in: Informatik - Forschung und Entwicklung, Springer-Verlag, 1990, s. 127-143.

Transaktionssysteme in Workstation/Server-Umgebungen

T. Härder, K. Meyer-Wegener
Universität Kaiserslautern

Überblick

Transaktionssysteme wurden ursprünglich als zentrale Systeme konzipiert, um im Auftrag von Endbenutzern einfache, vorgeplante Aufgaben sicher und effizient auszuführen. Heutige Systemarchitekturen unterstützen oft (systemseitig) verteilte Verarbeitung und verteilte Datenhaltung im Auftrag einer Transaktion. Künftige Konzepte sollten benutzerseitig den Einsatz von Workstations und systemseitig die Verfügbarkeit von heterogenen Servern vorsehen. Dadurch lassen sich das Verarbeitungspotential moderner Workstations für die Transaktionsverarbeitung nutzen und so komfortablere Schnittstellen/Benutzerdialoge sowie komplexere Abläufe der Transaktion ermöglichen. In solchen Systemen bedient sich eine Transaktion mehrerer unabhängiger Server, so daß eine verteilte Transaktionsverarbeitung zwischen Workstation und Servern abgewickelt werden muß. Dafür werden mögliche Verarbeitungsmodelle und ihre Implikationen auf das Transaktionskonzept untersucht.

Abstract

Transaction-processing systems were originally designed as centralized systems to reliably and efficiently execute simple preplanned tasks on behalf of end-users. Today's system architectures often support distributed processing and distributed data management on behalf of a transaction. Future concepts should consider the use of workstations at the user's side and the availability of heterogeneous servers at the system's side. Thereby, the processing power of modern workstations can be employed for transaction processing, and comfortable interfaces as well as user dialogues are rendered possible. In systems like these a transaction makes use of several independent servers, demanding distributed transaction processing among the workstation and the servers. For this case, applicable processing models and their implications for the transaction concept are investigated.

1. Verteilung von Betriebsmitteln und lokale Rechnerleistung

Durch den Einsatz von Transaktionssystemen wird eine dialogorientierte Sachbearbeitung direkt am Arbeitsplatz ermöglicht, wobei das System den Benutzer über Bildschirmformulare führt und bei der Auftragsformulierung unterstützt. An dieser problemorientierten Benutzerschnittstelle ist kein Bezug auf Programme oder Dateien erforderlich; Verarbeitungswünsche, d.h. die Ausführung eines Arbeitsvorgangs in der Anwendungswelt, werden vielmehr mit Hilfe von *anwendungsbezogenen Funktionen* formuliert, die über einen speziellen Namen *(Transaktionscode (TAC))* angesprochen oder über Menüs ausgewählt werden können. Zur Abwicklung dieser Funktionen verwaltet das Transaktionssystem eine Menge von auf die jeweilige Anwendung zugeschnittenen Programmen, die Arbeitsvorgänge oder einzelne Teilschritte in ihnen verkörpern. Über diese sogenannten *Transaktionsprogramme (TAPs)* sind alle Funktionen der Anwendung bis ins Detail rechnerintern vorgeplant (canned transactions); die "parametrischen Benutzer" versorgen sie nur noch mit aktuellen Parametern, die dem TAC mitgegeben werden.

Transaktionsprogramme haben in der Regel zur Erfüllung ihrer Aufgaben Zugriff auf eine große Datenbasis betrieblicher Daten. Weiterhin bringt es die Betriebscharakteristik eines Transaktionssystems mit sich, daß sie über unterschiedliche Leitungen und Netze mit einer Vielzahl von Terminals kommunizieren müssen. Oft rufen viele Benutzer gleichzeitig dieselbe Funktion (mit jeweils eigenen aktuellen Parametern) auf, so daß im Prinzip ein TAP "hochgradig parallel" benutzt werden muß [12]. Die eben skizzierten Probleme veranlaßten die Entwicklung sogenannter *DB/DC-Systeme*, die man als Kernstück eines Transaktionssystems auffassen kann. Die DC-Komponente (TP-Monitor) - für die Nachrichten- und TAP-Verwaltung zuständig - sorgt für Kommunikations- und Kontrollunabhängigkeit der TAPs, d.h., ein solches Programm kann unabhängig von Übertragungsverfahren und Terminaltyp durch das Konzept des virtuellen Terminals mit der Außenwelt kommunizieren und kann ohne Beachtung von Aspekten seiner Mehrfachbenutzung entworfen werden. Die DB-Komponente (DBS) dagegen garantiert einen hohen Grad an Datenunabhängigkeit für die DB-Zugriffe der TAPs (Sichtkonzept) und isoliert sie von allen Aspekten der Synchronisation und des Fehlerfalls (Logging/Recovery) durch das Transaktionskonzept [14] (Kontrollunabhängigkeit bei Zugriffen auf gemeinsame Daten).

