KfK 3229
Oktober 1981

DISCO: Entwicklung und Implementierung von Funktionen zur Verwaltung verteilter Datenbasen in Rechnernetzen

H. Breitwieser, O. Drobnik, E. Holler, U. Kersten,
R. Mailänder, W. Seemann
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 3229

DISCO: Entwicklung und Implementierung von Funktionen zur Verwaltung verteilter Datenbasen in Rechnernetzen*)

H. Breitwieser, O. Drobnik, E. Holler, U. Kersten
R. Mailänder, W. Seemann

*) Ergebnisbericht des mit Mitteln des Bundesministeriums für Forschung und Technologie (FKZ 081 5610) und der SIEMENS AKTIENGESELLSCHAFT geförderten Vorhabens "Entwicklung und Implementierung von Funktionen zur Verwaltung verteilter Datenbasen"

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Kurzfassung

Der Bericht beschreibt die Realisierung einer auf der logischen Dateiebene einzuordnenden Schnittstelle für verteilte Datenbasen in Kleinrechnernetzen mit einer Mehrbenutzerumgebung. Die Schnittstelle bietet aus Benutzersicht Transparenz bezüglich der Verteilung von Daten und Funktionen. Sie unterstützt die Behandlung von Ausfall/Wiederanlauf von Daten/Funktionseinheiten. Die Funktionseinheiten zur Verwaltung einer verteilten Datenbasis und ihre Interaktionen sind im Detail aufgezeigt. Eine Pilotimplementierung eines solchen Systems wird beschrieben.

DISCO: Development and implementation of functions for the management of distributed data bases in computer networks.

Abstract

This report describes the realization of a logical file interface for distributed data bases in small-computer networks with a multi-user environment. The interface offers transparency to the user with respect to the distribution of data and functions. It supports procedures to handle cases of failure/restart of data/functional units. The functional units of the distributed data base management system and their interactions are presented in detail. A prototype implementation of such a system is described.

Inhalt Seite

1. Einführung

Eine der wichtigsten Aufgaben eines Datenverarbeitungssystems ist die Verwaltung von Datenbeständen, d.h. die Abwicklung von Zugriffen auf gespeicherte Daten und die Organisation der Datenspeicherung.

Integrierte Datenhaltung /Loc78/ charakterisiert ein Datenverwaltungssystem, das sich so verhält, als ob

- diese Datenbestände nur an einer Stelle und nur einmal geführt würden
- eine einzige Instanz für Verwaltung, Veränderung, Zugriffsüberwachung und Sicherung verantwortlich sei
- jedem Anwender die Daten ausschließlich, d.h. unabhängig von gegebenenfalls gleichzeitig durchgeführten Zugriffen anderer Anwender, zur Verfügung stünden.

Die Definition des Begriffs "Integrierte Datenhaltung" beinhaltet keinerlei Aussage über die Struktur des für die Datenhaltung verwendeten DV-Systems.

Die konventionelle Vorgehensweise bei der Realisierung einer integrierten Datenhaltung besteht in der Bereitstellung einer für die Verwaltung aller Datenbestände zuständigen zentralen DV-Anlage. Dies führt dazu, daß die in vielen Fällen dezentralisierte Struktur eines Unternehmens, einer Organisation oder einer technischen Anwendung künstlich auf ein zentral strukturiertes Datenverwaltungssystem abgebildet werden muß.

Demgegenüber kann ein verteiltes DV-System mit dezentralisierter Datenhaltung an dezentralisierte Organisationsstrukturen leicht angepaßt werden; es ermöglicht, Informationen dort zu halten und zu verarbeiten, wo sie vorwiegend benötigt oder erzeugt werden. Wichtig für eine derartige Form der Datenhaltung ist jedoch, daß die Kriterien der integrierten Datenhaltung, wie eingangs defininiert, erfüllt sind: das System muß in der Lage sein, bei Bedarf

Gesamtsichten der verteilt im System abgelegten Information zu bieten, entsprechend der Forderung, daß die insgesamt gespeicherte Information einem Modell des für eine vorgegebene Anwendung oder Klasse von Anwendungen relevanten Teils der realen Welt entsprechen muß.

Weitere Vorteile einer dezentralisierten Form der integrierten Datenhaltung liegen in der Reduktion des Kommunikationsaufwands und in der verbesserten Antwortzeit: da die Datenbestände dort gehalten werden, wo sie vorwiegend gebraucht werden, entfällt die Notwendigkeit der ständigen Kommunikation mit einer Zentrale; dadurch bedingte Leistungsengpässe, die zu langen Antwortzeiten führen, können vermieden werden.

Ein weiterer positiver Aspekt der verteilten Datenhaltung ist der der Erweiterbarkeit und Rekonfigurierbarkeit: ein verteiltes DV-System kann durch Hinzufügung oder Entfernung von Verarbeitungs- und Speicherkapazität an sich verändernde Leistungsanforderungen angepaßt werden.

Vor allem für Anwendungen im technischen Bereich, die eine hohe Zuverlässigkeit und Verfügbarkeit des eingesetzten DV-Systems verlangen, bietet die verteilte Datenhaltung neue Möglichkeiten: Datenbestände oder Teildatenbestände können auch mehrfach gehalten werden; die dadurch erreichbare Redundanz gestattet in Verbindung mit der Speicherung von Teildatenbeständen auf unterschiedlichen Rechnern in einem verteilten DV-System die Realisierung einer fehlertoleranten Betriebsweise, derart, daß die Funktionsfähigkeit des Gesamtsystems auch bei Ausfall einzelner Rechner oder Teildatenbestände, wenn auch mit reduzierter Leistungsfähigkeit, erhalten bleibt.

Die Entwicklung in der DV-Technologie begünstigt den Trend hin zum Einsatz verteilter DV-Systeme: die Großintegration von Halbleiterbauelementen hat dazu geführt, daß heute sehr leistungsfähige ($\approx 10^6$ Operationen/sec) und kostengünstige Kleinrechner angeboten werden, die es gestatten, verteilte DV-Systeme mit einer an zentrale Großrechnersysteme heranreichenden Gesamtleistungskapazität bei vertretbarem Investitionsaufwand

einzurichten. Kommunikations- und Datenbanktechnologie haben, jede für sich, einen Entwicklungsstand erreicht, der ihren Routine-Einsatz rechtfertigt; dies wird belegt durch die gegenwärtigen Bemühungen zur Standardisierung in beiden Bereichen. Technologisches Neuland dagegen ist die Integration beider Technologien bei der Entwicklung von Systemen für die integrierte verteilte Datenhaltung.

Das im Juli 1976 begonnene Vorhaben DISCO (Dezentralisierte InformationshaltungsStrukturen für Mini-COmputersysteme) hatte die Konzipierung und Realisierung einer Systemarchitektur für die integrierte Datenhaltung in verteilten DV-Systemen zum Ziel und setzte sich aus zwei Abschnitten zusammen:

Der erste Abschnitt hatte den Charakter einer Durchführbarkeitsstudie und beinhaltete als Kern eine Zusammenstellung und Analyse möglicher System-Architekturen für die verteilte Datenhaltung; Ziel war es, ein detailliertes Konzept zu entwickeln, das die Möglichkeit der Realisierung auch auf Kleinrechern bietet und weitgehend auf der vorhandenen Softwaregrundausstattung am Markt verfügbarer DV-Systeme aufbaut. Die Ergebnisse der im Juni 1977 abgeschlossenen Arbeiten des ersten Abschnittes wurden in /HBD78/ dokumentiert.

Das in /HBD78/ als Ergebnis der Analyse möglicher Architekturen für die integrierte verteilte Datenhaltung beschriebene Systemkonzept weist eine auf der logischen Dateiebene liegende Schnittstelle mit Benutzergruppen-orientierter Verteilungsstruktur der Datenbestände als für eine Vielzahl von Anwendungen geeignete Form der integrierten Datenhaltung mit verteilten Kleinrechersystemen aus. Die logische Dateiebene wird mittels eines Systems interagierender Verwaltungsinstanzen, sogenannter Funktionsbausteine für Zugriffs- und Katalogverwaltung, realisiert, die über mehrere Arbeitsrechner eines verteilten DV-Systems verteilt werden können; die Benutzer der Schnittstelle sind von allen verteilungsspezifischen Aspekten der Datenhaltung weitgehend entkoppelt.

Der zweite Abschnitt des Vorhabens (Juli 1977 bis Dezember 1979) hatte die Realisierung des in /HBD78/ beschriebenen Systemkonzepts zum Ziel.

Im einzelnen wurden folgende Teilziele verfolgt:

- die Entwicklung der zum Aufbau der Verwaltungsinstanzen benötigten Datenmanagement-Funktionsbausteine für die Zugriffs- und Katalogverwaltung, einschließlich der zur Koordinierung paralleler Zugriffe erforderlichen Zugriffskontrollen

- die Realisierung eines für die Kommunikation zwischen den verteilten Verwaltungsinstanzen geeigneten Kommunikationssubsystems zur Abwicklung sowohl des Daten- wie auch des Kontrollnachrichtentransfers

- die Entwicklung von Verfahren zur Sicherung und Aufrechterhaltung der Funktion des verteilten Datenhaltungssystems beim Ausfall einzelner Komponenten (Datenbestände, Verwaltungsinstanzen)

- der Aufbau eines Modellierungssystems zur Verifikation des Detailentwurfs der Funktionsbausteine und zur Durchführung von Integrationsexperimenten

- die Pilotimplementierung eines Systems für die verteilte Datenhaltung unter Verwendung der entwickelten Funktionsbausteine.

2. Architekturkonzept und operationales Modell einer verteilten Dateiverwaltung (DISCO-Ansatz)

2.1. Das Architekturkonzept des DISCO-Systems

Das für DISCO entwickelte System /BKD78/ zur Realisierung einer logischen Dateiebene basiert auf dem in Bild 2.1. dargestellten Architekturkonzept: über eine Hierarchie von Ebenen erfolgt die Transformation der auf der logischen Dateiebene möglichen Operationen (Datendefinitionen und -manipulationen) auf die Ebene der lokalen Arbeitsrechner-Dateiverwaltung (Teil des lokalen Betriebssystems).

Die zuoberst liegende logische Dateiebene verhält sich gegenüber ihren "Benutzern" (z.B. den darüberliegenden Ebenen eines verteilten Datenbanksystems) transparent bezüglich der Verteilung von Daten, d.h. die Benutzer werden von der zugrundeliegenden aktuellen physikalischen Verteilungsstruktur der gespeicherten Daten isoliert.

Zwei Arten logischer Dateien werden angeboten: A-Dateien zum Aufbau von Zugriffspfaden und B-Dateien zur Aufnahme der Primär-Daten (der Daten, die den eigentlichen Informationsbestand des Datenhaltungssystems ausmachen).

Eine B-Datei wird innerhalb des verteilten DV-Systems durch einen symbolischen Namen eindeutig gekennzeichnet und umfaßt eine beliebige Anzahl bitstrukturierter Datensätze fester Länge (d.h. der Inhalt der Datensätze wird vom logischen Dateiverwaltungssystem lediglich als Bit-String ohne übergeordnete Struktur behandelt). Direkter Zugriff auf die Datensätze ist über sequentiell vergebene Satznummern möglich, sequentieller Zugriff erfolgt nach dem Cursor-Prinzip. Beide Zugriffsformen sind unabhängig voneinander und können parallel benutzt werden. Jeder Datensatz kann indirekt einer sogenannten "Benutzergruppe" zugeordnet werden; dies wird erreicht durch Einrichtung von sogenannten "Clustern", Mengen von Datensätzen, die jeweils spezifischen Benutzergruppen assoziiert sind. Dies ermöglicht einen beschleunigten Zugriff in den Fällen, in denen die Zugriffshäufigkeit auf bestimmte Teile einer B-Datei besonders

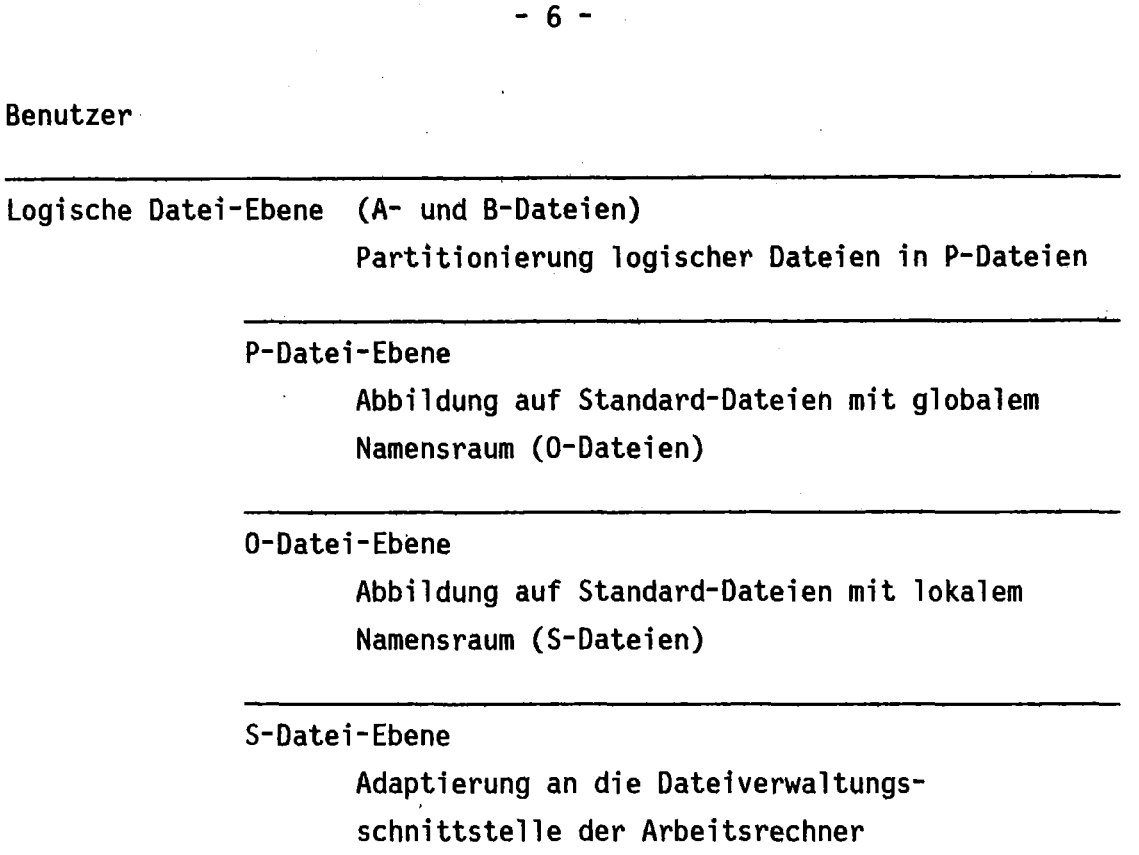


Bild 2.1: Architekturkonzept zur Realisierung einer globalen logischen Dateiebene in verteilten DV-Systemen (DISCO-Ansatz)

ausgeprägt ist. B-Dateien können partitioniert und über mehrere Arbeitsrechner eines verteilten DV-Systems verteilt sein.

Eine A-Datei wird in der gleichen Form durch symbolische Namen identifiziert wie eine B-Datei, kann jedoch aus Sätzen variabler Länge, auf die über Satzschlüssel zugegriffen wird, aufgebaut sein. Satzschlüssel besitzen eine feste Länge und eine vorgegebene einheitliche Position innerhalb jedes Datensatzes. Für sequentiellen Zugriff wird der Satzschlüssel als positive Zahl interpretiert.

A-Dateien sind nicht partitioniert und können mit einer oder mehreren Benutzergruppen gekoppelt sein. Wie auch für B-Dateien, gilt, daß dadurch für A-Dateien eine Nachbarschaftsbeziehung zu anderen A- bzw.

B-Dateien spezifiziert werden kann. In diesem Fall wird vom System versucht, die physikalische Nachbarschaft (gleicher Arbeitsrechner) für die angegebenen Dateien zu realisieren; dadurch wird eine schnellere Zugriffsabwicklung bei alternierendem Zugriff auf die "benachbart" eingerichteten Dateien möglich.

Die auf der logischen Dateiebene bereitgestellten Operatoren zur Datenmanipulation und Definition sind:

- Öffnen/Schließen von Dateien
- Einrichten/Löschen von Dateien
- Hinzufügen/Löschen von Datensätzen
- Bereitstellen von Datensätzen (direkter und sequentieller Zugriff), synchron (das aufrufende Programm wartet, bis der Datensatz bereitgestellt ist) oder asynchron
- Beginn/Ende einer Transaktion: die Einführung des Transaktionsbegriffes auf der logischen Dateiebene beinhaltet, daß entweder alle oder keine der durch eine Transaktion bewirkten Änderungen in der Datenbasis reflektiert werden (atomarer Charakter von Transaktionen zur Sicherung der internen Konsistenz der Datenbasis)
- Rücksetzen von Transaktionen: alle durch eine Transaktion bis zum Aufruf dieses Operators bewirkten Änderungen der Datenbasis werden rückgängig gemacht und alle gesperrten Datenobjekte freigegeben
- Sperren/Entsperren von Datensätzen (B-Dateien) oder gesamter (logischer) Dateien: dient der Konsistenzsicherung; es ist sukzessives Sperren erlaubt unter Berücksichtigung des Zwei-Phasen-Sperrprotokolls von /Esw76/: während der Wachstumsphase darf gesperrt werden, sobald aber ein Entsperren erfolgt, setzt die Schrumpfungsphase ein, in der keine weitere Sperroperation akzeptiert wird
- Bereitstellen von Satznummern für neu zu kreierende, in einen

bereits existierenden Cluster einzuordnende Datensätze

- Definition von Dateien: Festlegung von Dateinamen, Satzlänge, Schlüssel, Passworte, Relation zu anderen Dateien, Clusterspezifikation, Sicherheitsklassen (Grad der Redundanz zur Erhöhung der Verfügbarkeit einer Datei über Angabe der entsprechenden Sicherheitsklasse steuerbar) etc.

- Abbildung logischer Dateien auf physikalische Dateien (vgl. Bild 2.1)

Die unterste Dateiebene (Bild 2.1), die S-Ebene, definiert die Charakteristika der lokal in jedem zur integrierten Datenhaltung beitragenden Arbeitsrechner bereitzustellenden Dateien. Diese sogenannten S-Dateien können nur innerhalb des jeweiligen Arbeitsrechners eindeutig referenziert werden; d.h. ihr Namensraum hat lokalen Charakter. Die Merkmale der S-Datei sind:

- sie sind aus einer vorgegebenen Anzahl von Datenblöcken fester Länge aufgebaut
- die Blocklänge ist global einheitlich festgelegt
- Zugriff auf die Blöcke erfolgt nur direkt
- der Ort der Ablage (Rechner, Peripheriespeicher) kann gesteuert werden
- sie sind die Objekte der Sperrung und Sicherung

Folgende Operatoren existieren:

- Öffnen/Schließen von Dateien
- Bereitstellen von Blöcken
- Einrichten/Löschen von Dateien
- Sperren/Entsperren von Dateien
- Definition von Dateibeschreibungen, Festlegung von Dateinamen, Umfang, Passworten, Nachbarschaftsbeziehungen zu anderen S-Dateien

Die S-Dateiebene kann als Standard-Datei-Ebene verstanden werden, die dazu dient, die gegebenenfalls heterogenen lokalen Dateiverwaltungssysteme unterschiedlicher, in das verteilte DV-System einbezogener Rechner zu kompatibilisieren.

Die Abbildung der logischen Dateien (A- und B-Dateien) auf die S-Ebene erfolgt, wie in Bild 2.1 schematisch dargestellt, über zwei Zwischenebenen, die P- und O-Ebenen. Die Abbildung von B-Dateien auf die P-Ebene beinhaltet die Partitionierung einer B-Datei in mehrere Partitionen, die selbst jedoch nicht mehr partitioniert werden können. Ein Überlauf auf der B-Datei-Ebene muß deshalb durch Einrichtung neuer Partitionen zur Aufnahme der Überlauf-Datensätze abgefangen werden.

Während sich die Abbildung der Datensätze von B-Dateien auf Partitionen (die die gleiche Struktur wie S-Dateien aufweisen) einfach gestaltet, da alle Partitionen, auf die eine B-Datei abgebildet wird, über identische Blocklängen verfügen müssen, erfordert die Abbildung der A-Datei-Sätze komplexere Verknüpfungsmechanismen, die sich spezieller Baumstrukturen bedienen/Com79/.

Jede Partition ist genau einem Cluster zugeordnet; alle aus derselben B-Datei resultierenden Partitionen haben die gleiche Länge. Anzahl und Umfang der Cluster bestimmen daher die Zahl der einer B-Datei zuzuordnenden Partitionen.

Eine Partition kann physikalisch durch mehrere Kopien (Inkarnationen) realisiert sein; die Zuordnung dieser Inkarnationen, der O-Dateien, zu der (nur virtuell vorhandenen) Partition, erfolgt durch den Abbildungsmechanismus zwischen P- und O-Ebene. Neue Inkarnationen einer Partition können dynamisch kreiert werden. Die O-Datei-Ebene stellt zur Identifizierung der Inkarnationen von Partitionen einen globalen Namensraum bereit, der die eindeutige Benennung aller O-Dateien im verteilten DV-System ermöglicht. Da jede Partition genau einem Cluster zugeordnet ist, und damit einer Benutzergruppe, sind es auch die Inkarnationen der Partitionen, die O-Dateien. Benutzergruppen sind ihrerseits einem oder mehreren Rechnern

zugeordnet; eine Benutzergruppe bestimmt somit den Ort der Ablage der Inkarnationen einer Partition.

Durch Einrichtung mehrerer Inkarnationen (Kopien) derselben Partition kann die mittlere Zugriffszeit für Abfragen (lesende Zugriffe) bei der Verteilung einer Benutzergruppe über mehrere Rechner wesentlich verbessert werden. Dies gilt jedoch nicht für ändernde Zugriffe, wenn die Konsistenz der Kopien gewahrt werden soll: Änderungen einer Partition erfordern die Änderung aller ihrer Inkarnationen; dies ist mit erhöhtem Koordinierungsaufwand bei der Sperrung und Freigabe der Datenobjekte verbunden /Dro77/. Die Bestimmung der geeigneten Zahl der Inkarnationen einer Partition in Abhängigkeit vom Verhältnis ändernde Zugriffe/lesende Zugriffe wird zu einer Optimierungsaufgabe /Cas72/. Ein anderer Grund für die Einrichtung mehrerer Kopien der gleichen Partition ist die dadurch mögliche Steigerung der Verfügbarkeit des integrierten Datenhaltungssystems. Aus der Sicht der logischen Dateiebene eröffnet der Abbildungsmechanismus P-Ebene ⟶ O-Ebene die Möglichkeit, Cluster redundant zu führen.

Die Hauptaufgabe der Abbildung O-Datei-Ebene ⟶ S-Datei-Ebene ist die Transformation des globalen Namensraumes für O-Dateien in die lokalen Namensräume der S-Dateien auf den einzelnen Arbeitsrechnern des verteilten DV-Systems. Die bei der Einrichtung neuer S-Dateien erforderliche Auswahl des geeigneten Arbeitsrechners orientiert sich an der dem Datenhaltungssystem übermittelten Information über das Lokalitätsverhalten der zugeordneten Benutzergruppe(n).

Alle Dateiebenen des in DISCO realisierten Architekturkonzeptes, das auf verfügbaren lokalen Dateiverwaltungssystemen aufbaut, definieren lediglich virtuelle Dateien, die sich durch die auf ihnen möglichen Operationen und ihre für die Benennung verfügbaren Namensräume unterscheiden, und dienen dazu, die auf der logischen Dateiebene mögliche integrierte verteilte Datenhaltung auf die Ebene der lokalen Dateiverwaltung, auf der die Daten als lokale Dateien physikalisch gespeichert werden, zu transformieren.

2.2. Operationales Modell des DISCO-Systems

Während das oben erläuterte Architekturkonzept in erster Linie die Verteilungsstruktur der Daten beschreibt, zeigt das im folgenden behandelte operationale Modell auf, welche funktionellen Komponenten zur Realisierung dieses Architekturkonzeptes vorhanden sein müssen, wie sie verteilt werden können und welcher Art ihre Interaktionen sind.

Drei Klassen funktioneller Komponenten können identifiziert werden, wie Bild 2.2 verdeutlicht:

1. die lokale Vorverarbeitung vom Benutzer spezifizierter Operatorensequenzen (Transaktionen) und die Bereitstellung der Bearbeitungsresultate

2. die globale Bearbeitung von Operatoren unter Verwendung globaler Informationen; dies erfordert z.B. die Transformation der Operatoren in lokale Dateizugriffsoperationen

3. die Ausführung lokaler Dateizugriffe

Bezogen auf das in Bild 2.1 dargestellte Architekturmodell entspricht Klasse 1 der von der logischen Dateiebene bereitgestellten Schnittstelle, Klasse 2 beinhaltet die Abbildungsmechanismen zur Transformation von Operatoren der logischen Dateiebene in Operationen auf der S-Datei-Ebene einschließlich Rücktransformation der Ergebnisse und Klasse 3 die Transformation S-Datei-Operationen in Operationen im lokalen Dateiverwaltungssystem, Durchführung der lokalen Operation und Rücktransformation der Ergebnisse.

