAT;
             GET_ESTEU_PKB(EST_AUFT_KENN,EST_AUFT_PARA);
             CASE EST_AUFT_KENN
               ALT; /* Abarbeitung des Auftrags 1 */
                    /* z. B. Start Frequenzscan   */
               ALT; /* Abarbeitung des Auftrags 2 */
                    /* z. B. Stopp Frequenzscan   */
                  ...
             FIN;
           END;
           END;    /* TASK E_STEU */
```

Ein Vorteil dieser Struktur ist, daß ein aktivierter Prozeß (Task) immer einen der drei folgenden Zustände inne hat:

- Prozeß durchläuft Initialisierung
- Prozeß wartet auf Auftrag
- Prozeß bearbeitet Auftrag

Als letzter Modul wird der Kommandoentschlüßler gestartet, weil er die Tastatur frei gibt. Ist dies geschehen ist das System betriebsbereit.

3.3 Synchronisation über Prozeßkommunikationsblöcke

Zu jedem PROMOD-Modul der zweiten Ebene gibt es genau einen Modul in der dritten Ebene, den sogenannten Prozeßkommunikationsblock (PKB). Dieser PKB besteht aus zwei Funktionen, der GET-Funktion und der PUT-Funktion.

Die GET-Funktion wird nur von der dazugehörigen Task der zweiten Ebene genutzt, sie wartet (REQUEST PKB_SEMA) auf Aufträge von anderen Tasks (Modulen). Ist ein solcher Auftrag eingetroffen, so werden die Auftragskennung (AUFT_KENN) und die Auftragsparameter (AUFT_PARA) aus dem Auftragspuffer (PKB_POOL) ausgelesen und an die rufenden Task übergeben. Treffen keine Aufträge ein, so verharrt die Prozedur, und somit auch die Task, auf der Anweisung REQUEST PKB_SEMA.

Die PUT-Funktion wird von all den Tasks der zweiten Ebene genutzt, die an die entsprechenden Task einen Auftrag erteilen wollen. Bei der Erteilung eines solchen Auftrags werden die Auftragskennung (AUFT_KENN) und die Auftragsparameter (AUFT_PARA) in den Auftragspuffer (PKB_POOL) eingetragen und anschließend wird die Semaphore freigegen (RELEASE PKB_SEMA).

Aufbau der PKB-Datenstrukturen

```
DCL  ESTEU_PKB_SEMA   SEMA PRESET(0);

DCL  ESTEU_PKB_HEADER  STRUCT
               [ ( ANF_I,
                   END_I,
                   MAX_DS,
                   ANZ_DS,
                   L_PTR,
                   S_PTR )      FIXED     ] GLOBAL;

DCL  ESTEU_PKB_POOL(1:xx) STRUCT
                  [ AUFT_KENN    FIXED,
                    AUFT_PARA    CHAR(10) ] GLOBAL;
```

Bei Initialisierung des PKB-Headers müssen bezüglich des Auftragspuffers (PKB_POOL) folgende Angaben gemacht werden:

- Anfangsindex	ANF_I	typisch: 1
- Endindex	END_I	typisch: 20
- maximale Anzahl Datensätze	MAX_DS	typisch: 20
- Anzahl aktueller Datensätze	ANZ_DS	typisch: 0
- Lesepointer für Pool-Zugriff	L_PTR	typisch: 1
- Schreibpointer für Pool-Zugriff	S_PTR	typisch: 1

Da intern ein zählender Semaphore (z. B. ESTEU_PKB_SEMA) benutzt wird, ist es vom System her eine Pufferung von maximal 127 Aufträgen möglich.

Aufbau der PUT-Funktion

```
PUT_PKB:  PROCEDURE
           (AUFT_KENN FIXED, AUFT_PARA CHAR() IDENT) GLOBAL;
          IF ANZ_DS >= MAX_DS THEN
            /* Fehlermeldung */
          ELSE
            /* Daten in Pool eintragen */
            PKB_POOL(S_PTR).AUFT_KENN := AUFT_KENN;
            PKB_POOL(S_PTR).AUFT_PARA := AUFT_PARA;
            /* Schreibpointer weiterschalten */
            ANZ_DS := ANZ_DS + 1;
            S_PTR := S_PTR + 1;
            IF S_PTR > END_I THEN
              S_PTR := ANF_I;
            FIN;
            /* Semaphore freigeben */
            RELEASE PKB_SEMA;
          FIN;
          END;    /* PROCEDURE PUT_PKB */
```

Aufbau der GET-Funktion

```
GET_PKB:  PROCEDURE
           (AUFT_KENN FIXED  IDENT,
            AUFT_PARA CHAR() IDENT) GLOBAL;
          /* Semaphore belegen */
          REQUEST PKB_SEMA;
          /* Daten aus Pool lesen */
          AUFT_KENN := PKB_POOL(L_PTR).AUFT_KENN;
          AUFT_PARA := PKB_POOL(L_PTR).AUFT_PARA;
          ANZ_DS := ANZ_DS - 1;
          /* Lesepointer weiterschalten */
          L_PTR := L_PTR + 1;
          IF L_PTR > END_I THEN
            L_PTR := ANF_I;
          FIN;
          END;   /* PROCEDURE GET_PKB */
```

3.4_Vorteile_des_Entwicklungskonzepts

Die Entwicklung mit dem Werkzeug PROMOD bringt folgende Vorteile:

- Strukturierung des Systems
- modularer Aufbau des Systems
- Minimalisierung der Schnittstellen
- parallel zur Softwareentwicklung entstehende Softwaredokumentation

Der modularen Aufbau des Systems hat den Vorteil, daß sich jeder Zeit leicht neue Module oder Funktionen integrieren lassen.

Außerdem kann bereits zu einem sehr frühen Zeitpunkt ein Kernsystem zum laufen gebracht werden.

3.5 Testmethodik

Durch den Einstatz des Entwicklungswerkzeuges PROMOD konnten schon frühzeitig statische Fehler erkannt werden. Die Schnittstellen zwischen den Modulen können in der MD-Phase von PROMOD analysiert und auf Konsistenz geprüft werden.

Bei der Art der Aufgabenstellung ist die Hauptschwierigkeit in dem dynamischen Verhalten des Gesamtsystems zu sehen. Deswegen mußte besonderes Gewicht auf den Funktionstest gelegt werden. Hierbei wird das Zusammenspiel der Module untereinander geprüft. Implizit ist damit auch der korrekte Ablauf innerhalb eines Moduls nachgewiesen.

Aufgrund des Software-Systementwurfs ist es möglich einzelne Module rückwirkungsfrei zu prüfen. Dies wird dadurch erreicht, daß für jede Task je eine Schnittstelle (PKB) vorgesehen ist.

In einem abschließendem Test wird dann nochmals das gesamte System geprüft.

4. Resümee

Aufgrund der immer komplexer werdenden Systeme kann auf eine rechnergestützte Softwareentwicklung nicht mehr verzichtet werden.

Der Vorteil des modularen Aufbaus geht soweit, daß das komplette Kernsystem bestehend aus den Modulen afue, editor, editro_pkb, cli, cli_pkb und bild_verw fast ohne Änderung von einem frühren Projekt übernommen werden konnte. Mittlerweile ist dieses Kernsystem auch in weiteren Projekten Grundlage für die Softwareentwicklung. Daraus ergibt sich natürlich eine erhebliche Zeit- und damit auch Kostenersparnis.

Der Jobtransferdienst für IBM-Großrechner mit dem Betriebssystem VM

P. Holleczek, R. Kummer

Regionales Rechenzentrum Erlangen

(R R Z E)

1. Einleitung

Der Job-Transfer-(JT bzw. RJE = Remote Job Entry)-Dienst hat eine besondere Bedeutung als Kommunikationsdienst in der Forschungslandschaft.

Er erlaubt insbesondere einen wirtschaftlichen Zugriff auf Super-Rechner (z.B. CRAY, ETA) deren interaktiver Zugang oft stark eingeschränkt ist. Dabei führt der Job-Zugang klassischerweise nicht an den Super-Rechner selbst, sondern an einen größeren Universalrechner (z.B. DEC, CDC, IBM), der über genügend Hintergrund Speicherkapazität verfügt und der seinerseits mit Spezialprozeduren an den Super-Rechner angeschlossen ist.

Im Rahmen der Einführung von standardisierten Protokollen im Bereich des Deutschen Forschungs-Netz (DFN) kommen als erste Stufe für den Jobtransfer Protokolle zum Einsatz, die im Sinne des ISO/OSI Referenzmodells einschließlich der Transportschicht auf ISO-Standards, darüber jedoch auf einer nationalen Vereinbarung (DFN-RJE) beruhen. Dieser Dienst soll, sobald verfügbar, durch den ISO-Dienst JTM abgelöst werden.

Im vorliegenden Fall bestand die Aufgabe, den Dienst DFN-JT für das Betriebssystem IBM-VM zu implementieren. Bei dem Betriebssystem IBM-VM liegt, im Gegensatz zu dem bekannten "batch"-orientierten Betriebssystem MVS, das Schwergewicht auf einem interaktiven Zugang.

Als Programmiersprachen stehen PASCAL mit Betriebssystem-Aufrufen (Fa. IBM) und PEARL (Fa. Werum) zur Verfügung.

2. Aufgabenstellung

Im Rahmen eines gemeinsamen DFN-Projektes zwischen der GMD Darmstadt, dem ENC (European Networking Center) der Fa. IBM, Heidelberg, dem Rechenzentrum der Universität Stuttgart (RUS) und dem Regionalen Rechenzentrum der Universität Erlangen-Nürnberg (RRZE) wurde für IBM-Anlagen mit dem Betriebssystem VM ein Remote Job Entry- (RJE) und ein Filetransfer- (FT) Protokoll implementiert. Dabei entfiel an das RRZE die Aufgabe, das RJE-Protokoll zu realisieren. Da bislang noch keine "stabile" Normung für diesen Dienst vorliegt (JTM), wurde auf eine einfache Art eines RJE-Protokolls zurückgegriffen, das von der PIX-Arbeitsgruppe (/PHB85/) erstellt wurde.

Zur Entwicklung des Programms für RJE gehört auch die Festlegung einer Benutzer- und Operateurschnittstelle und der Anschluß eines BATCH-Verarbeitungssytems (Abwicklung von Aufträgen im Hintergrund /VMB85/) innerhalb des VM-Betriebssystems.

Der RJE-Dienst ist asymmetrisch. Er unterscheidet zwischen Jobs, die von einem Eingaberechner zu einem Verarbeitungsrechner und Output, der vom Verarbeitungsrechner zum Ausgaberechner transportiert wird. Der Verarbeitungsrechner muß die ankommenden Jobs an das BATCH-System weiterreichen, die fertigen Ergebnisse sammeln und an den Ausgaberechner zurückschicken. Eingaberechner und Ausgaberechner sind oft identisch.

Da in der Implementation in der Regel nicht zwischen Eingabe-/Ausgabe-/Verarbeitungsrechner unterschieden wird, muß das JT-Programm gleichzeitig folgende Datenströme verwalten:

Empfangen & Senden von Jobs vom bzw. zum Netz
Empfangen & Senden von Output vom bzw. zum Netz
Senden von Jobs zum BATCH-System
Empfangen von Output vom BATCH-System
Eingaben und Rückmeldung von bzw. zu einer Benutzerschnittstelle

Alle diese Vorgänge finden außerdem gleichzeitig für verschiedene logische Verbindungen statt.

Das JT-Programm hat also im hohen Maße viele Aufgaben gleichzeitig zu tun, die zudem meist asynchron angestoßen werden.

Es liegt daher nahe, zur Entwicklung des JT-Programms parallele Prozesse als Strukturierungshilfsmittel heranzuziehen, nach Möglichkeit unterstützt durch eine geeignete Programmiersprache.

Da das RJE-Protokoll lediglich die Schicht 7 (Anwendungsschicht) des ISO-Schichtenmodells darstellt, sind zumindest die Schichten 1 bis 4 vorauszusetzen. Auf die Schichten 5 (Sitzungsschicht) und 6 (Präsentationsschicht) konnte im Rahmen dieses Projektes verzichtet werden. Die Schicht 4 (Transportschicht, /T7084/) liegt für die VM-Systeme in verschiedenen Formen vor (s. Kap. 3).

3. Einbettung in die VM-Welt.

Das Grundbetriebssystem CP (Control Program, /CP87/) der IBM-Rechner der Serie 43xx, 93xx und 3090 ermöglicht die Bereitstellung von sogenannten Virtuellen Maschinen (VM). Darauf aufbauend findet innerhalb der Virtuellen Maschinen vor allem das Dialog- und den Programmablauf unterstützende Betriebssystem CMS (Conversational Monitor System, /CMS87/) Verwendung. Im folgenden wird also immer davon ausgegangen, daß sowohl das CP als auch das CMS zur Verfügung stehen.

Es laufen nicht alle die im Zusammenhang mit diesem Projekt entstandenen Programme in einer einzigen VM ab. Es ergibt sich folgendes Übersichtsbild für die verschiedenen Virtuellen Maschinen:

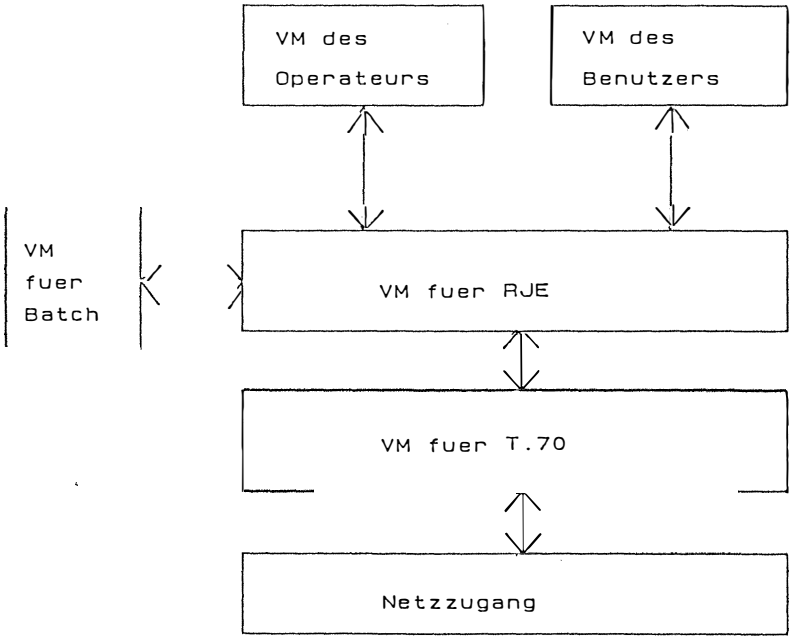


Bild 1: Übersicht über die verschiedenen Virtuellen Maschinen

Zur Beschreibung der Kommunikationen (Pfeile) zwischen den Virtuellen Maschinen sei auf das Kapitel 4.1 verwiesen.

Die einzelnen Virtuellen Maschinen lassen sich kurz, wie folgt, beschreiben:

- VM des Benutzers: Dem Benutzer des RJE stehen Kommandos zur Verfügung, um BATCH-Aufträge oder Listen an ferne Rechner zu senden und administrative Befehle einzugeben, wie z.B. den Status der Jobs abzufragen.

- VM des Operateurs: Bei einem Operateur handelt es sich um einen privilegierten Benutzer, der zwar die gleichen Kommandos wie ein normaler Benutzer absetzen kann, aber nicht der Beschränkung auf seine eigenen Jobs unterliegt. Er kann also alle Jobs von allen Benutzern in der RJE-Maschine administrieren.

- VM für BATCH: In dieser Virtuellen Maschine befindet sich das Programm zur Abwicklung von BATCH-Aufgaben. Dieses Programm trennt die Steueranweisungen von möglichen Eingabedaten, führt die Steueranweisungen aus und liefert die Ergebnisse des Auftrags an die RJE-Maschine zurück.

- VM für RJE: Wie bereits in Kapitel 2 erwähnt, wickelt die Maschine das DFN-RJE-Protokoll ab. Damit verbunden ist das Entgegennehmen, Speichern und Weiterleiten von BATCH-Aufträgen und -Ergebnisprotokollen aus dem DATEX-P-Netz, vom Benutzer und von der BATCH-Maschine. Außerdem müssen die Administrationsbefehle der Benutzer und des Operateurs ausgeführt werden.

- VM für T.70: Für Rechner mit Betriebssystem VM existieren zwei Programmpakete zur Abwicklung des Transportprotokolls:
  - Das Programm "TLMAIN" vom ENC Heidelberg (/ENC86/), das fertig gebunden vorliegt.
  - Die Makro-Sammlung "OTSS" (/OTSS87/) als offizielles Produkt der Firma IBM setzt wiederum die Makro-Sammlung "OSNS" (/OSNS87/) voraus. Dabei können die Prozeduren von OTSS nicht direkt genutzt werden, sondern müssen von einem in der Transportmaschine ablaufenden Anwendungsprogramm aufgerufen werden.

  Die genannten Transportprogramme erfüllen die CCITT-Norm (Comité Consultatif International Telegraphique et Telephonique) T.70 bzw. die Klasse 0 der ISO-Norm für das Transportprotokoll. Sie sind jeweils in PASCAL programmiert und fertig ausgetestet. Als zusätzliche Funktionenmenge sind Möglichkeiten zur Gebührenerfassung und zur Erstellung von Statistiken implementiert.

- Netzzugang: Um den Programmen für die Transportschicht (TLMAIN oder OTSS mit OSNS) den Zugang zu einem X.25-Netz (z.B. dem DATEX-P-Netz der Deutschen Bundespost) zu ermöglichen, sind verschiedene Vorrechner nötig. Wird das Programm TLMAIN benutzt, ist ein Rechner der Reihe Serie/1 notwendig. Bei der Verwendung von OTSS und OSNS ist ein Vorrechner des Typs 37X5 oder 3720 Voraussetzung. Als billigere Alternative dazu ist es auch möglich, den Zugang über den "Integrated Communications Adapter" (ICA) zu nutzen, so daß auf einen Vorrechner verzichtet werden kann.

4. Schnittstellen der RJE-Maschine zum vorhandenen Betriebssystem

4.1. Schnittstellen zur Kommunikation zwischen Virtuellen Maschinen bzw. zum Benutzer

Das Betriebssystem CP bietet zur Kommunikation zwischen Virtuellen Maschinen das IUCV-Paket (Inter-User Communications Vehicle, /CP87/) an, das den Austausch von sogenannten "Messages" (Nachrichten) erlaubt (Senden und Empfangen). Dadurch können Informationen von begrenzter Länge (z.B. 80 Zeichen einer Bildschirmzeile) von einer VM zu einer anderen gesendet werden:

MESSAGE userid ..... messagetext .....
mit userid ::= Name der Empfänger-VM.

Statt des CP-Kommandos "MESSAGE" wird das CMS-Kommando "TELL" verwendet, um auch Benutzer auf anderen über RSCS ("Remote Spool") erreichbaren Rechnern zu erreichen.

Müssen größere Datenmengen ausgetauscht werden, gibt es die Möglichkeit, Dateien von einer VM zu einer anderen zu senden. Um einen File senden zu können, benötigt man das CMS-Kommando SENDFILE (/CMS87/), für das Empfangen steht das CMS-Kommando RECEIVE zur Verfügung.

Mit Hilfe dieser Kommandos ist ein vollständiger Datenaustausch zwischen Virtuellen Maschinen sichergestellt. Wie diese Kommandos, die zunächst Dialogkommandos sind, vom RJE-Programm aus angesprochen werden, ist in Kapitel 4.2. behandelt.

Die Benutzer- und Operateurkommandos können von EXEC-Routinen, in REXX (Job-Control-Sprache /REX 83/) geschrieben, entgegengenommen und von diesen syntaktisch analysiert werden. Ist das Kommando syntaktisch richtig, wird es mit dem CMS-Kommando "TELL" and die RJE-Maschine gesendet.

4.2. Aufrufe des PEARL-Betriebssystems

Da die Implementierung in der Programmiersprache PEARL (/PEA78/, /PEA80/) erfolgt (siehe dazu Kapitel 5.2.1.) und sowohl CP als auch CMS die Sprache nicht unterstützen, existiert ein eigenes Laufzeit- und Betriebssystempaket für PEARL-Programme. Dieses Paket verwendet einige CMS-Makros bzw. CP-Prozeduraufrufe, die nachfolgend angeführt sind (/KNE86/):

- Makros für File-E/A
- Makros für Terminal-E/A
- Makros für Drucker-Ausgabe
- Supervisor-Calls und CP-Timer

5. Verwendete Spezifikationstechnik und Programmiersprachen

5.1. Spezifikationstechnik PASS

Am Regionalen Rechenzentrum Erlangen, und inzwischen auch am ENC, wird seit einiger Zeit mit der Spezifikationstechnik PASS (Parallel Activities Specification Scheme, /FLE84/) gearbeitet. Sie wurde in dem Arbeitstreffen "Spezifikationstechniken im DFN" (/SPE83/) im November 1983 in Darmstadt vorgestellt. Unter anderem wurde sie in dem DFN-Projekt zur Implementierung von Basis-Diensten (FT mit T.70, X.28/X.3/X.29, /BAS83/) eingesetzt.