Heute werden Transaktionssysteme typischerweise in Anwendungen eingesetzt, in denen eine große Anzahl von Auskunft-, Buchungs- oder Datenerfassungsvorgängen anfallen, also beispielsweise als Systeme zur Kontenbuchung, Platzreservierung oder Bestellabwicklung. Solche Systeme sind in der Regel als zentrale Systeme aufgebaut, wobei das Datenbanksystem und der TP-Monitor mit allen verfügbaren TAPs auf einem Rechner oder einem eng gekoppelten Mehrprozessorsystem ablaufen und alle Dienste für die angeschlossenen Arbeitsplätze erbringen. Datenhaltung sowie system- und anwendungsbezogene Verarbeitung erfolgen also ausschließlich zentral. Heute haben bereits viele Transaktionssysteme über 20000, einige sogar mehr als 100000 Terminals und 1000 Magnetplatten. Die höchsten Durchsatzforderungen für diese Systeme liegen im Bereich von 1000 Transaktionen pro Sekunde und darüber [9].

Neue, in den letzten Jahren herausgebildete Hardware- und Systemkonzepte erschließen aussichtsreiche Einsatzmöglichkeiten für andere, vor allem *dezentralisierte Organisationsformen* bei Transaktionssystemen. Dabei geht es insbesondere darum, die Leistungsfähigkeit heutiger/ künftiger Workstations optimal mit den Erfordernissen der betrieblichen Datenhaltung und der zentralen Dienste in einem Transaktionssystem zu verbinden. Dezentrale Konzepte unterstützen oft vorrangige Wünsche der Organisation [8] wie solche nach

- lokaler Autonomie bei der Kontrolle über die "eigenen" operationalen Daten
- effektiveren Dienstleistungen durch eigene Verwaltung der Systeme
- größerer Flexibilität hinsichtlich Funktionsausweitung und Kapazitätswachstum.

Aber auch von der technischen Seite gibt es gewichtige Gründe, ein großes Transaktionssystem zu dezentralisieren (unbeschränkte Verarbeitungskapazität, Verfügbarkeit, Verarbeitung "nahe am Arbeitsplatz" mit Reduktion der Antwortzeit u.a.).

Natürlich gehen mit der Dezentralisierung auch einige schwierige Probleme einher. Wenn die betriebliche Datenbasis verteilt ist, sind erhöhte Aufwendungen für die Konsistenzerhaltung der Daten (semantische Integritätskontrolle, Synchronisation, Logging/Recovery) erforderlich. Auch ist es meist nicht einfach, die Daten in geeigneter Weise zu partitionieren, um alle Ansprüche hinsichtlich Effizienz und Lokalität der Zugriffe zu befriedigen.

Jedoch wird jetzt auch immer häufiger die Forderung erhoben, im Rahmen von Transaktionssystemen selbständige Systeme mit Daten verschiedener Organisationen zu integrieren. Beispielsweise existieren gewachsene Reisebuchungssysteme mit heterogenen Datenbeständen, die weder vereinheitlicht werden können noch sollen. Trotzdem müssen z.B. aus Konkurrenzgründen alle Dienstleistungen in einem Anwendungssystem angeboten werden können. Künftige Transaktionssysteme sollten auch eine Organisationsform für solche Anforderungen bieten. Neben der *Zusammenfassung von existierenden Funktionen* unterstützt sie auch die *Entwicklung neuer, übergreifender Funktionen*.

