STEUERUNG EINER RELATIONALEN DATENBANKMASCHINE MIT PEARL

Dipl. Inform. Hans von Kleist-Retzow, Braunschweig
Dipl. Inform. Frank Hildebrandt, Braunschweig

Kurzfassung

Die im Rahmen des Forschungsprojekts RDBM konzipierte und realisierte Multiprozessor-Datenbankmaschine erfordert einen hohen Software-Aufwand für die Steuerung und Koordination der einzelnen Prozessoren. Zu Beginn des Projekts wurde PEARL als Implementierungssprache für diese Steuerung ausgewählt. Nach fast 5 Jahren mit PEARL kann man die Erfahrungen so zusammenfassen: Die Realzeit-Sprachkonstrukte sind praktikabel und erleichtern die Arbeit; ebenso das Typenkonzept in der benutzten PEARL-Implementierung. Vermißt wird ein Konzept zum Informationsaustausch zwischen Prozessen. Letztendlich hat nicht nur die Qualitat der Sprache, sondern auch die des Compilers schwerwiegende Auswirkungen auf die Produktivität des Programmierers.

Schlüsselwörter: Datenbanksystem, Datenbankmaschine, Prozeßkommunikation

Abstract

The multiprocessor database machine planned and implemented in the RDBM research project requires a high amount of software for the control and coordination of the processors. At the beginning of the project PEARL was chosen as implementation language. After almost 5 years with PEARL the experiences can be summarized as follows: The realtime language constructs are well suited for this application and so is the type concept in the PEARL implementation used. We missed a concept dealing with information interchange between processes. However, not only the quality of the language, but also that of the compiler is of special importance for the programmer´s productivity.

Keywords: database system, database machine, process communication

1. Einleitung

Das relationale Datenmodell wurde durch E.F. Codd 1970 in die Datenbankwelt eingeführt. Es hat gegenüber den bisherigen Ansätzen zur Verwaltung von Daten den großen Vorteil, daß sich der Benutzer nicht mehr um Zugriffspfade kümmern muß, wie sie z.B. im CODASYL-Modell definiert werden. Die ersten Implementierungen zeigten jedoch sehr schnell, daß dies zu schweren Effizienzproblemen führt.

In Software-Implementierungen wurden deshalb schon früh Indexstrukturen eingeführt, die die Abarbeitung von Kommandos an das Datenbanksystem beschleunigen helfen.

Ein anderer Ansatz besteht darin, die zeitaufwendigen Operationen durch spezielle Hardware zu unterstützen. So gibt es seit den frühen 70er Jahren eine Reihe von Konzepten für sogenannte Datenbankmaschinen. An der TU Braunschweig wird diese Richtung seit den Anfangen in Zusammenarbeit des Instituts für Datenverarbeitungsanlagen (Prof. Dr. Leilich) und des ehem. Lehrstuhls D für Informatik, jetzt Institut für Theoretische und Praktische Informatik, Abteilung für Betriebssysteme und Rechnerverbund (Prof. Dr. Stiege), verfolgt.

Das erste Projekt "SURE" fand ca. 1974-1978 statt. Zur Unterstützung des Suchens in großen Tabellen (Datenfilterung) wurde ein Suchrechner gebaut, der mit Hilfe spezieller Hardware und Ausnutzung von Pipeline-Konzepten alle Oberflächen einer Platte parallel durchsuchen kann. Es wurde dabei eine Datenrate von über 7 MB/sec. erreicht, wobei bis zu

14 verschiedene Suchanfragen gleichzeitig bearbeitet werden konnten. Die gesamte Platte (72 MB) konnte in ca. 20 Sek. durchsucht werden.

Das Nachfolgeprojekt "RDBM" läuft seit 1979. Es wird eine Maschine gebaut, die nicht nur die Suche unterstützt, sondern alle in relationalen Datenbanken vorkommenden Operationen; daher der Name Relationale DatenBankMaschine.

2. Gesamtstruktur der RDBM

Zur Durchführung der relationalen Operationen wurde ein System bestehend aus Steuerrechner und zusatzlichen Hardwarekomponenten entworfen. Ein Überblick gibt Bild 1.

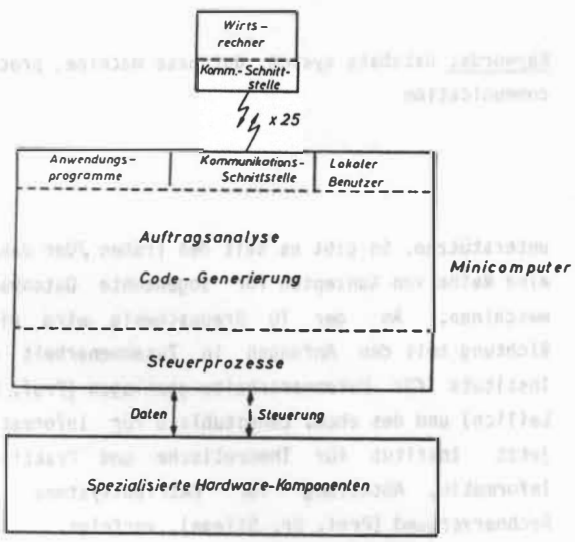

Bild 1: Gesamtstruktur der RDBM

Die Hardwarekomponenten bilden ein Multiprozessorsystem, wobei die Prozessoren über ein Bussystem gekoppelt sind.

Die Struktur der Hardware als auch der Software ist den auszuführenden Operationen angepaßt. Die Anforderungen eines relationalen Datenbanksystems hatten in erster Linie großen Einfluß auf die Struktur der Hardware, da es ja darum geht, relationale Operationen durch Hardware zu unterstützen. Die Software reflektiert diese Struktur in dem Teil, der sich mit der Steuerung der Hardwarekomponenten befaßt. Der andere Teil könnte auch in einem konventionellen Datenbanksystem vorhanden sein (Auftragsvorbereitung, Übersetzer, s.u.) bzw. zur Betriebssoftware eines normalen Rechners gehören (Kommunikationsschicht).

Da hier insbesondere der Einsatz von PEARL in der Steuerungsschicht dargestellt werden soll, wird im folgenden nur kurz auf die Hardware und die Software außerhalb der Steuerungen eingegangen.

2.1. Hardwarestruktur der RDBM

Zum leichteren Verstandnis des Aufgabenspektrums der Steuersoftware soll hier kurz die Struktur der Hardware vorgestellt werden. Eine ausführliche Beschreibung der Hardwarekonzepte einschließlich der Spezialhardware ist in /SCHW/ enthalten.