Entsprechend der oben aufgeführten drei Klassen funktioneller Komponenten wird eine Partitionierung des operationalen Modells eingeführt (vgl. Bild 2.2). Die Verteilungsstruktur der Inkarnationen funktioneller Komponenten, der Funktonseinheiten, wird bestimmt durch die unterschiedlichen Eigenschaften der Klassen:

Die Klassen 1 und 3 umfassen lokale Komponenten, von denen jede nur

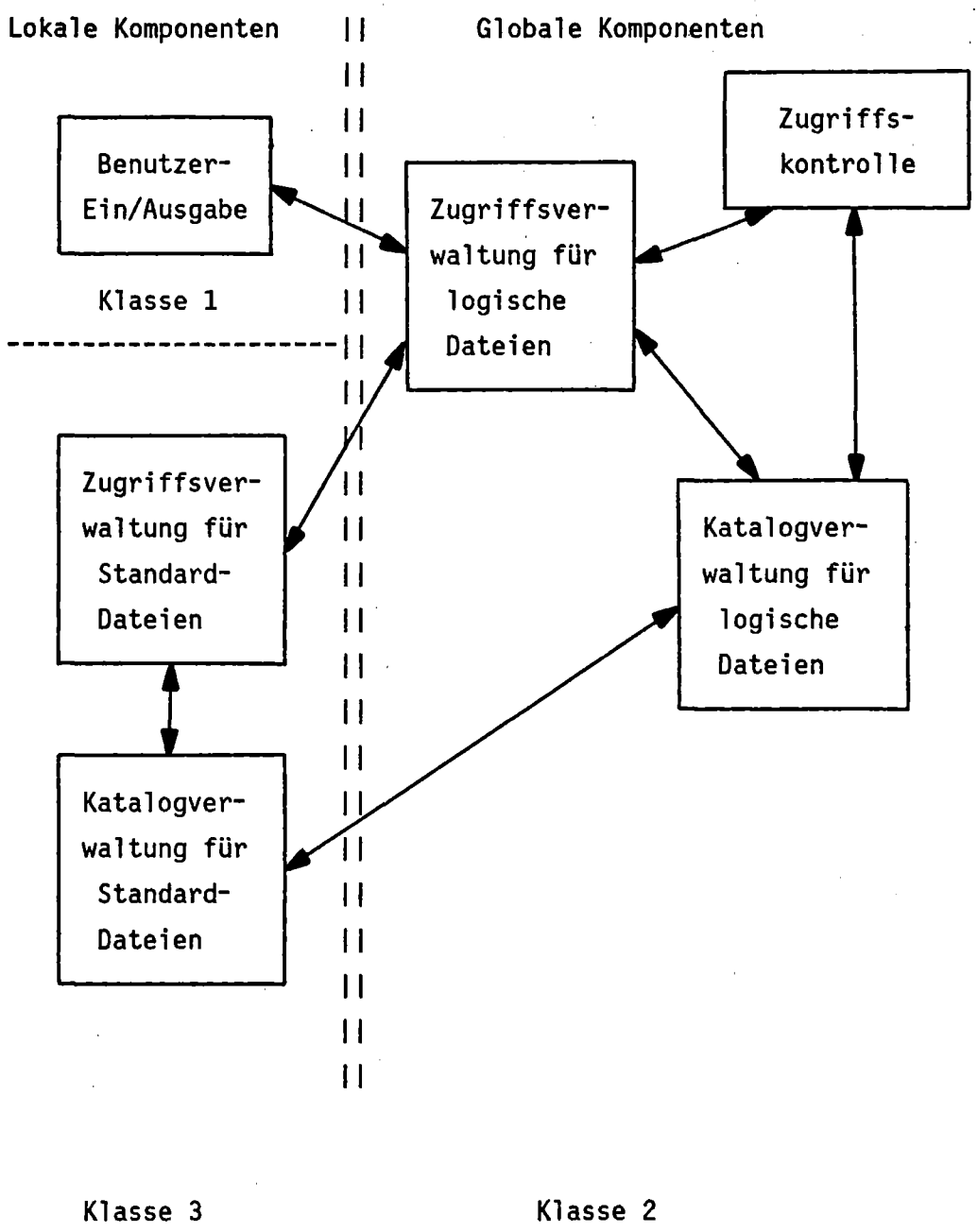


Bild 2.2: Operationales Modell des DISCO-Systems

auf den Arbeitsrechnern realisiert wird, die den Benutzern den Zugang zum System ermöglichen sollen bzw. die zur Datenspeicherung beitragen. Dies ermöglicht die Einbeziehung auch leistungsschwächerer Arbeitsrechner zur Abwicklung einzelner spezifischer Aufgaben, wie etwa der Verwaltung ausgewählter lokaler Datenbestände.

Im Gegensatz dazu können die Komponenten der Klasse 2 frei über

mehrere Arbeitsrechner verteilt werden; dies gilt sowohl bezüglich der Auswahl der Arbeitsrechner als auch für die Zahl der Inkarnationen einer einzelnen funktionellen Komponente.

Die Komponenten der Klasse 2 können als Kern des integrierten verteilten Datenhaltungssystems verstanden werden. Durch die gegebene Variationsmöglichkeit der Verteilungsstruktur kann dieser Systemkern entweder zentralisiert (alle Funktionseinheiten auf einem Arbeitsrechner) oder vollständig verteilt werden. Die Wahl der geeigneten Verteilungsstruktur wird durch die Anforderungen der Anwendung beeinflußt; Benutzerverhalten, Eigenschaften der verfügbaren Arbeitsrechner und des für den Nachrichtenaustausch zwischen den funktionellen Komponenten verwendeten Nachrichtentransportsystems sowie Umfang und Struktur der Datenbasis sind die zu berücksichtigenden Randbedingungen. Die Bestimmung der Verteilungsstruktur kann pragmatisch durchgeführt werden oder als Optimierungsaufgabe, die in Anlehnung an die in /Cas72, Chu69, LeM75, MaR76/ vorgeschlagene Vorgehensweise behandelt werden kann.

Die Aufgaben der einzelnen funktionellen Komponenten der Klassen 1 und 3 werden im Kapitel 3, der Klasse 2 in den Kapiteln 4 und 5 detailliert erläutert. Die Einbeziehung der Problematik "Ausfall/Wiederanlauf" in dem DISCO-Ansatz wird im Kapitel 6 aufgezeigt.

Die funktionellen Komponenten wurden im Rahmen eines Modellierungssystems (in der Programmiersprache SIMULA) detailliert spezifiziert und ihre Interaktionen in Simulationsexperimenten untersucht. Diese Arbeiten bildeten die Grundlage für die Pilotimplementierung eines Systems für verteilte Datenhaltung; Kapitel 7 gibt eine Übersicht über diese Pilotimplementierung.

3. Schnittstellen des DISCO-Systems

Entsprechend seiner Schichtenarchitektur hat das System DISCO zwei Schnittstellen zu seiner Umwelt:

- Schnittstelle zum Benutzer auf der Ebene der logischen Dateien
- Schnittstelle zur lokalen Dateiverwaltung auf der lokalen Ebene

Als Benutzer wird im folgenden jeder Prozeß bezeichnet, der mit dem System über die Benutzerschnittstelle kommuniziert. Das können sein: Anwendungsprogramme, Terminal-Monitore, Systemprozesse, die untere Schichten eines umfassenderen Systems (z.B. DBMS) realisieren.

3.1. Schnittstelle zum Benutzer

Die Schnittstelle zum Benutzer ermöglicht die Kommunikation zwischen Benutzer und System. Diese beinhaltet das Absetzen der DISCO-Operatoren sowie das Übergeben von Ergebnissen und Rückmeldungen.
Die Benutzerschnittstelle untergliedert sich in die funktionelle Komponente BEA und einen Anschlußmodul, der zu jedem Benutzerprogramm zuzubinden ist.

3.1.1. Anschlußmodul

Der Anschlußmodul dient dem Zweck, dem Benutzer zu ermöglichen auf einfache Art mit DISCO zu arbeiten. In ihm werden alle organisatorischen Vorbereitungen getroffen, die nötig sind, um die Kommunikaton zwischen dem System und dem Benutzer zu betreiben. Für den Benutzer erfolgt das Ansprechen des Systems durch Aufrufen der DISCO-Operatoren zur Datendefinition und -manipulation, wobei jedem Operator ein Unterprogramm entspricht. Die Parameter der Operatoren entsprechen dabei jenen des jeweiligen Unterprogramms. Der Benutzer kann somit Programmvariable direkt im Aufruf angeben, ohne erst Kontrollblöcke besetzen zu müssen, wie es z.T. bei anderen Systemen üblich ist.

Für den Systemimplementierer bietet dieser Ansatz den Vorteil, daß durch das Laufzeitsystem der verwendeten Programmiersprache bereits Syntax- u. Semantikprüfungen durchgeführt werden können.

Die Eigenschaft verschiedener Operatoren, dem Benutzer zu erlauben, optionale Parameter wegzulassen, bleibt auch bei diesem Ansatz erhalten. Dies wird erreicht durch geringfügige Eingriffe in die Systemroutinen, die die Parameterübergabe durchführen. Der Benutzer bemerkt hiervon nichts.

3.1.2. Semantik der Operatoren

In diesem Kapitel werden die Operatoren zur Datendefinition und -manipulation des Systems DISCO vorgestellt und ihre Semantik beschrieben. Die genaue Syntax der Operatoren wird in Anhang A definiert.

Hinweis: In eckigen Klammern stehende Parameter sind optional; bei geschweiften Klammern muß eine der angebotenen Möglichkeiten ausgewählt werden.

Definition der Parameter

RTC: enthält nach Ausführung des Operators den Returncode. Bei fehlerfreier Beendigung der Operation ist dieser Ø, sonst eine Codierung des aufgetretenen Fehlers.

EV: ist, wenn er zugelassen ist, stets optional und bewirkt, daß die Ausführung des Operators asynchron erfolgt. EV ist eine boolsche Größe und erhält den Wert .true., wenn die Ausführung beendet ist.
Der Benutzer kann somit das Ende der Operation abfragen, ohne durch den Operator WAIT die Kontrolle über das Programm abzugeben.

Datnam: Dateiname, bestehend aus maximal 16 Zeichen; darüberhinausgehende Zeichen bleiben unbeachtet. Alle Zeichen aus dem Zeichenvorrat der Programmiersprache sind

zugelassen.

SN: bezeichnet eine Satznummer, die für die betreffende Operation zugelassen sein muß.

Puffer: Speicherbereich im Benutzerprogramm, aus dem Daten gelesen bzw. in den Daten geschrieben werden.

Key: Speicherbereich im Benutzerprogramm, der den Schlüssel enthält bei Operatoren für A-Dateien.

Lng: gibt die Länge eines A-Satzes an.

Paßwort: hat die Form /Leseschl/Schreibschl/, wobei Leseschl, Schreibschl jeweils aus 1-6 Zeichen bestehen bzw. ganz entfallen können.

Beispiel: nur Leseschl: /abc//
nur Schreibschl: //xy/

Satzmanipulation bei B-Dateien

READ (RTC, Datnam, SN, Puffer [,EV])
Aus der Datei "Datnam" wird der Satz mit der Satznummer "SN" gelesen und im Speicherbereich "Puffer" abgelegt. Der Umfang von "Puffer" muß so bemessen sein, daß er den gelesenen Satz aufnehmen kann; ansonsten ist der Inhalt undefiniert.

READS (RTC, Datnam, $\{^{V}_{R}\}$, Puffer [,EV])
Aus der Datei "Datnam" wird der Satz gelesen und im Speicherbereich "Puffer" abgelegt, der in der angegebenen Richtung (Vorwärts/Rückwärts) auf den Satz folgt, der durch einen Satzzeiger referenziert wird. Der Satzzeiger zeigt anschließend auf den gelesenen Satz.
Wenn das Ende der Datei erreicht ist, ist der Operator unzulässig.

WRITE (RTC, Datnam, SN, Puffer [,EV])
WRITES (RTC, Datnam, Puffer [,EV])
Die in dem Speicherbereich "Puffer" stehenden Daten werden in der Datei "Datnam" an die Stelle geschrieben, die durch den Parameter

"SN" bzw. durch den Satzzeiger bestimmt wird. Bei WRITES wird der Satzzeiger nicht verändert.

Der Umfang vom "Puffer" muß der Satzlänge von "Datnam" entsprechen. Ist die Satzlänge größer als die übertragenen Daten, bleibt der Rest des bisherigen Satzes stehen, bei kleinerer Satzlänge wird nur der der Satzlänge entsprechende Anfangsteil der Daten in die Datei geschrieben, der Rest geht verloren.

Mit diesen Operatoren kann nur der Inhalt existierender Sätze geändert werden; es ist mit ihnen nicht möglich, neue Sätze in eine Datei einzuschreiben! Bezeichnet die angegebene Satznummer bzw. der Satzzeiger (nach SETCB möglich!) keinen vorhandenen Satz, wird die Ausführung des Operators abgebrochen !

INB (RTC, Datnam, SN, Puffer [,EV])
INBS (RTC, Datnam, Puffer [,EV])
Analog zum WRITE Operator werden die im "Puffer" stehenden Daten an die durch "SN" bezeichnete Stelle (bei INB) bzw. ans Ende der Datei (bei INBS) geschrieben. Die Satznummer "SN" darf bisher noch nicht vergeben sein. Diese Operatoren dienen dem Erzeugen neuer Sätze.

REMB (RTC, Datnam, SN [,EV])
Der Satz mit der Satznummer "SN" wird aus der Datei "Datnam" gelöscht. Die Satznummer kann anschließend neu vergeben werden.

NEW (RTC, Datnam, SN [,UG])
In dem Parameter "SN" wird eine bisher noch freie Satznummer der Datei "Datnam" dem Benutzer bekanntgemacht. Bei Angabe einer Benutzergruppe "UG" wird diese aus einem dieser Benutzergruppe zugeordneten Cluster beschafft.

Gleichzeitig wird die Satznummer vergeben, d.h. es existiert ab diesem Zeitpunkt ein Satz mit dieser Satznummer undefinierten Inhalts. Daten können mittels WRITE eingeschrieben werden, INB ist nicht möglich.

SETCB (RTC, Datnam, $\left\{\begin{matrix}\underline{SN}\\ \underline{FG}\\ \underline{ST}\end{matrix}\right\}$)

Der Satzzeiger der Datei "Datnam" wird auf den Satz mit Satznummer "SN" gesetzt, bzw. wenn statt einer Satznummer "FG" angegeben ist, auf das Ende, bei "ST" an den Anfang der Datei.

GETCB (RTC, Datnam, SN)
In "SN" wird der Wert des Satzzeigers der Datei "Datnam" übergeben.

Satzmanipulation bei A-Dateien

GET (RTC, Datnam, Key, Puffer [,EV])
Analog zu READ wird der Satz mit dem Schlüssel "Key" in "Puffer" eingelesen.

GETS (RTC, Datnam, $\{^{V}_{R}\}$, Puffer [,EV])
Analog zu READS wird der Satz, der dem durch den Satzzeiger bezeichneten Satz in der angegebenen Richtung folgt, in "Puffer" eingelesen und der Satzzeiger auf den gelesenen Satz verändert.

PUT (RTC, Datnam, Puffer, Lng [,EV])
Aus dem Speicherbereich "Puffer" wird ein Satz der Länge "Lng" in die Datei "Datnam" geschrieben. In der Datei muß bereits ein Satz sein, der den gleichen Schlüssel besitzt wie der durch PUT geschriebene. Die Satzlänge beider Sätze muß nicht übereinstimmen. Der neue Satz ersetzt den bisher vorhandenen. Die logische Ordnung der Datei bleibt erhalten.

PUTS (RTC, Datnam, Puffer, Lng [,EV])
Analog zu PUT ersetzt der in "Puffer" stehende Satz der Länge "Lng" den Satz in "Datnam", der durch den Satzzeiger bezeichnet ist. Die Schlüssel beider Sätze müssen übereinstimmen, bei der Satzlänge ist dies nicht erforderlich.

Die Bearbeitung von PUTS ist in der Regel schneller als die von PUT, da der Zugriff nicht über den Index erfolgen muß.

INA (RTC, Datnam, Puffer, Lng [,EV])
Der in "Puffer" stehende Satz der Länge "Lng" wird unter Berücksichtigung der durch den Schlüssel gegebenen Ordnung neu in die

Datei "Datnam" eingefügt. Es darf bisher kein Satz mit gleichem Schlüssel in der Datei vorhanden sein.

REMA (RTC, Datnam, Key [,EV])
Der Satz mit dem Schlüssel "Key" wird aus der Datei "Datnam" gelöscht.

SETCA (RTC, Datnam, Key)
Setzt den Satzzeiger der Datei "Datnam" auf den Satz mit dem Schlüssel "Key". Existiert kein solcher Satz, wird auf den in der Schlüsselordnung vorhergehenden Satz positioniert.

Anmerkung: Positionierung auf den Dateianfang kann durch Angabe eines leeren Schlüssels erreicht werden.

SETMAX (RTC, Datnam)
Positioniert den Satzzeiger der Datei "Datnam" ans Dateiende.

Dateitypunabhängige Operatoren

LOGON (RTC)
Mit diesem Operator muß eine Session mit dem DISCO-System eröffnet werden. Mit seiner Bearbeitung werden die Voraussetzungen für die Kommunikation mit dem System erfüllt.

LOGOFF
Dieser Operator beendet eine Session mit dem DISCO-System. Es kann wieder eine neue Session mit LOGON begonnen werden.

BTRANS (RTC) (BTRANS = BEGIN_TRANS)
Eine Transaktion wird begonnen. (Einzelheiten zum Begriff der Transaktion siehe Kapitel Kap.2 u.4). Zum Zeitpunkt der Bearbeitung dieses Operators darf keine weitere Transaktion des gleichen Benutzers begonnen sein, d.h. es ist keine Schachtelung von Transaktionen erlaubt.

ETRANS (RTC) (ETRANS = END_TRANS)
Eine mit BTRANS begonnene Transaktion wird beendet. Alle Änderungen an der Datenbasis werden wirksam.

UNDO (RTC)
Eine Transaktion wird abgebrochen. Alle seit dem letzten BTRANS von dem Benutzer gemachten Änderungen an der Datenbasis werden rückgängig gemacht. Die Datenbasis befindet sich im Zustand wie vor der Bearbeitung von BTRANS.

OPEN (RTC, Datnam [,Paßwort])
Die logische Datei "Datnam" wird eröffnet. OPEN muß vor dem 1. Operator, der auf eine Datei zugreift, erfolgt sein. Die Datei bleibt für den Benutzer geöffnet, bis sie mit CLOSE explizit , oder nach LOGOFF vom System aus geschlossen wird. Das Eröffnen einer Datei mit OPEN kann sowohl innerhalb einer Transaktion als auch außerhalb einer Transaktion erfolgen. Erfolgt es innerhalb einer Transaktion, so bleibt die Datei auch dann geöffnet, wenn die Transaktion zurückgesetzt wird, d.h. wenn der Benutzer die Transaktion mittels UNDO abbricht, oder diese vom System infolge eines Deadlocks (siehe Kap. 4.3) oder ähnliches beendet wird.

Wurde die Datei bei ihrer Definition (siehe DEFA, DEFB) mit einem Paßwort versehen, so wird ein Zugriff auf die Datei nur zugelassen, wenn diese mit dem jeweiligen Paßwort eröffnet wurde; d.h. für lesenden Zugriff muß der Leseschlüssel, für ändernden der Schreibschlüssel angegeben sein. Ein späteres Erweitern des bei OPEN angegebenen Schlüssels ist nicht möglich. Um dies zu erreichen, muß die Datei explizit mit CLOSE geschlossen und anschließend mit vollständigem Paßwort wieder eröffnet werden.

CLOSE (RTC, Datnam)
Die logische Datei "Datnam" wird geschlossen. Das Schließen kann sowohl innerhalb einer Transaktion erfolgen, als auch außerhalb. Sofern eine Datei innerhalb einer Transaktion geschlossen wird, und die Transaktion wird nicht durch ETRANS beendet, sondern vorher zurückgesetzt (siehe OPEN), so wird auch die Wirkung von CLOSE gelöscht, d.h. die Datei ist noch geöffnet.

LOCK (RTC, Datnam, $\{^{S}_{X}\}$ [,Satzn])
Die Datei "Datnam" wird im angegebenen Typ (S=shared, X=exclusive) (siehe Kap.4.) gesperrt. Die Sperre bezieht sich auf die ganze Datei,

sofern der Parameter "Satzn" fehlt, oder auf die in "Satzn" spezifizierten Sätze.

Sperren mit Typ=X ist erforderlich für alle Operatoren, die Änderungen der Datenbasis bewirken. Das sind: WRITE, WRITES, INB, INBS, REMB, NEW, PUT, PUTS, INA, REMA, DELETE.

Sperren mit Typ=S ist dann erforderlich, wenn der Benutzer sicherstellen will, daß die Daten während der Dauer seiner Transaktion nicht verändert werden.

Ein LOCK-Kommando kann nur innerhalb einer Transaktion bearbeitet werden. Eine Sperre bleibt bestehen, bis sie entweder explizit durch UNLOCK (bzgl. des LOCK-UNLOCK-Protokolls siehe Kap. 2.) oder vom System bei Beendigung der Transaktion aufgehoben wird.

Ein Ändern des Sperrtyps von S nach X ist nicht möglich. In einem solchen Fall muß die Transaktion abgebrochen und mit neuem LOCK-Kommando wiederholt werden.
Zu beachten ist, daß die Bearbeitung des LOCK-Operators zu Deadlocks führen kann (siehe Kap. 4.3), als deren Folge die Transaktion unter Umständen abgebrochen werden muß.

UNLOCK (RTC, Datnam)
Sämtliche Sperren, die bzgl. der Datei "Datnam" gehalten werden, werden freigegeben.

CREATE (RTC, Datnam)
Dieses Kommando muß außerhalb einer Transaktion erfolgen! Die Datei muß mit Schreibschlüssel eröffnet sein.

DELETE (RTC, Datnam)
Die Datei "Datnam" wird gelöscht. Ihre Definition bleibt erhalten, so daß durch ein CREATE-Kommando die Datei wieder erzeugt werden kann.

DELETE muß innerhalb einer Transaktion erfolgen und die Datei muß mit Typ=X gesperrt sein. Außerdem muß die Datei mit dem Schreibschüssel eröffnet worden sein. Die Löschung wird erst bei der Bearbeitung von ETRANS wirksam; wird die Transaktion vorzeitig abgebrochen, bleibt

die Datei erhalten.

WAIT (EV)
Das Benutzerprogramm wird fortgesetzt, wenn die Variable EV den Wert .true. besitzt.

Operatoren zur Dateidefinition

Beachte: Die folgenden Operatoren dürfen nur außerhalb von Transaktionen verwendet werden! Der Grund ist, daß sie ab dem Zeitpunkt ihrer Bearbeitung, im Gegensatz zur Definition von Transaktionen, global im System wirken (das gleiche gilt für CREATE).

DEFA (RTC, Datnam, PA1, PA2, ..., PA7)

PA1 :: = Richtsatzlänge = durchschnittlicher Umfang eines Satzes
PA2 :: = Abschätzung über den Umfang der Datei
PA3 :: = Paßwort, in der Form /Leseschl/Schreibschl/ (siehe Def. der Parameter)
PA4 :: = Sicherheitsklasse (= Anzahl der Kopien)
PA5 :: = Schlüsselanfang = relat. Byte-Adr. im Satz
PA6 :: = Schlüssellänge in Bytes
PA7 :: = Benutzergruppen (Angabe von 1-4 Benutzergruppen möglich)

Die Definition der A-Datei "Datnam" wird im Katalog vermerkt. Die Erzeugung erfolgt erst bei CREATE.
Die Richtsatzlänge und die Abschätzung des Umfangs der Datei dienen dazu, die Reservierung von Speicher bei CREATE zu steuern.

Der Parameter für das Paßwort muß bei der Definition angegeben werden. Soll die Datei ohne Restriktion zugänglich sein, so kann das durch leere Schlüssel erfolgen (///). Die Sicherheitsklasse gibt die minimale Anzahl von Kopien der Datei an; durch die Angabe der Benutzergruppen, die hauptsächlich mit der Datei arbeiten, kann die Zuordnung der Kopien zu den Rechnern gesteuert werden. Durch die Angaben in PA5 und PA6 wird der Ausschnitt eines Satzes festgelegt, der als Ordnungskriterium für die Datei fungiert. Damit wird auch die minimale Größe eines Satzes bestimmt, da jeder Satz mindestens den Schlüssel enthalten muß.

DEFB (RTC, Datnam, PA1, PA2, ...,PA7)

PA1 :: = Satzlänge
PA2 :: = Abschätzung des Umfangs
PA3 :: = Paßwort
PA4 :: = Sicherheitklasse
PA5 :: = Anzahl Sätze/Partition
PA6 :: = Anzahl Cluster (max. Anzahl z.Z. 8)
PA7 :: = Beschreibung der Cluster

Die Definition der B-Datei "Datnam" wird im Katalog vermerkt. Die Erzeugung der Datei erfolgt erst bei CREATE.
Mit den Parametern PA5, PA6, PA7 wird die Struktur der Datei festgelegt und damit auch die Möglichkeiten zur Verteilung der Datei im Rechnernetz. (Näheres über den Aufbau von B-Dateien siehe Kap. 5)

Die Beschreibung der Cluster erfolgt in der Form:

PA7 :: = $[\text{<Clusterdefinition>}]_1^{PA6}$

<Clusterdefinition> :: = Anzahl der Partitionen
Benutzergruppe 1
[Benutzergruppe 2]
[Benutzergruppe 3]

Die Bedeutung der restlichen Parameter entspricht denen von DEFA.