Mit der Verfügbarkeit von *Workstations* ist eine enorme Rechnerleistung für die Transaktionsverarbeitung direkt am Arbeitsplatz vorhanden. Solche Workstations können mit eigener Plattenkapazität, Hochleistungsgraphik und ggf. mit weiteren Spezialfunktionen ausgerüstet sein. Sie besitzen breitbandige Netzanschlüsse nicht nur für lokale Netze, sondern künftig auch für Weitverkehrsnetze (ISDN), und können neben Daten auch Texte, Bilder und Sprache übertragen und handhaben. Ihre Leistungsmerkmale lassen sich schon heute mit jeweils mehr als 10 MIPS Verarbeitungsleistung, 10 MBytes Hauptspeicherkapazität und 10 MBytes/sec Übertragungsleistung des Netzes charakterisieren. Die Verfügbarkeit solcher Ressourcen wird eine Verlagerung von möglichst großen Anteilen der aktuellen TA-Verarbeitung in die Workstation nahelegen. Herkömmliche TA-Abläufe werden jedoch ihre Verarbeitungskapazität nicht auslasten können, so daß ihre frei verfügbaren Reserven weit komplexere Funktionsabläufe "zum Nulltarif" erlauben.

Bei der Entwicklung künftiger Transaktionssysteme sollte man von einer "typischen" Systemgröße von $\leq 10^4$ Workstations und etwa 10-100 Server-Rechnern, die gemeinsame Dienste wie Datenhaltung, Archivierung usw. anbieten, ausgehen. Das herausragende Entwurfsproblem für Transaktionssysteme hierbei ist es, die lokale TA-Verarbeitung auf den Workstations mit den Eigenarten und Anforderungen der (ggf. verteilten) DB-Verarbeitung auf den Servern abzustimmen und zu optimieren, wobei vor allem das Leistungsverhalten jeder einzelnen Benutzerfunktion verbessert werden soll. Die Zusicherungen und Konsistenzgarantien des Transaktionskonzeptes sind natürlich zu gewährleisten.

In diesem Aufsatz geht es nicht um eine Ideengeschichte der Entwicklung von Transaktionssystemen, sondern eher um eine neue Anwendung bekannter Konzepte und eine Diskussion der sich daraus ergebenden Konsequenzen. Dazu führen wir zunächst ein allgemeingültiges Schichtenmodell für Transaktionssysteme ein, das sowohl zentrale als auch verteilte Systeme in einheitlicher Weise beschreibt. In Kap. 3 wird der Ablauf einer Transaktion in einer Workstation/Server-Umgebung erörtert, wobei vor allem die Möglichkeiten der Anbindung von Serverfunktionen an die lokal ablaufenden Anwendungsprogramme und die erforderlichen Interaktionsmuster untersucht werden. Da sich solche Transaktionen über mehrere Dialogschritte erstrecken und dadurch über längere Zeit Betriebsmittel (Daten) binden, werden neue Anforderungen an die DB-Synchronisation und -Recovery herangetragen. Die verschiedenen Aspekte, die bei der Realisierung eines geeigneten Transaktionskonzeptes zu berücksichtigen sind, werden in Kap. 4 diskutiert, bevor wir schließlich unsere Schlußfolgerungen ziehen und einen Ausblick geben.

2. Ein Schichtenmodell für ein Transaktionssystem

Frühe Transaktionssysteme wurden ausschließlich für den Einsatz in zentralen Rechnerumgebungen entwickelt [19, 15]; auch später berücksichtigten nur wenige Systementwürfe von vorneherein die Möglichkeiten einer verteilten Systemumgebung [3, 17]. Am Arbeitsplatz des Benutzers ist in der Regel nur ein "dummes" Terminal als Schnittstelle zu solchen Systemen verfügbar, über das lediglich die formulargestützten Ein-/Ausgaben erfolgen.