Diese Methode ist graphisch orientiert und erlaubt es, parallele Prozesse und ihre Interaktionen über Botschaften oder gemeinsame Objekte in übersichtlicher Weise darzustellen. Eigenständige Prozesse, die unabhängig voneinander (parallel) ablaufen können, sind als Strukturierungshilfsmittel allgemein bewährt, da sie voneinander unabhängige Abläufe leicht darstellbar machen. Als Hilfsmittel, um die Kommunikation zwischen den Prozessen anschaulich zu beschreiben, dienen Botschaften. Sie erlauben durch ihre kompakte Schreibweise, unabhängig von Implementationsdetails, die Synchronisation und den Datenaustausch zwischen Prozessen transparent zu machen.

Für eine genaue Beschreibung sei auf die angegebene Literatur verwiesen.

Das gesamte RJE-Protokoll ist mit der Spezifikationstechnik PASS beschrieben.

Als Beispiel sei auf die Bilder 2 ("Kommunikationsstruktur" = Darstellung aller Prozesse mit ihrer Kommunikation) und 3 ("Ablaufsteuerung" = Feinstruktur eines Prozesses) in Kaptiel 6 verwiesen.

5.2. Verwendete Programmiersprachen

5.2.1. PEARL als Sprache paralleler Prozesse

Da die Methode PASS eine klare Darstellung paralleler Prozesse erlaubt, liegt es nahe, eine Programmiersprache zu verwenden, die ebenfalls die Darstellung paralleler Prozesse erlaubt, um eine möglichst einfache Abbildung der Spezifikation auf Programmcode zu erzielen. In dem in Kapitel 4.1. erwähnten Projekt ("Implementierung von Basis-Diensten") wurde bereits erfolgreich die Abbildung von PASS auf die Programmiersprache PEARL (/PEA78/, /PEA80/) durchgeführt.

Es ist möglich, die in der Spezifikation definierten Botschaften in PEARL mit Hilfe von Prozeduren und Semaphoren nachzubilden (siehe 6.1.). Sogenannte "Schreibprozeduren" sollen dabei das Senden nachbilden, während "Leseprozeduren" das Empfangen darstellen. Um den Datenaustausch zu synchronisieren, werden "REQUEST"- und "RELEASE"-Operationen auf Semaphoren verwendet.

Für IBM-Rechner mit den Betriebssystemen MVS und VM existiert von der Firma WERUM, Lüneburg, ein PEARL-Compiler und -Laufzeitsystem (/WER78/, /WER80/). Dabei werden PEARL-Programme von einem PL/I geschriebenen Compiler im Assemblercode übersetzt und dann mit Hilfe des Assembler-Übersetzers in Maschinencode transferiert. Anschließend werden übersetzte Programme mit der PEARL-Laufzeitbibliothek zu einem lade- und lauffähigen Modul gebunden.

5.2.2. Zugriff auf Systemroutinen über PASCAL-Prozeduren

Wie bereits in Kapitel 3.1 erwähnt, gibt es CP- bzw. CMS-Kommandos, um die Kommunikation zwischen Virtuellen Maschinen zu erzielen. Um einen möglichst bequemen Anschluß für die FT- und RJE-Protokollprogramme zur Verfügung zu stellen, wurde vom ENC im Rahmen der Implementierung des Transportprotokolls das Prozedurenpaket ("Message Passing Handler" MPH4 /ENC86/) realisiert. In diesem Paket sind die wichtigsten Aufrufe zur Kommunikation mit der VM von T.70 ("CONNECT REQUEST", "DATA REQUEST", etc.) und mit anderen Virtuellen Maschinen ("SEND_MESSAGE", "RECEIVE_MESSAGE", etc.) zusammengefaßt. Eine genauere Beschreibung der Prozeduraufrufe befindet sich in /HOL85/.

Für die Durchführung weiterer CMS- und CP-Kommandos gibt es in PASCAL/VS (/PAL85/) die Möglichkeit, die Prozedur "CMS" (/PAP85/) zu verwenden. Sie dient zum Absetzen von Kommandos in CMS-Umgebung und liefert in Form eines ganzzahligen Rückgabeparame-

ters die Information, ob die Aktionen erfolgreich abgeschlossen oder mit einem Fehler beendet wurden. Damit können auch ganze EXEC-Routinen, z.B. REXX geschrieben, abgewickelt werden.

Da das RJE-Protokoll in PEARL codiert wurde, andererseits das MPH4 in PASCAL programmiert ist, war es notwendig, einen sauberen Übergang von PEARL nach PASCAL zu schaffen. Da in unserem Fall lediglich PASCAL-Prozeduren von PEARL-Programmen aufgerufen werden, reduziert sich das Problem auf den Vergleich der Datenstrukturen und auf einen geeigneten Mechanismus, um von PEARL-Programmen aus PASCAL-Prozeduren aufzurufen und die Parameter zu übergeben.

Bei den Datenstrukturen wurde auf die Verwendung von Strukturen ("RECORDS") verzichtet, da der Aufbau in den beiden Programmiersprachen zu unterschiedlich ist. Die folgende Liste gibt eine Abbildung von PASCAL- auf PEARL-Datentypen wider.

PASCAL	PEARL	
INTEGER	FIXED (n)	mit 16 < = n < = 32
POINTER	REF	
PACKED ARRAY [1..2*n] OF CHAR	(n) CHAR(2)	mit n > = 1
PACKED ARRAY [1..2*n] OF CHAR	CHAR (2*n)	mit n > = 1
STRING (n-2)	CHAR (n)	mit n > = 3, CHAR(1) = 0 CHAR(2) = tatsächliche Länge, STRING (1..n-2) = CHAR (3..n)

Beim Aufruf von PASCAL-Prozeduren sind einige Konventionen zu beachten (siehe dazu auch den "Programmer's Guide", /PAP85/). Insbesondere bei der Registerbelegung und den damit verbundenen reservierten Speicherbereichen von globalen und lokalen Daten sind strenge Vorgaben zu befolgen. Da sich die Firma WERUM in ihrer Implementation des PEARL-Compilers und -Laufzeitsystems sehr genau daran gehalten hat (/KNE86/), gibt es an dieser Stelle keine Probleme. Trotzdem sind bei der Übergabe von Prozedurparametern einige Regeln zu beachten, wie z.B.

- Die Parameterübergabe kann nur per Adresse, also "IDENT" (PEARL) bzw. "VAR" (PASCAL), erfolgen.
- Da PEARL im ASCII-Alphabet arbeitet, PASCAL dagegen im EBCDIC, müssen bei Text- bzw. Zeichenanalysen Codeumwandlungstabellen verwendet werden.

Da die Prozedurschnittstelle "MPH4" für die RJE- und die FT-Implementierung einheitlich bleiben soll, wurde für den Anschluß dieser Prozeduren an PEARL eine Prozedurzwischenschicht "PMPH4" in PASCAL entwickelt, die die obigen Regeln beachtet.

6. Implementation

6.1. Umsetzung der Prozesse und Botschaften aus der Spezifikation in PEARL-Programme

Die in der Spezifikation angegebenen Prozesse sind direkt in PEARL-Prozesse umgesetzt worden, wobei die Sender- und Empfängerprozesse mehrfach vorkommen. Die in PASS formulierten Botschaften werden in PEARL durch Prozeduren simuliert.

In der nachfolgenden "Kommunikationsstruktur" sind die PASS-Prozesse bzw. PEARL-Prozesse angegeben:

PASS-Prozeß	PEARL-Prozeß/-Prozedur
DISTRI	DISTRI
EMPFANG	EMPF
RDRWAI	RDRWAI
SENDER	SENDER
SERVER	STRTTA
TIMER	TIMER
TYGET	TYGETE

Um den Zusammenhang zwischen den Ablaufdiagrammen bzw. den Benutzermaschinen und dem Programmcode herzustellen, wurden die Knoten- und Botschaftsnamen als Kommentare in die Programme übernommen.

Bild 2 zeigt die am RJE-Programm beteiligten Prozesse und die zwischen ihnen ausgetauschten Botschaften (Kommunikationsstruktur aus PASS).

Bild 2: Kommunikationsstruktur

mit i gerade (='EMPFANG'),
j ungerade (='SENDER')

6.2. Zustandsdiagramme der Prozesse und des Gedächtnisses

Jeder der spezifizierten bzw. programmierten Prozesse befindet sich zu bestimmten Zeitpunkten in einem definierten Zustand. Fast alle Prozesse wechseln dabei zwischen den Zuständen "nicht aktiv" und "aktiv". Besondere Zustände bzw. Zustandsübergänge haben nur die Prozesse SENDER und EMPFANG, wie das nachfolgende Bild, vereinfacht, als Ausschnitt aus einer PASS-"Ablaufsteuerung" zeigt.

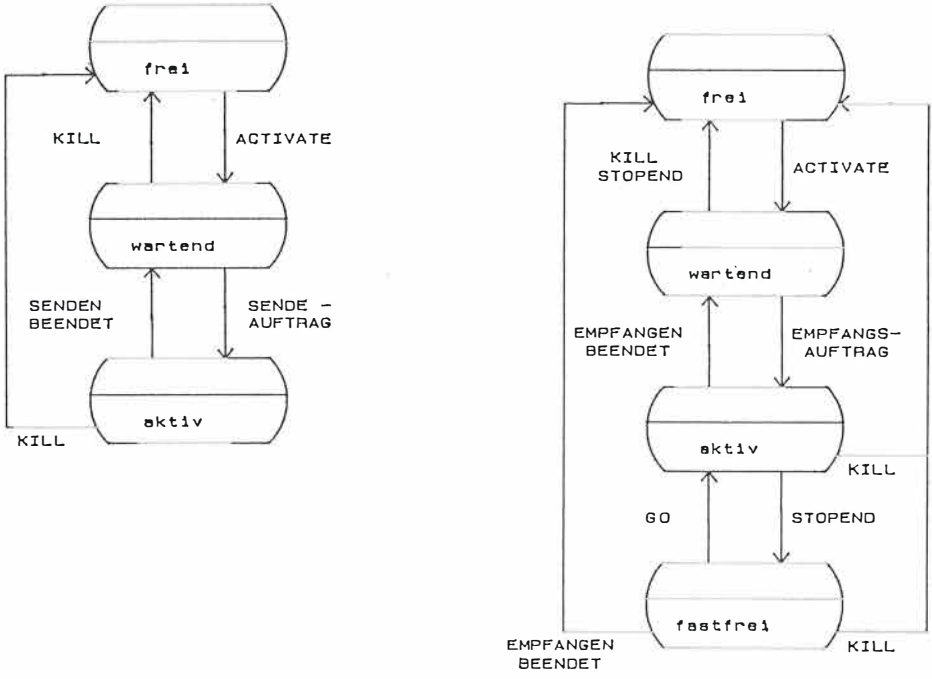


Bild 3: Zustandsdiagramm der Prozesse EMPFANG (links) und SENDER (rechts)

7. Erfahrungen

Die gewählte Implementationstechnik, d.h.

- die strikte Einhaltung des Software Life Cycle mit typ-down-Entwurf,
- die in der Spezifikations-Technik vorgesehenen Kommunikationsmittel zwischen parallelen Prozessen,
- die schematische Umsetzung der Spezifikation im Programm-Code,

erwies sich als außergewöhnlich erfolgreich. So konnte die vorgesehene Implementationszeit von 4 Mannjahren deutlich unterschritten werden, obwohl im Zusammenspiel von PEARL-/PASCAL-Laufzeitsystem und VM-Betriebssystem eine Reihe von Tücken vorborgen waren.

Mittlerweile ist RJE-VM an einer Reihe von Universitäten auf Rechner der unterschiedlichsten Größenordnung im Einsatz, angefangen z.B. von einer 4361 (ca. 1 MIPs) bis zu einer 3090-200 (ca. 30 MIPs). Das Programm hat sich bislang als stabil erwiesen, was die Wünsche nach einer immer komfortableren Benutzerschnittstelle nicht minderte.

8. Literatur

/BAS83/ Andres, Fleischmann, Hillmer, Holleczek, Kummer: "Pflichtenheft für die Basis-Dienste des Erlanger DFN-Anschlusses", Interner Arbeitsbericht Nr. 193, RRZE, Erlangen, 1983

/CMS87/ IBM Corporation: "VM/SP CMS Command and Macro Reference, Release 5", 1987

/CP87/ IBM Corporation: "VM/SP CP Command Reference for General Users, Release 5", 1987

/ENC86/ Haas, Kropp, Reinhardt, Schulz: "OSI Transport Layer Implementation for VM/SP", European Networking Center, Heidelberg, 1986

/FIS86/ J. Fischer: Private Mitteilungen, Erlangen, 1986

/FLE84/ A. Fleischmann: "Ein Konzept zur Darstellung und Realisierung von verteilten Prozeßautomatisierungssystemen", Mitteilungsblatt des RRZE, Erlangen, 1984

/HOL85/ R. Holliday: "MPH4 Functions and Internals", Heidelberg, 1985

/KNE86/ E. Kneuer: Private Mitteilungen, Lüneburg, Erlangen, 1986

/OSNS87/ IBM Corporation: "Open Systems Network Support (OSNS)", Release 2.0, Stuttgart, 1987

/OTSS87/ IBM Corporation: "Open Systems Transport and Session Suppport (OTSS)", Release 2.0, Stuttgart, 1987

/PAL85/ IBM Corporation: "PASCAL/VS - Language Reference Manual - Program Number: 5796-PNQ", 1985

/PAP85/ IBM Corporation: "PASCAL/VS - Programmer's Guide - Program Number: 5796-PNQ, 1985

/PEA78/ DIN 66253 Teil 2: "Programmiersprache PEARL, FULL PEARL", Beuth Verlag, Berlin, 1980

/PEA80/ DIN 66253 Teil 1: "Programmiersprache PEARL, BASIC PEARL", Beuth Verlag, Berlin, 1978

/PHB85/ Zentrale Projektleitung - DFN: "Protokollhandbuch Version 2", 1983

/REX83/ IBM-Corporation: "VM/SP System Product Interpreter Reference, Release 3", 1983

/SPE83/ Bauerfeld, Henken - ZPL-DFN: "Spezifikationstechniken im DFN", Tagungsband zum Arbeitstreffen am 28.-30. November 1983 in Darmstadt

/T7084/ ISO: "OSI - Connection Oriented Transport Protocol Specification, Draft International Standard ISO/DIS 8073", 1984

/VMB85/ IBM Corporation: "VM Batch Subsystem Program Description / Operations Manual", 1985

/WER78/ Werum, Windauer: "PEARL, Process and Experiment Automation Realtime Language", Vieweg & Sohn, Braunschweig, 1978

/WER80/ WERUM DV-Systeme GmbH: "PEARL Programming System for IBM 43xx with CMS - User's Guide", Lüneburg, 1986

Aufbau eines CIM-Konzeptes für einen Serienfertiger im Werkzeugmaschinenbau

Referent: Hans Weller
r w t GmbH
Talangerstr. 5-7
8033 Krailling

Veränderte Absatzmärkte verändern auch den Fertigungsmarkt. Diese Umstrukturierung beinhaltet Konfliktsituationen in allen Bereichen. Dieser Beitrag gibt die Erfahrung eines Systemhauses mit einem Werkzeugmaschinenhersteller wieder, bei welchem zwischenzeitlich CIM in der Einführungsphase ist. Welche Bereiche wie angegangen wurden, ist Thema dieses Vortrages.

Quellen:

- Die modulare Fabrik, von Prof. Horst Wildemann (gfmt)
- MAP Datenkommunikation in der automatisierten Fabrik, von Jürgen Suppan-Borowka, Thomas Simon (Datacom)
- Handbuch der flexiblen Automation, von Hans B. Kief (NC-Handbuch-Verlag)

Einführung

Reorganisation der Produktion:

Die veränderten Absatzmärkte mit ihren Auswirkungen auf die Produktion verlangen eine Reorganisation der Fertigung. Während in der Vergangenheit die quantitative Versorgung von Märkten Priorität hatte, liegt die Zukunft auf dem Schwerpunkt der Qualität. Diesen Trend kann der einzelne Verbraucher am eigenen Kaufverhalten nachvollziehen; beispielsweise sei hier nur das Käuferverhalten bei

- Kleidung
- Fotoapparaten
- Kraftfahrzeugen

erwähnt.

Diese Tatsache stellt die Fertigung vor neue Aufgaben wie

- sinkende Stückzahlen
- höhere Variantenvielfalt
- kürzere Produktlebenszeiten
- geringere Verkaufspreise
- höhere Lohnkosten
- mehr Wettbewerb, national und international

Aufgaben, die bei einer ersten Betrachtungsweise eine Reihe von unvereinbaren Zielkonflikten herausstellen.
Die Bewältigung dieser Aufgaben hat im Fertigungsbereich Aus-

wirkung auf fast alle Mitarbeiter, auf die Investitionsstrategie, auf den eingesetzten Maschinenpark sowie der bestehenden Arbeitsteilung in der Fertigungssegmentierung sowie den Einsatz von Technologien.

Den Überbegriff dieser Technologien bildet ein 'C' = Computer, die mittelfristige Zielsetzung den Begriff 'CIM' = Computer Integrated Manufacturing.

Die Einführung der C-Technologien vollzog sich in den Jahren 1965 - 1975; im Fertigungsbereich mit der NC = Numeric Control-gesteuerten Werkzeugmaschine, im Bereich der Arbeitsvorbereitung mit NC-Programmiersystemen, in der Konstruktion mit CAD-Systemen.

Der Einführungsprozess verlief teilweise schleppend, zumal die NC-Technologie es mit sich brachte, daß der Mensch als ablaufbestimmender Faktor vom Fertigungsprozess der Maschinen **entkoppelt wurde.** Sein Arbeitsinhalt veränderte sich, indem er die komplette

- Fertigungsvorbereitung
- die Versorgung der Maschine mit Betriebsmittel und Material
- die Ver- und Entsorgung mit Informationen

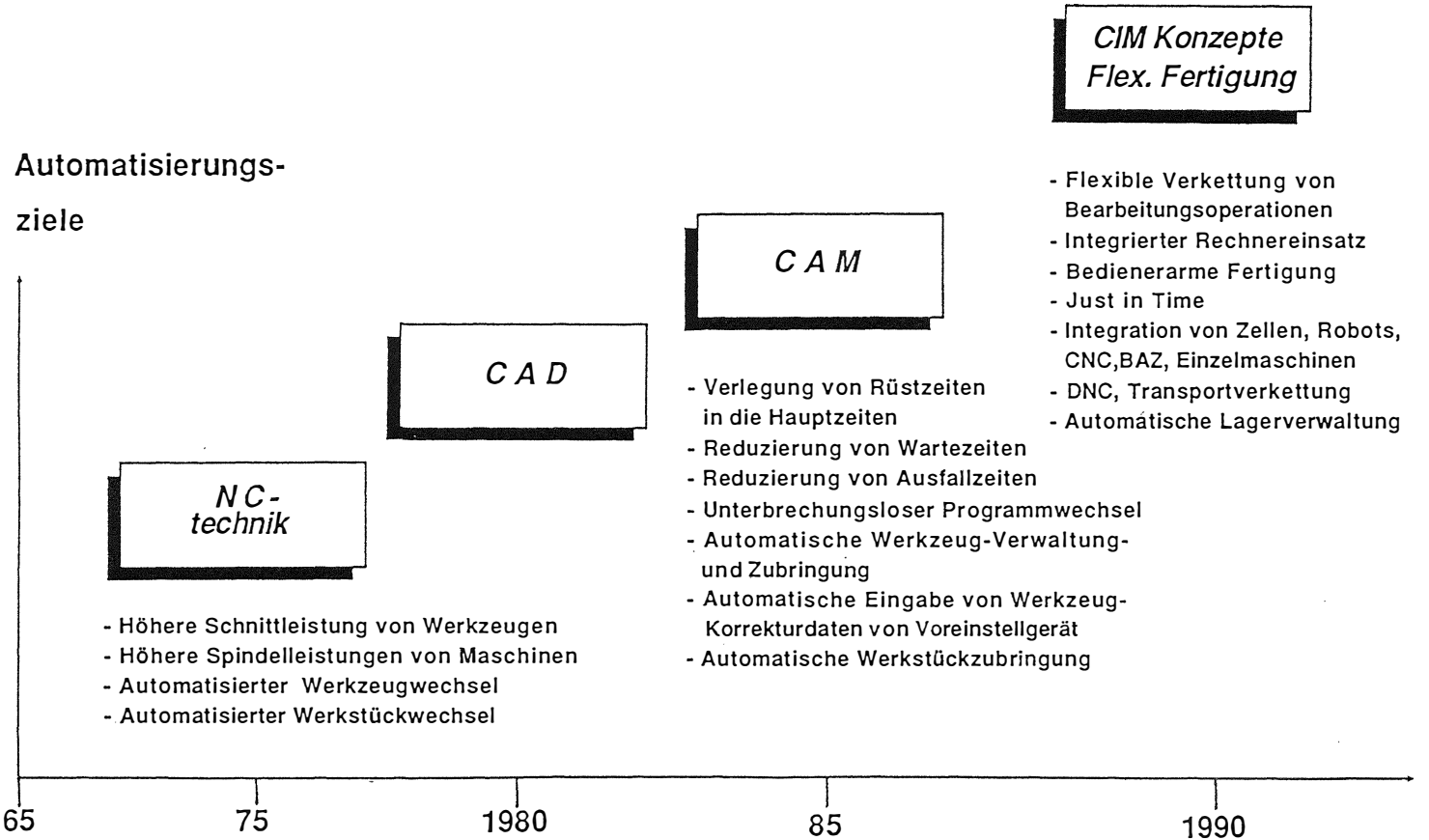
vor Start des Fertigungsprozesses durchzuführen hatte.