UNDEF (RTC, Datnam)
Die Definition der Datei "Datnam" wird aus dem Katalog gestrichen. Zusätzlich darf die Datei nicht eröffnet und muß bereits gelöscht sein (DELETE).

MAP (RTC, Datnam, Partition, PA1, PA2)

PA1 :: = Anzahl von Kopien (z.Z. 1-3)
PA2 :: = <O-Dateien-Liste>

mit

<O-Dateien-Liste> :: = $[\text{<Dateibeschreibung>}]_1^{PA1}$
<Dateibeschreibung>:: = <S-Dateiname> <Rechnernr>

<S-Dateiname> besteht aus 2 Zeichen
<Rechnernr> bezeichnet die Nummer des Zielrechners im Rechnernetz

Die logische Datei "Datnam" wird durch die spezifizierten O-Dateien realisiert. Damit wird eine vom System erzeugte Abbildung, die aus den Angaben bei der Definition gebildet wurde, redefiniert gesetzt. Die logische Datei darf weder erzeugt noch eröffnet sein.

DEFUG (RTC,UG, PA)
PA :: = $[(\text{Rechner})]_1^3$

Definition einer Benutzergruppe. Dem System wird eine neue Benutzergruppe bekanntgemacht, die hauptsächlich von den in "PA" angegebenen Rechnern aus mit dem System arbeitet. In "UG" wird die systeminterne Benutzergruppenkennung übergeben, die bei DEFA, DEFB, NEW verwendet werden kann.

UDEFUG (RTC,UG)
Die Benutzergruppe "UG" wird aus dem Katalog gelöscht. Sie kann nicht mehr angesprochen werden.

DEFGS (RTC, S-Dateiname, Rechnernr, Umfang)
Definition einer Standard-Datei. Auf dem in "Rechnernr" angegebenen Rechner des Netzes wird eine Datei mit Namen "S-Dateiname" (max. 2 Zeichen) angelegt. Ihr Umfang umfaßt die in "Umfang" angegebene Zahl von Blöcken .

UDEFGS (RTC, S-Dateiname, Rechnernr)
Die Standarddatei "S-Dateiname" wird aus dem Katalog des Rechners "Rechnernr" gestrichen.

3.1.3. Funktionelle Komponente Benutzer-Ein/Ausgabe

Die funktionelle Komponente BEA ist das Verbindungsglied zwischen Benutzerprozessen und dem System. Auf jedem Rechner, von welchem Benutzereingaben möglich sind, gibt es genau eine funktionelle Einheit BEA, die gegebenenfalls mehrere Benutzer bedient.

Jede funktionelle Einheit BEA ist mit genau einem Exemplar der

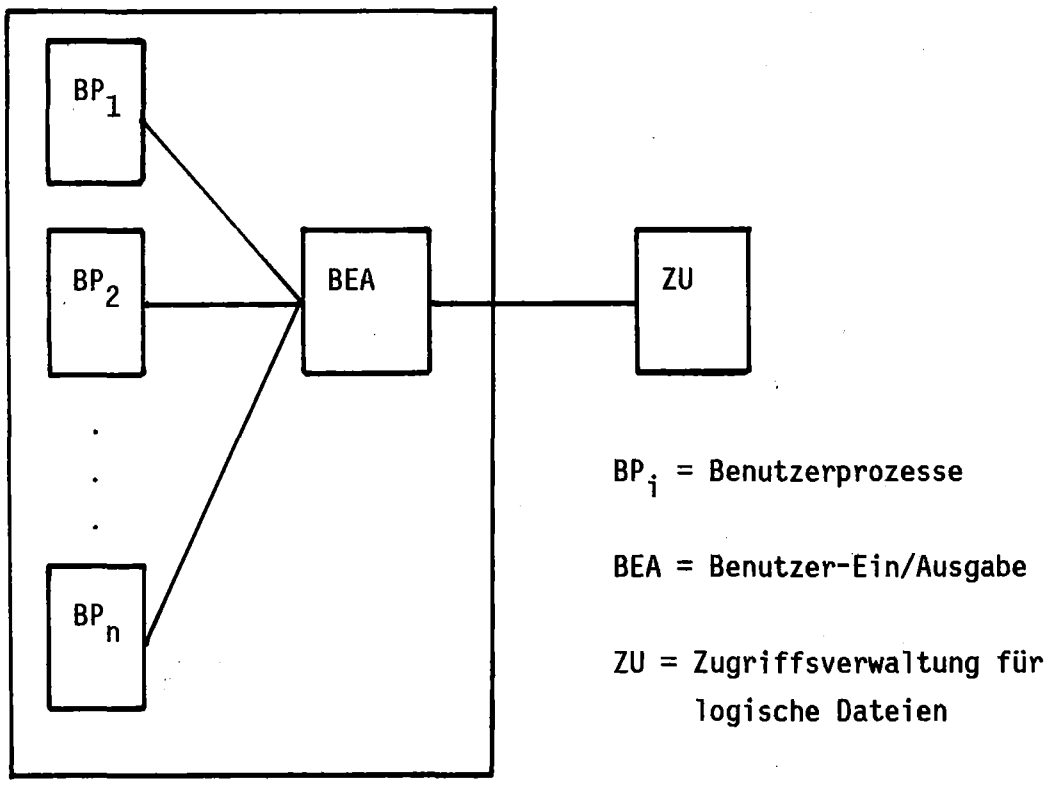


Bild 3.1: Verbindung zwischen Benutzerprozessen, BEA und ZU

Komponente ZU verbunden. Diese Funktionseinheit ZU kann sowohl auf dem gleichen Rechner als auch auf einem anderen Rechner des Netzes installiert sein (siehe Bild 3.1).

3.1.3.1. Aufgabe der Benutzer-Ein/Ausgabe

Die BEA nimmt Aufträge von Benutzerprozessen entgegen und leitet sie an die ihr zugeordnete ZU weiter. Umgekehrt empfängt sie von der ZU Rückmeldungen und Ergebnisse, die sie den entsprechenden Benutzerprozessen zuzuordnen und an diese auszugeben hat. Sie erfüllt somit eine Multiplexerfunktion in Bezug auf logische Kanäle, wodurch der Aufwand des Systems verringert und die Kommunikation mit dem Benutzer vereinfacht wird.

Desweiteren ermöglicht die BEA Benutzerprozessen, Aufträge asynchron zu bearbeiten. Das bedeutet, daß trotz unterschiedlicher Reihenfolge der Beendigung einzelner Aufträge die richtige Zuordnung von

Ergebnissen zu Auftragsparametern gewahrt bleibt.

Die funktionellen Aufgaben der BEA bestehen somit einmal darin, Kommunikationsverbindungen zur ZU und zu Benutzerprozessen herzustellen und zu unterhalten. Außerdem müssen Verwaltungsinformationen geführt werden, die die Zuordnung von Ergebnissen zu den entsprechenden Aufträgen gewährleisten.

3.1.3.2. Realisation der Kommunikation

Die BEA unterhält Kommunikationsverbindungen zu ZU und Benutzerprozessen. In diesem Kapitel wird beschrieben, wie diese im einzelnen realisiert werden.

Verbindung mit der ZU

Die Nachrichtenverbindung mit der ZU erfolgt mit Hilfe des Nachrichtentransportsystems, da nicht auszuschließen ist, daß die betroffene Funktionseinheit auf einem anderen Rechner installiert ist.

Verbindung mit Benutzerprogrammen

Die Vermittlung von Nachrichten vom Benutzerprogramm geschieht über das Nachrichtentransportsystem, um zu ermöglichen, daß die BEA jederzeit bereitstehende Nachrichten sowohl von ZU als auch von Benutzerprozessen empfangen kann.

Die Übermittlung von Rückmeldungen und Ergebnissen wird mit Funktionen der Interprozeßkommunikation des jeweiligen Betriebssystems des Host-Rechners ausgeführt, da nur so die Realisierung asynchroner Operatoren möglich ist.

Die Übertragung von Daten (Inhalt von Sätzen bei Lese/ Schreiboperationen) zwischen ZU und Benutzerprozeß erfolgt direkt in den bzw. aus dem entsprechenden Speicherbereich des Benutzerprozesses durch das Nachrichtentransportsystem.

3.2. Schnittstelle zur lokalen Dateiverwaltung

Der Konzeption des Systems DISCO liegt ein heterogenes Rechnernetz zu Grunde. Um Portabilität zu erreichen, wird das System jedoch auf einer einheitlichen, homogenen Dateischnittstelle - der S-Ebene (siehe Kap. 2) - aufgesetzt. An diese Schnittstelle müssen alle Dateiverwaltungssysteme der verschiedenen Rechner angepaßt werden. Es muß somit eine Abbildung der an der S-Ebene vorhandenen Operatoren und Datenstrukturen in die des jeweiligen lokalen Dateiverwaltungssystems stattfinden. Diese Abbildung wird in der Komponente "Anpassung" (ANP) vorgenommen.

Diese Komponente ist damit zuständig für den Zugriff auf die gespeicherten Daten und ihre Weiterleitung an den Empfänger, bzw. die Einspeicherung von Daten.

Daraus ergibt sich:

- eine Einheit der Komponente "Anpassung" liegt auf jedem Rechner, auf dem vom DISCO-System benutzte Daten gespeichert sind

- jede dieser Einheiten unterhält Kommunikationsverbindungen zu allen Einheiten der Komponente ZU

- von diesen nimmt sie Aufträge, die die Bearbeitung von Operatoren der S-Ebene beinhalten, entgegen, transformiert diese in Operationen des lokalen Dateiverwaltungssystems und sendet das Ergebnis zurück.

Die Abbildung der S-Ebene auf die Ebene des lokalen Dateiverwaltungssystems spaltet sich auf in die Komplexe

- Verwaltung von Speicherplatz,
  angesprochen durch die Operatoren DEF, UNDEF, CREATE, DELETE

- Zugriffe zu Daten
  entsprechend den Operatoren OPEN, CLOSE, READ, WRITE

3.2.1. Verwaltung von Speicherplatz

Die Verwaltung von Speicherplatz beinhaltet die Abbildung von S-Dateien auf lokale Dateien sowie die Führung der dafür benötigten Information (Katalog). Die Abbildung soll so erfolgen, daß der Zugriff zu den Daten optimiert werden kann. Eine zusätzliche Bedingung ist, die Organisation des Katalogs möglichst einfach zu gestalten, so daß sie leicht zu implementieren ist und wenig Speicherplatz benötigt (/Ker79a/).

S-Dateien haben die Eigenschaft, aus einer physisch kompakten Menge einer festen Anzahl von Blöcken mit einheitlicher Blocklänge (/HBD78/) zu bestehen. Eine geeignete Lösung besteht deshalb darin, die lokalen Dateien möglichst groß zu machen, d.h. sie über den ganzen Datenträger auszudehnen. Dadurch besteht die Möglichkeit, viele S-Dateien auf nur eine lokale Datei abzubilden. Der Hauptspeicher, der zur Verwaltung lokaler Dateien für Dateibeschreibungen und Transferpuffer benötigt wird, wird deshalb auf ein Minimum beschränkt.

S-Dateien können dynamisch angelegt und gelöscht werden. Innerhalb der lokalen Dateien, auf die S-Dateien abgebildet werden, muß deshalb eine Speicherverwaltung durchgeführt werden, um die jeweiligen Speicherbereiche zu beschaffen bzw. bis zur späteren Wiederverwendung freizugeben. Diese kann mittels bekannter Verfahren - Freispeicherliste, Bitliste oder ähnliches - realisiert werden.

Die Ablage der Kataloginformation geschieht zweckmäßigerweise in einer S-Datei, evtl. verbunden mit einer zusätzlichen Möglichkeit eines Schlüsselzugriffs. Es wird somit keine zusätzliche Speicherorganisation benötigt und die Katalogverwaltung besitzt eine erhöhte Portabilität.

3.2.2. Zugriffe zu Daten

S-Dateien ermöglichen direkten Zugriff auf Blöcke. Dies entspricht der Zugriffsform, die von den meisten Dateiverwaltungssystemen von

Kleinrechnern als einzige angeboten wird. Eine Abbildung von Zugriffen auf S-Dateien bedeutet somit die Ersetzung des Operators der S-Ebene durch den entsprechenden der L-Ebene. Da die Länge von S-Blöcken fest ist und zweckmäßigerweise der der lokalen Datei entspricht, ergibt die Blocknummer der S-Datei, auf die zugegriffen werden soll, addiert zu der Anfangsadresse der S-Datei innerhalb der lokalen Datei (siehe Kap. 3.2.1) die Blocknummer, auf die real zugegriffen wird.

Analog bewirken die Operatoren OPEN/CLOSE für S-Dateien eine Abbildung auf die entsprechenden Operatoren der lokalen Datei, sofern momentan keine andere S-Datei, die auf die gleiche lokale Datei abgebildet ist, in Bearbeitung ist; ansonsten bewirken diese Operatoren nur Einträge in Kontrollblöcke, die zur Verwaltung der Dateien benötigt werden.

4. Transaktionsverwaltung

In Kap. 3 wurde dargestellt, auf welche Weise eine Transaktion in das System gelangt. An dieser Stelle wird weiter darauf eingegangen, welche globalen funktionellen Komponenten in die weitere Abarbeitung einer Transaktion involviert sind und wie deren Verwaltung realisiert wird.

4.1. In die Transaktionsverwaltung involvierte Komponenten

Die globalen funktionellen Komponenten, die eine wichtige Rolle für die Bearbeitung einer Transaktion spielen, sind - wie aus Bild 2.2. ersichtlich:

- die Zugriffsverwaltung für logische Dateien
- die Katalogverwaltung für logische Dateien und
- die Zugriffskontrolle

4.1.1. Die Zugriffsverwaltung für logische Dateien

Die Zugriffsverwaltung für logische Dateien - im folgenden kurz ZU genannt - ist für die Transaktionsverwaltung von zentraler Bedeutung, weshalb ihre Aufgaben und die Pufferverwaltung hier näher betrachtet werden sollen.

4.1.1.1. Aufgaben der Zugriffsverwaltung

Die ZU übernimmt die semantische Analyse der vom Benutzer spezifizierten Operatoren, ihre Interpretation und Transformation in Operationen der S-Ebene sowie die Weiterleitung der S-Operatoren an die lokalen Funktionseinheiten "Anpassung" (ANP). Die Transformation wird gesteuert durch die in Dateibeschreibungsblöcken enthaltene Information der referenzierten logischen Dateien. Die Dateibeschreibungsblöcke ihrerseits entstammen dem Katalog für logische Dateien und werden von der Komponente "Katalogverwaltung für logische Dateien" bei Öffnung einer logischen Datei bereitgestellt

(siehe Kap. 4.1.2) /BCD78/.

Zugriffe auf logische Dateien sind nur innerhalb von Transaktionen zugelassen. Bei der Formulierung von Transaktionen sind strikte Regeln zu beachten, die von der ZU überprüft werden (siehe Kap. 2.u.3.). Vor der eigentlichen Durchführung des Zugriffs, d.h. vor dem Absetzen der Anweisungen an die Anpassung, ist zu überprüfen, ob die referenzierten Datenbestände entsprechend gesperrt sind. Die Sequenz der erforderlichen Sperr- und Entsperroperationen muß dabei dem Zwei-Phasen-Sperrprotokoll (siehe Kap. 2) /Esw76/ entsprechen und somit einer Wachstums- und Schrumpfungsphase bezüglich der Anzahl der einer Transaktion zugeordneten Datenobjekte folgen. Die Sperrung von Datenobjekten, ebenso wie ihre Freigabe, geschieht über die Komponente "Zugriffskontrolle" und wird dort für nachfolgende Prüfungen in Kontrollblöcken vermerkt.

Alle transaktionsbezogenen Operationen werden von der ZU überwacht, die damit für die eigentliche Transaktionsverwaltung verantwortlich ist /HBD78/.

4.1.1.2. Pufferorganisation

Zur Pufferverwaltung wird in DISCO ein spezielles Seitenverwaltungssystem eingesetzt, welches auch Überlaufsituationen beherrscht.

Das Konzept des Seitenpuffers wurde eingeführt, um die Verwaltung des Hauptspeicherbereichs innerhalb der ZU zu unterstützen /Ker79b, BCD78/. Der gesamte Transfer der Daten von bzw. zu den auf externen Speichereinheiten befindlichen S-Dateien läuft über den Seitenpuffer; er stellt also das Bindeglied zwischen Anpassung und ZU dar.

Jeder Funktionseinheit der Zugriffsverwaltung für logische Dateien wird genau ein Seitenpuffer zugeordnet. Ein Seitenpuffer besteht aus einer vordefinierten Anzahl von Seiten fester Länge, die jeweils genau einen Block aufnehmen können.

Jeder Datentransfer von und zum Hintergrundspeicher erfordert eine dieser Seiten als Puffer. Zu einem Zeitpunkt kann ein Seitenpuffer

Blöcke mehrerer S-Dateien beinhalten, die ihrerseits an unterschiedlichen Knoten des Rechnernetzes residieren können. Zur Adressierung dieser Blöcke findet eine Adressentransformation statt, die Blockadressen auf Seitenadressen umrechnet.

Stellvertretend für Blöcke werden Seiten manipuliert. Da diese Seiten Kopien von Blöcken der Datenbasis repräsentieren, muß der Inhalt einer Seite nach Änderungen wieder zurück in die Datenbasis geschrieben werden. Dies geschieht durch Überschreiben des entsprechenden Blockes bei Transaktionsende.

Zur Behandlung von Überlaufsituationen eines Seitenpuffers wird das Konzept einer temporären Datei (TD) eingeführt. Jeder ZU wird genau eine temporäre Datei zugeordnet. Eine temporäre Datei ist ebenfalls blockorientiert. Sie wird an dem schnellsten jeweils zur Verfügung stehenden Peripheriegerät abgelegt.

Wenn der Seitenpuffer voll ist und eine weitere Seite für einen zusätzlichen Block benötigt wird, wird der Inhalt einer der belegten Seiten auf der temporären Datei abgelegt. Ein geeignetes Seitenauswahlverfahren /Ker79b/ ermittelt die entsprechende auszulagernde Seite.

Bei der Konzeption des Seitenauswahlverfahrens wird der Aufwand berücksichtigt, der notwendig ist, um eine überlagerte Seite wieder im Seitenpuffer zu rekonstruieren. Im DISCO-System sind Zugriffe auf Daten im gesamten Rechnernetz möglich. Ein Zugriff auf einen Datenblock (S-Block), der von einem entfernten Rechner beschafft werden muß, gilt dabei teurer als ein lokaler Zugriff.
Aus diesem Grunde wurde eine Klasseneinteilung der Seiten vorgenommen, die in drei Typen von Seiten resultiert:

S1-Seiten: enthalten Blöcke, die geändert worden sind.

S2-Seiten: enthalten Blöcke, die nur gelesen werden und die ohne abgesetzten Zugriff aus der Datenbasis beschafft werden können.

S3-Seiten: enthalten Blöcke, die nur gelesen werden und die nur mit

abgesetztem Zugriff aus der Datenbasis beschafft werden können.

Wenn unter "<" die Relation mit der Bedeutung "Aufwand zur Rekonstruktion der links von "<" stehenden Seite kleiner als der rechts von "<" stehenden Seite" zu verstehen ist, so können mittels dieser Relation die Klassen folgendermaßen geordnet werden:

S1-Seiten < S2-Seiten < S3-Seiten

Deshalb werden vom Seitentauschalgorithmus neben anderen Randbedingungen vor allem S1-Seiten bevorzugt ausgelagert, da diese am Ende einer Transaktion auf jeden Fall auf die temporäre Datei geschrieben werden (siehe auch Kap. 4.2).

4.1.2. Katalogverwaltung für logische Dateien

Die Aufgaben der Katalogverwaltung für logische Dateien - im folgenden kurz KAT genannt - lassen sich aus operationaler Sicht in folgende Teilbereiche untergliedern /Bre78/ :

- Realisierung netzweiter Namensräume für logische Dateien, Gewährleistung netzweit eindeutiger Dateinamen für logische Dateien
- Bereitstellen der Transformationsstrukturen, die es der ZU ermöglichen, auf Sätze logischer Dateien zuzugreifen
- Berechnung optimaler Abbildungen für Dateischemata, Steuerung der Verteilung von Datenobjekten
- Erzeugen und Beseitigen von Dateien
- Realisierung einer Schnittstelle für den Datenbasisadministrator (DBA)
- Gewährleistung der Integrität von Dateibeschreibungen

Die OPEN-FILE-Liste (OFL) dient dabei als Hilfsmittel zur Sicherung der Konsistenz von Dateibeschreibungsblöcken (DBB). Der DBB einer

logischen Datei kann an mehreren ZUs zugleich angelegt sein. Bei allen Änderungen des DBB muß für die Konsistenz von allen bei den ZUs abgelegten DBBs als auch des entsprechenden Katalogeintrags selbst gesorgt werden. Die OFL hält hierzu für jede im System eröffnete logische Datei fest, welche ZUs diese logische Datei eröffnet haben. Die OFL wird redundant bei jeder Funktionseinheit der Katalogverwaltung geführt.
Wird bei einer ZU eine logische Datei eröffnet, übermittelt die Katalogverwaltung dieser ZU die entsprechenden Katalogeinträge. Mehrere ZUs können dabei einer KAT als Auftraggeber zugeordnet sein. Der Aufbau der Katalogeinträge wird in Kap. 5 noch näher beschrieben.

4.1.3. Zugriffskontrolle

Die Aufgaben der Zugriffskontrolle - im folgenden kurz KO genannt - ist die Sicherung der operationalen Integrität von Primär- und Sekundärdaten /Mai78/, /See79/. Bevor Objekte der Datenbasis bearbeitet werden, holt die ZU bei der KO die Zustimmung dafür ein. Die KO ist verantwortlich für das Sperren und Freigeben von Datenbeständen.

Um die Vorteile eines verteilten Systems auszunutzen, muß ein hoher Grad an Parallelität bei der Abarbeitung von Benutzeranfragen angestrebt werden. Dies wird durch eine dezentrale Kontrollstruktur /Dro77/ erreicht. Das bedeutet, entsprechend der Konfiguration des Systems und des Benutzerverhaltens können mehrere Inkarnationen der Komponente Zugriffskontrolle auf verschiedenen Rechnern vorhanden sein.

Die Zuordnung von Daten zu KOs erfolgt implizit über eine Zuordnung von Rechnern zu KOs. Die gesamte Datenbasis ist aufgeteilt in disjunkte Kontrollbereiche - eine Teilmenge von Rechnern des Systems. Jede KO ist für die Sperrungen in ihrem Kontrollbereich verantwortlich.
Bild 4.1 zeigt eine mögliche Zuordnung von KOs zu Kontrollbereichen.

Jede ZU ist genau einer KO zugeordnet, der sie die Sperr- und Freigabeanweisungen für Datenobjekte weiterleitet. Werden durch eine

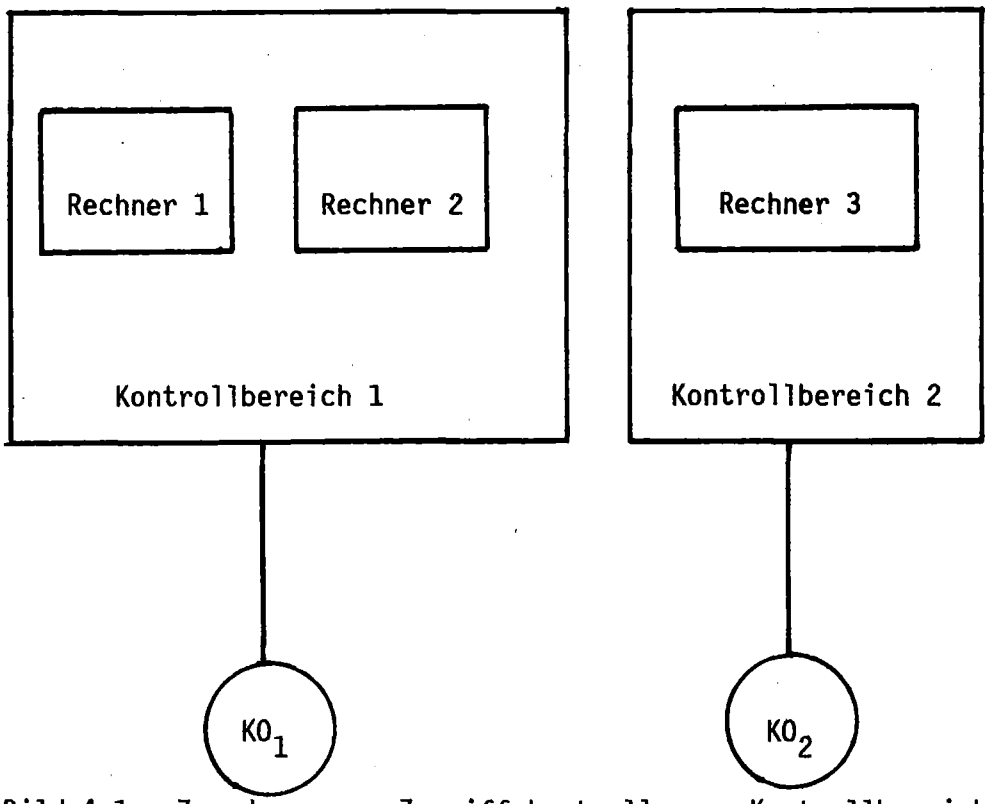


Bild 4.1: Zuordnung von Zugriffskontrollen zu Kontrollbereichen

Anweisung Objekte angesprochen. die von anderen KOs verwaltet werden, so setzt diese KO Folgeaufträge ab an die jeweils zuständigen KOs und überwacht auch das Eintreffen der Meldungen, die die Ausführung der Aufträge anzeigen. Erst wenn für jeden Folgeauftrag eine Erfolgsmeldung vorhanden ist, wird der Transaktion die Fortsetzung ihrer Aufgaben erlaubt.