2.1 Zentrale Verarbeitung und zentrale Datenhaltung

Um unsere Fragestellung genauer untersuchen zu können, muß zunächst ein *Architekturmodell* für solche Transaktionssysteme eingeführt werden. Im Rahmen unserer Diskussion ist vor allem die Analyse der Aufrufhierarchie bei der Transaktionsaktivierung und -bearbeitung nützlich, die in natürlicher Weise zu einem Systemmodell führt, das aus mehreren hierarchisch angeordneten Schichten besteht und DB- und DC-Komponenten sowie TAPs umfaßt (Bild 1). Dabei wird davon ausgegangen, daß die Terminals (relativ) direkt über feste Leitungen, Konzentratoren oder ein LAN an das zentrale Transaktionssystem angebunden sind. Die *oberste Systemschicht* ist dann für alle Aufgaben der Ein-/Ausgabe zu den Terminals verantwortlich (Kommunikationsunabhängigkeit), wobei eine Reihe von Betriebssystemdiensten genutzt werden. Der Begriff Kommunikationssystem deutet darauf hin, daß hier alle Aufgaben der externen Kommunikation abgewickelt werden. Es organisiert neben der Datenübertragung die Ein-/Ausgabewarteschlangen für die Terminalnachrichten. Eingabenachrichten werden vom TP-Monitor übernommen, analysiert und der eigentlichen Verarbeitung zugeführt. Über den TAC wird erkannt, welche Funktion auszuführen ist. Dazu ist ggf. für das entsprechende TAP eine Laufzeitumgebung (laden, initialisieren, Speicherplatz zuordnen) zu schaffen, bevor es die Ablaufkontrolle erhält.

Schicht 2 wird gebildet von der Menge der vom Transaktionssystem verwalteten TAPs, die quasi sandwich-artig vom TP-Monitor umschlossen werden. In der Aufrufhierarchie der Transaktionsbearbeitung übernimmt das ausgewählte TAP die Eingabenachricht und beginnt nach Routineprüfungen die eigentliche Verarbeitung, die durch eine Reihe von Datei- oder DB-Zugriffen unterstützt wird. Der TP-Monitor in *Schicht 3* ist dabei vor allem verantwortlich für die Koordination des Mehrbenutzer-Betriebes (Multi-Tasking [12]), wobei aus der Sicht des TAP trotz einer Vielzahl von (gleichen und parallelen) TAP-Aktivierungen eine Kontrollunabhängigkeit erzielt wird. Weiterhin registriert er alle Aufrufe an die Datenhaltung (aus Gründen der Fehlerbehandlung) und reicht sie entweder an das Dateisystem oder an das DBS weiter.

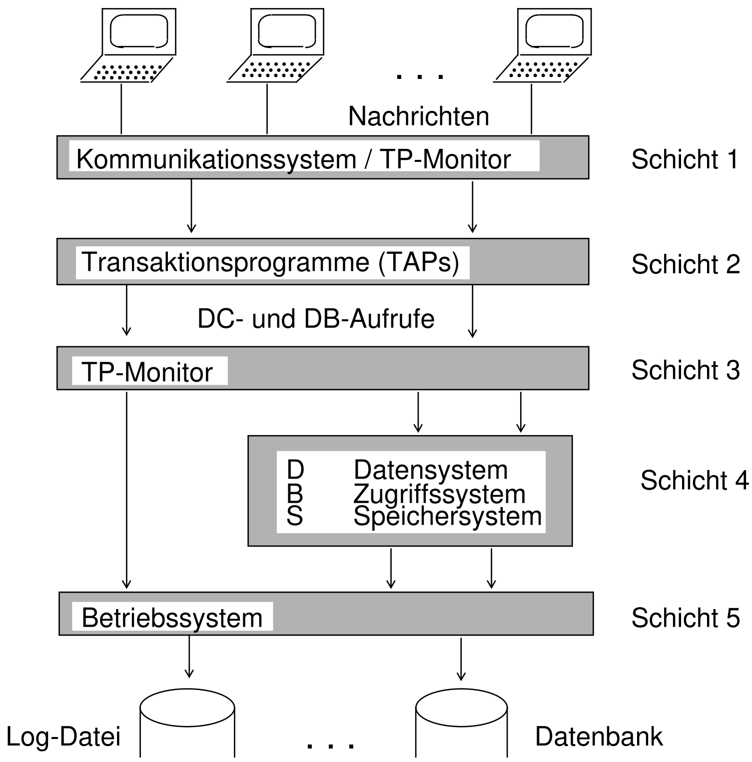

Bild 1: Schichtenmodell eines Transaktionssystems

Das DBS ist selbst ein komplexes System, bei dem man je nach Detaillierungsgrad der Betrachtung mehrere Ebenen unterscheiden kann. Für unsere Diskussion ist jedoch Schicht 4 ebenso wie Schicht 5, die Funktionen eines Dateisystems anbietet, nachrangig.