Die Rationalisierungsanstrengungen lagen vorwiegend in der Erhöhung der Hauptzeiten. Dies wurde erreicht durch

- höhere Schnittleistung von Werkzeugen
- höhere Spindelleistungen von Maschinen
- automatischer Werkzeugwechsel
- automatischer Werkstückwechsel

Die weiteren Ziele im Fertigungsbereich waren Verkettungsziele

AUTOMATISIERUNGSSCHRITTE



und die Verringerung von Nebenzeiten. Dies wurde durch

- Verlagerung der Rüstzeiten in die Hauptzeiten
- Reduzierung von Wartezeiten
- Reduzierung von Ausfallzeiten
- unterbrechungslose Programmwechsel
- automatische Werkzeugverwaltung und Zubringung
- automatische Eingabe der Werkzeugkorrekturwerte vom Voreinstellgerät in die CNC
- automatische Werkstückzubringung

Diese Automationsziele waren Voraussetzung für eine weitergehende Reduzierung der Maschinenstillstandszeiten wie

- Maschinenbedienung durch einen Operator
- unbeaufsichtigter Durchlaufbetrieb in Pausen und Nachtschichten.

'CAM' = Computer Automated Manufacturing, die Rechnergesteuerte Fertigung, ist eine weitere Automatisierungsstufe.
Die konsequente Ausnutzung von Maschinen und Verkettungssystemen führten zu einer Komplettbearbeitung, der flexiblen Verkettung von einzelnen Bearbeitungsoperationen ohne manuellen Eingriff.
Der integrierte Einsatz von Rechnern und Microprozessoren auf allen Gebieten ist ein weiterer Meilenstein, d.h., 'CIM'.
Die Zielsetzungen eines solchen Systems sind

- Just in Time - Fertigung
- bedienerarme Fertigung
- Integration des Maschinenparks von CNC-, Sonder-, Einzel-

maschinen und Bearbeitungszentren durch DNC-Systeme, Transportverkettung, autom. Lagersysteme im Datenverbund und mit entsprechender Rechnervernetzung.

Qualitative Vorteile der CIM-Technologie sind z.B.

- eine schnellere Angebotsabgabe
- genauere Kalkulation
  größere Flexibilität in der Planung und Fertigung
- kostengünstigere Fertigung trotz Anpassung an die Marktlage

Quantifizierbare Vorteile verspricht man sich mit CIM:

Verringerung von Entwurfskosten	15 - 3o %
Verringerung von Durchlaufzeiten	3o - 6o %
Steigerung in Produktivität und Manntagen	4o - 7o %
Verringerung des Materials in der Fertigung	3o - 6o %
Steigerung der Maschinennutzungszeiten	- 1oo - 2oo %
Ausschuß-Reduzierung	5o - 8o %
Steigerung der Kreativität von Ingenieuren	3 - 35 MT

Quellen: Com CAD/CAM
National Research Council USA

Das sind stolze Ziele, die viel Mühe und Kraft kosten, um sie zu erreichen. Wie ein Mittelstandsbetrieb eine solche Aufgabe angeht, zeigt nachfolgendes Fallbeispiel.

In jedem mittleren und größeren Unternehmen sind organisatorisch bestehende Strukturen anzutreffen. Diese Aussage gilt auch für die Datenverarbeitung Hard- und Software. Die Aufgabe eines Systemintegrators, wie rwt in diesem Falle, ist aus diesem Gesichtspunkt heraus zweigeteilt :

1. Neue Technologien einzuführen und bisher manuell durchgeführte Arbeiten zu automatisieren, beinhaltet den ersten Teil der Aufgabe,
2. bestehende Hard- und Software-Anwendungen in den neuen Lösungsansatz zu integrieren, den zweiten Teil der Aufgabe.

Dieser zweite Aufgabenteil betrifft nicht nur die Hard- und Softwareproblematik. Das Problemfeld umfaßt die Unternehmensorganisation, die im jeweiligen Anwendungsbereich tätigen Personen und die teilweise über 1o bis 15 Jahre eingesetzte Hard- und Software für den einzelnen Anwendungsbereich. Problematisch ist an dieser Aufgabe die Tatsache, daß eine Reihe von Zielsetzungen **Konfliktsituationen** beinhalten.

Grundsätzliche Zielkonflikte sind beispielsweise der Wunsch des Anwenders, 'Standardisierte Software' einzusetzen mit den Vorteilen

- permanente Weiterentwicklung dieser Pakete
- hohe Praxisnähe
- kleine Wartungskosten
- fehlerfreie Funktion
- keine Entwicklungskosten

Bezogen auf die Forderung einer EDV-Abteilung sind diese Vorteile

- bestehende Hardware zu nutzen
- Adaption, Modifikation und bestehende Anwendungspakete zu integrieren

Dabei fallen selbstverständlich Entwicklungskosten an. Diese Softwareversion ist separat zu behandeln, also kein 'Standard'. Die Wartung muß separat durchgeführt werden. Diese Version ist nicht automatisch in die Weiterentwicklung integrierbar.
Hier hilft nur die Kooperation zwischen der Anwendung und dem Lieferanten mit dem Ziel, diesen Zielkonflikt im Vorfeld auszuräumen. Die gemeinsame Zielsetzung, ein solches Projekt zwischen dem Hause Deckel und rwt zu starten, beinhaltete verschiedene Motivationspunkte.

Deckel-Ziele waren beispielsweise

- die Beteiligung von Deckel an rwt, und daraus einen Synergieeffekt zu generieren
- die Nutzung der Erfahrung von rwt in der eigenen Fertigung
- die Nutzung von rwt-CIM-Komponenten in der eigenen Fertigung
- die Automatisation von bisher manuell durchgeführten Arbeitsinhalten mit Standard-Software

rwt-Ziele waren

- die Deckel-Produktion als Kunden zu gewinnen
- der Einsatz von Standard-Anwendungssoftware und sogenannten 'Softwaretools' bei einem großen Maschinenbauer, wie ihn das

Haus Deckel darstellt

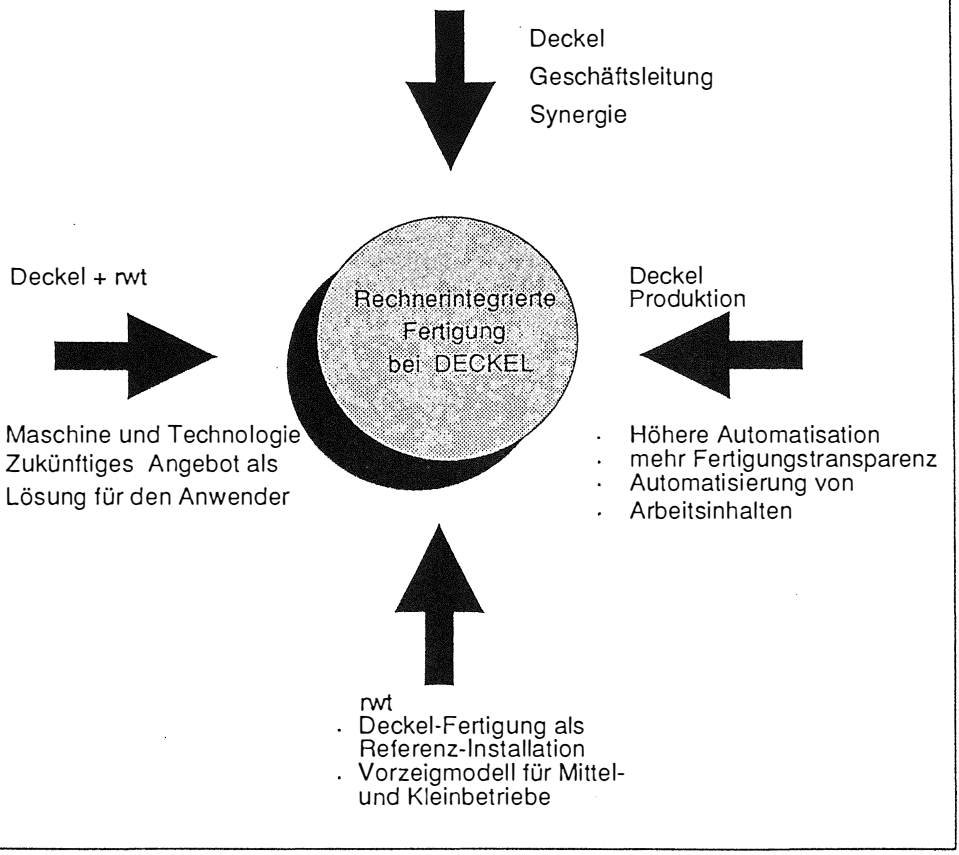
- die Deckel-Produktion als Pilotkunden für neue Anwender im CIM-bereich zu gewinnen

Gemeinsame Ziele von Deckel und rwt waren

- eine Referenz-Installation für den kleinen und mittleren Anwender; für ihn Lösungen zu schaffen und zu installieren.
- praxisnah zu demonstrieren, was der Einsatz von CIM-Lösungen bringt, für Deckel- und für rwt-Kunden, sowie den Nachweis zu erbringen, daß Deckel und rwt für komplexe Fertigungs-lösungen gemeinsam die Verantwortung übernehmen können

Die Vereinbarung dieser Ziele ist wichtig, da im Laufe der Konzeptionsphase Lösungen permanent darauf abgeprüft werden müssen. Die Zielsetzungen beider Unternehmen ergänzen sich also, so daß dieses Projekt relativ schnell angegangen werden konnte.

Erwartungshaltung der Beteiligten am Projektbeginn

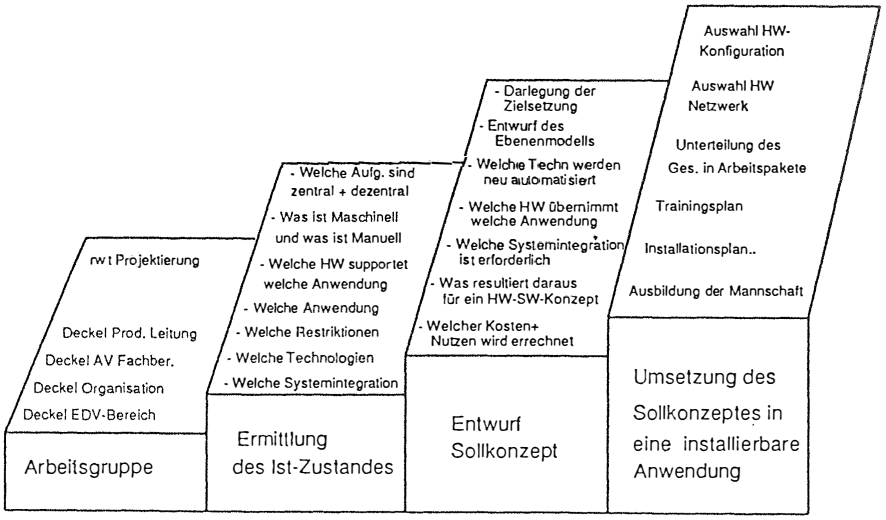


Welche Aufgaben standen nun zur Abdeckung dieser Ziele an ?

1. die Einführung von DNC
2. die Einführung von MDE an CNC-Maschinen
3. die Koordination von BDE mit einem bestehenden BDE-System
4. die Einführung eines elektronischen Leitstandes
5. der Aufbau einer Werkzeug- und Betriebsmittelverwaltung
6. die Integration von Fertigungszellen in ein Fertigungsleitsystem
7. die Integration der Anwendungssoftware zwischen den bereits vorhandenen Hardwaresystemen
8. die Umsetzung des Neukonzeptes in machbare Arbeitspakete

Zur Konzeptfindung wurde eine Arbeitsgruppe gebildet, welche aus den Bereichen Organisation, den betroffenen Fachbereichen, der Arbeitsvorbereitung, der Werkstatt und aus rwt-Mitarbeitern bestand.
Aufgabe dieser Arbeitsgruppe war die Dokumentation des Istzustandes und deren kurzfristige Fortschreibung. Die Durchführung übernahm dabei schwerpunktmäßig die Deckel-Mannschaft. rwt legte in dieser Projektphase den Schwerpunkt auf die Information der Arbeitsgruppe, welchen Leistungsumfang die unter 1 - 8 genannten Anwendungs-Softwarepakete beinhalten.
Zielsetzung der Arbeitsgruppe war die Dokumentation des Istzustandes in einem Deckel-/rwt-Ebenenmodell. Diese Dokumentation war der Ausgangspunkt für die Neukonzeption des CIM-Konzeptes.

Wie war die Vorgehensweise



Die Erfahrung in dieser Projektphase für beide Partner war

- die sprachliche Gleichschaltung der Projektgruppe, jeder sprach zwar von CIM, jedoch wurden Arbeitsinhalte, Benutzeranforderungen und die Bewertung von einzelnen Anwendersoftwarefunktionalitäten verschieden verstanden und in ihrer Priorität verschieden beurteilt.
- Unterschiede wurden hier deutlich, im besonderen zwischen Anwender und Organisation, indem die Prioritäten von Seiten der Organisation mehr auf die Integrationsfunktionalität gelegt wurde, während der Anwender größeren Wert auf die Benutzeroberfläche und die anwenderspezifischen Softwarefunktionen legte.

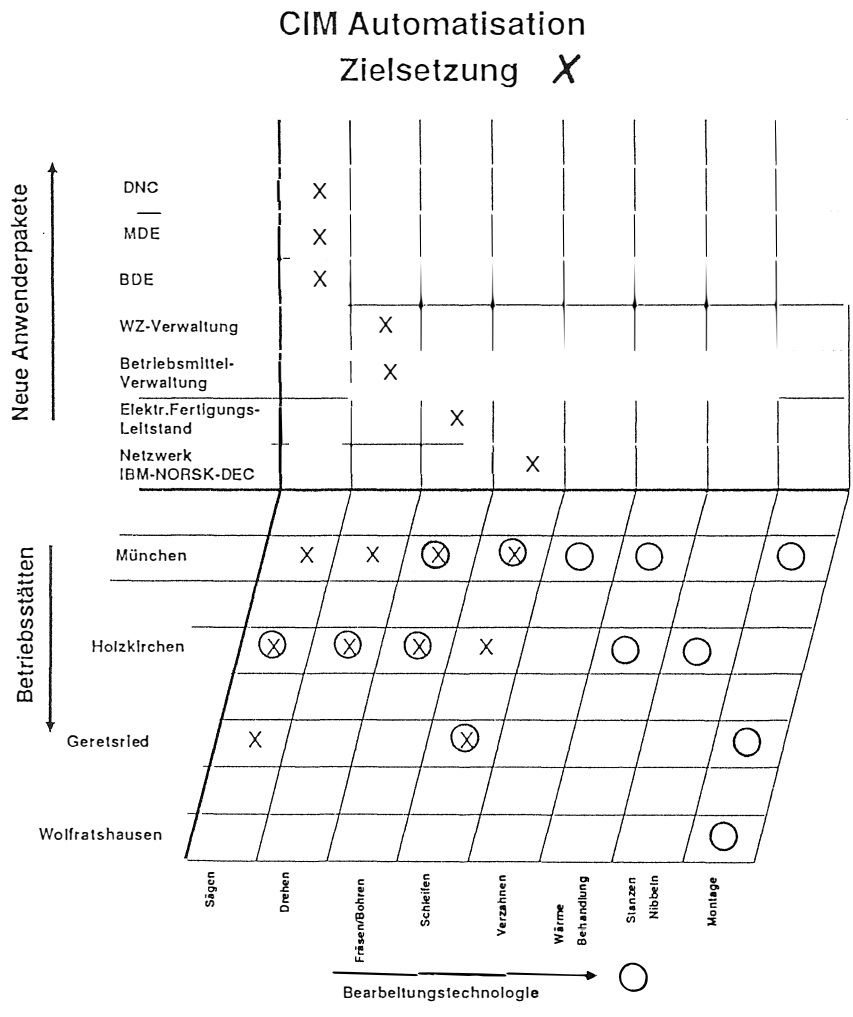
Relativ unkompliziert verlief die Zuordnung der einzelnen Arbeitsinhalte und der Anwendungssoftware im rwt-Ebenenmodell. Die Erfassung des Istzustandes war für die Arbeitsgruppe im wesentlichen eine Fleißaufgabe, nachdem die sprachliche Gleichschaltung erreicht war.

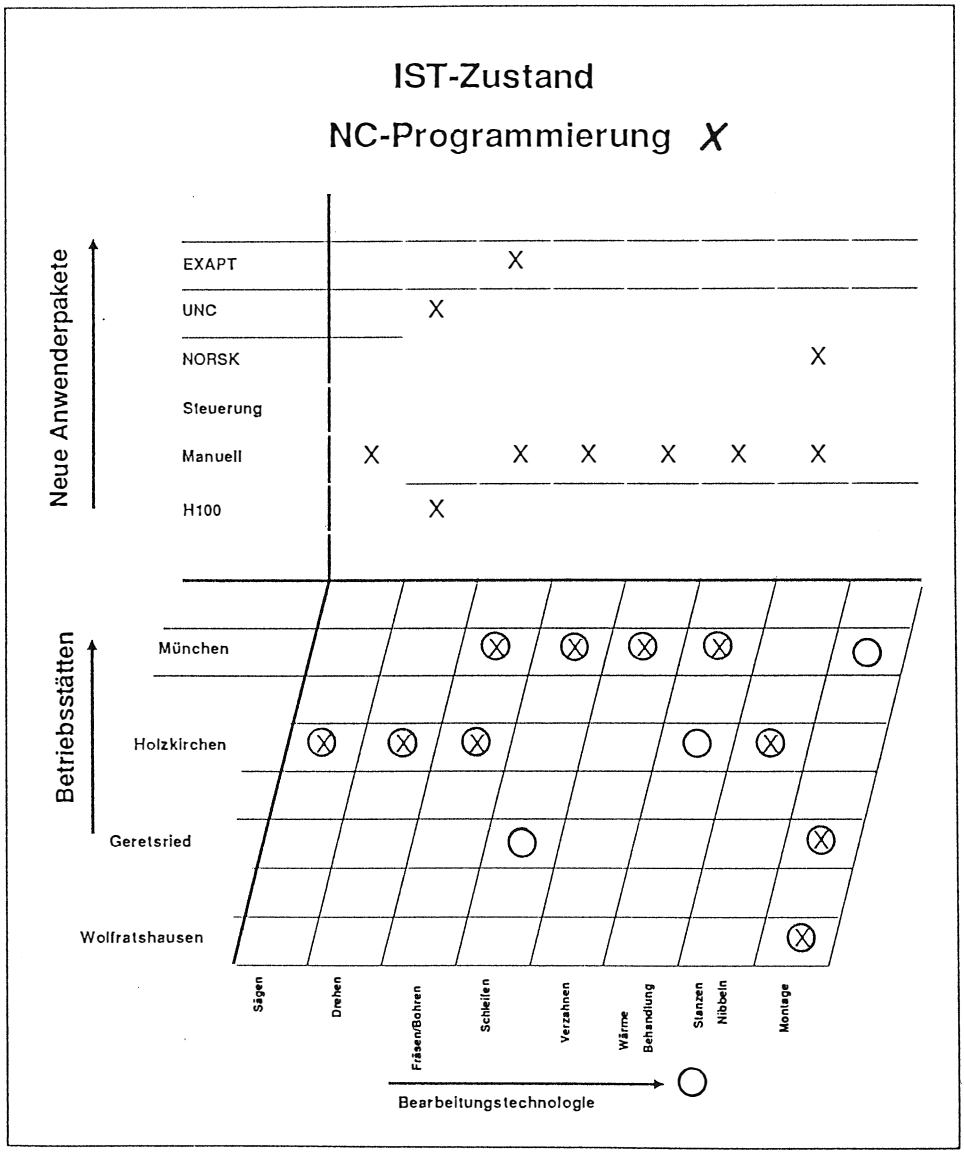
Die Automatisierung von bisher manuell durchgeführten Anwendungen konnte nach zwei Gesichtspunkten erfolgen :

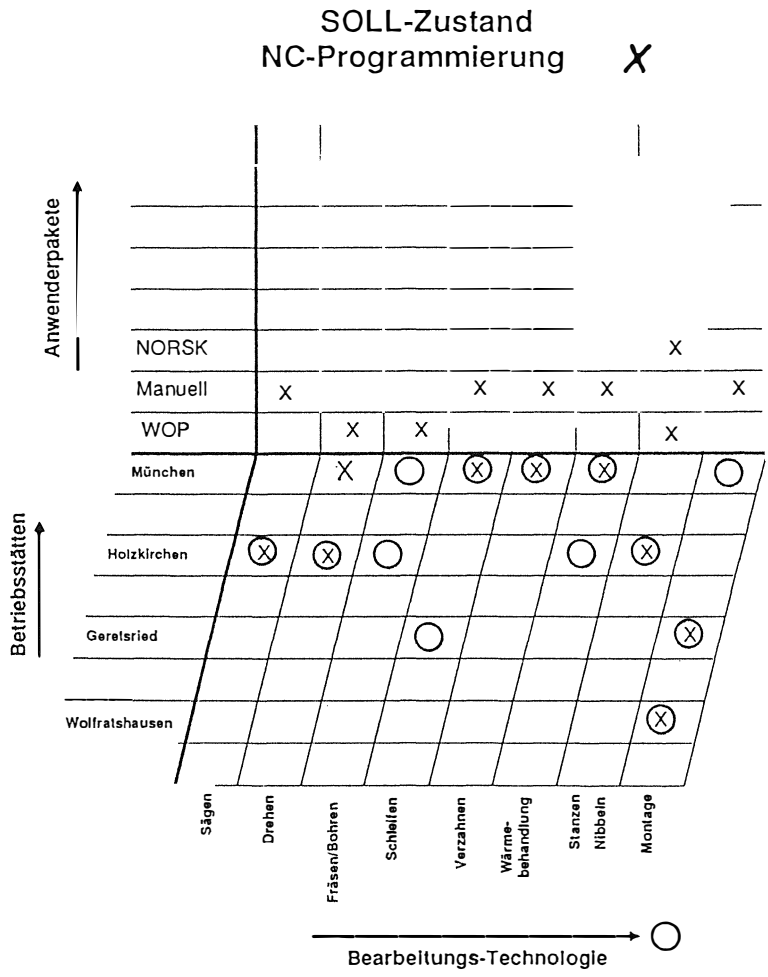
a) Die Fortschreibung der primär betriebswirtschaftlichen Funktionen in den Bereichen Fertigungsvorbereitung und Werkstatt.
b) Berücksichtigung der Standardanwendertools mit Priorität auf Automatisierung der Bereiche DNC/MDE/BDE, Werkzeug- bzw. Betriebsmittelverwaltung, elektronischer Leitstand und Werkstattorientiertes Programmierverfahren.