Die Semantik angeforderter Objekte, d.h. ob es sich um verschiedene Dateien oder um Kopien einer redundant vorhandenen Datei handelt, die für Änderungen alle gemeinsam exklusiv gesperrt werden müssen, ist bezüglich der Zugriffskontrolle transparent, d.h. eine KO sperrt Namen! Daraus ergibt sich, daß mit den Mechanismen der Zugriffskontrolle nicht nur der Zugriff auf Daten systemweit synchronisiert werden kann, sondern allgemein die exklusive Benutzung von Betriebsmittel ermöglicht wird.

Abgesehen von "schmutzigem Lesen" - d.h. ein Datenobjekt wird vor dem Lesen nicht gesperrt - gibt es im DISCO-System zwei Sperrungsarten:

- shared: Nur das Lesen ist erlaubt. Paralleler Zugriff mehrerer Transaktionen ist möglich

- exclusive: Lesen und Schreiben ist erlaubt. Zu einem Zeitpunkt hat jeweils nur die sperrende Transaktion Zugriff auf das Datenobjekt. Andere Transaktionen werden blockiert.

Die Verträglichkeit der Sperrarten zeigt Bild 4.2

		aktuelle Sperrart: shared	aktuelle Sperrart: exclusive
gewünschte Sperrart	shared	ja	nein
	exclusive	nein	nein

Bild 4.2: Kompatibilität der Sperrarten

Da im DISCO-System im Rahmen einer Transaktion das sukzessive Sperren von Datenobjekten erlaubt ist, besteht die Möglichkeit eines Deadlocks. Das Auffinden und Auflösen eines solchen Deadlocks ist ebenfalls Aufgabe der Zugriffskontrolle und wird in Kap. 4.3 noch näher erläutert. Um die Blockierungen im Gesamtsystem gering zu halten, ist durch ein Protokoll gewährleistet, daß eine KO eventuelle Folgeaufträge erst an die entsprechenden anderen KOs abschicken darf, wenn die bei ihr zu sperrenden Objekte erfolgreich gesperrt wurden.

4.2. Ablauf einer Transaktion

Eine Transaktion betrachten wir als die Einheit der Konsistenz. Die Auswirkungen auf die Datenbasis derjenigen Operatoren, die durch BEGIN_TRANS und END_TRANS eingeklammert sind, werden entweder als

Ganzes oder - im Fehlerfall - gar nicht wirksam.
Welche Operatoren im Rahmen einer Transaktion abgewickelt werden, wurde in Kap. 3 dargestellt.

Bild 4.3 zeigt den zeitlichen Ablauf einer Transaktion.

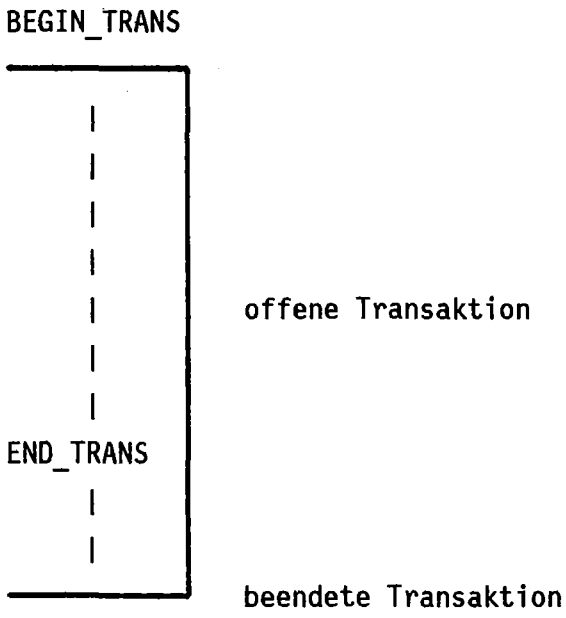


Bild 4.3: Ablauf einer Transaktion

Wir bezeichnen eine Transaktion als offen, sobald sie durch ein BEGIN_TRANS begonnen wurde, und als beendet, wenn alle in ihr enthaltenen Operatoren in der Datenbasis wirksam geworden sind und auch die Führung von Verwaltungsinformation abgeschlossen ist.

Die Zeitspanne zwischen Absetzen von END_TRANS und der Beendigung der Transaktion ist abhängig von der Anzahl der vorgenommenen Änderungen und davon, wieviele asynchrone Operatoren der Benutzer abgesetzt hat.

Es sei angenommen, ein Benutzer habe Daten gelesen und auch Änderungen in der Datenbasis vorgenommen. Sämtliche Aktionen werden aber nicht sofort in der Datenbasis wirksam, sondern erst - wie in

Kap 4.1 erwähnt - im Seitenpuffer. Änderungen in der Datenbasis selbst werden erst nach END_TRANS vorgenommen, also zeitlich verzögert. In DISCO wird also das Prinzip des deferred update angewandt.

Dieses Vorgehen des deferred update bedingt, daß

1. alle von einer Transaktion zum Zwecke der Änderung exklusiv gesperrten Objekte erst nach Beendigung der Transaktion anderen Transaktionen zugänglich sind. Dadurch wird gewährleistet, daß keine Abhängigkeiten zwischen Transaktionen auftreten können, die sonst beim Zurücksetzen der Transaktion zu berücksichtigen wären.

2. die Ausführung von UNLOCK-Operatoren bezüglich exklusiv gesperrter Objekte ebenfalls bis zum Transaktionsende verzögert wird.

3. das Zurücksetzen einzelner noch nicht beendeter Transaktionen sehr leicht durchführbar ist. Die Datenbasis bleibt bis zur Verarbeitung des END_TRANS Operators im (konsistenten) Zustand wie vor Beginn der Transaktion; es müssen also beim Zurücksetzen keine Datenbasisänderungen rückgängig gemacht werden.

Nach der Ausführung z.B. eines WRITE-Operators befindet sich die entsprechende Seite im Seitenpuffer. Zu Beginn der Abarbeitung des END_TRANS-Operators werden alle diejenigen Seiten, auf denen Änderungen vorgenommen wurden, auf der temporären Datei abgelegt, sofern dies nicht schon durch aufgetretene Überlaufsituationen geschehen ist. Sollten Systemfehler auftreten, können die Änderungen zu einem anderen Zeitpunkt nachgeholt werden (siehe auch Kap. 6).

Bei der weiteren Abarbeitung von END_TRANS werden die geänderten Seiten in die Datenbasis eingespielt, Zustandsinformationen aktualisiert und die Katalogeinträge - nach eventuellen Änderungen - wieder in einen konsistenten Zustand gebracht.

Ausgehend von Bild 4.3 wird der Ablauf einer Transaktion in Bild 4.4

etwas detaillierter dargestellt.

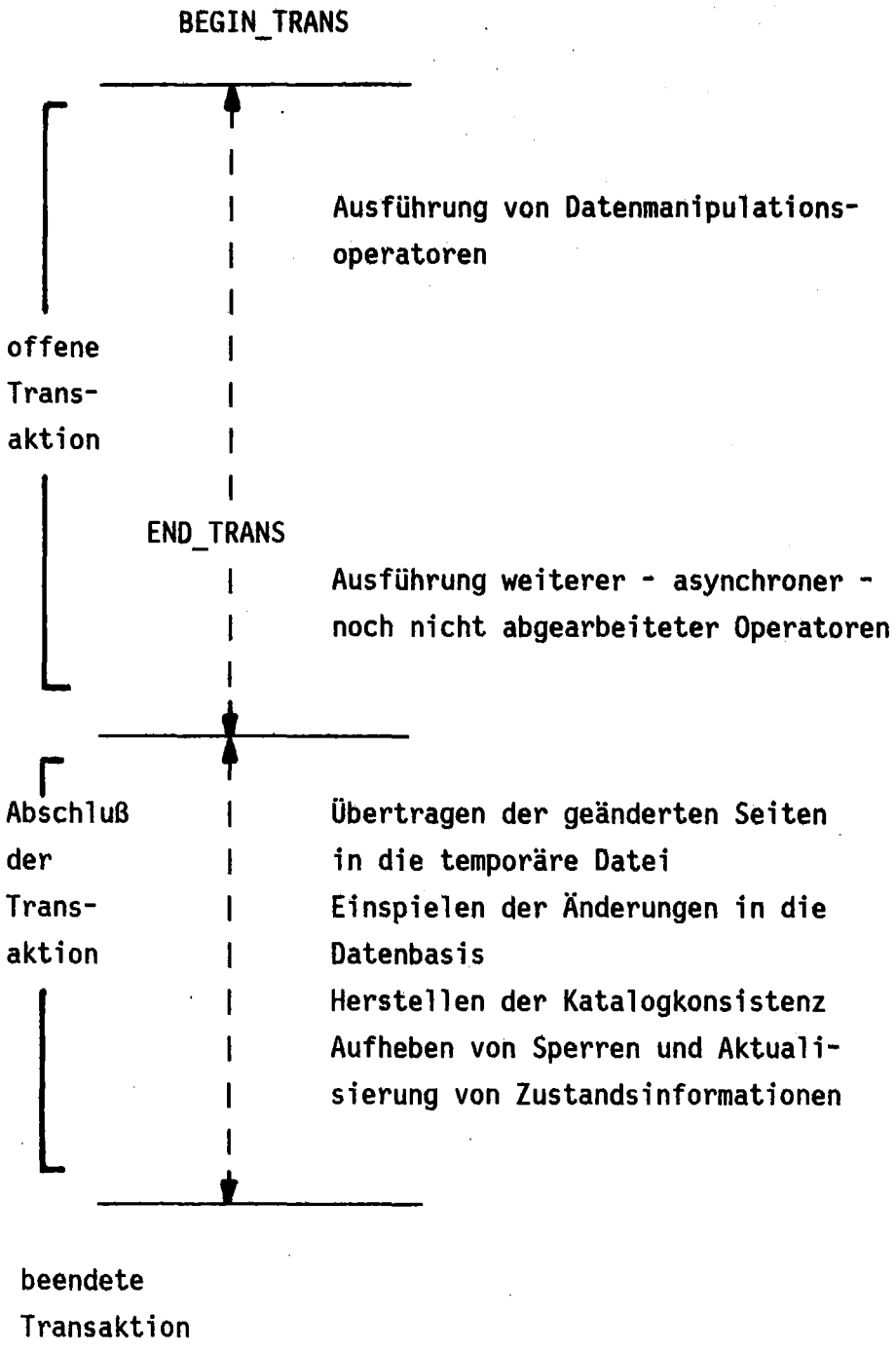


Bild 4.4: Ablauf einer Transaktion

4.3. Parallelität von Transaktionen

Um die Vorteile eines verteilten Systems auszunutzen zu können, muß ein hoher Grad an Parallelität bei der Bearbeitung von Benutzeraufträgen angestrebt werden. Da unter Umständen mehrere Transaktionen in Konkurrenz zueinander stehen, d.h. auf den gleichen Datenbeständen arbeiten wollen, müssen sie logisch serialisiert werden. Das geschieht dadurch, daß die Transaktionen die Datenobjekte mit den in Kap. 4.1.3 genannten Sperrarten für andere Transaktionen sperren. Versucht eine Transaktion, ein Datenobjekt, das bereits gesperrt ist, mit einer Sperrart zu sperren, die mit der vorhandenen unverträglich ist, so muß sie ihre Abarbeitung unterbrechen, und wir nennen sie blockiert.

Eine Möglichkeit für eine nur geringe gegenseitige Behinderung der Transaktionen wäre ein hoher Grad an Granularität, eine sehr feine Unterteilung der Sperrobjekte. Dies würde aber andererseits einen großen Overhead an Verwaltung von Sperrinformationen bedeuten. In DISCO können Dateien und Sätze gesperrt werden, wobei die Satzsperren auf Sperren von O-Dateien abgebildet werden.

Um den Benutzer nicht unnötigen Restriktionen auszusetzen, müssen nicht alle Datenobjekte, mit denen gearbeitet werden soll, gleich zu Beginn einer Transaktion gesperrt werden. Es ist sukzessives Sperren erlaubt, die Datenobjekte werden also erst dann mit Sperren belegt, wenn sie benötigt werden. In DISCO besteht die Möglichkeit, Daten redundant zu speichern. Diese Redundanz gewährleistet effiziente Parallelarbeit (was das Lesen von Daten betrifft) und Verfügbarkeit.

Wenn Daten verändert werden sollen, also exklusiv gesperrt werden, kann sich die redundante Verteilung der Daten zusammen mit der Möglichkeit des sukzessiven Sperrens als Nachteil herausstellen: Es besteht die Gefahr eines Deadlocks.

4.3.1. Deadlockbehandlung

Deadlocks können nur dann auftreten, wenn folgende fünf Bedingungen zugleich erfüllt sind:

1. Datenobjekte werden für Transaktionen gesperrt

2. zwei oder mehrere Transaktionen dürfen sich gleichzeitig um die Sperrung zusätzlicher Datenobjekte bewerben

3. eine Transaktion darf die Sperrung zusätzlicher Datenobjekte beantragen, obwohl sie bereits andere gesperrt hält

4. einer Transaktion dürfen von ihr gesperrte Datenobjekte nicht entzogen werden, bevor sie selbst diese nicht freigibt

5. es existiert eine zyklische Kette von Transaktionen derart, daß jede Transaktion Datenobjekte gesperrt hält, um die sich ihr Vorgänger in der Kette bewirbt.

Bild 4.5 zeigt in einem Blockierungsgraphen die möglichen Wartebeziehungen zwischen vier Transaktionen.

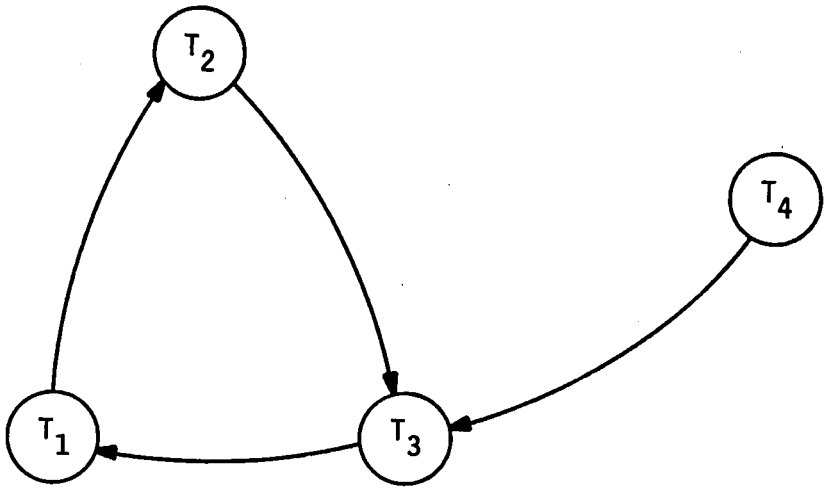


Bild 4.5: Wartebeziehungen zwischen vier Transaktionen

Bedingt durch die verteilte Kontrollstruktur der Zugriffskontrolle ergeben sich 2 verschiedene Deadlocksituationen:

- lokaler Deadlock

- globaler Deadlock

Lokaler Deadlock bedeutet, daß sich Transaktionen beim Zugriff auf Objekte, die alle von einer KO verwaltet werden, verklemmt haben, während in einen globalen Deadlock mindestens 2 Kontrolleinheiten involviert sind.

Die Deadlockbehandlung läßt sich in drei Bereiche unterteilen /Mai78/:

1. Erkennen des Vorhandenseins eines Deadlocks

2. Auswählen eines "Opfers", d.h. einer Transaktion, die zurückgesetzt werden soll, mit anschließender Rücksetzung dieser Transaktion

3. Zuweisung der freigegebenen Objekte an eine oder andere bisher blockierte Transaktionen.

Ein mögliches Verfahren zur Erkennung eines Deadlocks besteht darin, die Wartebeziehungen unter den blockierten Transaktionen in einem Blockierungsgraphen (WAIT-FOR-GRAPH) darzustellen.

Hinreichende und notwendige Bedingung für die Existenz eines Deadlocks ist das Vorhandensein eines Zyklus in diesem Graphen. Die Wartebeziehungen werden in einer Adjazenzmatrix dargestellt, in der mit Hilfe einer Wegmatrix ein Zyklus ermittelt wird. Das Vorgehen bei der Deadlockerkennung ist bei lokalem und globalem Deadlock gleich. Beide unterscheiden sich allerdings erheblich voneinander im Aufwand, da bei einem globalen Deadlock die Blockierungsinformation aller KOs über das Nachrichtentransportsystem eingeholt und in einem globalen Blockierungsgraphen zueinander in Beziehung gesetzt werden muß.

Die Deadlockanalyse wird periodisch gestartet, allerdings nur, wenn sich seit der letzten Analyse die Wartebeziehungen verändert haben.

Bevor jedoch die eigentlichen Aktivitäten der Deadlockbehandlung in Angriff genommen werden, wird erst geprüft, ob die Möglichkeit eines lokalen oder globalen Deadlocks besteht.

Ein lokaler Deadlock ist möglich, wenn mindesten zwei Transaktionen innerhalb eines Kontrollbereiches blockiert sind und beide Transaktionen schon Objekte gesperrt halten.

Die Möglichkeit für einen globalen Deadlock besteht, wenn in einer KO ein Folgeauftrag einer anderen KO blockiert wird, und

- die blockierende Transaktion selbst Folgeaufträge an andere KOs geschickt und noch keine Bestätigung für die erfolgreiche Sperrung erhalten hat oder

- die blockierende Transaktion selbst von einer anderen KO weitergeleitet wurde.

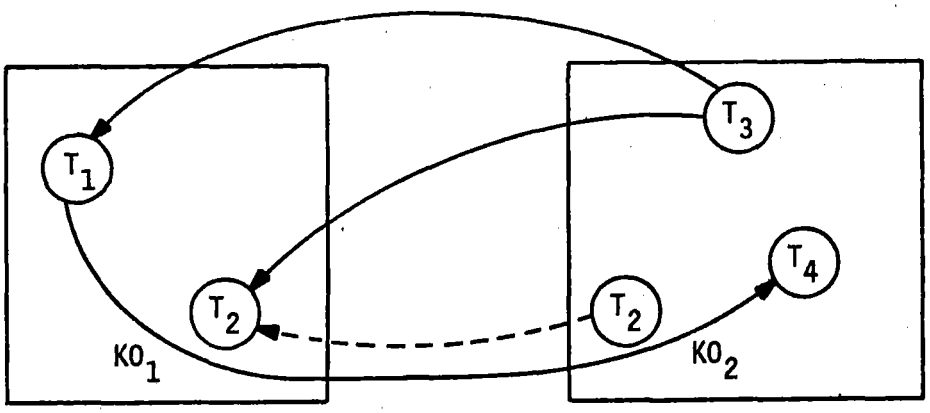
Bild 4.6 zeigt beide Möglichkeiten eines globalen Deadlocks aus der Sicht von KO_1. Transaktion 3 möchte Objekte sperren, die T_1 und T_2 (aus KO_2 kommend) schon gesperrt halten. Aus der Sicht von KO_1, die nicht die internen Wartebeziehungen in KO_2 kennt, besteht die Möglichkeit eines Deadlocks zwischen den Transaktionen 1,3 und 4 und/oder zwischen den Transaktionen 2 und 3.

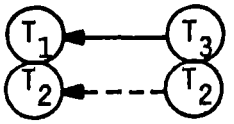
Wenn ein globaler Deadlock möglich erscheint, wird eine globale Deadlockanalyse initiiert.

Um zu gewährleisten, daß die im globalen Blockierungsgraphen dargestellten Wartebeziehungen die augenblicklichen realen Wartebeziehungen des Gesamtsystems widerspiegeln, ist es notwendig, daß jede KO der die Deadlockanalyse initiierenden KO ihre internen Wartebeziehungen übermittelt und so lange wartet, bis das Ergebnis der Analyse zurückgegeben wird. Dies hat den großen Nachteil, daß von der Initiierung der Analyse bis zu ihrem Ende kein Auftrag mehr im Gesamtsystem bearbeitet werden kann.

Um diese Ineffizienz zu vermeiden, wird in DISCO ein anderer Weg eingeschlagen /Mai80/:

Wie oben beschrieben, wird periodisch bei jeder KO überprüft, ob die Möglichkeit eines - hier globalen - Deadlocks besteht. Sollte eine KO





: T_1 blockiert T_3

: T_2 wurde von einer anderen KO weitergeleitet

Bild 4.6: Wartebeziehungen zwischen Transaktionen verschiedener Kontrollbereiche

feststellen, daß ein Deadlock bestehen könnte, initiiert sie eine globale Analyse, indem sie von allen anderen KOs die internen Wartebeziehungen anfordert.

Bis sie von allen KOs diese Information bekommen hat, steht sie jedoch nicht im "Leerlauf", sondern verarbeitet weiterhin eingehende Aufträge. Auch eine KO, die ihre internen Wartebeziehungen abgeschickt hat, muß nicht bis zum Ergebnis der Analyse warten, sondern kann gleich nach Absenden der Informationen weiterhin bei ihr eingehende Aufträge bearbeiten. Sie vermerkt lediglich, daß eine globale Deadlockanalyse initiiert ist, um zu vermeiden, daß sie selbst eine weitere Analyse startet.

Bild 4.7 zeigt den zeitlichen Ablauf zwischen drei KOs.

Im Zeitpunkt t_o stellt KO_1 fest, daß ein globaler Deadlock existieren könnte. Nachdem sie die Anforderungen der internen Wartebeziehungen der anderen KOs (A) abgeschickt hat, bearbeitet sie weiterhin bei ihr eingehende Aufträge. Im Zeitpunkt t_2 hat sie die Informationen (I) aller anderen KOs erhalten und untersucht von t_2 an den zusammengestellten globalen Blockierungsgraphen. Nach der Analyse schickt sie das Ergebnis der Untersuchung (E) an die anderen KOs, so

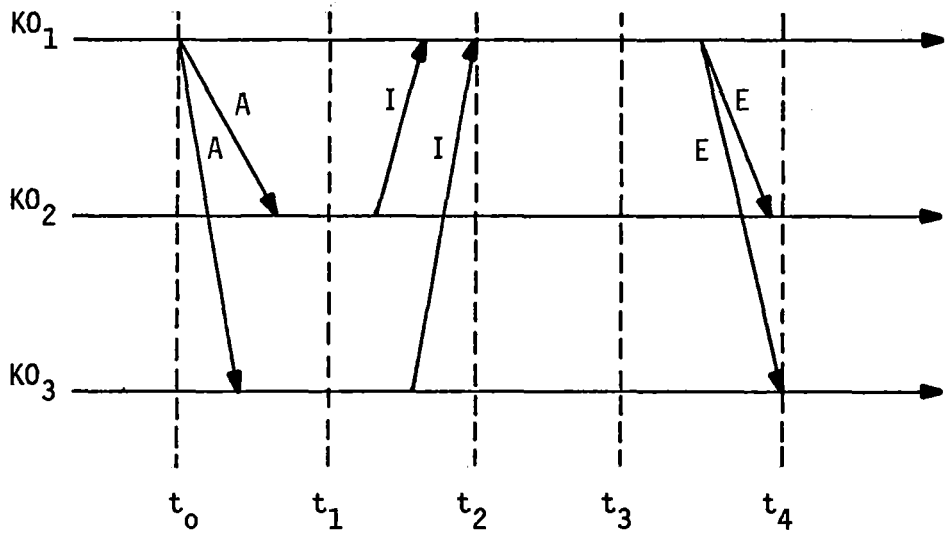


Bild 4.7: Zeitlicher Ablauf der Deadlockanalyse bei drei KOs

daß in t_4 die globale Deadlockanalyse beendet ist.

Diese Vorgehensweise hat den Nachteil, daß der Blockierungsgraph, der in t_2 untersucht wird, nicht die globalen Wartebeziehungen in einem festen Zeitpunkt t_x widerspiegelt, sondern aus internen Wartebeziehungen zusammengestellt ist, die in verschiedenen Zeitpunkten ermittelt wurden. Es besteht deshalb die Möglichkeit, daß nach Übersenden der internen Wartebeziehungen durch das Bearbeiten weiterer Aufträge neue Blockierungen entstehen, die im globalen Blockierungsgraph dann nicht enthalten sind.

Da die Wahrscheinlichkeit eines Deadlocks aber für gering gehalten wird, muß in Kauf genommen werden, daß neu auftretende Blockierungen erst bei der nächsten Analyse untersucht werden. Ausschlaggebend für dieses Vorgehen ist jedoch die Effizienzsteigerung des Gesamtsystems, da trotz globaler Deadlockanalyse (n-1) KOs für weitere Aufträge verfügbar sind.

4.3.2. Rücksetzen einer Transaktion

Um einen bestehenden Deadlock aufzulösen, muß mindestens eine der darin verwickelten Transaktionen zurückgesetzt werden, und die

dadurch freiwerdenden Objekte müssen an blockierte Transaktionen vergeben werden.

Das Problem besteht darin, eine Transaktion als Opfer zu wählen, bei der die durch die Rücksetzung entstehenden Kosten minimiert werden.