Die nachfolgend beschriebenen Prozessoren sind eine Eigenentwicklung auf Bitslice-Basis. Dem bei dieser Arbeitsweise hohen Entwicklungsaufwand stehen als Vorteile Schnelligkeit und Flexibilitat gegenüber. So können Spezialhardwarekomponenten, die aufwendige Teile der Aufgaben des jeweiligen Prozessors unterstützen, ohne allzu große Probleme eingebunden werden (Beispiel: Quasi-Assoziative Tabellen). Als weiterer Vorteil ergibt sich, daß haufig vorkommende Befehlsfolgen direkt in das Mikroprogramm verlegt werden können.

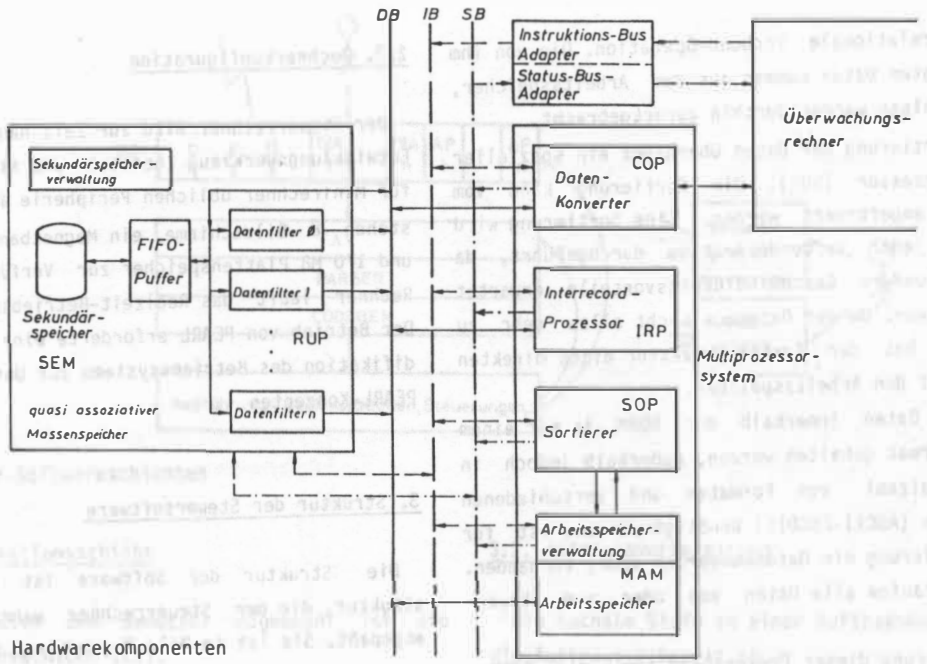

Bild 2: RDBM: Hardwarekomponenten

Bild 2 zeigt die Hardwarekomponenten der RDBM. Die hardware-zentrale Komponente ist das Bussystem bestehend aus 3 Bussen. Der Instruktionsbus (IB, Datenrate 4 MB/sec) transportiert die Instruktionen zu den einzelnen Prozessoren. Es handelt sich hierbei nicht um einzelne Prozessorbefehle, sondern dem jeweiligen Prozessor wird eine Aufgabe gegeben, die er mit Hilfe seines lokal gespeicherten Programms bearbeitet. Im allgemeinen wird ein Parametersatz übertragen.

Über den Statusbus (SB, Datenrate 1 MB/sec) werden Statusmeldungen der Prozessoren abgegeben, wie z.B. Fertig- oder Fehlermeldungen.

Auf dem Datenbus (DB, Datenrate 8 MB/sec) laufen die Daten zwischen den Prozessoren.

An die Busse angeschlossen sind 2 Speichersysteme, der Hauptspeicher (MAM, Main Memory) und der Sekundarspeicher (SEM, Secondary Memory), jeweils mit Speicherverwaltungsprozessoren (MAMM, SEMM, ...-Manager). Die übrigen Prozessoren werden von Speicherverwaltungsfunktionen freigehalten. Sie sprechen die Daten nur unter logischen Adressen an.

Hinzu kommen die eigentlichen Verarbeitungsprozessoren. Es gibt eine Anzahl von Prozessoren mit direktem Zugriff zum Sekundarspeicher, die die relationale Operation "Restriction" sowie das "Update" der Datensatze ausführen (RUP: Restriction and Update Processor). Um an dieser Stelle schnell genug zu sein, arbeiten hier mehrere Prozessoren parallel an derselben Aufgabe; die Synchronisation zum Massenspeicher hin übernimmt ein Hardware-FIFO-Puffer. Die strengen Anforderungen an die Geschwindigkeit erlauben es, auch schnellere Speicher als Magnetplatten zu betreiben. Die RUPs bilden zusammen mit dem Sekundarspeicher ein quasi-assoziatives Speichersystem (content adressable memory).

Während die RUPs also die Operationen ausführen, die sich auf Daten innerhalb eines Satzes beziehen, steht der Interrecordprocessor (IRP) für satzubergreifende Operationen zur Verfügung. Beispiele für die Operationen sind Minimum-, Maximum-, Summen- und Durchschnittsbildung sowie die wichtige und auf-

wendige relationale Verbund-Operation. Die von ihm verarbeiteten Daten kommen aus dem Arbeitsspeicher, die Ergebnisse werden dorthin zurückgebracht.

Die Sortierung der Daten übernimmt ein spezieller Sortierprozessor (SOP). Die Sortierung kann vom Benutzer angefordert werden. Eine Sortierung wird auch vor jeder Verbundoperation durchgeführt, da dann besondere Geschwindigkeitsvorteile erwartet werden können. Um den Datenbus nicht allzu sehr zu belasten, hat der Sortierprozessor einen direkten Zugriff auf den Arbeitsspeicher.

Da die Daten innerhalb der RDBM in nur einem Standardformat gehalten werden, außerhalb jedoch in einer Vielzahl von Formaten und verschiedenen Codierungen (ASCII-EBCDIC) benötigt werden, ist für die Umcodierung ein Datenkonverter (COP) vorhanden. Durch ihn laufen alle Daten von oder zum Steuerrechner.

Die Steuerung dieser Prozessoren sowie die Kommunikation mit der Außenwelt übernimmt ein Steuerrechner, wofür ein Prozeßrechner NORD-100 gewahlt wurde. Diese Aufgabe könnte von einem oder mehreren weiteren Prozessoren übernommen werden, jedoch ist es für die Soft- und Hardwareerstellung wichtig, einen "fertigen" Rechner zur Verfügung zu haben. Der Steuerrechner kann als weitere Komponente der Hardware gesehen werden. Er ist über Adapter hardwaremaßig an den Instruktions- sowie an den Statusbus angeschlossen. Der Datenaustausch erfolgt über den Datenkonverter. Auf der anderen, dem Benutzer zugewandten Seite, sind die Bildschirmgerate für lokale Benutzer, Anwendungsprogramme sowie die Anbindung von Wirtsrechnern mittels Datenfernübertragung vorgesehen (X.25).