Der Unterschied von Ansatz a) zu b) liegt im Kern darin, daß a) im wesentlichen die Fortschreibung auf bestehender Hardware mit entsprechender Umstellung der Anwendersoftware auf diese Hardware als Resultat gebracht hätte.
Der wesentliche Aufwand wäre hierbei, die Software zu portieren, der geringere Aufwand wäre das gesamte Thema Vernetzung. Nachteilig an diesem Lösungsansatz ist jedoch, daß das erzeugte Ergebnis eine Speziallösung für Deckel gewesen wäre mit einem wiederkehrenden Aufwand für Produktupdating bzw. Pflege der Anwendungssoftware. Dabei wäre auch nicht die Zielsetzung, ein Referenzmodell für Klein- und Mittelbetriebe darzustellen, erreicht worden.

Da in einem 3o-Minuten-Vortrag nicht detailliert auf jeden einzelnen Bereich eingegangen werden kann, möchte ich mit nachfolgender Folie einen kurzen Überblick über die CIM-Automatisation zeigen.


CIM Automatisation
Zielsetzung X
Neue Anwenderpakete
DNC
MDE
BDE
WZ-Verwaltung
Betriebsmittel-
Verwaltung
Elektr.Fertigungs-
Leitstand
Netzwerk
IBM-NORSK-DEC
Betriebsstätten
München
Holzkirchen
Geretsried
Wolfratshausen
Sägen
Drehen
Fräsen/Bohren
Schleifen
Verzahnen
Wärme
Behandlung
Stanzen
Nibbeln
Montage
Bearbeitungstechnologie


IST-Zustand
NC-Programmierung X
Neue Anwenderpakete
EXAPT
UNC
NORSK
Steuerung
Manuell
H100
Betriebsstätten
München
Holzkirchen
Geretsried
Wolfratshausen
Sägen
Drehen
Fräsen/Bohren
Schleifen
Verzahnen
Wärme Behandlung
Stanzen Nibbeln
Montage
Bearbeitungstechnologie


SOLL-Zustand
NC-Programmierung
X
Anwenderpakete
NORSK
Manuell
WOP
Betriebsstätten
München
Holzkirchen
Geretsried
Wolfratshausen
Sägen
Drehen
Fräsen/Bohren
Schleifen
Verzahnen
Wärme-
behandlung
Stanzen
Nibbeln
Montage
Bearbeitungs-Technologie

Die Übersicht dieser Folie zeigt die neuen Anwendungspakete, also die Tätigkeiten, welche in der Vergangenheit manuell durchgeführt wurden, und welche im Rahmen dieses CIM-Modells automatisiert werden. Neue Anwendungspakete sind hier der Bereich DNC, der Bereich Maschinendatenerfassung und der Bereich Betriebsdatenerfassung.
Als 2. Paket der Bereich Werkzeugverwaltung mit angeschlossener Betriebsmittelverwaltung.
Als 3. Paket der elektronische Leitstand.
Als 4. Paket die Vernetzung der Systeme IBM, Norsk Data und DEC.

Wie aus der Abbildung hervorgeht, liegen die Schwerpunkte der Automatisation in den Betriebsstätten München und Geretsried. Ein weiteres Arbeitspaket stellt die NC-Programmierung dar. Der Istzustand ist in nachfolgender Tabelle aufgeführt und zeigt die jeweiligen Anwendungspakete, zugeordnet zur Bearbeitungstechnologie und den Betriebsstätten. Auffallend ist hier, daß das Gesamtunternehmen zur Zeit fünf verschiedene NC-Programmiersysteme im Einsatz hat.

Wie im Sollzustand aufgezeigt wird, soll diese Anzahl auf 2 Programmiersysteme reduziert werden. Die Reduktion wird durch den Einsatz des neuen **WOP-NC-Paketes**, einer gemeinsamen Entwicklung u.a. von Traub, Deckel und rwt, erreicht.

Nachfolgende Dias sollen dem Betrachter einen kurzen Überblick in dieses neuartige Programmiersystem geben, was in der Kürze der Zeit nur als Momentaufnahme zu sehen ist.

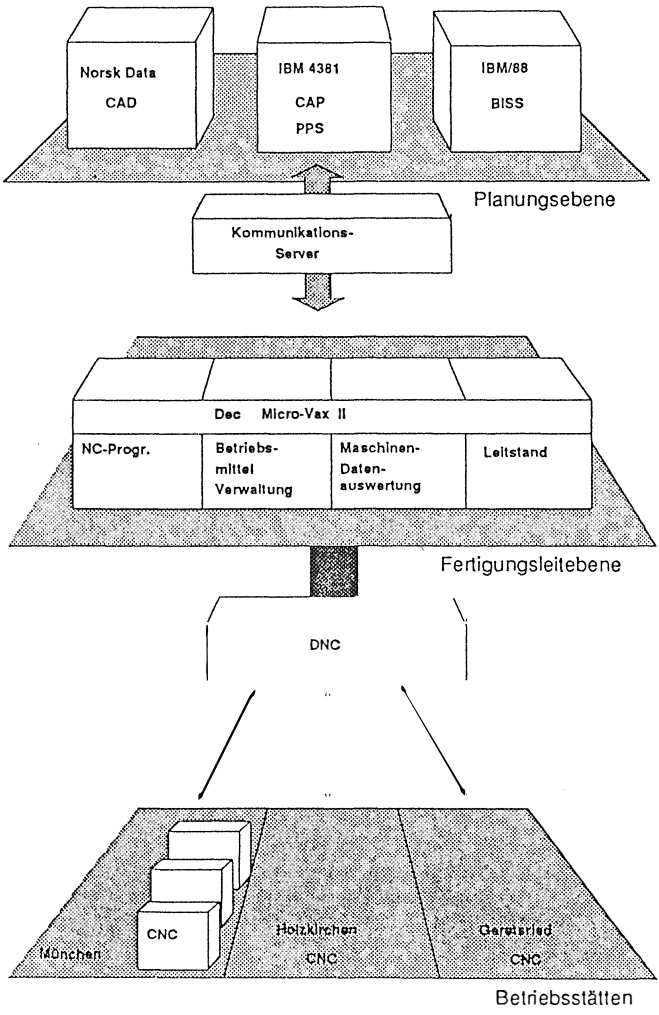
Dia 1 zeigt die Darstellung einer erzeugten Geometrie als Kontur, der rot dargestellte Bereich die Fräserlaufbahn bei der Ausarbeitung der Innenkontur.

Dia 2 zeigt das Modul Schnittdatenermittlung über die Auswahl der Bearbeitungsart, in diesem Fall Schruppen, und wie die Daten des zur Bearbeitung ausgewählten Werkzeuges aus der Werkzeugdatenbank herausselektiert und in das NC-Programm übernommen werden.

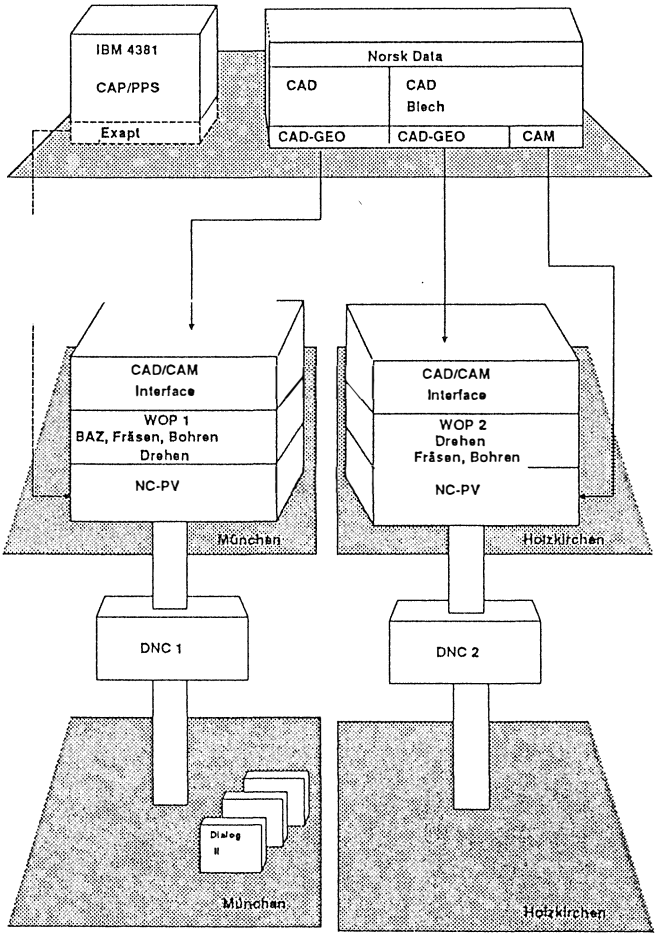
Dia 3 zeigt die Auswahl eines Bohrzyklus auf der linken Seite, die Eingabewerte in der Mitte mit den bereits aus den Werkzeugdaten vorbesetzten Werten, und rechts die grafische Zyklusdarstellung.

Der WOP-Arbeitsplatz ist eine integrierte Workstation - als IBM-PC aufgebaut, und bei dieser Installation als intelligentes Terminal an die MicroVAX angeschlossen.

Deckel-rwt-Ebenenmodell



Deckel-rwt Soll-Ebenenmodell

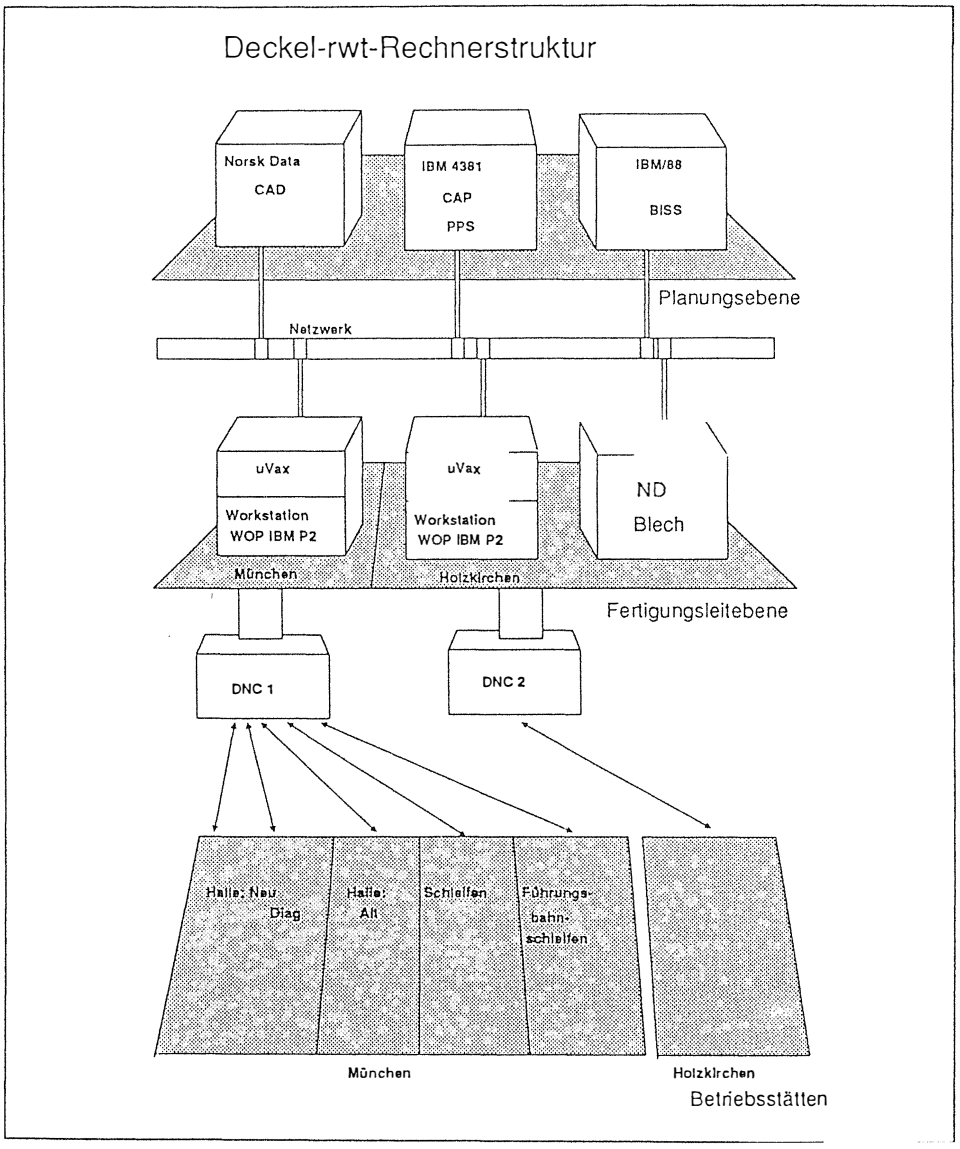


Betriebsstätten

WOP - Integration

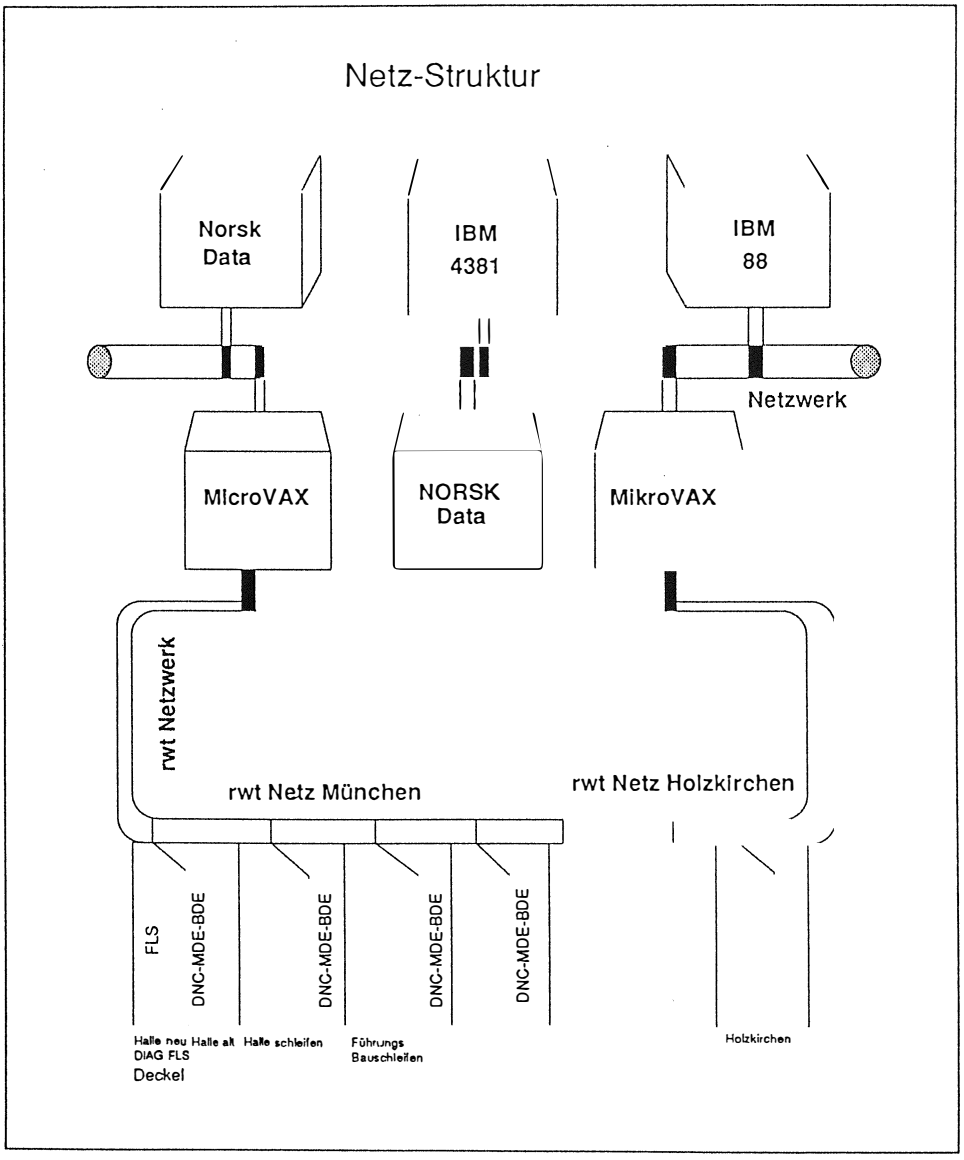
Die Integration der WOP-NC-Programmierung in das Gesamtkonzept zeigt nachfolgende Folie der CAD/CAM-Kopplung von WOP zum CAD-System von Norsk Data. Die Datenübergabe erfolgt mittels CAD/CAM-Geometrie-Interface auf Basis der Iges-Schnittstelle.
Die fertiggestellten NC-Programme werden auf der MicroVAX an die NC-Programmverwaltung weitergegeben, welche als Modul alle NC-Programme verwaltet und für die DNC-Freigabe zuständig ist.
Hervorzuheben ist die universelle Funktion der NC-Programmverwaltung, ein Datenbanksystem, welches sowohl Exaptprogramme, Norsk Data-Stanz- und Nibbelprogramme als auch WOP NC-Programme einheitlich verwaltet.
Nachfolgendes Dia wiederum als Eindruck über den Leistungsstandard zeigt die Merkmale des Verwaltungssystems am Beispiel eines NC-Programmfiles.
Die Bezeichnung, die DNC-Programmnummer, die Werkstückbezeichnung, die Arbeitsgangbezeichnung, die Maschine der Maschinengruppe sowie die abspeicherbaren Daten, Quellenprogramm mit Datum und Verfasser, Steuerdaten sowie den Optimierungsstand, die dazugehörige CAD-Datei sowie die letzten Änderungen mit Datum und dem letzten Zugriff.
Das nächste Dia zeigt am Beispiel einer Maschinendatei die Information über die Maschinengruppe, den Einsatzbereich, die letzte Programmanforderung mit Datum und Uhrzeit sowie die letzte Rückübertragung, den Zustand des übertragenen Programmes

wie Test oder optimiertes NC-Programm, den DNC-Strang zur CNC-Maschine, die Anzahl der möglichen Unterprogramme sowie den Status der NC-Datei nach der Programmrückübertragung. Ferner den Status der zum NC-Programm gehörigen Zusatzdateien wie CAD-Datei und sonstiges.