Diese Kosten setzen sich zusammen aus den Kosten für die Rücksetzung selbst und dem Rechenaufwand, der bisher zur Sperrung der Objekte der rückzusetzenden Transaktion nötig war, wobei dieser i.a. bei Wiederholung der Transaktion noch einmal erbracht werden muß. Zusätzlich können die von der Transaktion bisher durchgeführten Operationen in Betracht gezogen werden.

Als Maß für die Kosten kann die Zahl der gesperrten Objekte einer Transaktion genommen werden. Eine kleine Zahl bedeutete dann, daß die Kosten zur Rücksetzung dieser Transaktion geringer sind als bei Transaktionen mit einer größeren Zahl gesperrter Objekte.

Die Kosten der Rücksetzung selbst sind im DISCO-System relativ gering. Von einer Nachricht an den Benutzer abgesehen, sind nur die in die Bearbeitung der rückzusetzenden Transaktion involvierten KOs und die auftraggebende ZU von der Rücksetzung betroffen.

In den KOs sind nur sämtliche diese Transaktion betreffenden Informationsblöcke zu löschen. Die ZU muß Änderungen an den Dateibeschreibungsblöcken, die die rückzusetzende Transaktion gegebenenfalls verursacht hat, rückgängig machen.

Hier kommt auch das Konzept der temporären Datei zum Tragen. Da sämtliche Änderungen, die die Transaktion auf der Datenbasis vornehmen wollte, nur im Seitenpuffer und in der temporären Datei wirksam wurden, ist die Datenbasis selbst von einer Rücksetzung gar nicht betroffen.
Die Eintragungen im Seitenpuffer bzw. in der temporären Datei können einfach "vergessen" werden.

5. Schemaverwaltung

Die Schemaverwaltung umfaßt sowohl die Verwaltung von Benennungen und die Herstellung des Bezuges zu benannten Objekten als auch die Verwaltung der Schemadefinitionen für logische Dateien. Der Katalog ist das Hilfsmittel zur Realisierung dieser Aufgabe. Für DISCO faßt das Katalogsystem, integriert zu der funktionellen Komponente KAT, "Verwaltung von Katalogen logischer Dateien" (vgl. /HBD78/), die Kataloge für folgende Namensräume zusammen:

- logische Dateiebene
- P-Ebene
- O-Ebene

Die Zusammenfassung bringt den großen Vorteil einer effizienten Implementierung des Architekturmodells für logische Dateien, ohne wesentliche Einschränkungen für den Umgang mit Objekten der verschiedenen Namensräume machen zu müssen.

5.1. Struktur der Schemata der logischen Dateien

Die logische Dateiebene bietet A- und B-Dateien an. Unter Berücksichtigung der Architektur der logischen Dateien und der Operatoren der verschiedenen Ebenen ergeben sich Strukturen für A- und B-Dateien, wie in den Bildern 5.1 und 5.2 gezeigt.

Für A-Dateien gelten die in Kap 2. genannten Einschränkungen bzgl. der Verteilbarkeit der zugrundeliegenden B-Datei.
Die O-Dateien bilden die Abgrenzung gegenüber der S-Dateiebene der Anpassung: Die Anpassung legt die lokale Abbildung einer O-Datei auf Geräte, Sektoren usw. fest, KAT die Ablage einer O-Datei auf Rechner. Hieraus ergibt sich, daß aufgrund der grundlegenden Bedeutung der KAT als Basis für die globale, d.h. netzweite Handhabung von Objekten dieser funktionellen Komponente eine Reihe von Eigenschaften genereller Natur zu fordern sind:

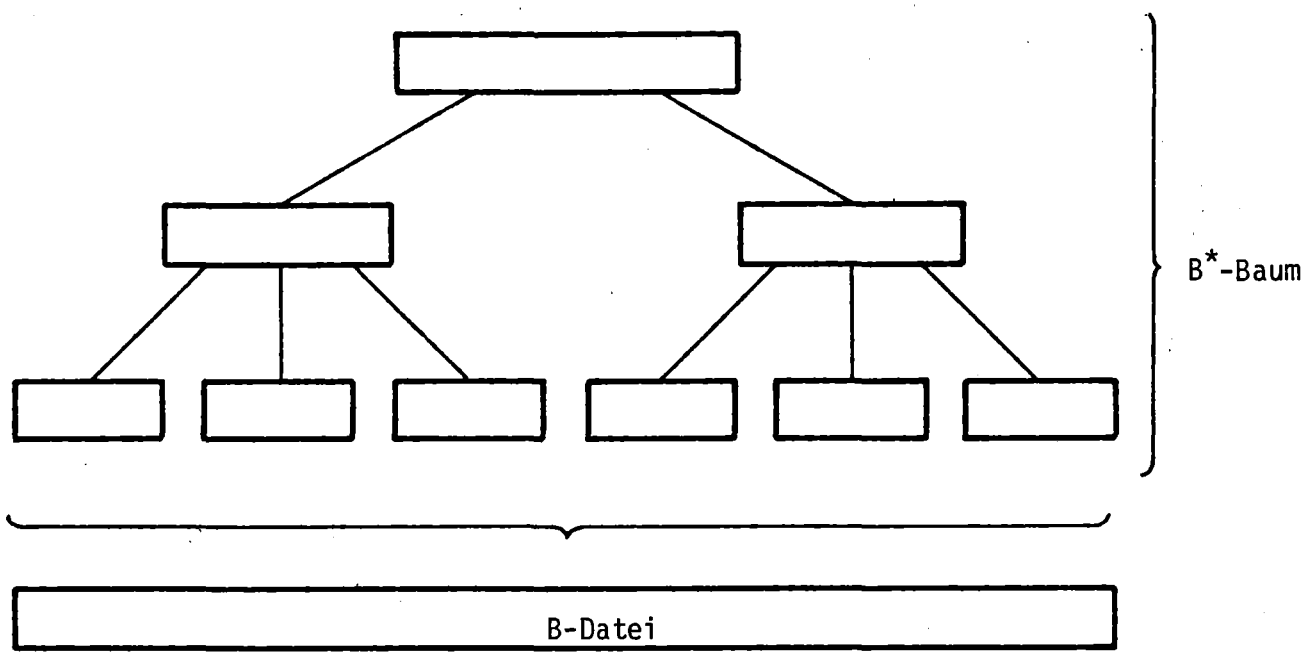


Bild 5.1: Struktur einer A-Datei

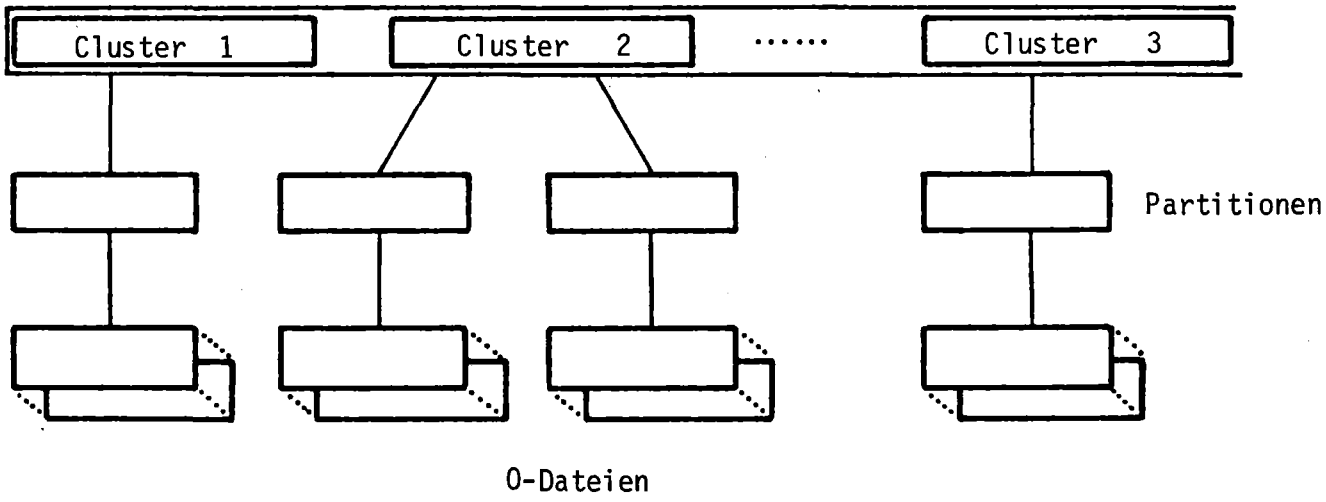


Bild 5.2: Struktur von B-Dateien

- Rechnerunabhängigkeit:
  Dies gilt sowohl für den Katalog als auch für die funktionelle Komponente KAT

- Berücksichtigung von Kleinrechnersystemen

- anpaßbare Leistungsfähigkeit für unterschiedliche Anwendungen

- Brauchbarkeit für unterschiedliche Netzkonfigurationen

- hohe Verfügbarkeit

- kompakte Schnittstellen

Diese Randbedingungen sind bei der Verwaltung der Schemata, die sich aus den Benutzerdefinitionen (Operator DEF, Anhang A) und den Strukturen gemäß Bild 5.1 und Bild 5.2 ergeben, einzuhalten.

Entsprechend den Forderungen nach Rechnerunabhängigkeit, Berücksichtigung von Kleinrechnersystemen und kompakten Schnittstellen, läßt sich für die Definition einer B-Datei (auf der auch eine A-Datei aufbaut) ein Schema gemäß Bild 5.3 ableiten.

Das Schema enthält alle Informationen, die der Benutzer mittels des Operators DEF angibt. Die den Clustern zugeordneten Benutzergruppennummern werden hier den die Cluster konstituierenden Partitionen zugeordnet. Partitionsnamen ergeben sich aus B-Dateiname und Index, Inkarnationen können unter Hinzunahme eines weiteren Index' eindeutig benannt werden. Damit können sämtliche Funktionen, auch DBA-Funktionen, (vgl. /Bre78/), parametrisiert werden. Aus Gründen des Kleinrechneraspekts müssen bei der Implementierung flexible Feldobergrenzen durch Konstanten ersetzt werden.

Für A-Dateien ergibt sich aufgrund der Implementierung mittels B^*-Bäumen das in Bild 5.4 gezeigte Schema.

Bei A-Dateien wird die Baum-Struktur auf eine eigene Partition abgebildet; ebenso die A-Sätze variabler Länge. Beide Partitionen

```
02  B-Dateiname          string
02  Füllgradanzeiger     int
02  Satzlänge            int
02  Paßworte

    03  Lesen     /1:6/ char
    03  Schreiben /1:6/ char

02  Sicherheitsklasse    int
02  Sätze pro Partition  int

02  Abbildung auf Partition /1:flex/[1)]
    03  erzeugt          bool
    03  Satzbelegung     /1:Sätze pro Partition/ bool
    03  O-Datei /1:flex/[2)]
        04  S-Datei      string
        04  Rechnernummer int
    03  Benutzergruppen  /1:n/ int
```

Bild 5.3: Schema einer B-Dateibeschreibung

können als Spezialfall einer B-Datei aufgefaßt werden, bei der die Satzlänge gleich der Standard-Blocklänge ist.

Die Aufgabe der Realisierung netzweiter Namensräume reduziert sich nach diesen Schemata auf die Gewährleistung der Eindeutigkeit logischer Dateinamen: Die Bezüge der logischen Datei zu den Objekten der verschiedenen Ebenen sind dann durch die Schemata und die Benennungsregeln gegeben.

---

1) Jedes Reihungselement definiert eine Partition. Die Anzahl der Reihungselemente ist gleich der die B-Datei konstituierenden Partitionen.
2) Diese O-Dateien sind Inkarnationen der Partition.

```
02 A-Dateiname    string
02 Paßworte
   03 Lesen /1:6/ char
   03 Schreiben /1:6/ char
02 Sicherheitsklasse  int
02 Schlüssellänge  int
02 Schlüsselposition  int
02 Blocknummer der Wurzel  int
02 Benutzergruppen /1:n/  int
02 Abbildung auf Partitionen
   03 Partition für Baum
      04 Umfang  int
      04 Blockbelegung /1:Umfang/  bool
      04 O-Datei /1:flex/
         05 S-Datei string
         05 Rechnernummer int
   03 Partition für Daten
      04 Umfang int
      04 Blockbelegung /1:Umfang/  bool
      04 O-Datei /1:flex/
         05 S-Datei string
         05 Rechnernummer  int
```

Bild 5.4: Schema einer A-Dateibeschreibung

5.2. Katalogstruktur

Der Katalog kann formal als eine Relation K mit dem Aufbau K(Objektname, Bezug)[1)] aufgefaßt werden, so daß jedes Tupel entweder ein Schema einer A- oder einer B-Datei darstellt. Für die für DISCO

---

1) Die Unterstreichung von Objektname zeigt an, daß Objektname Schlüsselattribut ist.

ausgewählte Struktur ist insbesondere in Hinsicht auf die Verteilbarkeit die Hinzunahme weiterer Attribute sinnvoll (vgl. /Bre78/). Dabei wird der Katalog K

K(NAME 1, NAME 2, TEILVERWEIS 1, TEILVERWEIS 2, INFORMATIONEN 1, INFORMATIONEN 2)

feiner strukturiert, wobei NAME 1 konkateniert mit NAME 2 den Dateinamen ergibt. Zusammengenommen bilden beide Namen den Primärschlüssel. Die zwei Teilverweise gemeinsam verweisen auf die Datei bzw. Adreßliste. Die beiden Informationen-Attribute enthalten zusätzliche Dateibeschreibungsinformationen, deren Bedeutung für die weiteren Betrachtungen nicht von Belang ist. K enthält für jedes Objekt (Datei-Schema) des Rechnernetzes ein Tupel. Weiterhin soll davon ausgegangen werden, daß die beiden Teilverweise auf standardisierte Adreßlisten zeigen, die die Dateiadressen auf einem speziellen Datenträger enthalten.

Insbesondere die Randbedingungen

- anpaßbare Leistungsfähigkeit für unterschiedliche Anwendungen
- Brauchbarkeit für unterschiedliche Netzkonfigurationen
- hohe Verfügbarkeit

ergeben die Strukturierung der Relation K als ein System von Teilkatalogen bestehend aus Masterkatalog MKAT und Subkatalog SUBKAT:

MKAT(NAME 1, VERWEIS 1, INFORMATION 1)
SUBKAT(NAME 2, VERWEIS 2, INFORMATION 2)

Der Katalog K wird zunächst auf zwei Teilkataloge projiziert, MKAT und SUBKAT. MKAT wird an allen Rechnern redundant geführt und dient als Zugriffspfad zu Katalog SUBKAT, der als ein System lokaler Teilkataloge $SUBKAT_i$ realisiert wird. MKAT wird gelegentlich auch als Hyperkatalog bezeichnet. Der Hyperkatalog kann entfallen und einheitlich durch eine Hashfunktion ersetzt werden (vgl. /WiT76/).

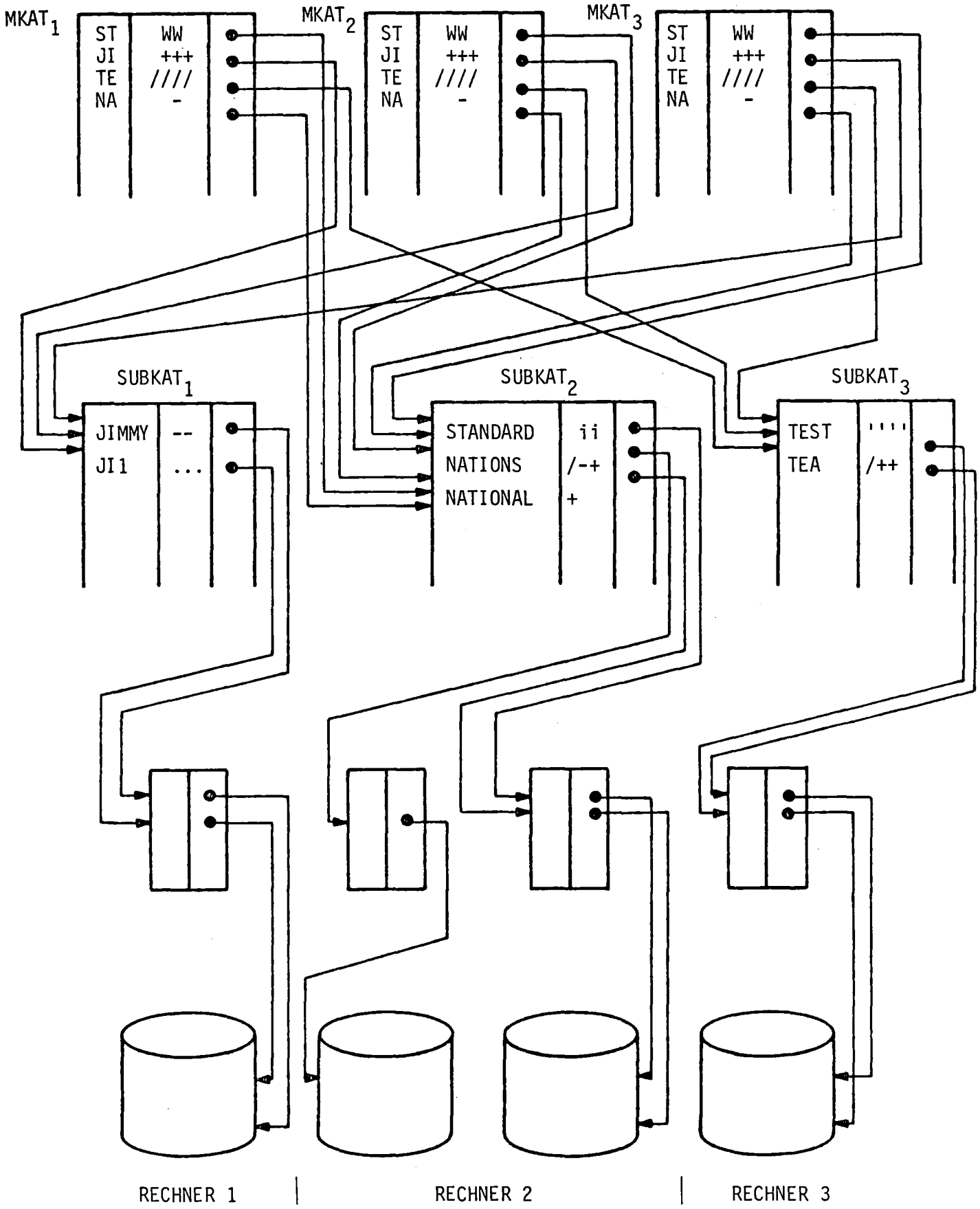


Bild 5.5: Beispiel für ein System hierarchischer Teilkataloge

In der Regel werden Vorschriften für die Verwendung von Namen erlassen wie die Verwendung qualifizierender Teilbezeichnungen. Sie dienen häufig zu organisatorischen Zwecken, z.B. die Verwaltung von Datenbeständen einzelner Benutzer. Die Benutzer müssen dann z.B. ihre Benutzernummer als Qualifier angeben. Alle ihre Dateien können dann in einem bestimmten Teilkatalog geführt werden. Außerdem kann so verhindert werden, daß zwei Benutzer dieselben Dateinamen verwenden wollen.

Das in Bild 5.5 gezeigte Beispiel macht deutlich, daß alle Dateien mit Dateinamen, die mit TE beginnen, auf Rechner R3 abgelegt sein müssen. Will man dagegen eine solche Datei auf einem anderen Rechner unterbringen, so können keine lokalen Teilkataloge verwendet werden. Um solche Einschränkungen zu verhindern, kann man eine weitere Hierarchiestufe hinzufügen. Anhand von NAME 1 sollte eindeutig festgelegt sein, welcher lokale Teilkatalog $SUBKAT_i$ weitere Informationen enthält. INFORMATION 1 würde global interessierende Attribute beschreiben - z.B. für die Zerteilung von Transaktionen -, während INFORMATION 2 Angaben zur Vervollständigung der Dateibeschreibungsinformation enthalten würde. Wird NAME 1 im Teilkatalog SUBKAT wiederholt, können bei VERWEIS 1 auch mehrere lokale Teilkataloge $SUBKAT_i$ angegeben werden. Innerhalb dieser Untermenge von Teilkatalogen müßte allerdings ein Broadcasting-Verfahren angewendet werden.

Eine sinnvolle Abänderung ergibt sich, wenn der Informationsgehalt der INFORMATION-Attribute sehr umfangreich ist: Der Hyperkatalog mit dem vollständigen Dateinamen verweist auf die Einträge mit vollständiger Information in den lokalen Teilkatalogen. Aufgrund der reduzierten Datenmenge ist das Verteilungsproblem für den Hyperkatalog wesentlich entschärft, besonders wenn die im Hyperkatalog enthaltenen Einträge vergleichsweise wenig änderungsintensiv sind.

Die für die Katalogorganisation aufgestellten Randbedingungen für ein Kleinrechner-Netzwerk, wie z.B. Kompaktheit der Schnittstellen, legen die Implementierung des Katalogs mit Hilfe von logischen Dateien nahe. Dies garantiert zugleich, daß für das Katalogschema kein

eigenes, neues Konzept einzuführen ist. Zusätzlich stehen die Konzepte der logischen Dateiebene, insbesondere das Transaktionskonzept, zur Verfügung.

Offensichtlich ist als Masterkatalog eine A-Datei vorzusehen, wobei als Schlüssel logische Dateinamen oder Teile davon zu verwenden sind. Der Rest einer Schemadefinition ist auf ein System lokaler Kataloge abzubilden. Es bieten sich drei Alternativen an:

- eine Anzahl von A-Dateien
- eine Anzahl von B-Dateien
- eine einzige B-Datei

Die dritte Alternative erweist sich vom Aufwand und vom operationalen Verhalten her als die geeignetste Lösung, wenn die einzelnen Partitionsbeschreibungen - Elemente der Reihung "Abbildung auf Partition" - auf einzelne Sätze abgebildet werden. Erhalten die Funktionseinheiten der Katalogverwaltung eigene Benutzergruppenidentifikationen, so werden die Cluster mit derart beschriebenen Benutzergruppen automatisch an der gewünschten Stelle angelegt und ergeben damit die Subkataloge.

Mit Hilfe der Abbildung des Katalogs auf zwei logische Dateien sind die Aufgabenstellungen teilweise gelöst:

- die A-Datei garantiert netzweit eindeutige Dateinamen
- die Bereitstellung von Schemabeschreibungen geschieht mit Hilfe der Lesezugriffe der logischen Dateiebene
- die Schnittstelle für den Datenbankadministrator (DBA) kann auf einfache Weise auf Operatoren der logischen Dateieben aufbauen.

Aus der Verwendung von Operatoren der logischen Dateiebene folgt, daß sich eine Funktionseinheit der Katalogverwaltung dazu an eine ZU wenden muß. Diese ZU soll im folgenden als Arbeits-ZU bezeichnet werden. Im allgemenen gilt, daß die auftraggebenden ZUs die Arbeits-ZU beinhalten. Die Tatsache, daß Arbeits-ZU und auftraggebende KAT auf verschiedenen Rechnern liegen können, würde im

Falle eines Auftrags der Arbeits-ZU an KAT das im Grunde unnötige Hin- und Herübertragen von Kataloginformationen bedingen. Durch die in Bild 5.6 gezeigte Abbildung der Schemabeschreibung wird der größte Overhead vermieden.