2.2. Rechnerkonfiguration

Der Steuerrechner wird zur Zeit hauptsachlich als Entwicklungswerkzeug benutzt und ist daher mit der für Minirechner üblichen Peripherie ausgestattet. Es stehen 8 Bildschirme, ein Magnetbandgerat, Drucker und 170 MB Plattenspeicher zur Verfügung. Auf dem Rechner lauft das Realzeit-Betriebssystem SINTRAN. Der Betrieb von PEARL erforderte eine spezielle Modifikation des Betriebssystems zur Unterstützung von PEARL-Konzepten.

3. Struktur der Steuersoftware

Die Struktur der Software ist der Aufgabenstruktur, die der Steuerrechner wahrzunehmen hat, angepaßt. Sie ist in Bild 3 graphisch dargestellt.

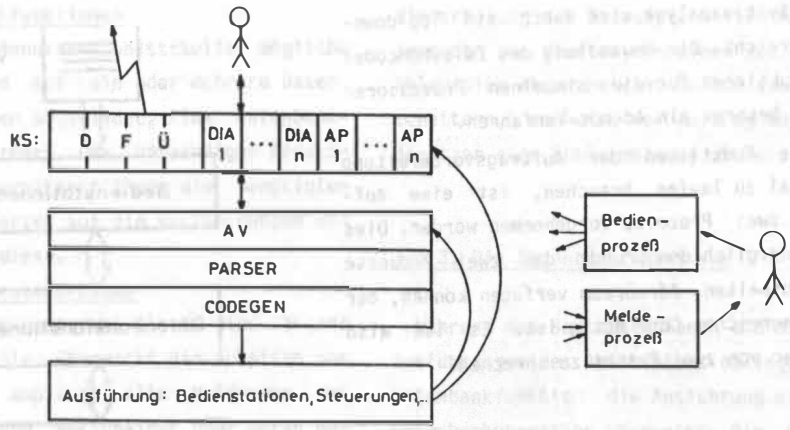

Bild 3: RDBM: Softwareschichten

3.1. Kommunikationsschicht

Am weitesten dem Benutzer zugewandt ist die Kommunikationsschicht (KS).

Sie besteht zum einen aus mehreren Dialogprozessen. Die Dialogprozesse bilden das Bindeglied zwischen lokalem Benutzer und den anderen RDBM-Teilen. Im Dialog können sowohl die Aufgaben des Datenbankadministrators als auch Ad-hoc-Anfragen der Benutzer bearbeitet werden. Für jeden Dialog steht ein Prozeß zur Verfügung.

Die andere Form der Kommunikation betrifft den Rechnerverbund. Die DFÜ-Schnittstelle erlaubt, von raumlich weit entfernten Rechnern aus auf die RDBM zuzugreifen. Die Komplexitat dieser Schnittstelle wird veranschaulicht durch die Tatsache, daß alleine die DFÜ-Schnittstelle aus 4 Prozessen besteht.

Schließlich können auch Anwendungsprogramme über eine definierte Schnittstelle mit einer Datenbank arbeiten. Ihnen steht derselbe Befehlsvorrat zur Verfügung, wie ihn auch die DFÜ-Schnittstelle bietet.

3.2. Auftragsvorbereitung

Die nachste Stufe in einer Auftragsbearbeitung ist die Auftragsvorbereitung.

In der Auftragsvorbereitung werden alle permanenten Verwaltungs- und Schemadaten gehalten. Der Teil der Benutzerkommandos, die sich nur auf diese Verwaltungsdaten beziehen, kann sofort bearbeitet werden. Es handelt sich hierbei um Kommandos wie: LOGIN, LOGOUT, Schemaanderungen, Schemaabfragen, etc. Andere Kommandos, die die Nutzdaten ansprechen, durchlaufen den Parser und die Codegenerierung. Nach dieser Übersetzung stehen sie zur Ausführung bereit und werden in die Warteschlange vor den Bedienstationen eingereiht.

3.2.1. Übersetzer

Als Datenbanksprache wird eine Variante von SQL benutzt. Um diese Hochsprache in Datenbankoperationen zu überführen, wird ein Übersetzer eingesetzt. Wie beim Übersetzer üblich, ist er in die Teile Syntaxanalyse (einschließlich lexikalischer Analyse) und Codegenerierung geteilt. Der Symboltabelle entspricht in diesem Fall das Datenbank-

schema. Die Syntaxanalyse wird durch ein Top-down-Verfahren erreicht. Die Umwandlung des Zwischencodes in die Instruktionen für die einzelnen Prozessoren der Hardware besorgt ein Ad-hoc-Verfahren.

Obwohl die Funktionen der Auftragsvorbereitung nicht parallel zu laufen brauchen, ist eine Aufteilung in zwei Prozesse vorgenommen worden. Dies hat jedoch lediglich den Grund, daß zwei Prozesse über den doppelten Adreßraum verfügen können, der für den umfangreichen Code nötig ist. Es ist also sinnvoll, eher von zwei Passes zu sprechen.

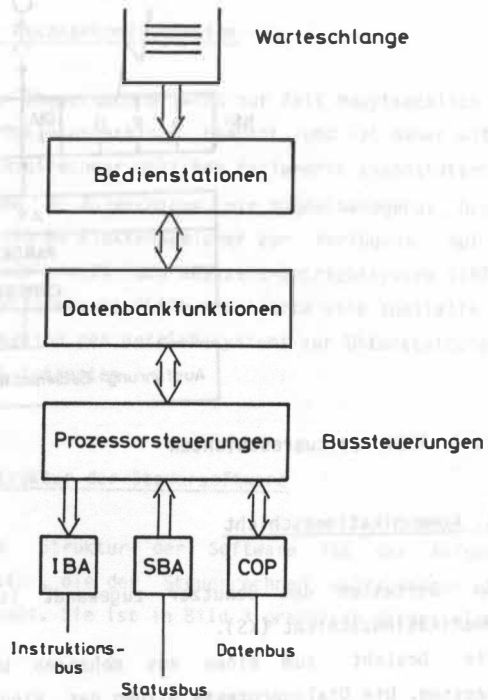

Bild 4: Struktur der Ausführungsschicht

3.3. Ausführung

Hat ein Benutzerauftrag den Parser und die Codegenerierung fehlerfrei durchlaufen, so wird er an die Ausführungsschicht übergeben. Die Aufgabe der Ausführung ist es, den Benutzerauftrag auf den Prozessoren abarbeiten zu lassen. Die Ausführungsschicht verwaltet die übersetzten Aufträge und überwacht deren Ausführung, während die Ausführung selbst in den Prozessoren vollzogen wird.