Deckel-rwt-Rechnerstruktur
Norsk Data
CAD
IBM 4381
CAP
PPS
IBM/88
BISS
Planungsebene
Netzwerk
uVax
Workstation
WOP IBM P2
uVax
Workstation
WOP IBM P2
ND
Blech
München
Holzkirchen
Fertigungsleitebene
DNC 1
DNC 2
Halle: Neu
Diag
Halle:
Alt
Schleifen
Führungs-
bahn-
schleifen
München
Holzkirchen
Betriebsstätten

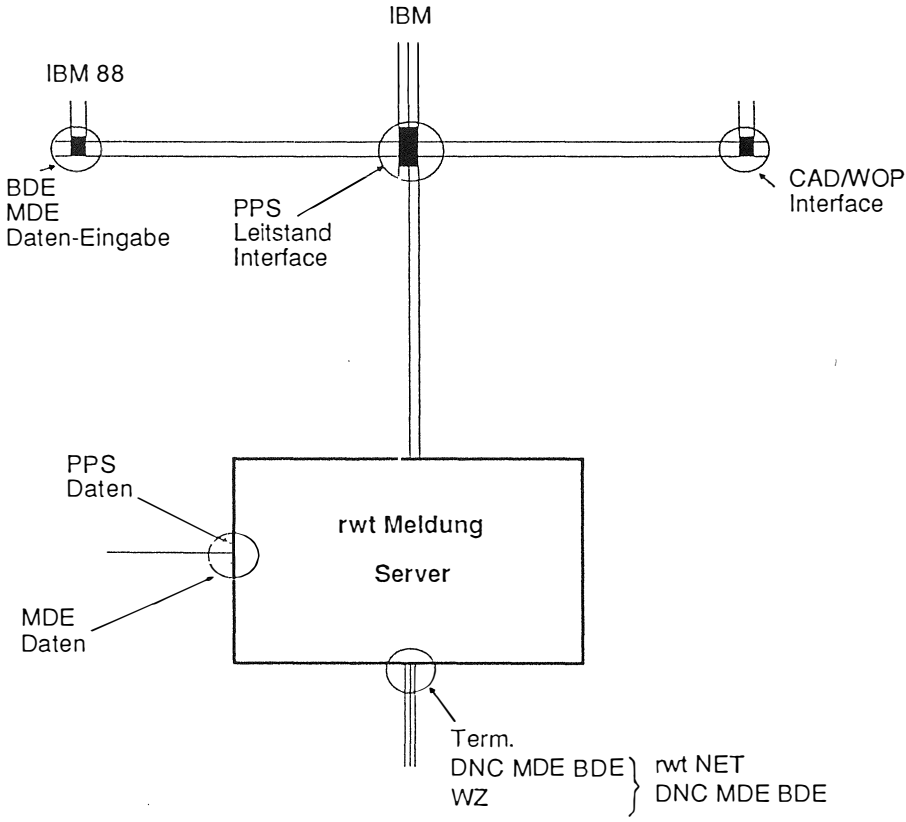
Vernetzung

Eine systemseitige Voraussetzung zur Realisierung eines solchen Konzeptes ist die Vernetzung der bestehenden Hardware mit der neu zu installierenden Hardware.
Die Vernetzung zwischen der Planungsebene und der Fertigungsleitebene erfolgt mittels **Standardnetzwerken**. Vernetzt werden dabei die Rechner Norsk Data, IBM 4381, IBM 88 und MicroVAX.
Die Vernetzung zwischen Fertigungsleitebene und Werkstattebene übernimmt das **rwt-Netzwerk**. Angekoppelt sind hier die einzelnen CNC-Maschinen, Voreinstellgeräte, Meßsysteme und Fertigungszellen sowie Handarbeitsplätze. Die Kommunikation zu den einzelnen Geräten wickelt das rwt-System DNC 6ooo ab.
Das DNC 6ooo transportiert die in den Betriebsstätten erfaßten Daten entweder 1 : 1 oder verdichtet zum jeweiligen Auswerterechner. Die Leitrechner (VAX) haben die Aufgabe, Datensammelsystem und Kommunikationsstationen zu bilden.
Nachfolgende Skizze zeigt einfach dargestellt die notwendige Interface-Ebene in der Anwendung, den IBM 88 als Auswerterechner für MDE/BDE, den IBM 4388 mit der Schnittstelle PPS-Leitstand, den Norsk Data mit der CAD/CAM-Schnittstelle zu WOP und zur NC-Programmverwaltung.


Netz-Struktur
Norsk Data
IBM 4381
IBM 88
Netzwerk
MicroVAX
NORSK Data
MikroVAX
rwt Netzwerk
rwt Netz München
rwt Netz Holzkirchen
FLS
DNC-MDE-BDE
DNC-MDE-BDE
DNC-MDE-BDE
DNC-MDE-BDE
Halle neu Halle alt
DIAG FLS
Deckel
Halle schleifen
Führungs
Bauschleifen
Holzkirchen

Interface Struktur

IBM System 88



Datenverteilung

Die Datenverteilung von DNC/MDE- und BDE-Daten liegt auf der MicroVAX. Die Kommunikationssoftware DNC 6ooo versorgt und entsorgt die Betriebsstätte mit allen Fertigungsinformationen. Der Kommunikationsrechner FÜ 6ooo (sh. Bild) wird bedarfsorientiert mit Bedienerterminals konfiguriert.

Dia 6 zeigt die möglichen Bedienkonsolen für DNC-Datenübertragung als Basispaket und Betriebs- oder Maschinendatenerfassung als erweitertes Anwendungspaket.

Dia 7 zeigt hier den Anschluß eines FÜ 6ooo Kommunikationsrechners an ein Voreinstellgerät.

Dia 8 zeigt die Bedienkonsole 'BB' am FÜ 6ooo Kommunikationsrechner zur grafischen Datenübertragung.

Dia 9 zeigt die Bereitstellung einer Spannsituation auf dem Bediengerät BB als Alternative zur Spannskizze.

Die Auswertung der DNC/MDE und BDE-Informationen kann wahlweise auf der MicroVAX oder auf einer anderen Hardware erfolgen. Das Auswertepaket standardisiert als VDI-Auswertung aufgebaut, wobei wahlweise eine Maschine oder eine Maschinengruppe über einen vom Benutzer festgelegten Zeitraum betrachtet werden kann. Über die vom System 6ooo per Signal (rote Taste) erfaßten Transaktionen und die entsprechende Klassifizierung in

- Belegungszeit (TB)
- Nutzungszeit (TN
- Instandsetzungszeit (TIN)
- Ruhezeit (TRU)
- Wartungszeit (TWRTG)

läßt sich wie im Bild dargestellt pro Maschine oder Maschinengruppe die technische Ausfallrate, die organisatorische Ausfallrate oder der Nutzungsgrad effektiv oder prozentual über den Auswertezeitraum hinweg ermitteln.

Dia 1o zeigt die Umsetzung dieser Werte in eine grafische Darstellung, wie sie von rwt in den USA realisiert wurde.

Werkzeug- und Betriebsmittelverwaltung

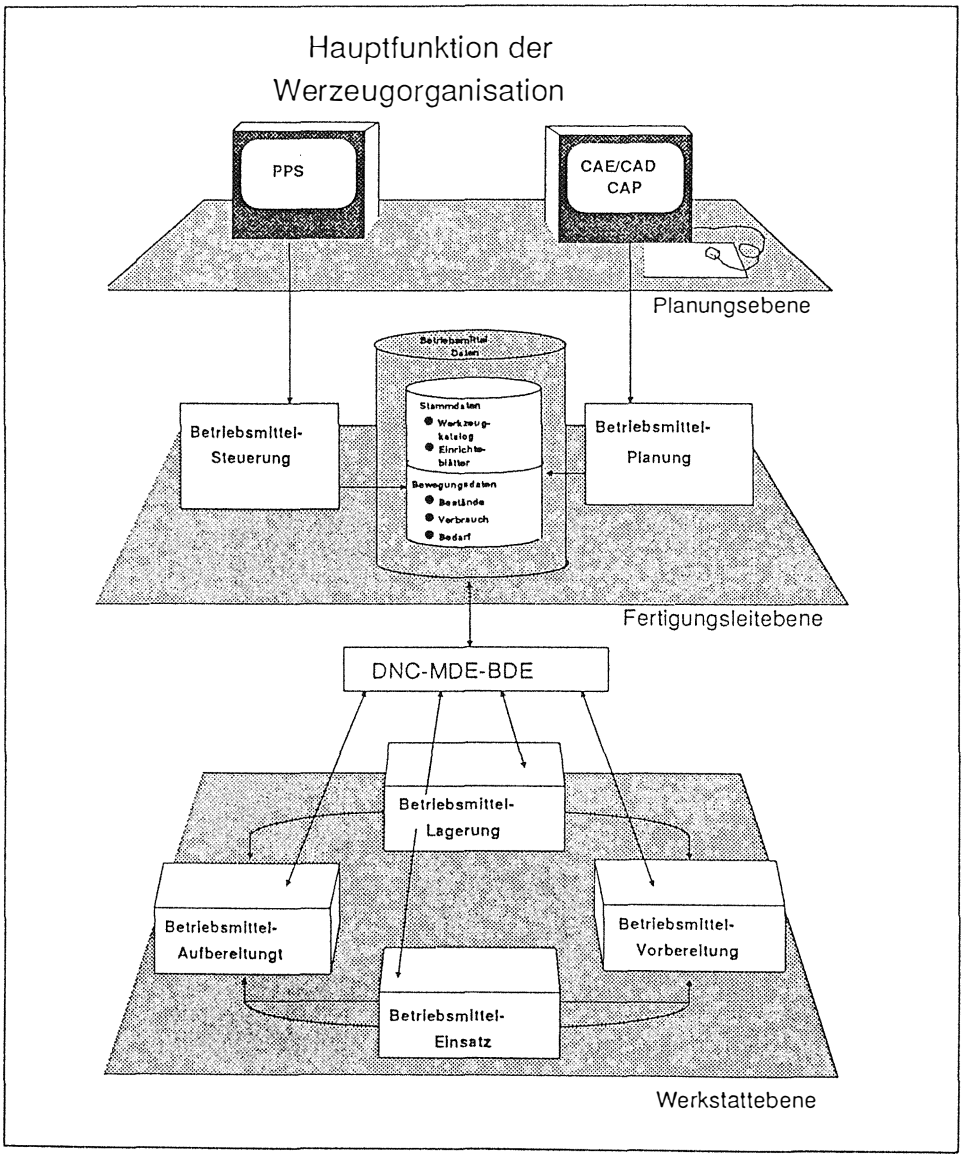
Weitere zu integrierende Anwendungspakete sind die Werkzeug- und Betriebsmittelverwaltung sowie der Fertigungsleitstand.
Der Leistungsumfang der Werkzeugverwaltung beginnt am Werkzeuglagerort mit den entsprechenden Komponenten, Betriebsmitteldaten, also den **Stammdaten**

- dem Werkzeugkatalog
- den Einrichteblättern
- den Bewegungsdaten
- den Beständen
- dem Verbrauch
- und dem Bedarf

Das Kommunikationsglied zur Werkzeugverwaltung zwischen Werkstattebene und Fertigungsleitrechner stellt wiederum das **DNC-System** dar. Die Datenverteilung von der Lagerung zur Betriebsmittelversorgung, zum Betriebsmitteleinsatz, zur Betriebsmittelaufbereitung und zurück zur Betriebsmitteleinlagerung wird über ein rwt-Datenbanksystem abgedeckt.
Die Versorgung der NC-Programmierung erfolgt durch eine direkte Ankopplung des WOP-Systems mit entsprechendem Übergabe-Interface an dieses Anwendungspaket.

Realisierung

Die Realisierung eines solchen Systems kann nur in **Arbeitspaketen** erfolgen.
Da sowohl die einzelnen Sachbearbeiter als auch das Produktionsmanagement lediglich einen geringen prozentualen Anteil ihres Zeitbedarfs investieren können, ohne die täglichen Fertigungsaufgaben zu vernachlässigen, ist die **Koordination** zwischen Hardwarebeschaffung, Einsatz der neuen Anwendungsgebiete und Umstellung der bestehenden Anwendung sowie die Vernetzung der Systeme in einem **Zeitplan genau abzustimmen.**
Die Ausbildung der Mitarbeiter sowie eine sauber organisierte Arbeitsteilung zwischen Organisation, Anwender und Lieferant ist Bestandteil einer Feinplanung.
Ein denkbarer Ansatz ist die kurzfristige Einführung des DNC-Systems, danach ein Ausbau der Maschinen- und Betriebsdatenerfassung. In einem weiteren Schritt könnte die Betriebsmittel- bzw. Werkzeugverwaltung eingeführt werden.
Der nächste Arbeitsschritt wäre dann die Integration der WOP-NC-Programmierung und des Fertigungsleitstandes. Bisher gemachte Erfahrungen bestätigen, daß es sehr wohl einen Sinn macht, auch für größere Maschinenbauunternehmen einen Systemintegrator mit in ein solches Integrationsprojekt einzubinden.
Das Zwischenergebnis zeigt, daß alle Beteiligten neue Erfahrungen machen, welche sich positv auf das Projekt auswirken. Die am Projektanfang gesteckten Ziele wurden bis heute nicht geändert.


Hauptfunktion der
Werzeugorganisation
PPS
CAE/CAD
CAP
Planungsebene
Betriebsmittel-
Steuerung
Betriebsmittel-
Planung
Fertigungsleitebene
DNC-MDE-BDE
Betriebsmittel-
Lagerung
Betriebsmittel-
Aufbereitungt
Betriebsmittel-
Vorbereitung
Betriebsmittel-
Einsatz
Werkstattebene

Die Realisierung dieses Projektes steht allerdings noch bevor. Insofern brachte die Erfahrung in der Konzeptionsphase die Motivation und die Sicherheit für alle Projektbeteiligten, daß die am Anfang gesteckten Ziele erreichbar sind.
Die Realisierung wird vom Anwender gewünscht. Unter anderem hat die bisherige Zusammenarbeit dazu beigetragen, daß der Anwender den Nutzen der einzelnen Arbeitspakete kennt. Er ist sich auch über seinen Beitrag im klaren der notwendig ist, um erfolgreich zu realisieren.

Dem Systemlieferanten brachte die Konzeptionsphase nicht immer nur die Zustimmung des Anwenders, doch gerade die kritische Betrachtung eines Standard-Produktes bzw. der Standardtools bringt den Lieferanten produktseitig voran und zeigt konzeptionelle Schwachstellen auf. Diese Erkenntnis verbessert den Standard und schafft die beste Lösung für den Kunden.

Strukturierung des Netzes eines regionalen EVU zur Verarbeitung großer Datenmengen

Hilmar Sethmacher

Krupp Atlas Elektronik GmbH
Sebaldsbrücker Heerstr. 235
2800 Bremen 44

o SCHLÜSSELWÖRTER

* Netzstruktur eines regionalen Energieversorgungsnetzes
* Konfiguration des Leitwartensystems
* Datenmodellierung für große Datenmengen
* Konstruktionsverfahren für die Datenaufbereitung
* Bedienung und Verarbeitung
* Anforderung an die Programmentwicklung
* Überblick über die PEARL-Implementation
* Ausblick

o ZUSAMMENFASSUNG

Es werden der Aufbau und die betrieblichen Anforderungen eines regionalen Energieversorgungsnetzes beschrieben. Hieraus ergibt sich die Notwendigkeit, große Datenmengen zu verarbeiten. Dies erfordert eine entsprechende Systemkonfiguration und eine geordnete Vorgehensweise bei der Systemspezifikation als Voraussetzung für die Systemrealisierung.
Es erfolgt die Vorstellung der ausgewählten Konfiguration des Leitwartensystems.
Anschließend wird das Ordnungssystem als eine strukturierte Möglichkeit der Datenmodellierung erläutert. Die Strukturierung des Datenmodells ist eine wichtige Voraussetzung für einfache Konstruktionsverfahren zur Datenaufbereitung und Bild/Prozeß-Verknüpfung bei der Verarbeitung umfangreicher Datenmengen.
Das Konstruktionsverfahren für die Datenaufbereitung wird vorgestellt.
Es erfolgt ein kurzer Einblick in Bedienung und Verarbeitung, um den Datenfluß im Gesamtsystem an Hand des beschriebenen Ordnungsssystems zu skizzieren.
Zum Abschluß erfolgt der Überblick über die PEARL-Implementation, den Realisierungszeitraum und die gewonnenen Erfahrungen sowie ein kurzer Ausblick für künftige Realisierungsvorhaben.

1. Einleitung

Netzleittechnik dient der zentralen Betriebsführung von Versorgungsnetzen z.B. für Elektrizität, Gas, Wasser, Fernwärme.

Ihre Komponenten bestehen aus Systemen der:

- Fernbedienungs- und Fernüberwachungstechnik
- Fernzählung und Fernmessung
- Tonfrequenz-Rundsteuerung
- Prozeßdatenverarbeitung
- Netzschutz
- zukünftig auch der integrierten Leittechnik in Stationen

Wir unterscheiden zwischen:
- Netzführung
- Lastführung
- Lastverteilung

Die Realisierung der Betriebsführungsaufgaben erfolgt mittels leistungsfähiger Rechner- und Übertragungssysteme.
Diese Systeme kommen in verschiedenen Hierarchiestufen zum Einsatz. Hier wird eine Anwendung der Netzführung aus dem Mittelspannungsbereich vorgestellt.
Die Vielfalt der Aufgaben in diesem Anwendungsbereich erfordern die schnelle Verarbeitung umfangreicher Datenmengen.
Zur Lösung der betrieblich hohen Anforderungen tragen betriebs-, rechner- und verarbeitungstechnische Struktur- und Ordnungsmaßnahmen bei.

2. Netzstruktur und Informationsumfang

Das regionale Energieversorgungsnetz ist in 8 Bezirke aufgeteilt worden.

Jeder Bezirk wird von einer Bezirkssteuerstelle überwacht, gesteuert und gewartet. Als Automatisierungseinrichtung wurden hier dezentral ein Knotenrechnersystem und die zugehörige Prozeßperipherie installiert.

An eine zentrale Netzleitstelle mit Zentralrechner und Datenspeicher sind die Knotenrechner über Modemtechnik angeschlossen Von den Bedienplätzen in der Netzleitstelle ist somit eine zentrale Bedienung für die Überwachung und Steuerung des Gesamtnetzes möglich. Das Netzleitsystem ist für den folgenden Informationsumfang ausgelegt:

15000	Schaltbefehle
15000	Schaltermeldungen
9000	Warn- und Kennmeldungen
25000	nachgeführte Zustandsdarstellungen von noch nicht fernwirktechnisch erfaßten Mastschaltern, Trennstellen u.ä.
1500	Meßwerte
80	Fernzählkanäle von den Einspeisestellen aus dem PreussenElektra-Netz
ca.65000	Prozeßvariable

Die Prozeßdatenpunkte sind verteilt auf:

	Netzleitstelle
8	Bezirkssteuerstellen
50	Umspannwerke
170	Schaltwerke bzw. -stationen

3. Konfiguration des Leitwartensystems

Für die Konfiguration der Rechnersysteme (Bild 1) waren die folgenden Gründe ausschlaggebend:

- Erhöhter Datendurchsatz - verkürzte Antwortzeit
- Verbessertes Echtzeit-Verhalten
- Erhöhte Zuverlässigkeit und Betriebssicherheit

Die Knotenrechner sind als Mehrprozessor-Simplexsysteme ausgelegt.
Das Zentralrechnersystem ist als Mehrprozessor-Synchron-Duplexsystem ausgelegt.
An das Zentralrechnersystem sind 3 Arbeitsplätze für Prozeßführung und Datenpflege angeschlossen.

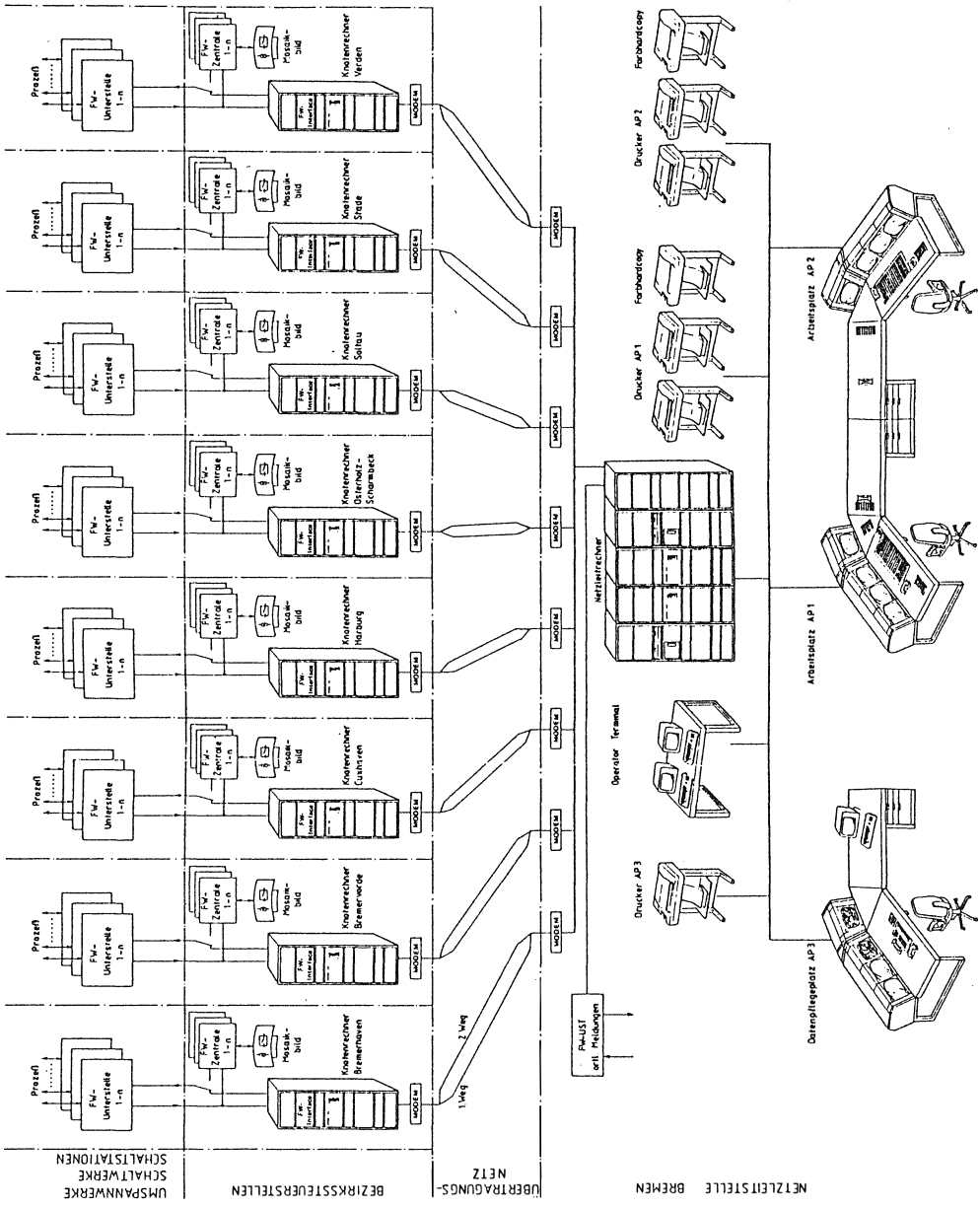


Bild 1: Konfiguration des Leitwartensystems

4. Datenmodellierung für große Datenmengen

4.1 Datenmodellierung

Für die Datenmodellierung wird ein Ordnungssystem bereitgestellt, das die folgenden wesentlichen Grundsätze für den Aufbau und die Erstellung von Datenmodellen berücksichtigt:

- o anwendbar für verschiedene Energiebereiche
- o technologisches Bezeichnungssystem für die Betriebsterminologie
- o Erzeugung eines internen technologischen Zugriffsschlüssels für optimierte Datenzugriffe
- o strukturweise Zusammenfassung von Betriebselementen
- o strukturweise Datenvorbeschreibung mit der Möglichkeit der Belegung von Normzuständen und Ermittlung der Strukturtopologie
- o strukturweise Bildkonstruktion mit Feldtypisierung
- o strukturweise Bild-/Prozeßverknüpfung mit der Möglichkeit der automatischen Erzeugung der Netztopologie und Plausibilitätsprüfungen für die Netzkonstruktion

4.2 Ordnungssystem

Adressierungssystem

An das Adressierungssystem werden folgende Anforderungen gestellt:

- Aufbau einer Dateiorganisation, die einen schnellen Zugriff auf Daten des Datenmodells ermöglicht.