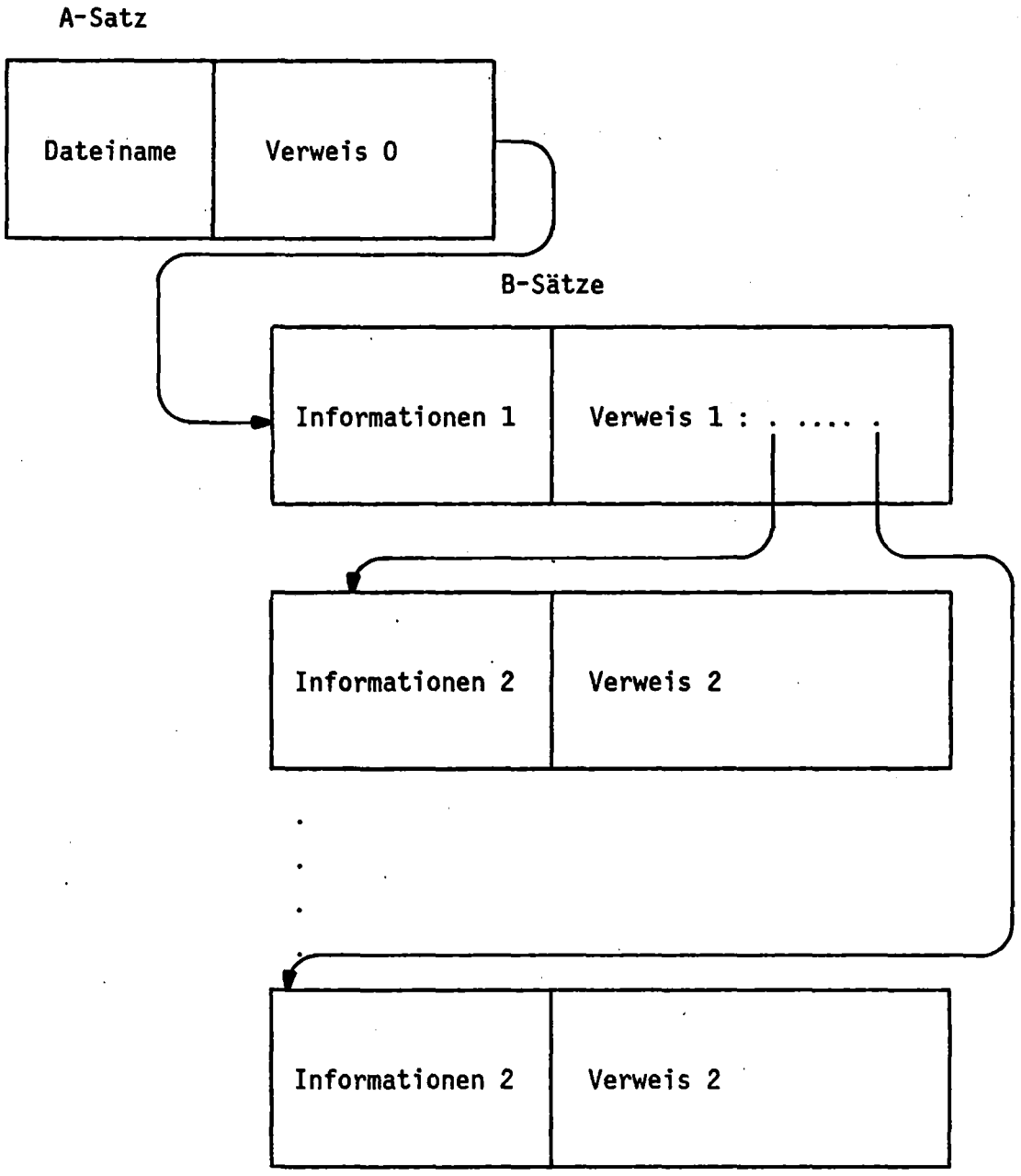


Bild 5.6: Abbildung der Schemadefinfitionen logischer Dateien auf das Katalogsystem A-Datei (Masterkatalog) und B-Datei (Subkataloge)

Informationen 1 umfassen die nicht partitionsbezogenen

Beschreibungsmerkmale, Informationen 2 vor allem Bitlisten für Satz- bzw. Blockbelegung. Für das Bereitstellen der Transformationsstrukturen, eine der häufigsten Anforderungen an die KAT, genügt es nun, wenn lediglich der erste B-Satz mit Informationen 1 und Verweise 1 zur Verfügung gestellt werden. Die restlichen Informationen können dann von den anfordernden ZUs unter Verwendung interner Transaktionen direkt beschafft werden (vgl. Kap. 7.2.3).

5.3. Erzeugung und Verwaltung von Schemata

Die Ausführung der beiden DEF-Operatoren beinhaltet das Erzeugen einer Dateibeschreibung. Diese wird im Katalog untergebracht, um für spätere Bezugnahmen zur Verfügung zu stehen.

5.3.1. Erzeugung von Schemata

Die Aufgabenverteilung zwischen KAT und SKAT weist dem SKAT hauptsächlich Speicherverwaltungsaufgaben und Erzeugen der S-Dateien zu, während die KAT globale Gesichtspunkte, Verteilung, Plazierung etc. festlegt.

Der KAT fällt insbesondere die Aufgabe zu, optimale Strategien bei der Abbildung logischer Dateien auf S-Dateien anzuwenden. Diese Strategien basieren einerseits auf Informationen, die mittels DEF vom Benutzer angegeben werden, andererseits auf systeminternen Daten, die sich teilweise erst im laufenden Betrieb ermitteln lassen. Für das Sammeln entsprechender Information hat die KAT Sorge zu tragen.

Die Abbildung von Benutzergruppenclustern, Zugriffspfaden und logischen Dateien muß unter Berücksichtigung folgender Faktoren erfolgen:

- Resourcen
  - . Rechner
  - . Funktionseinheiten
  - . Speicherperipherie

- Kommunikationsverbindungen
  - . Kapazität
  - . Übertragungszeiten
  - . Grad der Zuverlässigkeit
  - . Übertragungskosten
- Benutzerangaben (DEF, Angaben des DBA)
- Charakteristik von Transaktionen, insbesondere der Retrieval/Update-Rate
  - . bezogen auf Benutzergruppen
  - . bezogen auf Verwendung von Daten
  - . bezogen auf Anwendungen
- Betriebsorganisation (z.B. Stapel-Dialogbetrieb)
  - . Optimierungsziele
- Zustand des verteilten Dateiverwaltungssystems (VDVS) hinsichtlich
  - . aktueller Konfiguration, besonders bei Fehlersituationen
  - . Sperrungen
  - . Autorisierung
  - . konkurrierender Anforderungen

Zur Festlegung der Abbildung sind drei Funktionen notwendig:

- f1 berechnet den Umfang der Partitionen
- f2 den Redundanzgrad der einzelnen Partitionen
- f3 die Zuordnung der O-Dateien zu Rechnern.

Der Komplexitätsgrad dieser drei Funktionen ist je nach Berücksichtigung und Wichtung der oben aufgeführten Faktoren derart unterschiedlich, daß für alle Installationen maßgeschneiderte

Funktionen zu definieren sind. Allgemeine Verfahren sind aufgrund ihres Rechenzeitbedarfs nicht sinnvoll anwendbar.

5.3.2. Konsistenzhaltung von Schemabeschreibungen

Die Gewährleistung der Integrität von Schemabeschreibungen ist eine der aufwendigsten Aufgaben der Katalogverwaltung. Die KAT hat dafür zu sorgen, daß nicht nur die Integrität der im Katalog gehaltenen Schemabeschreibungen, sondern auch die bei den verschiedenen ZUs existierenden Dateibeschreibungen konsistent sind. In diesem Zusammenhang können drei Typen von Transaktionen unterschieden werden:

Typ 1: die Transaktion bewirkt keine Änderung an der Dateibeschreibung (z.B. kein INSERT aufgerufen)

Typ 2: die Transaktion bewirkt Änderungen an der Dateibeschreibung, ohne daß ein (bei einer B-Datei) impliziter Dateiüberlauf eintritt, der das Anlegen einer zusätzlichen Partition erfordert.

Typ 3: wie Typ 2, jedoch mit implizitem Dateiüberlauf

Bei Typ 2- und Typ 3-Transaktionen wird die Katalogverwaltung im Laufe der Bearbeitung von END_TRANS der die Änderungen verursachenden Transaktion von der ZU beauftragt, die Konsistenz der Schemabeschreibung im Katalog und bei anderen ZUs mit der neuen Schemabeschreibung herzustellen. Zur Identifikation derjenigen ZUs, bei denen eine solche Schemabeschreibung vorliegt, führen alle KATs redundant eine sogenannte OPEN-FILE-Liste. Diese wird nur dann modifiziert, wenn bei einer ZU eine Schemabeschreibung gelöscht oder neu beschafft wird.

Aufgrund der besonderen Abbildung der Schemabeschreibung für logische Dateien auf Katalogsätze können bei Typ 2-Transaktionen sogar die Konsistenzherstellungen von Partitionsbeschreibungen von verschiedenen, auf einer einzigen Datei arbeitenden Transaktionen zugleich ausgeführt werden.

Die Konsistenz der Schemabeschreibungen im (unter anderem redundant ausgelegten) Katalog ist allein durch die in Transaktionen gefaßten Verwaltungsaktionen gewährleistet.

6. Ausfall/Wiederanlauf

Das Ziel der Behandlung von Ausfällen in einem System liegt in der Erreichung der Verarbeitungsfortführung bei einer möglichst minimalen Einschränkung des gesamten Systemverhaltens. Dabei gilt, daß bei einem verteilten System wie DISCO ein entsprechend hoher Komplexitätsgrad berücksichtigt werden muß.

Die Übermittlung von Aufträgen, Daten und Rückmeldungen erfolgen über das Nachrichtentransportsystem (NTS). Demzufolge haben Ausfälle des NTS gravierende Folgen für die Zusammenarbeit zwischen den verschiedenen Funktionseinheiten, sofern diese nicht vom NTS selbstständig behoben werden können wie z.B. bei Übertragungsfehlern oder durch Routenwechsel. Wir gehen im folgenden davon aus, daß ein Ausfall von Systemteilen von den entsprechenden Funktionseinheiten erkannt werden kann. Weitergehende Untersuchungen finden sich in /Mül80/.

Die Ausfallbehandlung in DISCO stützt sich wesentlich auf die Durchführung der Transaktionsverarbeitung und des deferred update Konzepts (s. Kap.4.) und wird deshalb hier weiter detailliert. Wir bezeichnen eine Transaktion als offen, solange sie rücksetzbar ist. Während der END_TRANS Bearbeitung wird zuerst die Abarbeitung aller asynchronen Aufträge abgewartet. Danach wird die zur Transaktion gehörige Information auf die temporäre Datei (TD) geschrieben:

- alle geänderten Seiten aus dem Seitenpuffer, die noch nicht auf TD sind nebst Zuordnungsinformation (Dateiname und Blocknummer)
- Liste nachzuholender Operationen wie z.B. Konsistenzherstellung des Katalogs, vgl. 5.3.

Das Ablegen der obigen Information auf der temporären Datei geschieht aufgrund der Berücksichtigung des Falles, daß während des Einschreibens in die Datenbasis ein Fehler eintreten kann. Wenn solch ein Fehlerfall den Verlust des Hauptspeicherinhaltes bewirkt, könnte die Änderung der Datenbasis nicht mehr zu einem späteren Zeitpunkt abgeschlossen werden, da alle hauptspeicherresidente Information

vernichtet ist. Somit befände sich die Datenbasis in einem inkonsistenten Zustand. Das Ablegen auf der temporären Datei ermöglicht nun ebenfalls diese Fehlersituation zu beherrschen. Unter der Voraussetzung, daß die Speichereinheit der temporären Datei intakt bleibt, ist es nun möglich, das Einschreiben aller von der Transaktion geänderten Blöcke nochmals vorzunehmen.

Nun werden alle KOs, die Änderungssperren bzgl. dieser Transaktion halten, verständigt, diese zu markieren. Nach der erfolgreichen Rückmeldung an die aussendende ZU wird ein sogenannter Sicherungspunkt erreicht und eine Marke auf TD geschrieben. Mit dem Schreiben des Sicherungspunktes wird ein Zustand erreicht, der sicherstellt, daß alle notwendigen Informationen auf TD aufgezeichnet sind, damit zu einem beliebigen späteren Zeitpunkt alle durch die Transaktion verursachten Änderungen mit Hilfe der TD in die Datenbasis eingebracht werden können.

Die Transaktion wird von diesem Zeitpunkt als gebunden bezeichnet, d.h. sie gilt für den Benutzerprozeß als beendet. In dieser Bindungsphase werden die eigentlichen Änderungen in die Datenbasis eingespielt, danach die Sperren gelöst; dieser Vorgang ist im Fehlerfall zu einem späteren Zeitpunkt wiederholbar. Der zeitliche Ablauf ist in Bild 6.1 dargestellt.

6.1. Ausfall von Daten und Abhilfemaßnahmen

Unter Ausfall von Daten wollen wir verstehen, daß aus der Sicht der Zugriffsverwaltung Daten nicht zugreifbar sind. Diese Situation kann bedingt sein durch fehlerhaften Datenträger, Gerätefehler oder Ausfall der Anpassungseinheit auf dem jeweiligen Rechner. Die beiden ersten Fehlerarten liegen außerhalb des DISCO-Systems und erfordern betriebsspezifische Maßnahmen. Abhilfe wäre durch redundante Hardware möglich, transparent für die Anpassungseinheit von DISCO. Ansonsten Maßnahmen wie bei Anpassungs-Ausfall.

Beim Ausfall einer Anpassungseinheit sind alle lesenden oder schreibenden Aufträge betreffs Daten an diesem Rechner nicht durchführbar und müssen abgebrochen werden. Die Folge ist, daß davon

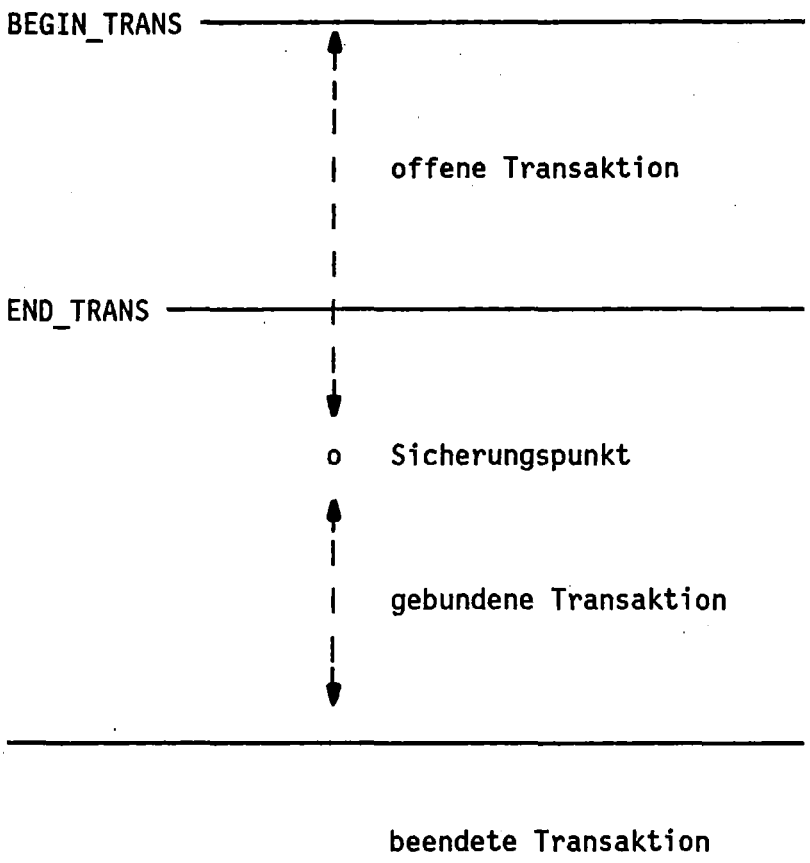


Bild 6.1: Ablauf einer Transaktion

betroffene offene Transaktionen zurückzusetzen sind und gebundene Transaktionen nicht beendet werden können. Um diesem Nachteil zu begegnen, bietet DISCO mit dem Konzept der Sicherheitsklasse (s. Kap.2.) die Möglichkeit, Daten redundant auf verschiedenen Rechnern abzulegen. Dadurch kann im Fehlerfall die Verfügbarkeit des Gesamtsystems erhalten bleiben. Bei lesenden Aufträgen wird dann einfach auf eine der Kopien zugegriffen.

Anders ist die Vorgehensweise bei schreibenden Aufträgen. Die Transaktionsverarbeitung verlangt, daß alle Kopien der Datenbasis während der Bindungsphase auf den gleichen, konsistenten Zustand gebracht werden. Bei einem Ausfall einer Kopie hätte dies zur Folge, daß diese Kopie nicht geändert werden könnte, die entsprechenden Sperren bestehen blieben und somit die gesamte Transaktion nicht beendet werden könnte. Dies wäre von erheblicher Beeinträchtigung für

das Systemverhalten. Als Lösung bietet sich an, nicht zugängliche Kopien aus dem Katalog zu streichen, sofern mindestens eine Kopie in der Datenbasis zugreifbar ist. Dadurch kann die Transaktion beendet werden und der Gesamtbetrieb reibungslos weiterlaufen. Zu einem späteren Zeitpunkt dann, wenn die ausgefallenen Daten wieder zugreifbar sind, müssen sie mit Hilfe einer Kopie auf den neuesten Stand gebracht und wieder in den Katalog eingetragen werden.

Problematisch wird es, wenn der Datenausfall eine temporäre Datei umfaßt. Während offene Transaktionen, die mit der ausgefallenen TD arbeiten, ohne weiteres zurückgesetzt werden können, gilt dies nicht für Transaktionen in der Bindungsphase. Hier muß davon ausgegangen werden, daß sich die Datenbasis in einem inkonsistenten Zustand befindet.

Darum muß dafür gesorgt werden, daß die Bindungsphase der Transaktion nach der Wiedereingliederung der TD nochmals durchlaufen wird, was durch Abprüfen der Sicherungsmarke durchführbar ist. Nachteilig ist die evtl. lange Blockierung der Transaktionsbeendigung. Sollte die TD unrettbar zerstört sein, hat dies fatale Folgen, die gesamte Datenbasis muß auf einen früheren, konsistenten Zustand zurückgesetzt werden (vgl. Kap.6.3). Eine höhere Sicherheit läßt sich erreichen, wenn bei Erstellung des Sicherungspunktes alle notwendigen Informationen inklusive der geänderten Datenobjekte in einer weiteren, redundanten Sicherungsdatei abgelegt werden, die sich auf einem anderen Rechner befindet. Dadurch kann, selbst beim zusätzlichen Ausfall der zugehörigen Zugriffsverwaltung (z.B. bei einem Rechnerausfall), mit Hilfe einer anderen ZU die Transaktion beendet und die Verfügbarkeit des Systems gewahrt werden.

6.2. Ausfall von globalen Funktionseinheiten

Von entscheidender Bedeutung ist die Behandlung von Ausfällen der globalen Funktionseinheiten (FE) Zugriffsverwaltung (ZU), Zugriffskontrolle (KO) und Katalogverwaltung (KAT). Das nach einem Ausfall verbleibende Restsystem muß eine geeignete Ausfallbehandlung durchführen, um inkonsistente Zustände zu vermeiden, die weiteren

Aufgaben fortzuführen und die spätere Wiedereingliederung der ausgefallenen Funktionseinheit vorzubereiten. Als Maßnahmen dazu dienen:

- Zurücksetzen von Aufträgen, die an die ausgefallene Funktionseinheit gerichtet waren,
- Transaktionen je nach Ausführungsgrad zurücksetzen oder ruhen zu lassen,
- inkonsistente Datenobjekte weiterhin gesperrt zu halten.

Einige wesentliche Gesichtspunkte sollen in den folgenden Ausführungen exemplarisch an der Ausfallbehandlung bei ZU und KO aufgezeigt werden. Eine umfassendere Darstellung findet sich in /Mül80/.

Der Ausfall einer Zugriffsverwaltung ZU_a hat Auswirkungen auf alle von ihr verwalteten Transaktionen. Es wird davon ausgegangen, daß nach einem Ausfall stets alle hauptspeicherresidente Information, wie Kontrollblöcke etc., verloren ist. Dadurch ist es unmöglich, den Zustand von offenen Transaktionen später zu rekonstruieren und es folgt, daß sie zurückgesetzt werden müssen. Bezüglich der Zugriffskontrolle bedeutet dies nach der Ausfallerkennung eine Benachrichtigung aller KOs, damit alle von der ZU_a beantragten Sperren bezüglich offener Transaktionen aufgehoben werden. Sperren von gebundenen Transaktionen bleiben bestehen. Sämtliche Aufträge der ausgefallenen ZU_a an Katalogverwaltungseinheiten werden gelöscht.

Eine gesonderte Betrachtung erfordert die KAT, deren zugeordnete ZU (Arbeits-ZU, vgl. Kap.5.2) ausgefallen ist. Die von KAT an ZU_a ergangenen Transaktionen (K-Transaktionen genannt) werden zurückgesetzt. Dies gilt sowohl für offene als auch für bereits gebundene K-Transaktionen. Der Grund für die Sonderbehandlung der K-Transaktionen liegt darin, daß diese Änderungen auf den Katalogen MKAT bzw. SUBKAT bewirken und die Kataloge, zumindest Teile des gesamten Katalogsystems, gesperrt sind. Würden die Sperren wie bei "normalen" gebundenen Transaktionen gehalten, so wären die gesperrten Katalogteile erst nach der Wiedereingliederung von ZU_a wieder

zugänglich, was eine starke Beeinträchtigung des gesamten DISCO-Systems zur Folge hätte. Deshalb werden im Fehlerfall auch gebundene K-Transaktionen zurückgesetzt und später wiederholt. In /Mül80/ wird gezeigt, daß hierbei keine Inkonsistenzen auftreten können, da erstens K-Transaktionen beliebig oft wiederholbar sind und zweitens sich verschiedene K-Transaktionen bei der Manipulation von Katalogeinträgen nicht überschneiden können. Das zweite Argument bedeutet, daß andere K-Transaktionen evtl. inkonsistente Katalogteile nicht sehen können. Dies folgt daraus, daß ja K-Transaktionen aufgrund von Benutzer-Transaktionen ausgelöst werden, die ihrerseits die entsprechenden Datenobjekte gesperrt haben müssen, und diese Sperren werden ja stets gehalten.

Wie bereits erwähnt, ist für die Konsistenzerhaltung der Datenbasis das Sperren von Datenobjekten notwendig, welches von den KOs vorgenommen wird. Welche Auswirkungen ergeben sich nun bei dem Ausfall einer solchen KO?

Die Reaktion auf den Ausfall einer KO ist ziemlich geradlinig. Die ZUs setzen diejenigen offenen Transaktionen zurück, die Objekte bei der KO_a gesperrt haben. Die gebundenen Transaktionen werden weiter fortgesetzt. Bei Wiederanlauf besorgt die ausgefallene KO_a ihre Sperrinformation für gebundene Transaktionen von allen ZUs und trägt sie in die Sperrtabelle ein.

Unter der Annahme, daß niemals mehr als eine Funktionseinheit ausfällt, kann obige Vorgehensweise angewandt werden. Problematisch wird der Fall, wenn zusätzlich noch eine ZU ausgefallen ist, was bei Konfigurationen mit ZU und KO auf einem Rechner recht wahrscheinlich ist. Da die wiederanlaufende KO die Sperrinformation der gebundenen Transaktion von allen ZUs benötigt, wirkt sich ein bestehender Ausfall einer ZU folgendermaßen aus:

Die KO kann ihren normalen Betrieb nicht aufnehmen (d.h. keine Sperranforderungen bearbeiten), weil Ungewißheit herrscht, ob bei der ausgefallenen ZU gebundene Transaktionen existieren, die Sperren bei der wiederangelaufenen KO halten. Die Folge ist eine Lähmung des Systembetriebs, weil Daten aus dem ganzen Bereich, den die KO

verwaltet, nicht gesperrt werden können und damit für neue Transaktionen unzugänglich sind.

Um dies zu vermeiden, wurde eine Lösung entwickelt, die darauf beruht, daß bestimmte Kontrollinformation redundant in einer anderen KO abgelegt wird. Diese KO nennen wir Stellvertreter $SV(KO_x)$ bzgl. KO_x. Für jede KO_x existiert genau eine vorher fest zugeordnete $SV(KO_x)$, die zusätzlich zu ihrer Sperrinformation die Sperrinformationen bzgl. der gebundenen Transaktionen von KO_x verwaltet.

Bei der Bindung der Transaktion muß jede KO_x, die Objekte für diese Transaktion gesperrt hat, der betreffenden ZU quittieren, daß sie die Sperren noch aufrecht erhält. Zuvor übermittelt sie ein Paket mit der zugehörigen Sperrinformation an die $SV(KO_x)$. Diese nimmt das Paket entgegen und fügt es in ihre SV-Sperrtabelle ein. Diese Tabelle enthält die ganze Sperrinformation der gebundenen Transaktionen der zu vertretenden KO_x. Danach sendet die $SV(KO_x)$ eine Quittung an die KO_x, und diese kann den Übergang in die Bindungsphase für die Transaktion der ZU gestatten.

Dieses sogenannte Stellvertreter-Prinzip vermeidet nun die Gefahr der Beeinträchtigung des Systembetriebs nach dem Wiederanlauf, verhindert jedoch nicht, daß während des Ausfalls von KO_a kein der KO_a zugeordnetes Datenobjekt gesperrt werden kann. Ein weitergehendes Stellvertreter-Konzept (s. /Mül80/) bietet nun auch diese Problemlösung an, indem $SV(KO_a)$ das gesamte Aufgabenspektrum von der ausgefallenen KO_a übernimmt und ihrerseits einen neuen Stellvertreter erhält.

6.3. Rekonstruktion

6.3.1. Rekonstruktion der Datenbasis

Bei Auftreten von "katastrophalen" Fehlern - das sind Fehler, die nicht durch die im vorigen dargestellten Maßnahmen behebbar sind - ist es notwendig, die gesamte Datenbasis zu rekonstruieren. Ein solcher Fehler ist zum Beispiel der Ausfall von Speichermedien, deren

Teil der Datenbasis nicht redundant auf einem anderen Speicher vorliegt.

Es gilt, einen globalkonsistenten Zustand wiederherzustellen. Unter "globalkonsistent" versteht man einen Zustand der Datenbasis, der gegenseitige Abhängigkeiten von Transaktionen, die sich aus dem zeitlichen Ablauf ergaben, berücksichtigt. Das bedingt, daß es beispielsweise beim Ausfall eines Teils der Datenbasis nicht genügt, nach der Reparatur des Speichers irgendeine Kopie nur dieses Teils von einem Sicherungsband zurück zu kopieren. Es muß deshalb entweder die gesamte Datenbasis auf einen konsistenten Zustand zurückgesetzt werden, oder man muß nach dem Kopieren des ausgefallenen Teils gewährleisten, daß alle bis zum Ausfall durchgeführten Änderungen von abgeschlossenen Transaktionen nachgezogen werden.

Das wichtigste Hilfsmittel für die Rekonfiguration ist ein sogenanntes Sicherungsband, auf dem in größeren periodischen Abständen die gesamte Datenbasis abgespeichert wird. Da diese globalkonsistent sein muß, wird in DISCO dafür gesorgt, daß zu Beginn des Aufzeichnens keine ändernde Transaktion im System befindlich ist. Dann wird die gesamte Datenbasis gesperrt mit Sperrmodus LOCK(S). Nun können Benutzertransaktionen wieder zugelassen werden, die auf der gesamten Datenbasis lesenden Zugriff haben und nach erfolgreichem Kopieren von Teilen der DB und dem resultierenden sukzessiven Freigeben dieser Teile über diese auch schreibend verfügen können.