Das in Bild 1 gezeigte Schichtenmodell verdeutlicht nur die statischen Aspekte der Aufrufhierarchie. Für ihre Realisierung ist ein breites Spektrum von Prozeß- und Taskzuordnungen denkbar [13]; eine konkrete Konfiguration bestimmt dann den spezifischen Aufwand für die Interprozeßkommunikation (DC-Operationen, DBS-Aufrufe, 2-Phasen-Commit), legt aber gleichzeitig auch den Grad an Prozeßisolation (Schutzbedürfnis) sowie weitere Systemaspekte wie Betriebssystemunabhängigkeit (Portabilität) fest.

Die *Dynamik* der Transaktionsverarbeitung soll hier nur kurz am Beispiel der Kontenbuchung skizziert werden [13]:

```
BUCHUNG:
   BEGIN-DC;
   READ MESSAGE FROM TERMINAL;                                  {100 Bytes}
   EXTRACT KONTONR, DELTA, SCHALTERNR, ZWEIGSTELLENNR
   BEGIN_TRANSACTION;
      READ KONTO   KEY IS KONTONR;
      IF NOT FOUND OR KONTOSTAND + DELTA < 0 THEN
         DO
            prepare negative response;
            RESTORE_TRANSACTION
         END
      ELSE
         DO
            KONTOSTAND := KONTOSTAND + DELTA;
            REWRITE KONTO   KEY IS KONTONR;
            prepare record BUCHUNGEN;
            WRITE BUCHUNGEN;
            READ SCHALTER   KEY IS SCHALTERNR;
            KASSENSTAND := KASSENSTAND + DELTA;
            REWRITE SCHALTER   KEY IS SCHALTERNR;
            READ ZWEIGSTELLE   KEY IS ZWEIGSTELLENNR;
            GELDBESTAND := GELDBESTAND + DELTA;
            REWRITE ZWEIGSTELLE   KEY IS ZWEIGSTELLENNR;
            prepare response;
            COMMIT;
         END
      WRITE MESSAGE TO TERMINAL;                                {200 Bytes}
   END-DC;
```

Dieses TAP enthält 5 DC- und 9 DB-Operationen, was charakteristisch für eine kurze Transaktion ist. Zur Abarbeitung fallen eine Ein- und eine Ausgabenachricht an, die Anzahl der Prozeßwechsel beträgt für eine typische Konfiguration etwa 20; weiterhin sind noch mindestens 5 E/A-Vorgänge für DB-Seiten und Logging erforderlich. Es ist zu beachten, daß bei dieser zentralen Lösung Kooperation zwischen Prozessen relativ billig (Interprozeßkommunikation in einem Rechner, ggf. durch gemeinsamen Speicher unterstützt) und E/A stets lokal sind.

2.2 Verteilte Verarbeitung und verteilte Datenhaltung

Das eben skizzierte Schichtenmodell ist grundsätzlich geeignet, für die verteilte Verarbeitung und verteilte Datenhaltung in Transaktionssystemen verallgemeinert zu werden. In einem verteilten System ist die Menge der TAPs und der Daten über mehrere Knoten verteilt, wobei als Betriebsformen Partitionierung, partielle Redundanz sowie vollständige Replikation sowohl von Programmen als auch Daten denkbar sind. (Hier sollen nur solche Architekturen, die auf allgemein einsetzbaren Rechnern aufbauen, betrachtet werden; Funktionsspezialisierung ist im ortsverteilten Fall ohnehin ein schlechtes Prinzip). Prinzipiell kann nun die Transaktionsverarbeitung in jedem Knoten mit Hilfe des Schichtenmodells nach Bild 1 beschrieben werden, wobei das Modell zunächst nur eine lokale Sicht auf die durch den Knoten verwalteten Betriebsmitteln erlaubt. Dabei ist allerdings zu berücksichtigen, daß zumindest dem Benutzer gegenüber die Verteilung des Systems (TAPs und Daten) zu verbergen ist (Orts-, Fehler-, Replikationstransparenz usw.). Das bedeutet aber, daß eine Schicht des Modells eine *globale Systemsicht* besitzen muß, um durch Kommunikation mit den lokalen Sichten der einzelnen Knoten die "zentrale" Sicht des Benutzers rekonstruieren zu können. Auf diese Weise kann also diese zentrale (ortsunabhängige) Sicht auf jedem Knoten simuliert werden.