Ein Auftrag entspricht einer Eingabe eines Benutzers. Er wird in mehrere Teilaufträge zerlegt. Ein solcher Teilauftrag wird als eine Datenbankfunktion bezeichnet. Diese Bezeichnung ergibt sich daraus, daß eine Datenbankfunktion in etwa den Umfang einer relationalen Datenbankoperation hat.

Ein Auftrag aus ein oder mehreren Datenbankfunktionen wird als ein Ablaufplan bezeichnet.

Die Ausführungsschicht kann folgendermaßen gegliedert werden (s. Bild 4):

- Die Auftragswarteschlange
  Jeder von der Auftragsvorbereitung erzeugte Ablaufplan wird durch Einketten in diese Warteschlange an die Ausführung übergeben.
- Die Bedienstationen
  Zur Überwachung einer Ablaufplanausführung existieren mehrere Prozesse. Diese werden als Bedienstationen bezeichnet. Mehrere Bedienstationen ermöglichen die parallele Ausführung von Ablaufplanen.

- Die Datenbankfunktionen
  Jede in der Benutzerschnittstelle mögliche Anfrage wird auf ein oder mehrere Datenbankfunktionen abgebildet. Eine Datenbankfunktion fordert die notwendigen Prozessoren an, übermittelt ihnen die benötigten Parameter, wartet auf die Rückmeldungen und verarbeitet diese.
- Die Prozessorsteuerungen
  Für jeden Prozessor existiert eine eigene Steuerung. Sie überwacht die Arbeiten des Prozessors, empfängt alle Meldungen des Prozessors und entscheidet über deren Behandlung.

3.3.1. Die Auftragswarteschlange

Die mögliche Parallelitat von Ablaufplanen ist durch die Anzahl der Bedienstationen begrenzt. Für weitere Ablaufpläne ist eine Warteschlange vorgesehen, in der die Ablaufpläne in der Reihenfolge ihrer Ankunft verkettet und nach der FIFO-Strategie weitergeleitet werden.

3.3.2. Die Bedienstationen

Eine Bedienstation ist ein Prozeß, der die Ausführung eines Ablaufplans überwacht. Es sind drei Bedienstationen vorhanden, wodurch die parallele Ausführung von drei Ablaufplänen ermöglicht wird.

Jede Bedienstation bearbeitet einen Ablaufplan zur Zeit. Erst nach der vollstandigen Ausführung wird ein neuer Ablaufplan aus der Warteschlange abgerufen.

Die einzelnen Ablaufplanbefehle - die Datenbankfunktionen - werden sequentiell abgearbeitet.

Wird in einem Prozessor ein Verarbeitungsfehler erkannt, so hat die Bedienstation die Möglichkeit, den Ablaufplan sofort abzubrechen oder - durch die Wiederholung des Ablaufplans von Anfang an - den Versuch zu starten, in einem zweiten Durchgang den Ablaufplan abzuarbeiten. Sollte auch dieser Versuch scheitern, so wird der Auftrag abgebrochen und dem Benutzer eine Fehlermeldung übermittelt.

3.3.3. Die Datenbankfunktionen

Wahrend den Bedienstationen die Kontrolle über die Ausführung eines Ablaufplans obliegt, wird von einer Datenbankfunktion die Ausführung einer relationalen Datenbankoperation überwacht. Die Datenbankfunktion übergibt den jeweiligen Prozessorsteuerungen den für diesen Auftrag spezifischen Parametersatz und wartet auf die Antwort der Prozessorsteuerungen. Die Datenbankfunktion besitzt die Fahigkeit, mehreren Prozessorsteuerungen Parameter zu übergeben, ehe die Antworten entgegengenommen werden, so daß die Prozessoren parallel die ihnen ubertragenen Aufgaben ausführen können.

Die Datenbankfunktion erhält für jeden Auftrag, den sie an einen Prozessor geschickt hat, eine Meldung, ob der Auftrag korrekt bearbeitet wurde.

Neben der Parallelitat mehrerer Benutzerauftrage kommt hier die Parallelitat innerhalb eines Benutzerauftrags bzw. innerhalb einer Datenbankfunktion hinzu.

Folgendes Beispiel dazu sei hier erwahnt:

Die Anfrage lautet:

```
SELECT MIN(GEHALT)
  FROM PERSONAL
```

Die Anfrage soll das niedrigste Gehalt liefern, das von einer Person empfangen wird.

Bei dieser Anfrage sind der SEMM, die RUPs, der MAMM und der IRP beteiligt (siehe Kapitel 2.1.).

Der SEMM erhalt den Auftrag, alle Satze der Relation PERSONAL an die RUPs zu liefern.

Die RUPs erhalten den Auftrag, eine Projektion auf

das Attribut GEHALT vorzunehmen.

Der MAMM erhält den Auftrag, Platz für zwei temporare Relationen (Ergebnis der RUPs und Ergebnis des IRP) bereitzustellen.

Der IRP erhält den Auftrag, das Minimum zu bestimmen.

Bei dieser Anfrage wird jeder Satz der Relation PERSONAL - auf ein Attribut verkürzt - an den MAMM übertragen. Der IRP fordert jeden dieser Sätze beim MAMM an, um das Minimum bestimmen zu können.

Hier besteht nun die Möglichkeit, durch Parallelität innerhalb eines Benutzerauftrags die Ausführung zu beschleunigen: Während die RUPs die Projektion der Sätze durchführen, kann der IRP die im MAMM bereits angekommenen Sätze auf das Minimum hin überprüfen.

Trotz der einfachen Anfrage kann hier bereits ein hoher Parallelitätsgrad erzielt werden.

Die Datenbankfunktionen sind als ein Paket von Unterprogrammen realisiert, die von den Bedienstationen aufgerufen werden können. Eine Datenbankfunktion kann auch von mehreren Bedienstationen gleichzeitig aufgerufen werden (reentrant). Die Kontrolle über die Belegung der Prozessoren obliegt den Prozessorsteuerungen.

Der Grundbestand an Datenbankfunktionen soll im folgenden aufgeführt werden. Durch Zusammensetzen mehrerer Datenbankfunktionen des Grundbestandes können beliebig komplexe Datenbankfunktionen gebildet werden, die eventuell die Möglichkeiten der Parallelität besser ausnutzen (siehe vorhergehendes Beispiel):

- Suche Einzelsatz
- Suche Menge
- Ändere Einzelsatz
- Ändere Menge
- Einfügen mit Primärschlüsselüberprüfung
- Einfügen ohne Primärschlüsselüberprüfung
- Ausgabe an den Benutzer
- IRP-Auftrag
- Sortier-Auftrag
- Starte Transaktion
- Beende Transaktion
- Abbruch Transaktion
- Löschen von Relationen
- Lesesperre anfordern

Die zuletzt genannte Datenbankfunktion fällt hier aus dem Rahmen. Sie wird bereits im Steuerrechner ausgeführt, da diese Funktion hardwaremäßig nicht unterstützt wird.