Durch dieses Vorgehen wird die notwendige Systembeeinträchtigung beim Sicherungskopieren während des laufenden Betriebs so gering wie möglich gehalten. Das Kopieren wird von einer Funktionseinheit, einer ZU, zentral durchgeführt, nachdem sich das DISCO-System durch Annahmeverweigerung neuer und Rücksetzung laufender Transaktionen in einem globalkonsistenten Zustand befindet. Auf das Sicherungsband muß zuerst der gesamte Katalog aufgezeichnet werden, danach dann die Datenbasis. Dabei genügt es, daß von Kopien (P-Dateien) nur jeweils eine Inkarnation gesichert wird. Als Schutz gegen mögliche Fehler der Bandaufzeichnung kann diese bei gegebener Systemkonfiguration redundant vorgenommen werden. Dies geschieht ohne wesentliche Zeiteinbußen, indem die Schreibaufträge quasi parallel an

verschiedene Anpassungseinheiten gesandt werden, über die die Bandaufzeichnung gesteuert wird.

Die zeitlichen Abstände, in denen Sicherungsaufzeichnungen stattfinden und auf die die Datenbasis zurückgesetzt werden kann, sind für manche Anwendungen zu groß. Eine Maßnahme, um auf nur kurz zurück liegende sogenannte Checkpoints (an denen Konsistenz gewährleistet werden kann) wieder aufsetzen zu können, ist die Führung von Journalen, auch Logfile genannt. Sie wird in der gegenwärtigen Ausbaustufe von DISCO nicht vorgenommen, soll jedoch nicht unerwähnt bleiben. Auf dem Journal werden, beginnend mit der letzten Sicherungsaufzeichnung, alle notwendigen Informationen aufgezeichnet, um ausgehend von der Sicherungskopie neuere, konsistente Zustände wiederherzustellen. Das Journal umfaßt im wesentlichen Zustandsinformation von Transaktionen, alle Änderungen auf Datenobjekten und Katalog, sowie Checkpointmarken. Die Schwierigkeit in der Journalführung liegt darin, daß sie aus Effizienzgründen nicht zentral vorgenommen werden soll (Flaschenhals Effekt).

Der Vorteil eines zentralen Journals liegt in der chronologischen Aufzeichnung, woraus implizit die Abhängigkeiten zwischen Transaktionen ablesbar sind. Da diese Abhängigkeiten auch bei dezentraler, entkoppelter Journalführung erkennbar sein müssen, muß jedes Journal die dazu erforderlichen, zusätzlichen Informationen enthalten. Neben des erhöhten Informationsbedarfs ist auch der Aufwand im Rekonstruktionsfall höher als bei der zentralen Journalführung.

Weitere Probleme ergeben sich beim Schreiben von Checkpointmarken, wenn minimale Systembelastung gefordert wird. Der gesamte Fragenkomplex im Zusammenhang mit verteilten Systemen wird in der Forschung gerade in ersten Ansätzen, z.B. /ScD80/, behandelt, und muß noch weiter behandelt werden.

6.3.2. Rekonstruktion des operationalen Systems

Die Rekonstruktion des operationalen Systems bedeutet in DISCO die

Überführung des Systems nach einem Fehlerfall in ein funktionsfähiges Restsystem, welches das vollständige Spektrum an Verarbeitungsmöglichkeiten, mit Ausnahme des Zugriffs auf ausgefallene Datenbestände, anbietet. Dies kann nur erfolgen, wenn bezüglich jeder globalen funktionellen Komponente noch jeweils mindestens eine Funktionseinheit existiert.

Entscheidender Anteil bei der Durchführung der Rekonfiguration liegt bei den Funktionseinheiten der Zugriffskontrolle (KO), die alle untereinander in Verbindung stehen und die notwendige Koordination vornehmen. Wenn nun eine KO selbst ausfällt, ist dieser Ausfall vorrangig zu behandeln. Die Vorgehensweise entspricht dem erweiterten Stellvertreterkonzept, wie es in Kap.6.2. erwähnt wurde, und soll hier nicht weiter detailliert werden. Die KOs müssen Informationen über die bestehende Konfiguration und die Ausfälle halten. Aufgrund dieser Ausfälle werden dann neue Zuordnungen von BEA zu ZU, bzw. ZU zu KAT festgelegt und Ausfallnachbehandlungen bei den entsprechenden Funktionseinheiten angestoßen.

7. Pilotimplementierung

Für die Pilotimplementierung werden verschiedene Ebenen und Schichten unterschieden, die nicht mit denen des Architekturmodells zu verwechseln sind.

7.1. Implementierungsebenen

Die Implementierung des operationalen Modells erfolgte auf einem Rechnernetz bestehend aus drei Rechnern der Serie Siemens 300, gekoppelt mittels Siemens Koppeleinheiten gemäß der in Bild 7.1 gezeigten Topologie.

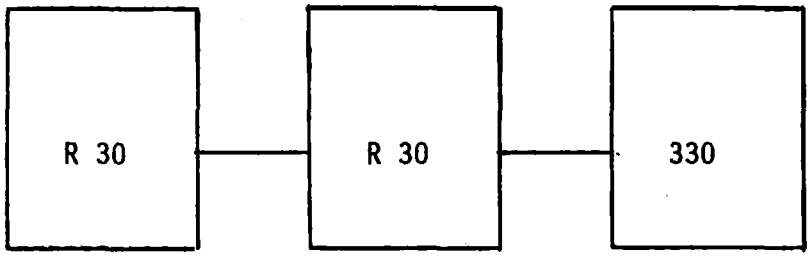


Bild 7.1 : Konfiguration für die Pilotimplementierung

Grundlage der Implementierung sind die Programmiersprache Prozeß-FORTRAN, deren Erweiterungen gegenüber Standard-FORTRAN bezüglich Bitmanipulationen verwendet wurden, ferner die ORG-Betriebssysteme und die von der Transportstation des RPS-Systems angebotenen Kommunikationsprimitive für die Interprozeßkommunikation. Alle Funktionseinheiten stellen einen Prozeß im Sinne des Siemens ORG Betriebssystems dar.

Die Wahl der Implementierungssprache FORTRAN erfolgte aufgrund ihrer Verfügbarkeit auf praktisch allen in Frage kommenden Kleinrechnersystemen und des Reifegrades der dazu gehörenden Übersetzer und eingebauten Betriebssystem-Schnittstellen.

Gemeinsame Implementierungsstrategie für die funktionellen Komponenten ist die Schaffung von Schichten, die häufig benutzte Funktionen aufeinander aufbauend zu erstellen ermöglichen:

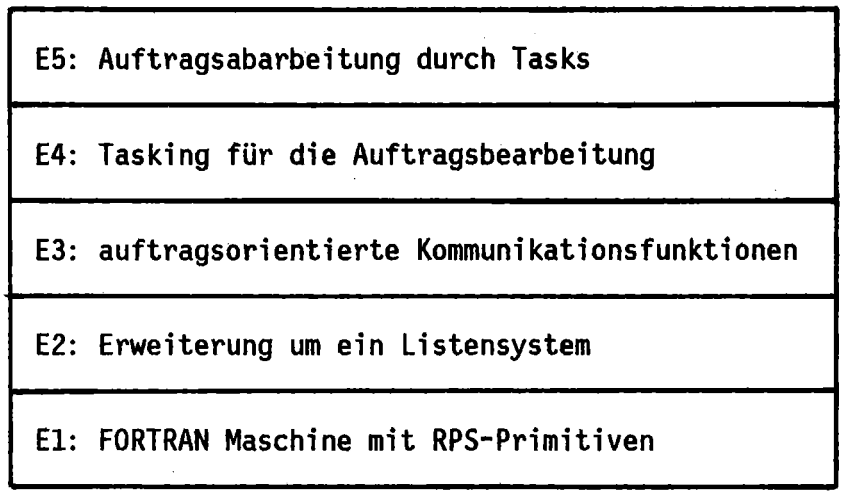


Bild 7.2: Implementierungsebenen

7.1.1. Die Ebene E1: FORTRAN Maschine mit RPS-Primitiven

Die Funktionen zur Interprozeßkommunikation

- SEND/RECEIVE
- OPEN/CLOSE
- WAIT
- CHECK/PLACE

sind in /Mai79/ beschrieben. Da der Aufbau der Kommunikationskanäle nach diesem Konzept (vgl. /HBD78/ Kap. 7) in der Anlaufphase eines DISCO-Systems erfolgt, basiert die Kommunikation hauptsächlich auf den SEND/RECEIVE-Funktionen. Anhand der Returncodes, die mittels der sogenannten Fortsetzkennung dem Aufruf zugeordnet werden können, ergibt sich, ob z.B. eine Nachricht beim Empfänger abgelegt wurde, ob Timeout wegen fehlendem RECEIVE auf der Empfängerseite oder andere Fehlersituationen vorliegen.
Alle funktionellen Komponenten kommen mit der E1 Ebene aus, mit Ausnahme der Benutzer-Ein/Ausgabe:

Zur Koordinierung der Benutzerprogramme, die mit DISCO arbeiten, mußte, um auch asynchrone Operatoren (vgl. Anhang A) realisieren zu können, auf das nicht mit FORTRAN-Standardfunktionen angebotene Koordinierungszählerkonzept des ORG-Betriebssystems zurückgegriffen werden /ORG79/.

In unserer Implementierung wurden die DISCO-Operatoren mittels einer CALL-Schnittstelle zunächst in das Programmiersystem Siemens Prozeß-FORTRAN 300 eingebettet. Zur besseren Lesbarkeit der Notationen wurden hierzu kleinere Ergänzungen notwendig, damit eine variable Anzahl von Parametern übergeben werden kann.

7.1.2. Die Ebene E2: Erweiterung um ein Listensystem

Hier handelt es sich um eine Erweiterung mit Listenverarbeitungsfunktionen, die einfach verzeigerte Listen variabler Elementlänge verarbeiten. So können z.B. Listenelemente beliebiger Länge mittels GETLE beschafft oder mittels RETLE an die Freispeicherverwaltung zurückgegeben werden. Das Freispeicherverwaltungskonzept gestattet das Unterbringen beliebig vieler Zeiger in einem Listenelement.

Aufbauend auf diesen Funktionen können die verschiedenen notwendigen Kontrollblöcke in den einzelnen Funktionseinheiten und die Kommunikations- und Task-Steuerung ohne den sonst üblichen Speicherverschnitt bequem realisiert werden. Implementierungstechnisch werden in FORTRAN Listenelemente in einem Ausschnitt des Feldes LISTE untergebracht.

7.1.3. Die Ebene E3: auftragsorientierte Kommunikationsfunktionen

Um eine nachrichtengesteuerte Auftragsbearbeitung implementieren zu können, muß das Konzept der Ebene E1 erweitert werden. Zur effizienten Übertragung von Nachrichten werden Auftrags- und Datennachrichten auf eigenen Kanälen übermittelt (vgl. /Bre78/). Dies ermöglicht

- effiziente platzsparende Nachrichtenpufferung

- bevorzugte Auftragsübertragung

- Zusammenfassung der Bearbeitung grundlegender Fehlerfälle

- Kommunikationsauftragspufferung in Warteschlangen zur optimalen Auslastung von Nachrichtentransportsystem und Zentralprozessor

Da Auftragsbeschreibungen im allgemeinen wesentlich kürzer als Daten sind, kann mit Hilfe eines Standardrahmens, einem sogenannten Paket, der größte Teil von Nachrichten übertragen werden. Bei Bedarf wird ein umfangreicherer Auftrag auf mehrere Pakete aufgeteilt. Dies ermöglicht die permanente Empfangsbereitschaft von Aufträgen, ohne unvertretbar viel Nachrichtenpuffer zur Verfügung stellen zu müssen. Kurze Auftragsnachrichten können Datennachrichten überholen, Daten und Aufträge können simultan empfangen werden.

Über den Auftragskanal können Protokolle zur Behandlung fehlerhafter Kommunikationsvorgänge abgewickelt werden, nicht operatorspezifische Fehlerbehandlungsmaßnahmen können auf Ebene E3 ablaufen.

Die Paketformate für Aufträge gestatten wegen der permanenten Empfangsbereitschaft das Akkumulieren von Aufträgen, so daß die Möglichkeiten unterschiedlicher Bearbeitungsreihenfolgen und weitergehender Ablaufoptimierung bestehen.

7.1.4. Die Ebene E4: Tasking für die Auftragsbearbeitung

Die Ebene E4 umfaßt die Funktionen für eine elementare Taskverwaltung mit nachrichtengesteuerter Taskaktivierung. Für die Einführung eines solchen Konzepts in Ebene E4 waren folgende Gründe ausschlaggebend:

- das ORG-Betriebssystem läßt lediglich eine relativ kleine Anzahl von Prozessen zu und fordert die Generierung der für die Maximalzahl erforderlichen Prozeßparameterblöcke

- dem ORG liegt die Philosophie, daß ein Programm genau einen Prozeß ergibt, zugrunde

- es gibt keinen Standard für Intra-Prozeßkommunikation

Die Konzeption des operationalen Modells beruht darauf, daß ein Auftrag in Unteraufträge zerlegt wird, die von anderen Funktionseinheiten, die im allgemeinen auf unterschiedlichen Knoten des Rechnernetzes installiert sind, bearbeitet werden müssen. Dies erfordert fallweise längere Wartezeiten, wenn die Auftragsbearbeitung unterbrochen ist. Um überhaupt die vorhandenen Optimierungsmöglichkeiten, z.B. bei asynchronen READ-Aufträgen ausnutzen zu können, ist zur Erlangung eines vernünftigen Durchsatzes die Implementierung der Operatoren in Form von reentranten Programmen und deren Ablauf als Prozeß notwendig. Die vergleichsweise statische Auffassung zu Prozeß- und Programmstrukturen im ORG-Betriebssystem und die Überlegung, daß eine in FORTRAN programmierte Taskverwaltung auch leicht auf andere Systeme übertragbar ist, sowie die so möglichen Konventionen zur Intertaskkommunikation unter Vermeidung des IPC-Overheads, legten diese Lösung nahe. Die Funktionen der Ebene E4 werden in Kap. 7.2 genauer beschrieben.

7.1.5. Die Ebene E5: Auftragsabarbeitung durch Tasks

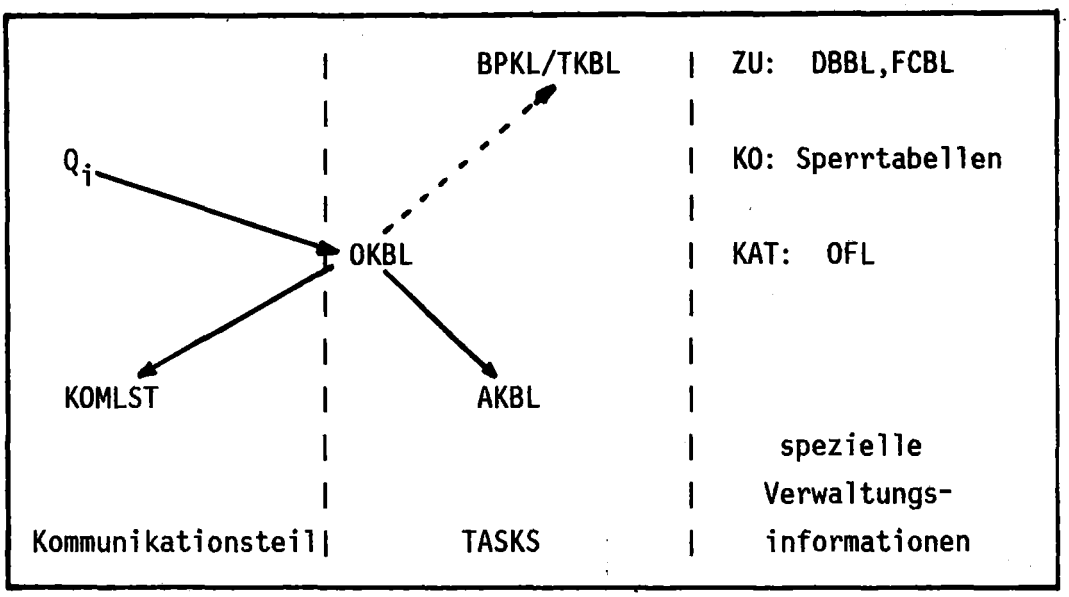
Diese Ebene enthält die eigentlichen Funktionen, die die verschiedenen Aufträge, die eine Funktionseinheit bearbeitet, erledigen. Diese Unterprogramme sind als Tasks entsprechend Ebene E4 implementiert, so daß für jeden Auftrag, auch wenn alle Aufträge von derselben Art sind, eine Task aktiviert wird. Zusätzlich sind interne Tasks, mit deren Hilfe Aufgaben auf Taskfamilien übertragen werden, auf Ebene E5 angesiedelt.

7.2. Ablaufsteuerung und Kontrollstrukturen

Die folgenden Beschreibungen der Ablaufsteuerung beziehen sich auf die globalen funktionellen Komponenten ZU, KAT und KO. Für BEA und ANP genügen vereinfachte Strukturen, selbst für die KO gibt es Vereinfachungen, weil zur KO lediglich Aufträge im Sinne der Ebene E3 gesandt werden.

7.2.1. Kontrollstrukturen

Für alle globalen Funktionseinheiten können die Kontrollstrukturen grob in 3 Teile aufgeteilt werden (vgl. Bild 7.3).



Q_i:	Eingabe-Warteschlange
KOMLST:	Kommunikationsauftrags-Warteschlange
OKBL:	Operator-Kontrollblock-Liste
AKBL:	(Unter-) Auftrags-Kontrollblock-Liste
BKBL/TKBL:	Benutzerprozeß-/Transaktions-Kontrollblock-Liste
DBBL/FCBL:	Listen von Dateibeschreibungen für B- bzw A-Dateien
OFL:	Open-File-List

Bild 7.3: Kontrollstrukturen in einer Funktionseinheit

Zentrale Kontrollstruktur ist die OKB-Liste. Sie definiert die Aufträge an die Funktionseinheit. Ein OKB dient zugleich als Taskkontrollblock. Die OKBL enthält alle ablauffähigen Tasks sowie die wegen Unteraufträgen blockierten Tasks, angezeigt durch den Zustand des OKB, der angibt, wieviele Unteraufträge noch erfüllt werden müssen. Es gibt sowohl Kommunikationsaufträge, die in die

KOMLST des Kommunikationsteils aufgenommen und danach abgearbeitet werden, als auch Aufträge, die im allgemeinen von anderen Funktionseinheiten zu bearbeiten sind. Zu bearbeitende Aufträge werden aus den Warteschlangen Q_i von der Tasksteuerung entnommen, in die OKBL eingereiht und aktiviert.

Der OKB enthält folgende Informationen:

- Adresse des Auftraggebers (Port)
- Identifikation (im allgemeinen Benutzerprozeßidentifikation kombiniert mit Auftragsidentifikation)
- Auftragsbezeichnung
- auftragsabhängige Parameter
- Unterbrechungsmarke
- Zustand (Anzahl der nicht erledigten Unteraufträge)
- lokale Taskvariablen

Der Port als Adresse des Absenders ermöglicht, die notwendigen Ergebnisse und Rückmeldungen an den Auftraggeber zu übermitteln, die Identifikation eröffnet den Zugang zu den Transaktionsverwaltungsinformationen, die sowohl im übergreifenden Benutzerprozeßkontrollblock als auch im Transaktionskontrollblock abgelegt sind.

Mittels der Auftragsbezeichnung werden von der Tasksteuerung die zugehörigen Tasks aktiviert bzw. reaktiviert. Die auftragsabhängigen Parameter können auch in Form von Paketen an den eigentlichen OKB angeschlossen sein, so daß beliebig viele Parameter zulässig sind.

Die drei letzten Attribute sind Taskattribute, wobei lokale Taskvariablen taskspezifische Zwischenergebnisse enthalten. Die Unterbrechungsmarke gibt an, wo blockierte Tasks bei ihrer Reaktivierung wieder fortgesetzt werden.

Im weiteren Sinn können funktionseinheitenspezifische

Verwaltungsinformationen zu den Kontrollstrukturen gezählt werden. Für die ZU sind dabei an erster Stelle die Listen mit den Dateibeschreibungsblöcken für A- bzw. B-Dateien (DBBL bzw. FCBL) zu nennen, für die KO die Sperrtabellen und Wartelisten und für die KAT die OFL, die zur Kontrolle der Kopien von Dateibeschreibungsblöcken dient.

7.2.2. Funktionen zum Tasking

Die Implementierung von Tasking-Funktionen wird weniger in Aufrufen von FUNCTIONs oder SUBROUTINEs als in der Einhaltung fest vorgeschriebener Programmierkonventionen realisiert, die es gestatten, nach diesen Richtlinien programmierte SUBROUTINEs als Tasks zu betrachten.

7.2.2.1. Erzeugen

Eine Operator-Task ist erzeugt, wenn ein OKB (vgl. Bild 7.3) in die OKBL eingereiht ist und eine gültige Auftragsbezeichnung enthält, sowie mittels LISTE [1] (<OKB-adr> + PORTX), LISTE(<OKB-adr> + IDX) und LISTE(<OKB-adr> + IDX + 1) eine innerhalb der ZU eindeutige Identifikation vorliegt.
Die Task ist ablauffähig, wenn LISTE(<OKB-adr> + ZUST).EQ.Ø und LISTE(<OKB-adr> + UBMX) eine gültige Unterbrechungsmarke (integer*2) größer Ø enthält. PORTX, IDX, ZUSTX und UBMX sind Relativindizes mit deren Hilfe PORT (des Auftraggebers, z. B. BEA - Einheit), Identifikation des Auftrags (diese nimmt 2 Feldelemente ein und wird vom Auftraggeber vergeben), Zustand des Auftrags und Unterbrechungsmarke der Auftrags-Tasks adressiert werden.

7.2.2.2. Starten

Eine Task wird, wenn sie ablauffähig ist, von der Taskverwaltung der Ebene E4 gemäß ihrer Priorität gestartet. Ein direkter Start kann

---

1) Im Feld LISTE werden die Listenelemente untergebracht.

auch mit CALL vorgenommen werden.

7.2.2.3. Anhalten

Eine Task wird mittels folgender Maßnahmen angehalten:

- in LISTE(<OKB-adr> + ZUSTX) den Zustand entsprechend der Anzahl der zu bearbeitenden Unteraufträge ablegen
- in LISTE(<OKB-adr> + UBMX) die Marke der Stelle, wo der Prozeß angehalten werden soll, die sogenannte Unterbrechungsmarke hinterlegen
- lokale Taskvariablen retten
- RETURN

7.2.2.4. Fortsetzen

Eine Task wird von der Ablaufsteuerung festgesetzt, wenn sie ablauffähig, am weitesten vorne in der OKBL eingereiht ist und keine Aufgaben der Kommunikationsschicht E3 zu erledigen sind.

Damit eine Task überhaupt ablauffähig werden kann, nachdem sie sich angehalten hat, müssen die Unteraufträge (⟶ AKBs) und Kommunikationsaufträge, auf die die Task wartet, ausgeführt sein. Bei jeder Beendigung eines Unter- oder Kommunikationsauftrags wird LISTE(<OKB-adr> + ZUSTX) um 1 erniedrigt.

7.2.3. Taskfamilien

Alle Operatoren der logischen Dateiebene werden in den Funktionseinheiten als Task realisiert. So gibt es in der ZU z.B. Tasks (⟶ SUBROUTINEs gemäß Kap. 7.2.2) für READ, OPEN, BEGIN_TRANS usw., in der KAT z.B. für OPEN, DEF usw.

Insbesondere für die KAT werden z.B. beim Eintragen einer Schemadefinition in den Katalog bei der Abarbeitung von DEF ganze

Transaktionen der logischen Dateiebene ausgeführt. Ähnliches geschieht in der ZU selber, wenn Partitionsbeschreibungen fehlen und beschafft werden.

Mit der Einführung zusätzlicher Tasks, den internen Tasks, die formal wie Aufträge einer Funktionseinheit an sich selbst behandelt werden, ist eine logische Schachtelung von Taskaktivitäten möglich. Es entstehen mittels Identifikationen zusammengehaltene OKB-AKB-Ketten, die die Schachtelung repräsentieren. Insbesondere läßt sich hiermit auch eine an Tasks orientierte Segmentierung erzielen. Ganze Taskaktionen wie

```
BEGIN_TRANS
LOCK
READ
  .
  .
  .
READ
END_TRANS
```

zur Beschaffung von Partitionsbeschreibungen werden zu Tasks zusammengefaßt, die bei Bedarf relativ bequem erzeugt und aktiviert werden können.

7.3. Installation eines DISCO-Systems

In einem durch ein Kommunikationssystem verbundenen Rechnernetz (RPS /Häl79, Kos79/) werden die einzelnen Funktionseinheiten als Programme installiert.

7.3.1. Konfigurierung

Hierfür ist das Programm Masterinit, BMINIT, zuständig. Mit Hilfe eines Dialogteils kann am Bildschirm eine beliebige Konfiguration eines DISCO-Systems in dem zugrundeliegenden Rechnernetz definiert werden. Wenn das Programm BMINIT an jedem Rechner des Netzes

installiert ist, kann ausgehend von jedem beliebigen Knoten des Rechnernetzes ein neu konfiguriertes DISCO-System hochgefahren werden, sofern die erforderlichen Funktionseinheiten in gebundener, ladefähiger Form vorliegen.