- für Ein- und Ausgaben ohne Umsetzung einsetzbar.

Es wird folgendes Adressierungsschema verwendet:

- Netzart
- Numerische Ebene (Spg-Ebene)
- Station/Anlage
- Anlagen- bzw. Netzteil
- Netzelement
- Info-Quelle
- Info-Objekt
- Status

Netzart

Netze können in die folgenden Gruppen eingeteilt werden:

- Transportnetz
- Verteilernetz
- Abnehmernetz

Anlage- bzw. Netzteile

Ein Anlagenteil ist ein abgrenzbarer Teilbereich einer Station (z.B. Schaltfelder).

Netzelement

Netzelemente sind die Grundbausteine eines Netzes.(z.B. Schaltgeräte, Erder, Meßstellen usw.) Anlagenteile sind Kollektive von Netzelementen.

Info-Quelle

Die Info-Quelle gibt die Quelle an, von der die Information stammt (z.B. Meßstelle).

Info-Objekt

Informationen beziehen sich stets auf Objekte, die die kleinste adressierbare Einheit darstellen.
Info-Objekte eines Schaltelementes sind u.a.
- Schaltbefehl und
- Fernsteuerbarkeit

Info-Objekte einer Meßstelle sind u.a.
- aktuelle Wert- und Grenzwertprüfung

Auch Markierungen stellen in der Regel Objekte dar.

Status

Im Status wird der aktuelle Status des Objektes geführt

5. Konstruktionsverfahren für die Datenaufbereitung

Das Verfahren dient zur

- o Definition der Elementtypen
- o Definition von Schaltfeldern
- o Zusammenfassung von Datenpunkten
- o automatischen Vorbereitung der Netztopologie
- o automatischen Ermittlung der technologischen Adressen
- o automatischen Erzeugung und Belegung der Netzdatenbank
- o stationsweisen und fernwirktypspezifisch geordneten Zuordnung der Fernwirkdaten zu den Datenpunkten
- o Zuordnung der Schaltfelder zu Bildmakros

Das Konstruktionskonzept basiert auf dem hierarchisch geordneten Knotenmodell, das sich zusammensetzt aus den

- o physikalischen Knoten zwischen den Betriebsmitteln (Elementebene)
- o elektrischen Knoten als Zusammenfassung einer Menge von physikalischen Knoten zu Schaltfeldern (Zusammenfassung von Datenpunkten)
- o Verbindung zwischen den elektrischen Knoten (Verknüpfung von Schaltfeldern)

Die Verbindung aller physikalischen Knoten ergibt ein vollständiges Netz, das die Rohdatenebene beschreibt.

Die Konstruktion der Rohdatenebene erfolgt durch einen Automatisierungsprozeß, der die Typisierung von Betriebsmitteln und Anlagenteilen vorsieht.

Bild 2 verdeutlicht diese Art von Netzkonstruktion.

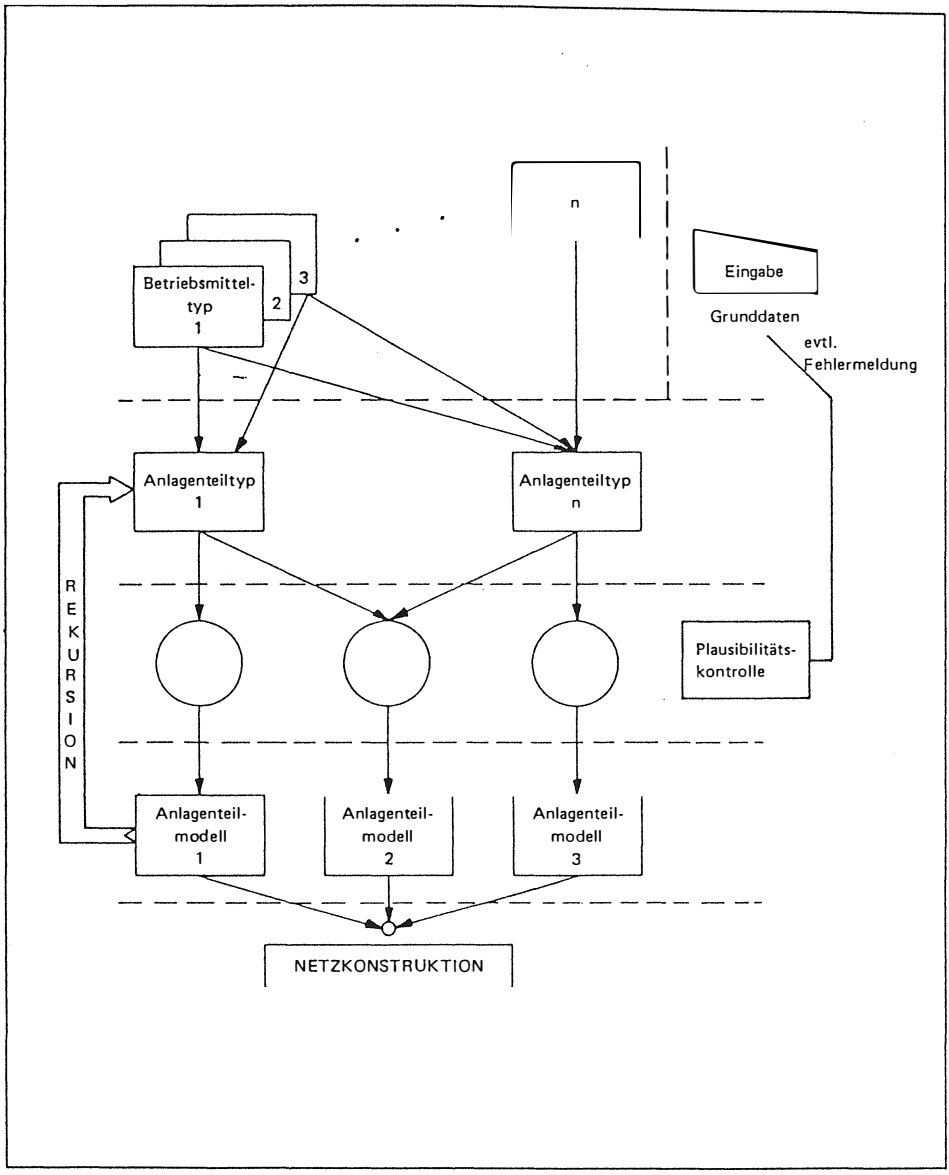


Bild 2: Typisierung von Betriebsmitteln und Anlagenteilen als Hilfsmittel bei der Netzkonstruktion

Das nachfolgende Beispiel veranschaulicht die Definition eines Anlagenteiltyps (Abzweigfeld).

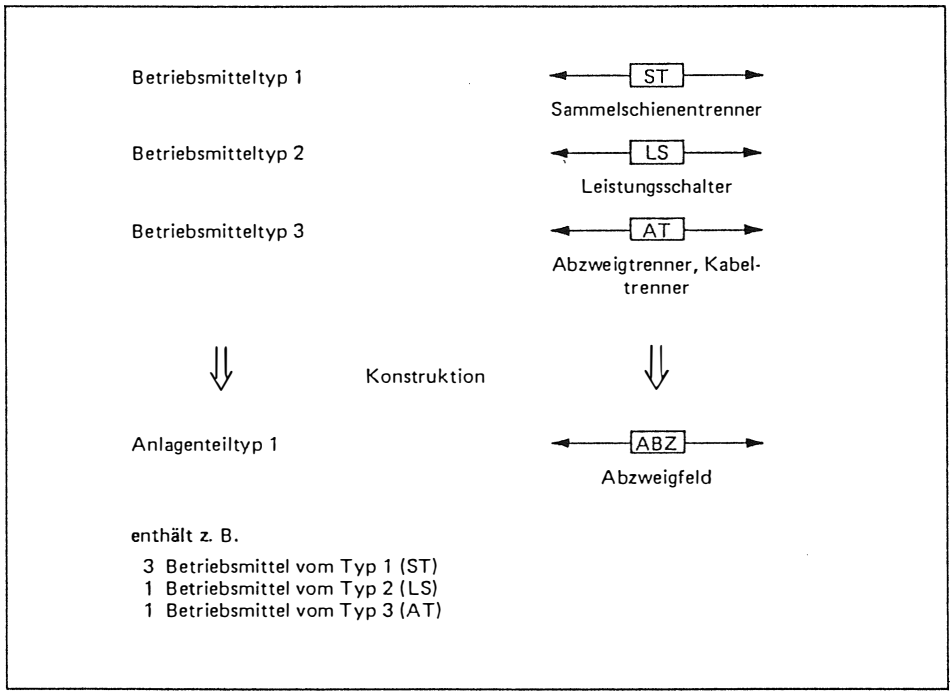


Bild 3: Definition eines Anlagenteiltyps, hier: Abzweigfeld

Für diesen Teil der Netzmodellierung wird ein formulargestütztes Konstruktionsverfahren verwendet.
Das erzeugte Abzweigfeld wird einer Plausibilitätskontrolle unterzogen und anschließend einem Anlagenteilmodell fest zugeordnet (vgl. Bild 2, Bild 4).

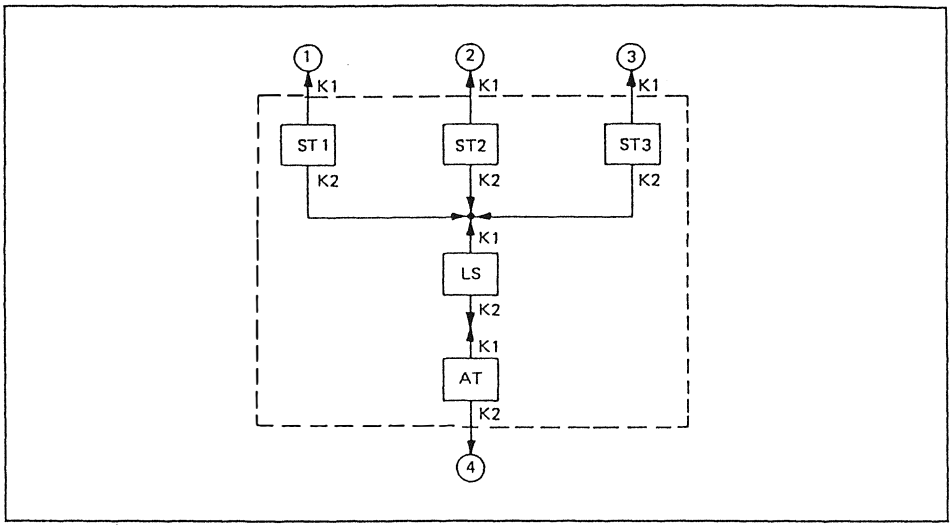


Bild 4: Anlagenteilmodell: Abzweigfeld

Das Anlagenteilmodell entspricht nunmehr einer Struktur mit 4 möglichen Verknüpfungspunkten.

Dieses Modell wird bei der Gesamtnetzkonstruktion mehrfach aufgerufen und einem realen Schaltfeld zugeordnet.

Der Vorteil dieser Konstruktionstechnik besteht darin, daß die rechnerunterstützte Netzmodellkonstruktion nicht so zeitaufwendig ist wie die elementweise Dateneingabe. Ein weiterer Vorteil besteht darin, daß Anlagenteiltypen zusammengefaßt werden können und das erzeugte Anlagenteilmodell wieder in einen Anlagenteiltyp überführt werden kann. Dieser Vorgang entspricht einem rekursiven Verfahren.

Der Topologie-Generator (Bild 5) leitet aus den Rohdaten des konstruierten Netzes eine Verknüpfungstabelle ab, die die folgenden Verknüpfungsebenen berücksichtigt:

- o Betriebsmittel/Schaltelemente
- o Anlagenteile/Schaltfelder
- o Stationen

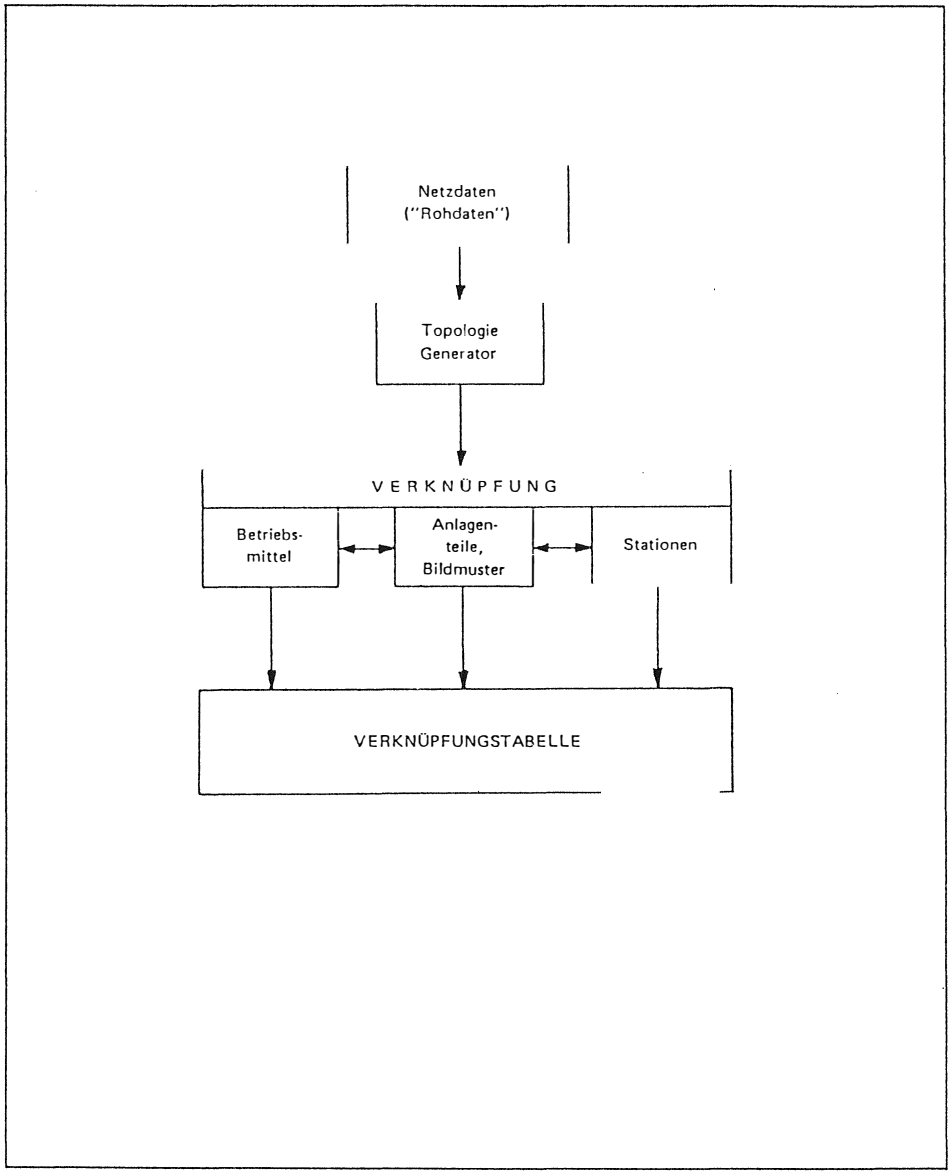


Bild 5: Ablauf Topologie-Generator

Das Verknüpfungskonzept der Betriebsmittel innerhalb eines Anlagenteils unterstützt die Verfahren

- o Abprüfung der elektrischen Schaltungsbedingungen
- o Ausführen von Schaltbefehlen
- o topologische Zustandsermittlung

6. Bedienung und Verarbeitung

In den Bedienungs- und Verarbeitungsphasen werden die in Bildern und Datenmodell erzeugten Strukturen zur Erzielung eines betriebsgerechten Antwort-/Zeitverhaltens ausgenutzt.
Die Prozeßbedienung wird über eine technologische Funktionstastatur ausgeführt (Bild 6). Die Funktionstasten sind in Sektionen aufgeteilt gemäß der Kriterien

- o Bezirkssteuerstellen / Netzleitstelle
- o Großbild, Teilnetzbild, Stationsbild
- o Betrieb/Simulation/Schaltprogrammdefinition
- o Einzelsteuern/Schaltfolge/Schaltprogramm
- o Übersichten/Protokolle/Dialoge

Schalthandlungen im Atlas-Warten-Leitsystem AWL 1300 E

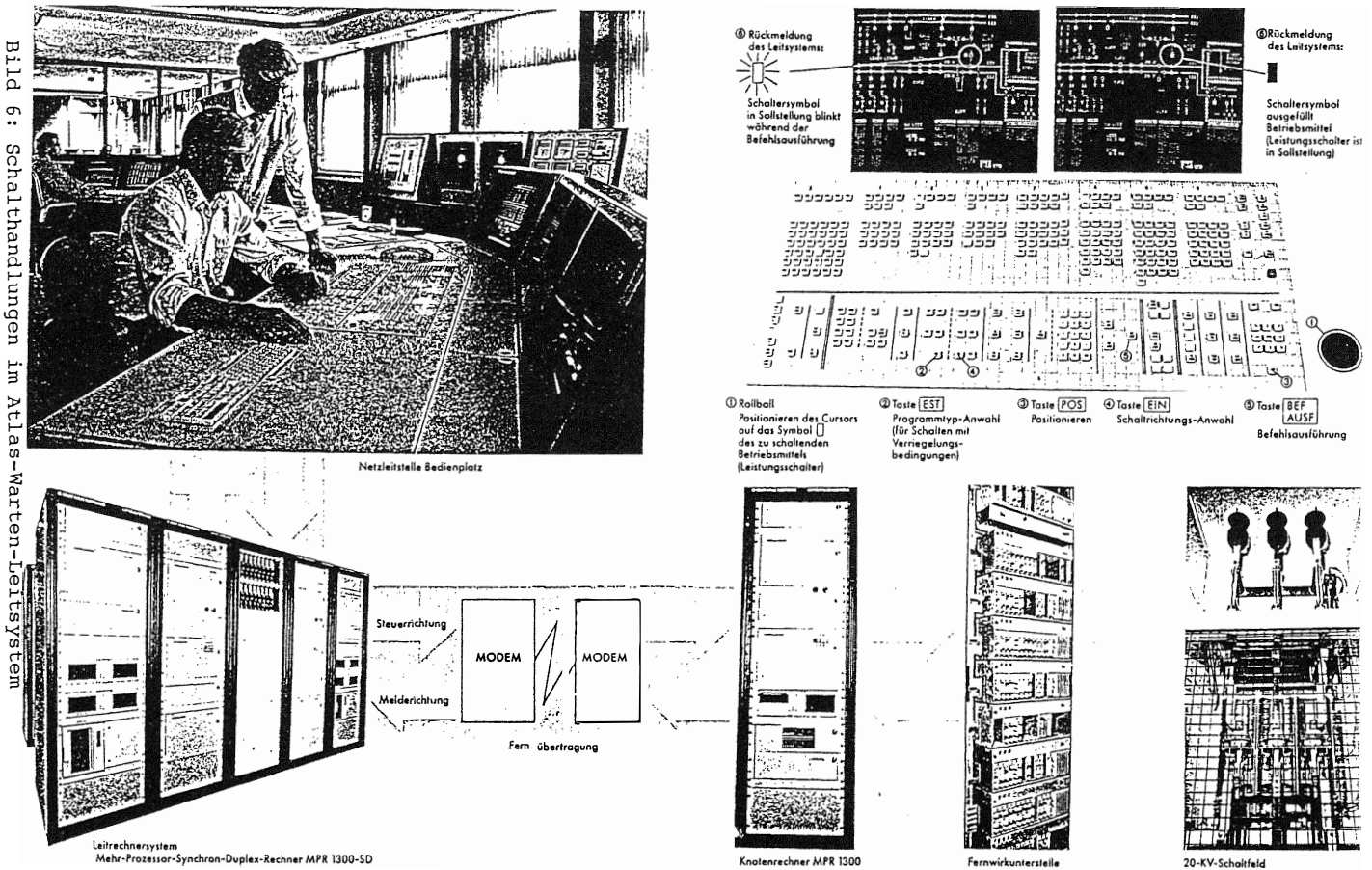


Bild 6: Schalthandlungen im Atlas-Warten-Leitsystem AWL 1300 E

7. Anforderung an die Programmentwicklung

Wesentliche Voraussetzungen für die Programmentwicklung waren

- o Ordnungssystem
- o einheitliche Datenmodellstrukturen
- o PEARL-Precompiler für die zentrale Verwaltung der Datenstrukturdefinitionen
- o Dateiumsetzerprogramme
- o zentrale Verwaltung und Instanzierung des Quelldatenbestandes
- o Generierprogramme für die Erzeugung verarbeitungsgerechter Dateisichten aus dem Quelldatenbestand
- o einheitliches Programmfunktionsgruppengerüst
- o einheitliches Namenskonzept
- o On-Line-Testmonitor

8. Überblick über die PEARL-Implementation

Das Atlas-Warten-Leitsystem AWL 1300 E ist ein modulares Programmsystem, das vorwiegend in PEARL programmiert ist.