Dieser Bestand an Datenbankfunktionen reicht bereits aus, alle in der Benutzerschnittstelle möglichen Anfragen beantworten zu können.

3.3.4. Die Prozessorsteuerungen

Jeder Prozessor der RDBM wird durch eine eigene Steuerung kontrolliert. Diese Steuerung überwacht und organisiert die Belegung und das Arbeiten des Prozessors. Aufgaben für eine Prozessorsteuerung kommen einerseits von einer Datenbankfunktion, andererseits als Rück- bzw. Statusmeldung vom Prozessor. Eine Steuerung besteht deshalb aus zwei Teilen:

- Auftragsannahme
- Meldungsverarbeitung

Die Auftragsannahme ist als ein Unterprogramm realisiert, das von den Datenbankfunktionen aufgerufen werden kann.

Falls der Prozessor bereit ist, wird der empfangene Auftrag in Verwaltungstabellen der Prozessorsteuerung vermerkt und dem Prozessor zugestellt. Damit ist die Aufgabe der Auftragsannahme beendet. Der Datenbankfunktion wird mitgeteilt, ob der Auftrag dem Prozessor übergeben wurde.

Bei exklusiv belegbaren Prozessoren wird mit einem Semaphor gewährleistet, daß nur ein Auftrag zur Zeit bearbeitet wird.

Die Meldungsverarbeitung ist ein eigener Prozeß und hat Zugriff auf die Verwaltungstabellen der Auf-

tragsannahme. Jede vom Prozessor eintreffende Rückmeldung wird hier entgegengenommen und verarbeitet. Anschließend wird der jeweiligen Bedienstation übermittelt, ob der Auftrag korrekt bearbeitet wurde.

Ein wichtiger Teil der Meldungsverarbeitung ist hierbei die Fehlerbehandlung. In einem Fehlerfall hat sie zu entscheiden, was mit dem Prozessor geschieht. Der jeweiligen Bedienstation muß sie eine Fehlermeldung schicken, in der ein Vorschlag für die weitere Bearbeitung des Auftrags gemacht wird.

Muß der Prozessor nach einem Fehler neu initialisiert werden, so wird für jeden Auftrag, an dem er gerade arbeitete, eine Fehlermeldung an die jeweilige Bedienstation geschickt. Solange der Prozessor nicht wieder bereit ist, wird von der Auftragsannahme kein neuer Auftrag akzeptiert.

3.3.5. Beispiel einer Auftragsausführung

In den vorangehenden Kapiteln wurden die einzelnen Komponenten der Ausführung vorgestellt. Zur Veranschaulichung soll an dieser Stelle anhand eines Beispiels die Ausführung eines Auftrags dargestellt werden. Aus Vereinfachungsgründen wird ein Auftrag gewählt, der aus einer Datenbankfunktion mit lediglich einem Prozessoraufruf besteht.

Eine Situation, in der dies ein realistischer Fall ist, ist der Auftrag "Beende Transaktion". Der Auftrag entspricht der gleichnamigen Datenbankfunktion, die lediglich den SEMM anspricht.

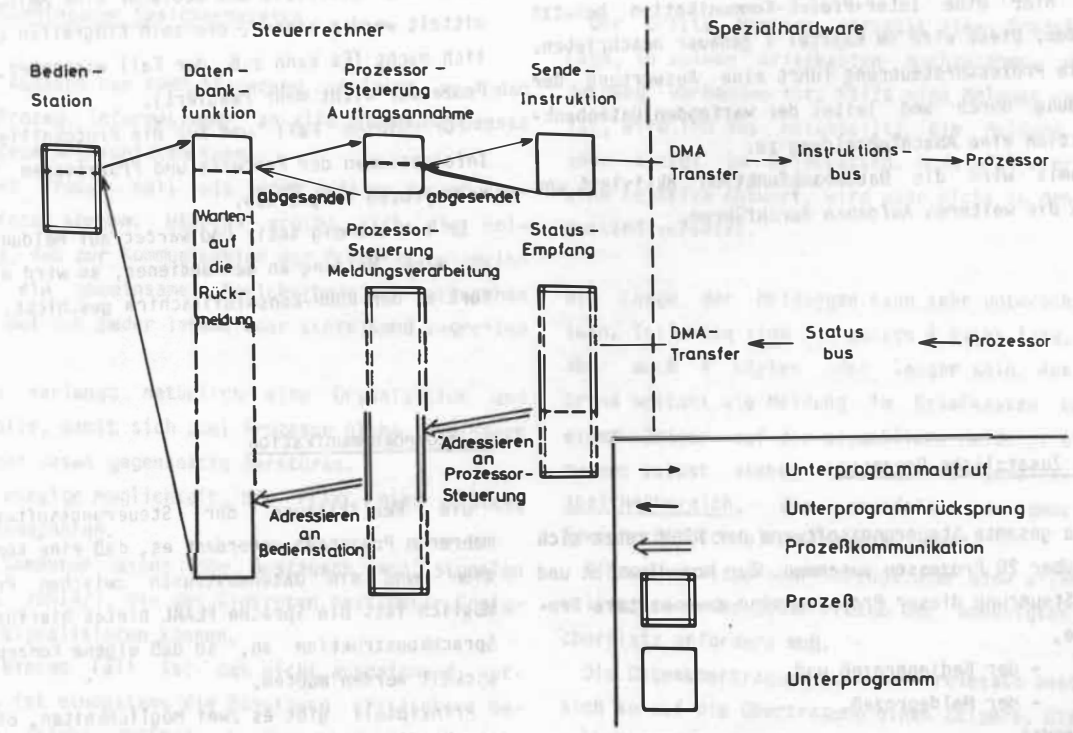

Bild 5: Die Ausführung eines Auftrags

Nachdem die Bedienstation den Auftrag nach erfolgter Übersetzung in Form eines Ablaufplans übernommen hat, erfolgt der Aufruf der entsprechenden Datenbankfunktion. Die Datenbankfunktion leitet den Auftrag an die Auftragsannahme der jeweiligen Prozessorsteuerung weiter. Die Auftragsannahme prüft, daß der Prozessor bereit ist, notiert den Auftrag und schickt ihn über den Instruktionsbus an den Prozessor.

Das Weiterleiten des Auftrags erfolgte durch Unterprogrammaufrufe. Nach der Rückkehr aus dem Unterprogramm der Prozessorsteuerung wartet nun die Datenbankfunktion auf die Antwort des Prozessors.