An Stelle einer dialoggeführten Definition kann der Systemadministrator auch eine an dem jeweiligen Rechner hinterlegte Definition als Konfiguration definieren. Mit Hilfe des Dialogteils von Masterinit kann selbstverständlich jederzeit eine solche unter DSCOØ1 hinterlegte Standardkonfiguration umdefiniert werden.

7.3.2. Anlauf und Betrieb

Im Zuge der Konfigurierung werden von Masterinit die Konfigurationsdaten an die lokalen Knoteninit übermittelt (vgl. Bild 7.4).

Für den Start der Funktionseinheiten an einem Rechner ist das Programm Knoteninit, BKINIT, zuständig. Dieses startet und lädt, falls erforderlich, die an seinem Rechner vorgesehenen Funktionseinheiten, z.B. BEA, KO usw.

Bevor die einzelnen Funktionseinheiten selber betriebsbereit sind, müssen diese ihre konfigurationsabhängigen Parameter initialisieren und ihre Kommunikationskanäle zu den anderen Funktionseinheiten gemäß der globalen Konfiguration des DISCO- Systems aufbauen.

Diese Informationen werden für die Funktionseinheiten eines Rechners von Knoteninit bereitgestellt. Schnittstelle ist die Datei DSCØ2, auf die die einzelnen Funktionseinheiten zugreifen.

BKINIT bezieht diese für den lokalen Rechner erforderlichen Informationen über das RPS-Kommunikationssystem von Masterinit. Auftrags- und, wenn notwendig Datenkanäle, werden gemäß Bild 7.5 aufgebaut. Die Portnummer ist dabei eine Funktion aus Rechnernummer des Kommunikationspartners und dessen Komponententyp. Nach Initialisieren dieser Kanäle ist DISCO betriebsbereit und kann von mit Anschlußmoduln versehenen Programmen angesprochen werden.

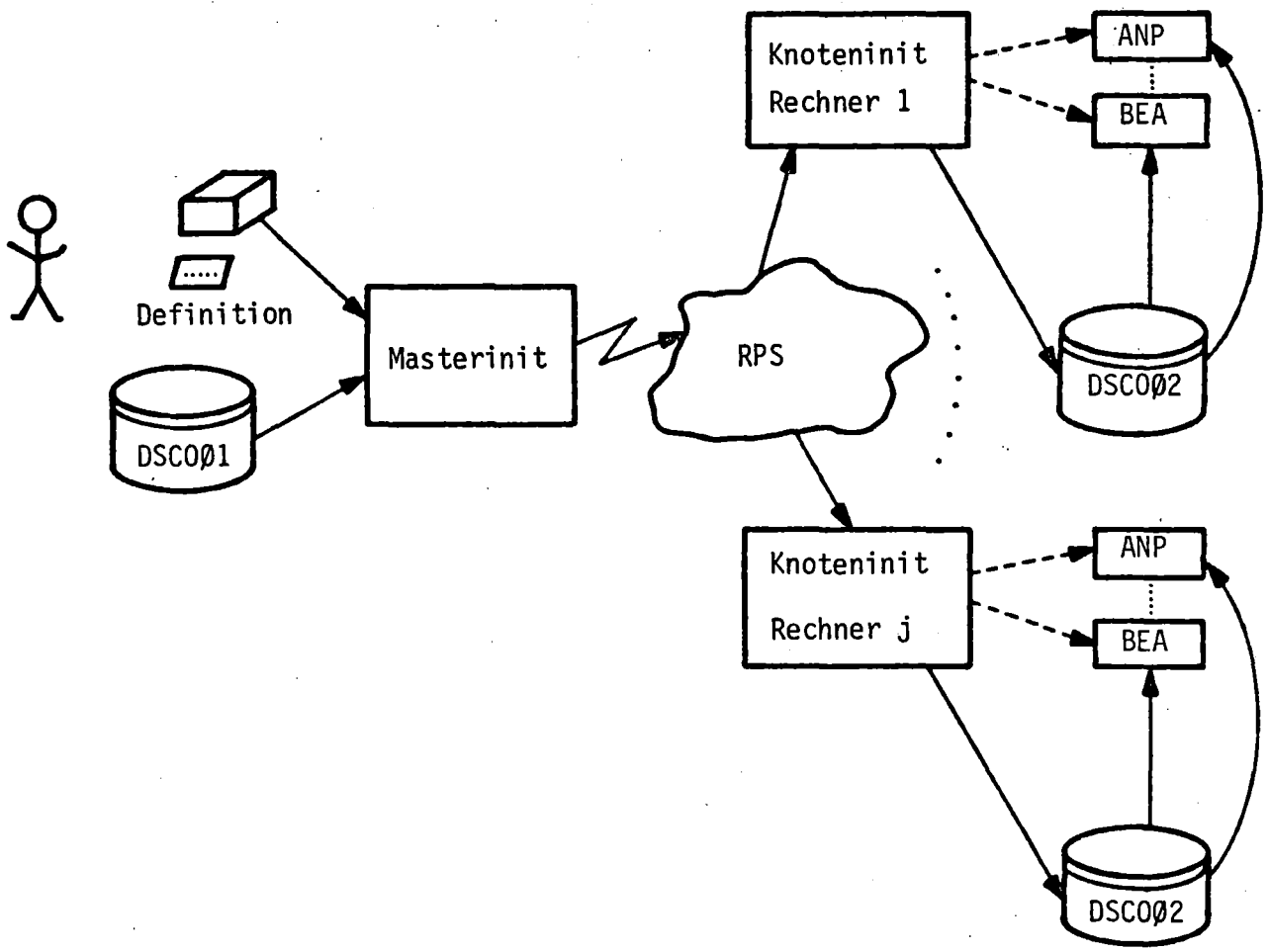


Bild 7.4: Schema einer Systeminitialisierung

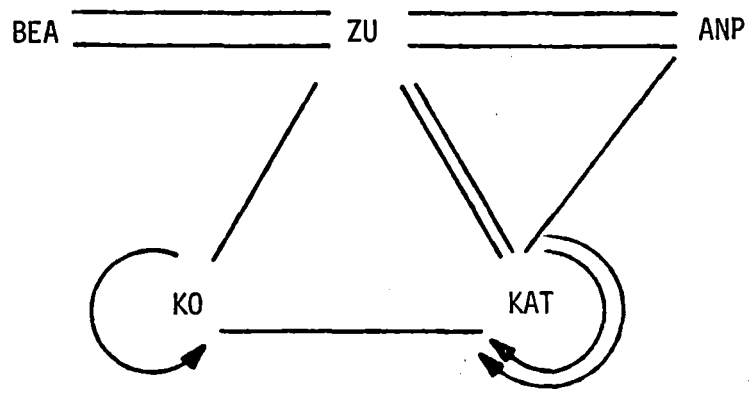


Bild 7.5: Auftrags- und Datenkanäle zwischen Funktionseinheiten

8. Abschließende Bemerkungen

Die Arbeiten in diesem Vorhaben hatten die Realisierng einer auf der logischen Satzebene einzuordnenden Schnittstelle für eine integrierte Datenhaltung in verteilten DV-Systemen zum Ziel. Das erarbeitete Konzept einer verteilten Dateiverwaltung ist sowohl als Basis für eine verteilte Datenbank als auch als eigenständiges System verwendbar.

Das System unterstützt ein breites Anwendungsspektrum. Die Verteilung der Daten ist für den Benutzer nicht sichtbar. Es ist jedoch möglich, zu Benutzergruppen Datenbestände zu assoziieren und darüber die Residenz von Datenbeständen für den optimalen Zugriff zu steuern. Es wird ein modernes Transaktionskonzept zur Sicherung der Konsistenz der Datenbestände und zur Unterstützung einer fehlertoleranten Systemführung verwendet. Mehrere Benutzer können parallel auf den Datenbeständen operieren. Die Verwaltung synchronisiert konkurrierende Zugriffe, eventuell auftretende Deadlocks zwischen Transaktionen werden erkannt und beseitigt.

Die Konzepte für die Verteilung von Daten und Funktionen bieten weit über das bisher bekannte Maß hinausgehende Anpassungsmöglichkeiten an unterschiedlichste Konfigurationen verteilter DV-Systeme. So können die Rechner des verteilten DV-Systems unterschiedliche Kapazität besitzen, z.B. ein Verbund von Kleinrechnern und mittleren Rechnern mit unterschiedlicher Verarbeitungs- und/oder Speicherkapazität; die Einbeziehung dedizierter Prozessoren für Dateiverwaltungsaufgaben ist ebenso möglich.

Aspekte der Zuverlässigkeit und Fehlertoleranz sind in der Struktur der verteilten Dateiverwaltung mitberücksichtigt. Über Sicherheitsklassen ist der Grad der Redundanz von Datenbeständen im verteilten DV-System steuerbar. Das Stellvertreterkonzept für Funktionseinheiten bildet die Basis für die automatische Übernahme des Aufgabenbereiches einer ausgefallenen Funktionseinheit durch die restlichen funktionsfähigen Funktionseinheiten. Zusätzliche Clusterstützung bieten Transaktionsjournale. Die Maßnahmen bei

Ausfall/Wiederanlauf von Daten/Funktionseinheiten sind dabei anwendungsspezifisch in das System zu integrieren.

Eine Prototypimplementierung der verteilten Dateiverwaltung erfolgte für ein aus Prozeßrechnern (Typ Siemens 300) bestehendes verteiltes DV-System und steht kurz vor dem Abschluß.

Zu Beginn der Implementierungsarbeiten stand noch kein für unsere Zwecke geeignetes Nachrichtentransportsystem (Software) zur Verfügung. Es mußte daher eine Eigenentwicklung verwendet werden.

Es ist geplant, mittels Experimenten mit dieser Pilotimplementierung Kenntnisse über das Leistungsverhalten verteilter Datenverwaltungssysteme unter besonderer Berücksichtigung des Fehlertoleranzaspektes zu gewinnen. Weitergehende Arbeiten werden neuere Konzepte für Nachrichtentransportsysteme in das vorgestellte Konzept einer integrierten Datenhaltung für verteilte DV-Systeme einbeziehen. Als Ergebnis dieser Arbeiten wird eine verteilte Dateiverwaltung mit einer umfangreicheren Schnittstelle zum Benutzer sowie einer verbesserten Fehlertoleranz unter Ausnutzung von Sicherungskonzepten lokaler Dateiverwaltungssysteme erwartet.

Anhang A

Syntax der Aufrufe der Operatoren, die an der Benutzerschnittstelle von DISCO zur Verfügung stehen.

```
Aufruf an DISCO  :: = CALL <Op-Term>
<Op-Term>        :: =

READ   (<RTC>, <Datnam>, <SN>, <Benvar> [,<EV>] ) |
READS  (<RTC>, <Datnam>, <Richtung>, <Benvar> [,<EV>] ) |
WRITE  (<RTC>, <Datnam>, <SN>, <Benvar> [,<EV>] ) |
WRITES (<RTC>, <Datnam>,       <Benvar> [,<EV>] ) |
INB    (<RTC>, <Datnam>, <SN>, <Benvar> [,<EV>] ) |
INBS   (<RTC>, <Datnam>,       <Benvar> [,<EV>] ) |
REMB   (<RTC>, <Datnam>, <SN> [,<EV>] ) |
NEW    (<RTC>, <Datnam>, <SN> [ <User>] ) |
SETCB  (<RTC>, <Datnam>, <SN>) |
GETCB  (<RTC>, <Datnam>, <SN>) |

GET    (<RTC>, <Datnam>, <Key>, <Benvar> [,<EV>] ) |
GETS   (<RTC>, <Datnam>, <Richtung>, <Benvar> [,<EV>] ) |
PUT    (<RTC>, <Datnam>, <Benvar>, <Lng> [,<EV>] ) |
PUTS   (<RTC>, <Datnam>, <Benvar>, <Lng> [,<EV>] ) |
INA    (<RTC>, <Datnam>, <Benvar>, <Lng> [,<EV>] ) |
REMA   (<RTC>, <Datnam>, <Key>   [,<EV>] ) |
SETCA  (<RTC>, <Datnam>, <Key>) |
SETMAX (<RTC>, <Datnam>) |

LOGON  (<RTC>)|LOGOFF |
BTRANS (<RTC>) |
ETRANS (<RTC>) |
UNDO   (<RTC>) |
OPEN   (<RTC>, <Datnam> [,<Paßwort> ] ) |
CLOSE  (<RTC>, <Datnam>) |
LOCK   (<RTC>, <Datnam>, <Typ> [ ,<Satzn> ] ) |
UNLOCK (<RTC>, <Datnam>) |
WAIT   (<EV>) |
```

```
DEFB   (<RTC>, <Datnam>, <PA1>, <PA2>, ..., <PA7>) |
DEFA   (<RTC>, <Datnam>, <PA1>, <PA2>, ..., <PA7>) |
MAP    (<RTC>, <Datnam>, <Partition>, <PA1>, <PA7>) |
CREATE (<RTC>, <Datnam>) |
DELETE (<RTC>, <Datnam>) |
UNDEF  (<RTC>, <Datnam>) |
DEFUG  (<RTC>, <User>, <PA7>) |
UDEFUG (<RTC>, <User>)

<RTC>         :: = int
<Datnam>      :: = dim
<Partition>   :: = int
<SN>          :: = int
<Benvar>      :: = dim
<EV>          :: = bool
<Richtung>    :: = int
<User>        :: = int
<Key>         :: = dim
<Lng>         :: = int
<Paßwort>     :: = dim
<Typ>         :: = int
<Satzn>       :: = dim

<PA1>         :: = int
<PA2>         :: = int
<PA3>         :: = dim
<PA4>         :: = int
<PA5>         :: = int
<PA6>         :: = int
<PA7>         :: = dim

int   :: = integer*2 Variable oder Ausdruck gemäß FORTRAN-Konvention
dim   :: = Feld oder Literal
bool  :: = logical*2
```

Anhang B

Interaktive Terminalschnittstelle zum Aufruf von DISCO-Operatoren

Mit Hilfe der interaktiven Schnittstelle kann ein Endbenutzer die Operatoren des Systems DISCO vom Bildschirm aus aufrufen. Er kann sich damit insbesondere die Ergebnisse von Leseabfragen direkt auf dem Bildschirm ausgeben lassen bzw. neue Dateneingaben direkt vom Bildschirm aus vornehmen, ohne erst spezielle Programme zu schreiben.

Das System zeigt durch die Meldung READY, daß es bereit ist und eine Anfrage eingegeben werden kann. Falls ein Auftrag nicht erfolgreich durchgeführt werden konnte, wird eine Fehlermeldung ausgegeben. Durch nachfolgendes READY wird angezeigt, daß eine neue Eingabe möglich ist.

Im folgenden wird die Syntax der Terminaleingaben aufgelistet und gegebenenfalls durch Beispiele erläutert. Die Semantik entspricht der der entsprechenden Operatoren von DISCO (siehe Kap. 3.1.2)

Operatoren zum Beginn bzw. Abschluß einer Sitzung

: pnr : LOGON

Mit diesem Kommando wird der Dialog mit DISCO eröffnet; "pnr" gibt dabei die Programmnummer an, unter der das Schnittstellenprogramm dem Betriebssystem des Rechners bekannt ist (nur bei Siemens-Rechnern).

LOGOFF

Der Dialog wrd beendet. Eine neue Sitzung kann mit LOGON eröffnet werden.

Operatoren zur Transaktionsbearbeitung

{BTRANS / BT} Beginn einer Transaktion

{ETRANS / ET} Ende einer Transaktion

UNDO Die laufende Transaktion wird abgebrochen.

Operatoren zur Dateimanipulation

Vorbemerkung

1) Bei einigen Operatoren kann die Angabe des Dateinamen entfallen. In solchen Fällen wird jeweils der zuletzt angegebene Dateiname substituiert. Zu Beginn einer Sitzung ist der Parameter undefiniert!

2) Bei den Schreiboperatoren WRITE, WRITES, PUT, PUTS, INA, INB wird der jeweils zu schreibende Satz in der nachfolgenden Zeile übergeben.

3) Alle Operatoren werden synchron bearbeitet!

Manipulation von B-Dateien

$$\left\{\begin{matrix}\text{READS}\\ \text{RS}\end{matrix}\right\} \; [\langle\text{Datnam}\rangle \; [\; \left\{\begin{matrix}\text{V}\\ \text{R}\end{matrix}\right\} \;] \;]$$

Fehlt die Richtungsangabe (V/R), so wird vorwärts gelesen.

Beispiel:
Gültige Eingaben sind:

RS : = Lesen des nachfolgenden Satzes aus der Datei, auf die zuletzt zugegriffen wurde

RS ABC : = Lesen des nachfolgenden Satzes aus der Datei ABC
READS XYZ : = Lesen des vorhergehenden Satzes aus der Datei XYZ

$$\left\{\begin{matrix}\text{READ}\\ \text{R}\end{matrix}\right\} \quad \langle\text{Datnam}\rangle \quad \langle\text{SN}\rangle$$

Beispiel:

R D1 25 : = Lesen von Satznr 25 aus der Datei D1

{WRITES / WS} [<Datnam>]

{WRITE / W} <Datnam> <SN>

INB [<Datnam> [<SN>]]

Bei Angabe sämtlicher Parameter entspricht dieser Operator dem entsprechenden DISCO-Operator. Wird keine Satznummer angegeben, wird der DISCO-Operator INBS verwendet.

REMB <Datnam> <SN>

NEW [<Datnam> [<User>]]

Auf dem Bildschirm erscheint die Ausgabe: "Satznummer : ....."

SETCB <Datnam> {<SN> / FG / ST}

GETCB [<Datnam>]

Auf dem Bildschirm erscheint die Ausgabe:
"Satzzeiger zeigt auf Satznummer : ...."

Manipulation von A-Dateien

GETS [<Datnam>[{V / R}]]

Bearbeitung erfolgt analog READS

GET <Datnam> <Key>

PUTS [<Datnam>]

PUT [<Datnam>]

INA [<Datnam>]

REMA <Datnam> <Key>

SETCA <Datnam> <Key>

SETMAX [<Datnam>]

Dateitypunabhängige Operatoren:

OPEN <Datnam>[/[<Leseschl>]/[<Schreibschl>]/]

Beispiel: OPEN ABC /123/uvw/
OPEN XY //ab/

CLOSE [<Datnam>]

{LOCK / L} [<Datnam> [[{X / S}][SN]$_{1}^{14}$]]

Wenn kein Sperrtyp angegeben ist, wird Typ = S substituiert, wenn keine Satznummern angegeben sind, wird die ganze Datei gesperrt.

Beispiel:

LOCK XYZ : = Die ganze Datei XYZ wird mit Sperrtyp S gesperrt
LOCK ABC X 15,23 : = Die Sätze 15 und 23 der Datei ABC
werden mit Sperrtyp X gesperrt.

UNLOCK [<Datnam>]
CREATE <Datnam>
{DELETE / DEL} [<Datnam>]

Operatoren zur Dateidefinition

Hinweis:
Angaben bei schließender eckiger Klammer (]$_{1}^{i}$) bedeuten, daß der Klammerinhalt mindestens 1, höchstens i mal angegeben werden kann!

DEFA <Datnam> <Satzlänge>, <Abschätzung des Umfangs>
/[<Leseschl>]/[<Schreibschl>]/ <Sicherheitklasse>,
<Schlüsselanfang>, <Schlüssellänge>,[<Benutzergruppe>]$_{1}^{4}$

DEFB <Datnam> <Satzlänge>, <Abschätzung des Umfangs>
/[<Leseschl>]/[<Schreibschl>]/ <Sicherheitsklasse>
<Anzahl Sätze pro Partition>, <Anzahl Cluster>
[: <Anzahl Partition>,[Benutzergruppe]$_{1}^{3}$]$_{1}^{\text{Anzahl Cluster}}$

UNDEF <Datnam>

MAP <Datnam> <Partition> <Anzahl Kopien> , [<Dateiname> <Rechnernr>] $_{1}^{\text{Anzahl Kopien}}$

DEFGS <Dateiname> <Rechnernr>, <Umfang>

UDEFGS <Dateiname> <Rechnernr>

UDEFUG <User>

DEFUG [<Rechnernr>]$_{1}^{3}$

Auf dem Bildschirm erscheint die Meldung "Neue Benutzergruppe : ...."

Hilfsfunktion

Durch Eingabe des Operators HELP kann sich der Benutzer Informationen über das System, die Syntax und die Semantik der Operatoren auf dem Bildschirm auflisten lassen.

Absetzen von Kommandos an das Betriebssystem

Nach der Bearbeitung des LOGON-Operators ist die Bildschirmeingabe programmgesteuert. Der Benutzer kann nur noch gültige Kommandos eingeben, andernfalls erscheint die Meldung: "Falsche Eingabe". Durch ein vorangehendes Steuerzeichen & wird es dem Benutzer ermöglicht, Kommandos an das Betriebssystem oder andere Programme abzusetzen, ohne seine begonnene Sitzung zu beenden.

Beispiel:

```
&/STRT : P5;    : = Das Programm Nr. 5 wird gestartet
&:7:Ende;       : = Programm Nr. 7 wird mit der Nachricht "Ende"
                    beendet
```

Literatur

/BCD78/ Breitwieser, H., Conrad, U., Drobnik, O., Keil, C., Kersten, U.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1978

/BKD78/ Breitwieser, H., Kersten, U., Drobnik, O.: "DISCO: A distributed file management system for heterogenous computer networks", International Conference of Management of Data, ACM Italian Chapter, Milan, June 29-30, 1978, Conference Proceedings, pp. 157-173

/Bre78/ Breitwieser, H.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1978

/BrK79/ Breitwieser, H., Kersten, U.: "Transaction and Catalog Management of the Distributed File Managment System DISCO", Proceedings of the 5th International Conference on Very Large Data Bases, Rio de Janeiro, Brazil, October 1979, pp. 340-350

/Cas72/ Casey, R.G.: "Allocation of copies of files in an information network", Proceedings AFIPS 1972 SJCC, Vol. 40, AFIPS Press, Montsvale, N.J. pp. 617-626

/Chu69/ Chu, W.W.: "Optimal file allocation in a multiple computer system", IEEE Trans. on Computers, C-18, 10. Oct. 1969, pp. 885-889

/Com79/ Comer, D.: "The ubiquitous B-Tree", ACM Computing Surveys, Vol. 11, No 2, June 1979

/Dro77/ Drobnik, O.: "Verfahren zur Sicherung der operationalen Integrität in verteilten Datenbasen bei dezentraler Kontrollstruktur", Dissertation, Universität Karlsruhe, 1977

/Esw76/ Eswaran, K.P. et al.: "On the notions of consistency and predicate locks in a data base system", CACM, Vol. 19, No. 11, November 1976

/Häl79/ Hälsig, M.: "Nachrichtentransport im RPS-System", Siemens-Anwenderkreis, 10. Jahrestagung, Berlin 1979

/HBD78/ Holler, E., Breitwieser, H., Drobnik, O., Keil, C., Kersten, U., Knöpker, R., Krieger, J.: "DISCO: Eine Datenbankarchitektur für verteilte DV-Systeme auf der Basis logischer Dateien", Kernforschungszentrum Karlsruhe, Bericht 2851, März 1978

/Ker79a/ Kersten, U.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1979

/Ker79b/ Kersten, U.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1979

/Kos79/ Kossmann, P.: "Filetransfer, Telexfunktionen und Fernbedienung im RPS-System", Siemens-Anwenderkreis, 10. Jahrestagung, Berlin 1979

/LeM75/ Levin, K.D., Morgan, H.L.: "Optimizing distributed databases", National Computer Conference 1975, AFIPS Conference Proceedings, Vol.44, pp. 473-478

/Loc78/ Lockemann, P.: "Verteilte Datenbanken - wissenschaftlich und technisch gelöst", 3. Internationaler Kongreß für Datenverarbeitung, IKD Berlin, 1978, 12-15 Sept. 1978, Workshop Reihe VIII, Proceedings AMK-Berlin, S. 1-11

/Mai78/ Mailänder, R.: "Entwurf und Implementierung eines dezentral organisierten Sperrmechanismus' für verteilte Datenhaltung in Kleinrechnernetzen", Diplomarbeit, Universität Karlsruhe, 1978

/Mai79/ Mailänder, R.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1979

/Mai80/ Mailänder, R.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1980

/MaR76/ Mahmoud, S., Riordon, J.S.: "Optimal allocation of resources in distributed information networks", ACM Transaction on Database Systems 1, March 1rst 1976, pp. 66-78

/Mül80/ Müller, M.: "Ausfall und Wiederanlauf in einem verteilten Dateiverwaltungssystem", Diplomarbeit, Universität Karlsruhe, 1980

/ORG79/ Siemens Programm-Service: "Organisationsprogramm, Beschreibung für ORG 300-PV", Best. Nr. P7 1100-B0340-X-X-35

/ScD80/ Schlageter, G., Dadam, P.: "Reconstruction of consistent global states in distributed databases", Proceedings of the International Symposium on Distributed Data Bases, Paris, France, March 12-14, 1980, ed. Delobel, C., Litwin, W., North Holland Publishing Company

/See79/ Seemann, W.: Kernforschungszentrum Karlsruhe, unveröffentlichter Bericht, 1979

/WiT76/ Widdoes, C., Trabb-Pardo, L.: "Distributed Databases at Stanford", in C.W. Strevens, University of Warwick, Computer Center Report, 1976