Nicht in PEARL geschrieben sind:

- Betriebssystem
- Grundprogramme
- Dienstprogramme
- Datenhaltesystem
  (dieses besitzt eine PEARL-Schnittstelle und wurde vor 4 Jahren hier auf der PEARL-Tagung vorgestellt)

Das System beruht auf ca. 100 Mannjahren PEARL-Programm-Entwicklung. Im Rahmen des hier vorgestellten Projektes sind weitere 20 Mannjahre an Entwicklung für PEARL-Programme (einschl Test und Inbetriebnahme) aufgewendet worden.
Diese Programm-Produktion dauerte 18 Monate, die Projektlaufzeit 3 Jahre.
Das System AWL 1300 E ist implementiert unter dem Multiuser-Betriebssystem MOS von Krupp Atlas Elektronik.
Es besteht aus 100 solcher User oder Programmfunktionsgruppen, die allerdings nicht immer gleichzeitig laufen.
Das System ist aufgeteilt auf mehrere reale oder virtuelle Rechner, und zwar:

- Leitrechner (in der Regel einer) mit 75 Tasks
- Arbeitsplatzrechner (hier 3) mit je 75 Tasks
- Knotenrechner (hier 8) mit 75 Tasks
- Querschnittssoftware (verteilt) mit 10 Tasks

Das ergibt 430 Tasks.
Zur Synchronisation werden ca. 430 Semaphoren verwendet.

Die Kommunikation zwischen den Programmfunktionsgruppen (PFG) oder Usern erfolgt über bis zu 700 Queues, die von einer Kommunikations-Software verwaltet werden. Diese stellt einen Software-Bus dar.

Jede PFG aller realen oder virtuellen Rechner besitzt eine Sende-, eine Empfangs- und eine Rückmeldequeue. Die Kommunikationsuser arbeiten auf globalen Datensegmenten, in denen die Zuordnung der Queues zu den Tasks der einzelnen PFG's in den einzelnen Rechnern enthalten ist. Die Segmente werden bei Systemstart von den Kommunikations-PFG's aus einem Initialisierungsfile aufgebaut. Den Transport der Daten übernimmt das KAE-Queueing-System (Kommunikationsoftware im Betriebssystem).

9. Ausblick

Die Systemanforderungen werden erhöht durch die

- o Komplexität der Aufgaben
- o Integration von Fremdsystemen
- o flexible Fenster- und Menuetechnik bei der Gestaltung der Bedienoberfläche
- o Steigerung des Bildinformationsgehaltes bei Verwendung vollgrafischer Bildelemente
- o Integration schneller Übertragungssysteme (z.B. LWL)

Krupp Atlas Elektronik wird diese Anforderungen bei der Weiterentwicklung des Atlas-Warten-Leitsystems berücksichtigen.

Literatur

/1/ H.Püffel: Darstellung und Eingabe der Informationen beim Netzleitsystem ORWEL der ÜNH, Hannover-Messe INDUSTRIE 88

/2/ H.Wulf: Vorstellung eines PEARL-Projektes in der Wasser-Gewinnung und -Verteilung, PEARL-Workshop 1987

/3/ VDEW-Arbeitskreis: Netzabbild in Prozeßrechnern

/4/ Jaekel, Röllinger, Wieching: Datenmodell zur Abbildung beliebig strukturierbarer elektrischer Netze, etz. Bd. 105 (1984)

Eine PEARL-Umgebung unter einem "Echtzeit"-UNIX

Krupp Atlas Elektronik GmbH
Hans-Dieter Worm
Sebaldsbrücker Heerstr. 235
2800 Bremen 44
☎ 0421/457-3153

1. Übersicht

Es werden Hintergrund, Konzept und Zukunftsperspektive einer Rechnerentwicklung und die Verbindung zweier Betriebssysteme beschrieben. Die Ausgangslage ist eine 16bit Prozeßrechnerfamilie EPR/MPR1300 mit dem Realzeitbetriebssystem MOS1300 und für Anwendungen die Sprache PEARL. Es wird eine 32bit Rechnerfamilie entwickelt, die die Produktpalette nach oben hin erweitern soll. Dabei stützt sich die Entwicklung auf marktübliche Standards, ohne die in unserem Haus übliche Aufwärtskompatibilität, bzw. Portierbarkeit der Software aus dem Auge zu verlieren. Erstmalig laufen die Rechner der neuen Familie mit dem Betriebssystem UNIX System V Release 3.1.

Vor diesem Hintergrund wird ein Konzept entwickelt das es gestattet, beide Betriebssysteme gleichzeitig auf einem Rechner zu betreiben. Der Vortrag beschreibt die Implementation der beiden Systeme unter Ausnutzung der Mehrprozessorarchitektur und die Kommunikationsmechanismen untereinander. Anhand eines Beispiels wird die Kommunikation einer PEARL-*task* unter dem MOS mit einem C-Programm unter UNIX gezeigt.

Als Zukunftsperspektiven werden weitere Entwicklungen oder Planungen geschildert, wie die Verwendung von UNIX als *file server* für das MOS, oder die Einbindung eines MOS-Filesystems in ein UNIX-Filesystem und die Anwendung des Konzepts für die Verbindung von MPR1300 MOS zu MPR2300 UNIX.

2. Ausgangssituation und Hintergrund

Die Ausgangsbasis besteht aus einer Rechnerfamilie EPR/MPR1300, deren Rechner auf einem *Bitslice*-Prozessor basieren. Die Firmware dieses *Bitslice*-Prozessors gestattete es uns, eine sehr effiziente Implementierung unserer Sprache META zu entwickeln. Die Firmware wurde in einer META-ähnlichen Sprache entwickelt, um auch auf dieser Basis portierbar auf weitere Rechner des Hauses zu sein. Auf Basis von META wurde unser Realzeitbetriebssystem MOS1300 entwickelt, das somit ebenfalls portierbar ist. Durch die gleiche Firmware war es möglich, ein ebenso effizientes PEARL zu implementieren (siehe [1]). Unter Verwendung von PEARL wurden in unserem Hause viele Softwareprodukte entwickelt. Dabei nimmt die Entwicklung des Atlas Warten Leitsystems einen wesentlichen Anteil ein. Dieses AWL wurde mehrfach in großen Leitsystemen der Ver- und Entsorgung installiert. Unter anderem aus diesem Einsatzgebiet kam dann die Anforderung nach einer Erweiterung der bestehenden Rechnerlinie. Dabei waren folgende zusätzlichen Forderungen gegenüber der bisherigen Rechnerlinie zu berücksichtigen:

- 32 bit Technologie,
  Einsatz eines Standard-Prozessors,
- Einsatz eines Standard-Bus-Systems
- zusätzlicher Einsatz eines Standard-Betriebssystems

Nach eingehenden Untersuchungen wurde die Entscheidung getroffen, einen Mehrprozessorrechner MPR2300 auf Basis des Motorola 68020/30 Prozessors zu entwickeln. Als Bussystem entschied man sich für den VMEbus. Das Betriebssystem wurde vom MPR1300 übertragen und angepaßt bzw. erweitert. Das MOS2300 läuft aber nicht wie beim MPR1300 auf der Firmware-Schale, sondern auf einem Nukleus, dem MOS-Kern. Dieser Kern wurde in der Programmiersprache META-K erstellt, die sich wesentlich an den Strukturen unser Standardsprache META orientiert, aber optimal auf Programmierung von Prozessoren der Motorola 68000 Serie abgestimmt ist. Das Konzept ähnelt dem Verfahren, das bereits für die Erstellung der Firmware des MPR1300 verwendet wurde. Die Basis-Programme des MOS1300, wie Editoren etc., wurden ebenfalls portiert. Dazu gehören natürlich auch die Compiler, die Testmonitoren und sonstige Hilfsprogramme für die Programmiersprachen META und PEARL. Zu diesen Portierungen möchte ich noch erwähnen, daß wir keine schlichte Portierung unter Mitnahme der 16-Bit-Einschränkungen wollten, sondern zusätzlich zu der reinen Übertragungsarbeit, (natürlich) eine Überarbeitung der Programmsysteme erfolgte, um an den sinnvollen Stellen auf 32 Bit umzustellen. Als letzte Forderung des Vertriebes blieb noch die Implementation eines Standard-Betriebssystems. Wir implementierten also eine Portierung des original AT&T

UNIX System V Release 3.1. Damit hatten wir einen Mehrprozessorrechner, den wir sozusagen "pur" mit einem Mehrprozessor MOS, oder alternativ als Einprozessorrechner, mit UNIX betreiben konnten. Mit Erreichen dieses Status kamen nun sofort die Forderungen, auf beiden Betriebssystemen jeweils die *tools* des anderen Betriebssystems zu implementieren. Diese Forderung hätte jedoch eine quasi Verdopplung der Entwicklungsaufwendungen zur Folge gehabt, ganz zu schweigen von den unterschiedlichen Philosophien, die die Systeme MOS und UNIX verfolgen. Hier war also eine sinnvolle Verbindung der Systeme gefordert.

3. Der UNIX/MOS-Übergang

3.1 Das Konzept

Wie sieht in diesem Umfeld eine sinnvolle Verbindung der Systeme aus? Der erste Gedanke einer Rechnerkopplung der beiden Systeme ist sicher eine der Lösungen, die man nicht außer acht lassen kann. Im Projektgeschäft kann das aber eine zu teure Lösung sein, man benötigt immerhin einen kompletten zweiten Rechner. Also war eine Verbindung in einem Rechner gefordert. Die Idee, UNIX als eine *task* unter MOS laufen zu lassen, scheitert an zu vielen Änderungen im UNIX-Kern. Damit entfernt man sich zu weit vom Standard und hat Probleme mit der Überführung auf neue Releases von AT&T. Außerdem würde die Fehlersuche erheblich erschwert werden. Also entwickelten wir ein Konzept, daß die Systeme unter Ausnutzung der Mehrprozessorarchitektur auf einem gemeinsamen VMEbus arbeiten läßt. Jedes System sollte im ersten Ansatz seine eigene Peripherie erhalten, so daß die weitestmögliche Entkopplung der Systeme erreicht wird. Nur so kann später die Forderung nach Nutzung derselben Verbindung zwischen zwei Rechnersystemen mit MOS und UNIX und eine Verbindung mit MPR1300-Rechnern realisiert werden. Diese einfache Verbindung gestattet es später auch, I/O-Servicefunktionen des einen Systems für das andere nachzurüsten und somit einen Teil der doppelten Peripherie wieder einzusparen. Das Konzept wurde in mehreren Teilschritten realisiert. Im ersten Schritt wurden die beiden Systeme auf einen Rechner gebracht und die gesamte *Boot*- und Startphase entwickelt. Dies war der komplizierteste Teil der Entwicklung. Der zweite Schritt bestand in der Realisierung einer schnellen Kommunikationsverbindung zwischen den Systemen, wobei wir zunächst mit einer einfachen Speicherkopplung begannen und zur Zeit auf ein *buffered pipe protocol* übergehen. Im dritten Schritt wurden dann drei *tools* entwickelt, die den Datentransport zwischen den Systemen unterstützen und das Arbeiten auf dem anderen Betriebssystem ermöglichen. Diese *tools* bieten sowohl eine Bediener-, als auch eine Softwareschnittstelle.

3.2 *Boot* und Start der Systeme

Nach dem *reset* des Rechner-Systems lädt eine *boot*-Routine, entweder automatisch oder dialoggesteuert, die beiden Systeme und startet sie auf den jeweils angegebenen Prozessoren. Im automatischen *boot* wird ein UNIX-Filesystem vorrausgesetzt, in dem ein Textfile genau die Angaben enthält, die auch per Dialog eingegeben werden können. Im Dialog ist anzugeben, welches System zu laden, und auf welchem Prozessor es zu starten ist. Dabei ist die Angabe des Systems ein Filename im UNIX-Filesystem, z.B. /usr/src/uts/unix oder /kae/mos. Ebenso kann das MOS auch über ein *autoload device* (*softdisk* oder *pseudotape*) geladen werden. Wenn z.B. ein MPR2300 mit den Prozessoren 0 bis 4 ausgerüstet ist, so wird über

auto: 0-3>/kae/mos
auto: 4>/unix
auto: "*return*"

den Prozessoren 0 bis 3 das MOS und dem Prozessor 4 das UNIX zugeordnet. Dann werden alle 5 Prozessoren gestartet. Die Systeme haben generierbare disjunkte Speicherbereiche, die sie jeweils überprüfen um ihren Speicherplatz zu ermitteln. Die Systeme starten nahezu unabhängig voneinander. Lediglich die Verteilung der *clock*-Impulse wurde zunächst dem MOS zugeordnet, so daß das UNIX-System am Ende der Startphase auf den ersten *clock interrupt* wartet. Im MOS wird daher während der Initialisierung dem MOS-Kern per Schalter gesteuert mitgeteilt, daß der *clock interrupt* an den generierten UNIX-Prozessor weiterzugeben ist. Auf alle weiteren *shared resources* wurde in den ersten Implementationen bewußt verzichtet. Somit haben z.B. beide System getrennte *system operator terminals*, die später wieder auf einem Gerät abgebildet werden können. Die Peripherie des Rechners ist nun durch entsprechende Vereinbarungen und Systemgenerierungen jeweils genau einem Betriebssystem zugeordnet. Benötigen die Geräte *interrupts*, so werden sie so konfiguriert, daß die *interrupt request line* des entsprechenden Systems benutzt wird, z.B. MOS *line* 2, UNIX *line* 3.

3.2 Das Transportsystem

Das Transportsystem stellt in der Terminologie der Rechnerkopplungen lediglich die unterste Ebene dar, die physikalische Transportebene. Ein wesentlicher Punkt für die Flexibilität der Lösung ist ihre Einfachheit. Nur dadurch, daß das Transportsystem quasi ein Draht zwischen den Systemen ist, kann diese Schicht auch durch eine komfortable Rechnerkopplung zwischen getrennten Rechnern ersetzt werden, z.B. Ethernet und TCP/IP. Das einfachste Transportsystem ist eine Speicherkopplung über *shared memory*.

Beiden Systemen wird ein gemeinsamer statischer Speicherbereich durch die Systemgenerierung bekannt gemacht, in dem zwei *mailboxes* realisiert sind. Jede *box* enthält eine Kontroll-Struktur in der über *flags* oder Variable der jeweilige Status und z.B. die Länge der enthaltenen *message* angezeigt werden. Für jedes System ist eine *box* für das Senden von Nachrichten zum Partner vorgesehen, die das System aktiv zu füllen hat. Das jeweils andere System entnimmt die Nachricht aus dieser *box*, nachdem es per *interrupt* über das Vorliegen einer Meldung informiert wurde.

3.3 Realisierung

Sowohl auf der MOS, wie auch auf der UNIX Seite sind die Transportsysteme durch Treiber realisiert, d.h. unter UNIX als einfacher *character*-orientierter Treiber. Auf der MOS-Seite gehört zu dem Treiber immer ein Software-Prozessor dazu, sozusagen der asynchrone Teil des Treibers. Für die Entwicklung war es wichtig, keinerlei Eingriffe in die UNIX-Kern-*Sources* zu machen, also nur dort zu ändern, wo die Systeme es erlauben. So kann garantiert werden, daß der UNIX/MOS-Übergang auch auf weitere UNIX-Releases übertragen werden kann. Die Speicherkopplung wird von den Treibern so genutzt, daß Nachrichten von den jeweiligen Treibern in diese Speicher kopiert, bzw. aus diesen Speichern heraus kopiert werden. Die lesenden Teile der Treiber enthalten *buffer*-Mechanismen, auf der MOS-Seite sind das *queues* (siehe [2]), auf der UNIX-Seite wurden zunächst eine Reihe statischer *buffer* dafür zur Verfügung gestellt. Der jeweils aktive oder schreibende Teil bei dieser Kommunikation prüft, ob seine entsprechende *box* zu Verfügung steht, füllt sie mit den entsprechenden Daten, setzt die Elemente der Kontrollstruktur und informiert seinen Kommunikationspartner durch einen Interprozessor-*interrupt*. Dieser *interrupt* wird durch die Ansprache des *location monitors* des entsprechenden MOS-Prozessors vom UNIX-Treiber erzeugt, bzw. umgekehrt, wenn der MOS-Treiber seine *box* gefüllt hat. Der so geweckte *interrupt*-Teil des Treibers entnimmt nun die Nachricht und kopiert sie in seinen *buffer* bzw. seine *queue*. Die weitere Bearbeitung erfolgt nach den in dem jeweiligen Betriebssystem üblichen Regeln.

3.4 Verbindungsaufbau

Verbindungen zwischen den beiden Systemen bestehen zwischen einer *queue* und einem Prozeß, vertreten durch seine *process identification* (*pid*). Vom MOS her werden die Kommunikationspartner der UNIX-Seite als *remote queues* betrachtet. Es gibt im MOS eine lokale *queue*, in die alle *remote*-Nachrichten geschickt werden. Die Nachrichten enthalten als Adresse eine *remote queue* und werden von einem Kopplungstreiber weiter

verteilt, d.h. in diesem Fall an UNIX gesendet. Der Verbindungsaufbau zwischen den Partnern wird durch einen MOS Treiber unterstützt. Es gibt ein *info device*, bei dem Anwendungen angemeldet werden, oder das auf Anfrage die bekannten Anwendungen mitteilt. Dieses *info device* ordnet Anmeldungen nach Themenkreisen. Zur Zeit sind definiert: Terminals, *file transport* und *file system switch*. Diese Themenkreise existieren, um im System implizite Dienstleistungen bei entsprechenden Anmeldungen zu starten. Es können jederzeit beliebige andere Themenkreise eingerichtet werden, die dann keine spezielle Behandlung erfahren. Themenkreise sind durch Nummern gekennzeichnet. Eine Anwendung, die eine Dienstleistung offeriert, meldet sich beim *info device* an, indem sie als Parameter ihre *queue* angibt, auf der sie Aufträge empfängt. Weitere Parameter sind der Themenkreis und zur Unterscheidung in einem Themenkreis ein *identifier*, bestehend aus 4 Zeichen. Eine weitere Anwendung, die einen Service in Anspruch nehmen möchte, kann sich beim *info device* erkundigen, welche Anwendungen angemeldet sind. Dazu wird ein *info* Auftrag in die *queue* des *info device* geschrieben. Als Antwort erhält der Absender eine Liste der Anmeldungen bestehend aus Themenkreis, *queue* Nummer und *identifier*. Eine MOS-*task*, die als *server* arbeitet, wird typischerweise eine *queue* eröffnen, sich mit der *queue* Nummer, dem Themenkreis und einem eigenen *identifier* beim MOS *info device* anmelden und anschließend auf Aufträge in dieser *queue* warten. Aufträge werden beantwortet, indem der *server* eine Nachricht in die *queue* schreibt, die bei Auftragserteilung als Absender angegeben wurde. Antwort*queues* werden also nicht beim *info device* angemeldet. Ein UNIX-Prozeß, der einen MOS-Service in Anspruch nehmen möchte, wird typischerweise ein *open* auf das Kommunikations*device* ausführen und einen *info* Auftrag als *write* auf die *queue* des *info devices* absetzen. Die Antwort gelangt über eine *remote queue* in den UNIX-Kern (*queue* Nummer = *pid* des UNIX-Prozesses), von wo sie über einen *read*-Auftrag zum UNIX-*user* gelangt. Aus den erhaltenen Informationen, kann das UNIX-Programm sich dann den MOS-*server* aussuchen und die Aufträge in dessen *queue* schreiben. Die Antworten erhält es durch den *server*, der Absender und Adresse in *remote header* vertauscht und sein Telegramm an die lokale Vermittlungs*queue* sendet.