Der Prozessor sendet eine Meldung über den Statusbus. Diese wird von einem Prozeß entgegengenommen, der nur dazu da ist, die Statusmeldungen der Prozessoren anzunehmen und an die entsprechenden Prozessorsteuerungen weiterzureichen.

Da die Prozessorsteuerung ein eigener Prozeß ist, muß hier eine Inter-Prozeß-Kommunikation benutzt werden. Diese wird im Kapitel 4 genauer beschrieben.

Die Prozessorsteuerung führt eine Auswertung der Meldung durch und leitet der wartenden Datenbankfunktion eine Abschlußmeldung zu.

Damit wird die Datenbankfunktion aktiviert und kann die weiteren Aufgaben durchführen.

3.4. Zusatzliche Prozesse

Die gesamte Steuerungssoftware der RDBM setzt sich aus über 20 Prozessen zusammen. Zur Koordination und zur Steuerung dieser Prozesse sind zwei weitere Prozesse,

- der Bedienprozeß und
- der Meldeprozeß,

vorhanden.

3.4.1. Bedienprozeß

Der Bedienprozeß dient zum Steuern der gesamten RDBM (einschließlich Hardware). Er ermöglicht das Starten und Beenden der RDBM und stellt zusatzliche Funktionen bereit, so daß Tests und Informationsausgaben der Prozesse und Prozessoren möglich sind.

Für das Arbeiten der RDBM ist es nicht erforderlich, daß der Bedienprozeß aktiv ist. Solange keine Funktionen des Bedienprozesses benutzt werden sollen, kann der Prozeß passiv sein.

3.4.2. Meldeprozeß

Wie gerade erwahnt wurde, muß der Bedienprozeß beim Lauf der RDBM nicht aktiv sein. Andererseits aber muß jederzeit dem Bediener eine Meldung übermittelt werden können, die sein Eingreifen erforderlich macht (Es kann z.B. der Fall eintreten, daß ein Prozessor nicht mehr reagiert).

Für diesen Fall und für die Protokollierung von Informationen der Prozesse und Prozessoren ist der Meldeprozeß vorgesehen.

Er ist standig aktiv und wartet auf Meldungen. Ist es eine Meldung an den Bediener, so wird diese sofort an den RDBM-Konsolbildschirm geschickt.

4. Prozeßkommunikation

Die Realisierung der Steuerungssoftware in mehreren Prozessen erfordert es, daß eine Kommunikation und ein Datenaustausch zwischen Prozessen möglich ist. Die Sprache PEARL bietet hierfür keine Sprachkonstruktion an, so daß eigene Konzepte entwickelt werden mußten.

Prinzipiell gibt es zwei Möglichkeiten, einen Informationsaustausch durchzuführen:

- Die kommunizierenden Prozesse müssen einen gemeinsamen Speicherbereich haben. Ein Prozeß generiert eine Information und speichert sie in diesem Speicherbereich ab. Ein zweiter Prozeß greift lesend auf diesen Bereich zu.
  Die Adreßräume der beiden Prozesse müssen sich somit überschneiden.
- Das Betriebssystem bzw. die Prozeßverwaltung muß Mechanismen anbieten, die eine Übermittlung von Informationen durchführen können.

Beide Methoden der Prozeßkommunikation werden in der Steuerung der RDBM verwandt.

4.1. Gemeinsamer Speicherbereich

Das Konzept der RDBM-Steuerung verlangt es, daß ein Prozeß Informationen an alle anderen Prozesse der Steuerung schicken kann.

Jeder Prozeß soll mit jedem anderen Prozeß kommunizieren können. Hieraus ergibt sich die Folgerung, daß zur Kommunikation der Prozesse untereinander ein gemeinsamer Speicherbereich vorzusehen ist, auf den jeder lesend oder schreibend zugreifen kann.

Dies verlangt natürlich eine Organisation und Kontrolle, damit sich zwei Prozesse nicht behindern oder gar Daten gegenseitig zerstören.

Die einzige Möglichkeit, die PEARL hier bietet, sind Semaphoren.

Ein Semaphor dient dem Austausch von Signalen (siehe /BRIN/), die das Eintreten bestimmter Ereignisse signalisieren können.

In diesem Fall ist das nicht ausreichend. Notwendig ist mindestens die Benutzung kritischer Bereiche (siehe /BRIN/), besser aber die Bereitstellung von Monitoren für das Lesen wie auch für das Schreiben von Meldungen.

In der RDBM-Steuerung werden jedem Prozeß drei Monitore zur Prozeßkommunikation bereitgestellt:

- Lesen einer Meldung
- Schreiben einer Meldung
- Prüfen, ob eine Meldung vorhanden ist.

Jeder Prozeß besitzt einen "Briefkasten", in dem die für ihn eingetroffenen Meldungen abgelegt sind.

Der Empfangsprozeß entscheidet, wann er eine Meldung lesen möchte. Durch den Aufruf des Lesemonitors erhält er die nächste Meldung aus seinem Briefkasten.

Sollte keine Meldung vorhanden sein, so wird der Prozeß in einen Wartezustand versetzt, bis die nachste Meldung für ihn eingetroffen ist.

Durch den Aufruf des zweiten Monitors kann ein Prozeß jedem beliebigen Prozeß eine Meldung in dessen Briefkasten legen.

Der dritte Monitor versetzt einen Prozeß in die Lage, in seinem Briefkasten nachzusehen, ob eine Meldung vorhanden ist. Falls eine Meldung vorhanden ist, wird ihm das mitgeteilt; die Meldung selbst aber bleibt im Briefkasten. Andernfalls erhalt er eine negative Antwort, wird aber nicht in den Wartezustand versetzt.

Die Lange der Meldungen kann sehr unterschiedlich sein. Teilweise sind sie gerade 4 Bytes lang, konnen aber auch 4 KBytes oder langer sein. Aus diesem Grund besteht die Meldung im Briefkasten nur aus einem Zeiger auf die eigentliche Meldung. Die Meldungen selbst stehen in einem allgemeinen Freispeicherbereich, der ebenfalls im gemeinsamen Speicherbereich aller Prozesse liegt.

Dieser Freispeicher verfügt über eine eigene Verwaltung, bei der jeder Prozeß den benötigten Speicherplatz anfordern muß.

Die Datenübertragung zwischen Prozessen beschrankt sich so auf die Übertragung eines Zeigers. Die Daten selbst werden einmal im Freispeicher abgelegt und brauchen beim Weiterreichen an einen weiteren Prozeß nicht kopiert zu werden.

Die prozeßübergeordnete Freispeicherverwaltung erlaubt es, die Anforderung und die Freigabe eines Speicherbereichs von verschiedenen Prozessen aus durchzuführen.

4.2. Systemroutinen

Zur Prozeßkommunikation zwischen Prozessen, deren Adreßraum sich nicht überlappen soll, ist die Unterstutzung des Betriebssystems erforderlich. Es müssen Mechanismen angeboten werden, die es erlauben, analog dem soeben erlauterten "Briefkasten" Informationen von einem Prozeß einem anderen Prozeß zu übermitteln.

Das Betriebssystem des Steuerrechners stellt derartige Mechanismen bereit.

Auf Betriebssystemebene werden mehrere Ringpuffer (internal devices) verwaltet, die von Prozessen zum Lesen oder Schreiben reserviert werden können.

Der Vorteil dieser Ringpuffer liegt in der absoluten Trennung der Adreßraume. Alle Anwendungsprogramme, die Datenbankoperationen aufrufen und auf dem Steuerrechner ablaufen (siehe Kap. 3.1.), müssen über Ringpuffer mit dem Datenbanksystem kommunizieren. Erstens soll Anwendungsprogrammen der gesamte Adreßraum zur Verfügung stehen, und zweitens muß es ausgeschlossen werden, daß ein Benutzer die Benutzerüberprüfung umgehen oder Teile des Datenbanksystems selbst verändern oder zerstören könnte.

Der Nachteil der Ringpuffer ist, daß durch die byteweise Übertragung viel Dverhead entsteht.

4.3. Realisierung der Prozeßkommunikation

Die Beschreibung der Kommunikationsmechanismen zeigte die Vor- und die Nachteile der jeweiligen Verfahren auf.

Der gemeinsame Speicherbereich reduziert zwar den Adreßraum eines Prozesses, bietet dafür aber eine schnelle Kommunikation. Die Möglichkeit der Teilung eines Prozesses in mehrere Prozesse wiegt den Nachteil des gemeinsamen Speicherbereichs wieder auf.

Die Kommunikation aller internen Prozesse untereinander verlauft über den gemeinsamen Speicherbereich.

Der Datenaustausch mit den Prozessen der Kommunikationsschicht (Datenfernübertragung, Programmschnittstelle und lokale Bildschirme) wird zum Schutz vor Benutzermißbrauchen über Ringpuffer abgewickelt.

Schließlich werden alle Meldungen an den Meldeprozeß ebenfalls über einen Ringpuffer geleitet. Die Gründe hierfür liegen darin, den Meldeprozeß von den anderen Prozessen völlig abzukoppeln, so daß Fehler wahrend der Testphase nicht die Protokollierung selbst beeintrachtigen können.

5. Erfahrungen mit PEARL

Obwohl hier in erster Linie die Eigenschaften von PEARL im Vordergrund stehen, muß der Einfluß der Sprachimplementierung gesehen werden; denn Fehler und Ineffizienzen können die tagliche Arbeit enorm behindern. Inwieweit diese Schwierigkeit allein von der Implementierung oder auch vom Sprachkonzept herrühren, kann nur durch weiterführende Untersuchungen entschieden werden.

Im folgenden sollen zunachst die Erfahrungen mit dem vorhandenen PEARL-System und dann mit dem PEARL-Sprachkonzept beschrieben werden.

5.1. Erfahrungen mit dem vorhandenen Compiler

An dieser Stelle müssen zunachst einige Erlauterungen zur Situation im Projekt gemacht werden.

Zu Projektbeginn wurde beschlossen, daß der Aufbau der Steuerung in der Programmiersprache PEARL erfolgen sollte. Unter diesem Gesichtspunkt erfolgte die Rechnerauswahl. Trotz Zusicherung wurde letzt-

endlich die Wartung des PEARL-Compilers vom Rechnerhersteller nicht übernommen. So entstand die Situation, eine Programmiersprache zu verwenden, die vom Rechnerhersteller nicht unterstützt wird. Dies führte zu vielen zeitaufwendigen Problemen, die u.a. selbst ausgeführte Änderungen im Betriebssystem erforderten.

Problematisch wurde auch die Wartung des Compilers. Das RDBM-Projekt stellt die einzige Anwendung eines PEARL-Compilers auf einem NORD-Rechner dar. Damit liegt beim Compiler-Entwickler kein großes Interesse an einer fehlerfreien Implementierung vor. Trotz der ersten Auslieferung vor fast 5 Jahren tauchen immer noch Fehler auf. Leider ist es auch so, daß die Reaktion auf Fehlermeldungen sehr schleppend verläuft. Auf einige Fehler, die u.a. das erste Mal bereits vor über 2 Jahren gemeldet wurden, erfolgte bisher keine Erklärung, erst recht aber keine Beseitigung. Daß einer dieser Fehler einen Absturz des Compilers herbeiführt, muß einfach erwähnt werden.

Die einzige Möglichkeit, in PEARL auf Fehler bei Ein-/Ausgabeoperationen zu reagieren, bietet die SIGNAL-Konstruktion. Der benutzte Compiler ist aber nicht imstande, einen solchen Code zu erzeugen, daß nach dem Durchlauf einer ON-Anweisung das Programm fortgesetzt wird, was einem Unterprogrammverhalten entsprechen würde. Stattdessen muß eine ON-Routine mit einem GOTO-Befehl verlassen werden, was das SIGNAL-Konzept für die Mehrzahl der aufzufangenden Fehler unbrauchbar macht, da man nicht weiß, wo das Programm unterbrochen wurde.

Zur Optimierung eines Programms ist es notwendig, oft durchlaufene Programmteile zu lokalisieren. Hierzu liefert i.a. das ablaufende Programm Statistikinformationen, die man gebrauchlicherweise per Compiler-Option anschaltet. Der vorhandene PEARL-Compiler bietet diese für die Optimierung der Steuerrechner-Software wichtige Möglichkeit nicht.

Die Deklarationen von Ein/Ausgabekanalen (DATIONs) unterliegen starken Einschrankungen, die zum einen nicht schriftlich dokumentiert und zum anderen auch vom Compiler nicht überprüft werden. Für das Laden spielt es eine große Rolle, in welchem Modul die DATION-Deklaration integriert wird. Werden Prozesse auf verschiedene Speicherbereiche geladen, die über einen gemeinsamen Bereich kommunizieren, so kann dies zu einem inkorrekten Verhalten der Prozesse bis hin zum Absturz führen. Das Laden verlangt genaue Kenntnis über den Zweck und den Inhalt der vom Compiler erzeugten Codedateien. Hierzu müßte es eine ausführliche Dokumentation geben.

Für die Bewertung einer Compiler-Implementierung haben Laufzeiten des erzeugten Codes eine große Bedeutung. Speziell bei einer Prozeßsteuerungssprache spielt die erreichbare Laufzeit eine wichtige Rolle.

Die Übersetzungs- und Laufzeiten einiger Beispiele wurden für FORTRAN-77 und PEARL untersucht:

1) Matrixmultiplikation
   Es wurden in einer Schleife zwei 50x50-Matrizen miteinander multipliziert.