3.5 Anwendungen

Basierend auf diesen Mechanismen wurden Anwendungsprogramme entwickelt, die zum Betreiben des UNIX/MOS-Übergangs benötigt werden: ein *file transfer* und ein transparentes Terminal. Ausgangspunkt dieser *tools* ist, daß UNIX seine Stärken als Entwicklungsumgebung hat, während das MOS als Ablaufumgebung dominiert. Daher wurde unter

dem MOS das *info device* realisiert, so daß Programme unter UNIX, also von der Entwicklungsumgebung aus, auf Dienstleistungen des MOS zugreifen können. Unter UNIX wurde ein Programm (fts = *file transport service*) entwickelt, das *files* zwischen den Systemen kopiert. Im MOS wird während der Initialisierung eine *task* gestartet, die sich beim *info device* als *server* für den Themenkreis *file transport* anmeldet. Das *fts* erfragt nun über das MOS *info device*, welche *queue* den *file transport* unterstützt und bearbeitet mit seinem MOS-Partner die Kopieraufträge zwischen den Systemen. Dabei sind Datenkonvertierungen zwischen den Systemen notwendig. MOS *files* enthalten typischerweise eine *record*-Struktur, während UNIX *files* lediglich als *byte stream* realisiert sind. Die beteiligten Transportprogramme konvertieren Textfiles zwischen *records* und entsprechend eingestreuten *new lines.* Bei den binären Files ist per Option wählbar, ob die *record*-Struktur des MOS-Files mit zu übertragen ist oder nicht. Soll die Struktur bei späteren Transporten wieder reproduzierbar sein, werden sämtliche *record*-Informationen in die Daten eingetragen.

Das transparente Terminal wurde in zwei Varianten entwickelt, als manuell zu bedienendes Terminal und als programmgesteuertes Termial. Damit kann ein *user* unter UNIX sein Terminal transparent auf die MOS-Seite durchschalten, das sich dann identisch zu einem MOS-Terminal verhält. Ein spezielles *control character* bringt ihn auf die *user* Ebene zurück. Das programmgesteuerte Terminal ist von einem Anwendungsprogramm aus per *fork/exec* zu starten und dann über *messages* vom Programm aus zu bedienen. Mit Hilfe dieses Programmes ist man in der Lage eine Art von *remote job entry* zu implementieren. Die beiden Terminalprogramme verhalten sich ansonsten gleich. In der Startphase des Terminalprogramms wählt es einen MOS-*server* aus dem Themenkreis Terminal des MOS *info devices*. Durch den Auftrag *request terminal* an das MOS *info device* wird die Terminalverbindung initiiert. Dies ist einer der Fälle, wo das *info device* für den Themenkreis Terminal eine spezielle Systemfunktion auszulösen hat. Es wird bei Anforderung eines Terminals ein entsprechendes Treiberprogramm, der Terminalprozessor, gestartet, der wiederum dafür sorgt, daß sich ein *remote* Terminaltreiber bei dem Programm für das transparente Terminal meldet. Nach dieser Initialisierungsphase gibt der UNIX-Prozeß alle Terminaleingaben an das MOS weiter, wo der Terminalprozessor mit dem Treiber dafür sorgt, daß die Eingaben wie Terminaleingaben aussehen und behandelt werden. Die Antworten werden über *remote queues* an den UNIX-Prozeß zurückgesandt und an das Terminal ausgegeben.

3.6 MOS PEARL Kommunikation zu UNIX C

Im nebenstehenden Programm soll anhand eines einfachen Beispiels der Einsatz des UNIX/MOS-Überganges demonstriert werden. Hintergrund ist hier der Einsatz des MOS als "Realzeit-Verstärker" des UNIX-Systems. Das PEARL-Programm offeriert als Dienstleistung beispielhaft die zyklische Messwerterfassung, deren Daten unter UNIX z.B. archiviert, oder in Form von Langzeitstatistiken einer Datenbank zugeführt werden sollen. Zunächst werden die Strukturen für die *remote queue header* und die Telegramme definiert. Der *remote queue header* kann auch bei lokalen *queues* verwendet werden. Es werden jeweils für Empfänger und Absender (*dest, source*) die *queue*-Nummern und die zugehörigen *link*-Prozessoren (Rechner) definiert. Eine weitere *queue* wird bei *remote* Verbindungen als Vermittlungs*queue* vom MOS eingetragen, um bei Antworten den lokalen Vermittler zu kennzeichnen. Telegramme bestehen aus dieser *header*-Struktur und einem Datenteil. In diesem Beispiel gibt es das Format für die Anmeldung beim *info device*, ein Auftragstelegramm zum Starten der zyklischen Erfassung und ein Datentelegramm, das als Ergebnis der Erfassung an das UNIX-Programm zurückgesendet wird.

Das Dienstleistungsprogramm eröffnet zunächst seine Auftrags*queue* und meldet sich dann beim *info device* an. Dazu wird das entsprechend formatierte Telegramm in die lokale *queue* eingetragen. Das *info device* bestätigt die Anmeldung mit einem Telegramm in der Auftrags*queue*. Anschließend erfolgt in der Schleife ein Lesen der Auftrags*queue*. Zeigt die Option einen Erfassungsauftrag an, so wird die *task* CYCL entsprechend gestartet. Im anderen Fall wird die *queue* geschlossen und das Programm terminiert.

In der *task* CYCL wird nach der Meßwerterfassung und Datenaufbereitung ein Antworttelegramm an die beauftragende UNIX-Software geschickt. Dazu werden die Daten von *destination queue* und Linkprozessor der Auftrags*queue* als Absender in den *header* des Datentelegramms eingetragen, während die Absenderparameter des Auftragstelegramms (*source queue* und Linkprozessor) als Empfänger eingetragen werden. Das Telegramm wird an die lokale Vermittlungs*queue* geschickt.

Das zugehörige C-Programm führt ein *open* auf ein *special file* /dev/muc aus und schreibt über dieses MOS-UNIX-Kopplungs*device* einen Informationsauftrag (OPT = 1) an das *info device* auf der MOS-Seite. Es erhält als Antwort auf einen weiteren *read* die Liste der angemeldeten Dienstleistungen und wählt über den Themenkreis "zyklische Erfassung" (= MESSEN) und den *identifier* "CYCL" seinen Partner, bzw. dessen *queue* aus. In diese *queue* wird nun der Auftrag für eine Meßperiode geschrieben. Als Absender trägt der Treiber die *process id* ein. Die Antworttelegramme werden in einer Schleife durch *read* Aufträge gelesen und verarbeitet.

```
MODULE MESRV; /* MESSWERT ERFASSUNGS SERVER */
PROBLEM;
TYPE     RMQHD    STRUCT  [ DESTQ    FIXED,     /* DESTINATION QUEUE     */
                            DESTL    FIXED,     /* DEST. LINK PROCESSOR */
                            SRCQ     FIXED,     /* SOURCE QUEUE          */
                            SRCL     FIXED,     /* SRC. LINK PROCESSOR   */
                            LOCALQ   FIXED];    /* LOCAL QUEUE           */
DCL      INFO     STRUCT  [ QHD      RMQHD,     /* REMOTE QUEUE HEADER */
                            OPT      FIXED,     /* INFO OPTION           */
                            LNG      FIXED,     /* EXTENDED LENGTH       */
                            STA      FIXED,     /* STATUS WORD           */
                            THEMA    FIXED,     /* THEMENKREIS           */
                            ID       CHAR (4),  /* IDENTIFIER            */
                            QUEUE    FIXED,     /* SERVICE QUEUE NUMBER*/
                            QSTA     FIXED];    /* QUEUE STATUS          */
DCL      JOB      STRUCT  [ QHD      RMQHD,     /* REMOTE QUEUE HEADER */
                            OPT      FIXED,     /* MESS-OPTION           */
                            INTERVALL  DURATION,
                            ENDTIME  CLOCK];
DCL      MEDAT    STRUCT  [ QHD      RMQHD,     /* REMOTE QUEUE HEADER */
                            WERT     FIXED];    /* MESSWERT              */
DCL  STV  STRUCT [ ( STA , LNG , QNR )  FIXED ];
DCL  OPB  STRUCT [ ( TYP , MAXL , MAXN )  FIXED ];
MAIN: TASK;
   OPB.TYP = 0;  OPB.MAXL = 50;  OPB.MAXN = 5;
   OPEN_QUEUE( QNR , STV , OPB );
   INFO.THEMA = MESSEN; INFO.ID = "CYCL"; INFO.QUEUE = QNR; INFO.STA = 0;
   PUT_REQUEST ( IQNR , STV , INFO , INFOLNG ); /* ANMELDUNG INFO DEVICE */
   GET_REQUEST ( QNR , STV , INFO , INFOLNG );  /* ANMELDUNGSQUITTUNG    */
   REPEAT;
      GET_REQUEST ( QNR , STV , JOB , JOBLNG ); /* AUFTRAG LESEN */
      IF ( JOB.OPT  EQ  MESSEN )
         THEN  ALL JOB.INTERVALL  UNTIL  JOB.ENDTIME  ACTIVATE  CYCL;
         ELSE  CLOSE_QUEUE ( QNR , STV );     TERMINATE;
      FIN;
   END;
END;
CYCL:  TASK;
   /* MESSWERTERFASSUNG UND DATENAUFBEREITUNG */
   MEDAT.QHD.DESTQ   = JOB.QHD.SRCQ;      /* SET DESTINATION QUEUE     */
   MEDAT.QHD.DESTL   = JOB.QHD.SRCL;      /* SET DESTINATION LINK-PROC */
   MEDAT.QHD.SRCQ    = JOB.QHD.DESTQ;     /* SET SOURCE QUEUE          */
   MEDAT.QHD.SRCL    = JOB.QHD.DESTL;     /* SET SOURCE LINK PROC      */
   PUT_REQUEST ( MEDAT.QHD.LOCALQ , STV , MEDAT , MEDATLNG );
END;
```

Unter UNIX würde man für solche Aufgaben typischerweise ein Startprogramm für die Auswahl des Servicepartners erstellen. In diesem Programm würde dann die Dialogführung mit dem Bediener und die Übermittlung der Erfassungsaufträge an das MOS durchgeführt. Für die Auswertung würde über *fork/exec* ein eigenes Programm gestartet.

4. Weiterentwicklungen

4.1. *Buffered Pipe Protocol*

Das *buffered pipe protocol* liefert eine Standardmethode zum Austausch von Nachrichten zwischen Prozessoren, die auf einem VMEbus arbeiten und Zugriff auf einen gemeinsamen Speicher oder einen gemeinsamen Adressbereich haben [3]. Schon diese Zieldefinition von Motorola zeigt die Eignung dieses Protokoll für die hier vorliegende Aufgabe. Als weitere Entwicklung wird zusätzlich zur geschilderten Speicherkopplung dieses *buffered pipe protocol* implementiert. Der Hintergrund dabei ist, daß dieses Protokoll für sogenannte *targets* angeboten wird. Das heißt für die Entwicklung von *firmware* oder *controller* Software für Baugruppen mit Motorola Prozessoren kann dieses Protokoll implementiert werden. Wir werden über dieses Protokoll mit dem Konzept des UNIX/MOS -Übergangs solche *targets* an unsere Betriebssysteme koppeln. Sowohl für den UNIX/MOS-Übergang, wie auch für die Entwicklungsumgebung für *target*-Systeme spezifizieren wir ein gemeinsames Testkonzept. Dabei soll ein *low level debugger* auf dem System des zu testenden Objekts laufen und das Objekt kontrollieren. Unter UNIX wird dann der sprachorientierte Teil des *debuggers* laufen und die Eingaben behandeln. Die Kommunikation erfolgt über den UNIX/MOS-Übergang bzw. über die *target* Anbindung. Für den Bediener steht damit immer eine einheitliche Umgebung zur Verfügung. Inwieweit dieses Verfahren auch zum Test von UNIX-*usern* geeignet ist, wird zur Zeit geprüft. Grundlage dieses *debuggers* wird der PEARL-Testmonitor des MPR1300 werden.

4.2 *Streams*

Der oben geschilderte Treiber und auch der Treiber für das *buffered pipe protocol* werden auf *streams* Treiber bzw. Module umgestellt. *Streams* sind ein neuer Mechanismus im AT&T UNIX (ähnlich den *sockets* im Berkley UNIX), der hauptsächlich als Ablaufumgebung und Unterstützung von Protkoll-Software von Kommunikationen entwickelt wurde. Durch die Umstellung auf *streams* können Teile des Verbindungsaufbaus und des *handshakes* zwischen den Systemen aus den Anwendungsprogrammen in *streams modules* übernommen werden. Desweiteren kann dann der "Draht" zwischen den Systemen, die Speicherkopplung (oder das *buffered pipe protocol*), problemlos durch andere Kopplungen und Prozeduren

ersetzt werden. Das setzt natürlich voraus, daß die Kopplungsprozeduren als *streams modules* und Treiber realisiert sind. Mit dieser Umstellung wird es möglich, die *queues* unter UNIX einzuführen. Ein UNIX-Prozeß kann dann mehrere Verbindungen gleichzeitig unterhalten (Entkopplung von der *process id*), da über *streams* ein *multiplex/demultiplex* Modul einfach implementiert werden kann. Zusätzlich wird es dann einfach möglich, ein *info device* unter UNIX zu entwickeln. Damit würde der Übergang zwischen beiden Systemen symmetrisch, was insbesondere bei der Verbindung mehrerer Rechner von Vorteil ist. Jeder Rechner verwaltet dann eine Liste seiner Dienstleistungen, die über *remote* Verbindungen von anderen System abgefragt und genutzt werden können.

4.3 VME-Subsysteme und Rechner-Rechner Übergang

Unter Verwendung dieser Kopplungssoftware und einer VME-VME Kopplungshardware werden wir den UNIX/MOS-Übergang von einem VMEbus-System auf ein System von einem VMEbus mit bis zu 3 VME-Subsystemen übertragen. Diese Konfigurationen wurden bereits beim EPR/MPR1300 durch Grund- und Erweiterungseinschübe realisiert. Die Koppelhardware verbindet zwei VMEbus-Systeme miteinander, wobei ein VMEbus die Funktion eines *master* übernimmt. Zugriffe vom *master* in ein VME-Subsystem sind für die Programme transparent. Vom Subsystem zum *master* ist eine *interrupt* gesteuerte Kommunikation realisiert. Damit können die Systeme noch weiter entkoppelt werden, und die gegenseitigen Einflüße werden weiter reduziert, die Fehlersuche wird erleichtert. Diese Konfiguration wird dann auch auf die Kopplung zweier Rechner mit getrenntem VMEbus erweitert, um größere Systeme zu realisieren. Der UNIX/MOS-Übergang ist nach dem vorgestellten Konzept geeignet, die Systeme MPR1300 MOS mit dem MPR2300 UNIX zu verbinden. Diese Systemverbindung wird dadurch realisiert, daß die Erweiterungen des MOS2300 auf das MOS1300 übertragen werden. Das ist einfach möglich, da beide Betriebssysteme in META programmiert sind.

4.4 *File system switch* und *file server*

Als weitere Entwicklung läuft zur Zeit die Einbindung eines MOS Filesystems in ein UNIX Filesystem. Unter Benutzung des UNIX *file system switch* kann ein MOS Filesystem in den Baum des UNIX Filesystems eingehängt werden (*mount*). Die *files* des MOS sind dann für den UNIX Bediener und die Programme nicht von UNIX *files* zu unterscheiden. Erstmalig mit dem Release 3 des System V hat AT&T diesen *file system switch* im UNIX Kern implementiert (siehe auch [4]). Es handelt sich um eine Schnittstelle im Kern, dessen Syntax und Semantik für die verschiedenartigen Zugriffe auf *files* und Filesysteme

spezifiziert ist. Zur Einbindung eines neuen Filesystems sind ca. 25 neue Kernroutinen zu entwickeln, die für das MOS-Filesystem an den notwendigen Stellen über den UNIX/MOS-Übergang mit einem entsprechenden Partner kommunizieren.
Der umgekehrte Weg, die Implementation eines MOS *file systems* auf einer UNIX-Platte, und die *server* Funktionen, sind bereits spezifiziert. Diese Entwicklung ist bei Kleinsystemen sinnvoll, wenn in der Realzeitumgebung wenig Plattenaktivitäten erforderlich sind. Auf der UNIX-Platte wird ein *minor device* für ein MOS Filesystem reserviert. Das UNIX übernimmt dann eine *server* Funktion auf der Basis von Block-I/O Aufträgen. Unter dem MOS ist dafür ein Pseudotreiber zu entwickeln, der als neuer Plattentreiber oder *harddisk*-Treiber in das MOS eingebunden wird und die Kommunikation mit dem UNIX *server* abwickelt.

5. Perspektiven

5.1 Multiprozessor-UNIX

Zur Zeit wird im Rahmen eines Verbundprojektes für unseren Rechner ein Multiprozessor-UNIX auf Basis des *local shared memory* Konzeptes entwickelt [5]. Nach dem bisherigen Kenntnisstand ist es möglich, den UNIX/MOS-Übergang auch für dieses Multiprozessor-UNIX zu implementieren, ohne dessen *source code* zu benötigen. Wir werden das nach Abschluß der Multiprozessor-UNIX-Entwicklung im Laufe des nächsten Jahres implementieren und damit den Nachweis erbringen, das die Portierung des UNIX/MOS-Übergangs auf ein "anderes UNIX" lediglich eine Portierung von Treibern ist.
Damit bildet die Rechnerfamilie, angefangen vom EPR/MPR1300 mit Ein- und Mehrprozessor-MOS bis zum MPR2300 mit Mehrprozessor-MOS und Mehrprozessor-UNIX zusammen mit dem UNIX-MOS-Übergang, die Basis für die Realisierung komplexer dezentraler Systeme.

5.2 Realzeit-UNIX

Zur Zeit der Drucklegung dieses Papiers gibt es noch keine, bzw. noch wenig Informationen über die sogenannten Realzeit-Erweiterungen des UNIX System V Release 4, der Ende 1989 als ß-Release für den 3B2 angekündigt ist. Diese und ähnliche Entwicklung, z.B. AIX, wird man weiter aufmerksam beobachten müssen. Daraus wird sich evtl. einmal ein standardisiertes Echtzeit-Betriebssystem entwickeln. Ebensolche Aussichten bestehen allerdings für das VMEexec Projekt von Motorola, bzw. die RTEID. Im Augenblick ist jedenfalls kein Anbieter zu erkennen, der einen deutlichen Vorsprung in Form eines entsprechenden Marktanteils hat, so daß man einen de facto Standard feststellen

könnte. Diese Strategie wird unter anderem von X/OPEN angewendet um Standards festzustellen, in den *Portability Guides* festzuschreiben und damit zu fördern. Dort gibt es zur Zeit keine Verlautbarungen über die Standardisierung von Realzeiteigenschaften. Allerdings arbeiten sowohl AT&T, wie auch X/OPEN im IEEE–Komitee P 1003 mit, die im Rahmen der herstellerunabhängigen Schnittstellendefinition von Standard-Betriebssystemen unter P 1003.4 *Realtime facilities* spezifizieren (siehe [6]). Damit ist anzunehmen, daß das neue Release 4 von AT&T diesem Standard entspricht. Es wird aber einige Zeit dauern, bis dieses System verbreitet ist und es bleibt abzuwarten, ob es sich gegen andere Produkte durchsetzen kann. Somit wird uns ein vom Markt anerkanntes standardisiertes Realzeitsystem wohl noch für etliche Zeit vorenthalten werden. Diese Lücke wollen wir bei Krupp Atlas Elektronik für unsere Rechner mit der Kombination unserer bewährten MOS-Realzeitsysteme und dem standardisierten UNIX ausfüllen.

Literatur

[1] Brüning, C., Landsberg, G., Krupp Atlas Elektronik, Bremen:
Implementation von PEARL auf dem Mehrprozessorrechner MPR1300.
Tagungsband PEARL 84, Düsseldorf, Hrsg.: PEARL-Verein e.V.

[2] Bockhoff, W., Krupp Atlas Elektronik, Bremen:
Ein Botschaftssystem mit PEARL-Schnittstelle zur Kommunikation in verteilten Systemen. Tagungsband PEARL 83, Düsseldorf, Hrsg.: PEARL-Verein e.V.

[3] Motorola Inc.:
Common Enviroment User's Manual, Appendix A Buffered Pipe Protocol.
Motorola Microcomputer Division 1986

[4] Meyer, A., Stollmann GmbH, Hamburg:
Ein schnelles Filesystem unter dem UNIX 5.3 Filesystem-Switch
Tagungsunterlagen GUUG-Jahrestagung '88

[5] Klemke, G., Stollmann GmbH, Hamburg:
SUPRENUM - ein Parallel-Superrechner.
Tagungsunterlagen GUUG-Jahrestagung '88

[6] Windauer, H., Werum GmbH, Lüneburg:
UNIX als Entwicklungsumgebung für Realzeit-Anwendungen - Ein Überblick –
Tagungsband PEARL 87, Boppard, Hrsg.: PEARL-Verein e.V.