   FORTRAN-77 (CPU-Zeit):

Obersetzungszeit:	1,2 sec
Laufzeit:	2,5 sec

   PEARL (CPU-Zeit):

Obersetzungszeit:	44 sec
Laufzeit:	4,9 sec
	(ohne Index-Check)

2) Ackermann-Funktion (3,6)

   FORTRAN-77:

Obersetzungszeit:	1,0 sec
Laufzeit:	10,1 sec

PEARL:

Obersetzungszeit: 20 sec
Laufzeit: 55 sec

Hierbei werden 172233 Prozeduraufrufe durchgeführt. Während FORTRAN pro Aufruf 58 µs benötigt, werden vom PEARL-Programm 319 µs pro Aufruf gebraucht.

3) Ein/Ausgabe auf Datei
Es wurden 10000 Integer/FIXED-Werte auf eine Datei geschrieben bzw. von einer Datei gelesen.

Lesen : FORTRAN-77 : 38 sec
PEARL (ALPHIC DIM(*,80)) : 154 sec

Schreiben : FORTRAN-77 : 38 sec
PEARL: 198 sec

4) Aufruf einer leeren Prozedur

keine Parameter	FORTRAN-77 :	38 µs
	PEARL :	123 µs
ein Parameter	FORTRAN-77 :	43 µs
	PEARL :	166 µs
fünf Parameter	FORTRAN-77 :	52 µs
	PEARL :	336 µs

Wie in PEARL erfolgt auch in FORTRAN-77 eine Stackanforderung, so daß FORTRAN-77 diesbezüglich keinen Vorteil gegenüber PEARL hat.

Diese Zeiten sagen von sich aus genug aus. Wo die Gründe für derartig krasse Unterschiede liegen, kann nur vermutet werden: Der PEARL-Compiler benutzt eine Zwischensprache, aus der dann der maschinenabhängige Code erzeugt wird. Womöglich ist diese Zwischensprache schlecht für NORD-Rechner geeignet. Weitere Gründe können in einer ineffizienten Realisierung des PEARL-Laufzeitsystems liegen. Daß solche Laufzeitunterschiede in einer Realzeitsprache eigentlich nicht zu tolerieren sind, sollte ziemlich klar sein.

Die Probleme mit den DATIONs und den SIGNALs führten im Projekt dazu, daß eigene Ein/Ausgaberoutinen in Assembler geschrieben wurden. Ein/Ausgaben, die mit PEARL 1-2 Minuten benötigten, sind jetzt in wenigen Sekunden erledigt.

5.2. Erfahrungen mit dem Sprachkonzept

Die Vor- und Nachteile von PEARL als Programmiersprache können nur im Vergleich zu anderen Programmiersprachen gesehen werden.

Für eine Realzeitsprache ist die Existenz von Semaphoren zur Prozeßsynchronisation notwendig. Die Benutzung von Semaphoren ist gut gelöst.

Die Verwendung von Datenstrukturen und Zeigern erlaubt eine übersichtliche und verstandliche Gestaltung von Programmen.

Doch folgende Punkte im Sprachkonzept sollten einige Oberlegungen wert sein:

- Eine Reaktion auf Ein/Ausgabefehler ist nur über die SIGNAL-Konstruktion möglich. Das individuelle Abfangen und Behandeln eines Fehlers ist sehr umständlich oder auch garnicht möglich. Wird z.B. End-of-File gemeldet, so muß dies an einem Punkt durchaus als Fehler behandelt werden, während es an einem anderen Punkt sogar erwartet wird. In diesem Fall ist eine Information darüber notwendig, bei welchem Programmschritt der Fehler erkannt wurde.
- Soll ein Programm als ein Realzeitprogramm ablaufen, so muß zur Obersetzungszeit die Bildschirmnummer fest vorgegeben werden. Ein Programm, das abwechselnd an verschie-

denen Bildschirmen laufen soll, muß für jeden möglichen Bildschirm eine DATION-Deklaration enthalten. Bei Kenntnis über die Intern-Struktur des Codes kann diese Platzverschwendung vermieden werden, führt aber zu nicht portablen Programmen.

- In einem PUT- und GET-Befehl muß bereits zur Übersetzungszeit festgelegt sein, wieviele Bytes ausgegeben bzw. eingelesen werden sollen. Dies führt zu einem enormen Verwaltungsaufwand beim Ausgeben bzw. Einlesen variabel langer Daten.
- Obwohl PEARL eine Prozeß- und Realzeitsprache ist, verfügt sie über keine Konstruktionen zur Prozeßkommunikation.
- Eine einfache Kommunikation ware schon dadurch möglich, daß einem Prozeß beim Aktivieren bzw. Fortsetzen ein oder mehrere Parameter übergeben werden könnten.
- PEARL bietet keine Möglichkeit, auf verschiedene Ereignisse zu warten und zu reagieren. Jede Ereignisquelle erfordert einen eigenen Prozeß. Dies führt schnell zu einer großen Zahl von Prozessen. Jeder neue Prozeß erhöht die Anzahl der Prozeßwechsel im Betriebssystem und verlangt zusätzliche Prozeßkommunikationen.

5.3. Fazit

Zusammenfassend kann das Konzept von PEARL als gut angesehen werden, wobei aber eine volle Unterstützung vom Rechnerhersteller erforderlich ist.

Das SIGNAL-Konzept sollte im ganzen überdacht werden. Die Möglichkeit der synchronen Fehlerbehandlung ist für eine Programmiersprache wünschenswert.

Eine Prozeß- und Realzeitsprache sollte über bessere Konstruktionen zur Prozeßkommunikation verfügen.

Die Laufzeiten des erzeugten Codes ist für eine Realzeitsprache nicht akzeptabel.

PEARL zu verwenden, muß für das Projekt als eine Fehlentscheidung betrachtet werden.

6. Literatur

/BRIN/ P.Brinch Hansen,
Operating System Principles,
Prentice Hall 1973

/AUER/ H. Auer, W. Hell, et. al.,
RDBM - A Relational Data Base Machine,
Inform. Systems Vol. 6, No. 2, 1981

/SCHW/ H. Schweppe, H. Ch. Zeidler, et. al.,
RDBM - A Dedicated Multiprocessor System for Database Managment,
in
D. K. Hsiao,
Advanced Database Machine Architecture,
Prentice Hall 1983

---

von Kleist-Retzow, Hans
Hildebrandt, Frank

TU Braunschweig
Institut für Datenverarbeitungsanlagen
Institut für Informatik
Abteilung Betriebssysteme und Rechnerverbund
Postfach 3329
3300 Braunschweig
Telefon: 0531/391-3292