In [16, 22] werden verschiedene Systemlösungen für verteilte Transaktionssysteme diskutiert, wobei gezeigt wird, daß grundsätzlich jede Modellschicht zur Bildung der globalen Sicht herangezogen werden kann:

- In Schicht 1 heißt das Lösungsprinzip *"Transaction Routing"*, d.h., die Funktionsaufrufe sind Einheiten der Verteilung, was bei gut partitionierbaren Betriebsmitteln und gleichmäßigem Lastaufkommen mit ausgeprägtem Lokalitätsverhalten (ein hoher Anteil der Aufrufe wird lokal verarbeitet) eine befriedigende Lösung sein kann. Dabei sind aber keine Funktionen möglich, die Betriebsmittel mehrerer Standorte benutzen. Übergreifende Auswertungen muß der Benutzer selbst vornehmen.
- Soll in Schicht 2 (d.h. zunächst einmal *im TAP*) die globale Sicht hergestellt werden, so werden die TAPs abhängig von den Verteilungsaspekten. Zugriffe auf lokale Daten (direkter DB-Aufruf) werden anders ausgeführt als solche auf entfernte (z.B. remote procedure call). Restrukturierungen und Umverteilungen der Daten schlagen also bis auf den Programmtext der TAPs durch.
- In Schicht 3 übernimmt der TP-Monitor die entsprechende Rolle. Er bestimmt den Knoten, der eine Datenanforderung bearbeiten kann, leitet den Auftrag weiter (*Function Request Shipping*) und empfängt das Ergebnis, das er dann dem TAP zur Verfügung stellt.
- Schicht 4 charakterisiert den Einsatz eines *verteilten DBS (VDBS)*, wobei typische Lösungen die Verteilungs-/Kommunikationsaufgaben im Datensystem ansiedeln (Minimierung der Kommunikation, mengenorientierte Anforderungen).
- Auch Schicht 5 ist als Verteilungskomponente denkbar, wenn ein *verteiltes Betriebssystem* vorliegt. Eine solche Lösung erscheint ebenso wie solche im Zugriffs- oder Speichersystem des DBS aus Aufwandsgründen (große Anzahl von Kommunikationsvorgängen) wenig attraktiv.

Das so erweiterte Schichtenmodell, das nun die Ortsverteilung der Transaktionsverarbeitung und der Datenhaltung umfaßt, ist so allgemein, daß prinzipiell auch die Kooperation *heterogener* TP-Monitore und Datenbanksysteme damit beschrieben werden kann. Im Bereich der verteilten DBS sind bisher keine funktionsfähigen Systeme mit heterogenen Teilsystemen (unterschiedliche Datenmodelle) bekannt geworden; selbst für homogene Systeme sind überzeugende Implementierungen rar. Ortsverteilte DBS im strengen Sinne (Gewährleistung aller Aspekte der Verteilungstransparenz) werden es ohnehin schwer haben, sich im Markt durchzusetzen, selbst wenn alle technischen Probleme gelöst sein werden. Es sind zu große Administrationsprobleme zu erwarten, welche vor allem

- die globale Konsistenzerhaltung aller Daten
- die Aktualität des konzeptuellen Schemas
- Migrationsprobleme bei Schemaänderungen usw.

betreffen, da stets Konflikte und Verzögerungen durch Autonomieansprüche, Sonderwünsche und Fehler lokaler Organisationen auftreten können. (Diese Probleme können am ehesten ausgeschaltet werden, wenn das verteilte DBS durch eine Organisation und in einem Raum betrieben wird). Es ist ohnehin fraglich, ob alle konzipierten Leistungen verteilter DBS großen Zuspruch finden ("transcontinental joins" sind wohl nicht lebensnotwendig). Generell läßt sich sagen, daß das von ihnen verkörperte Ubiquitätsprinzip [30], das potentiell allen Anwendungen zu jeder Zeit eine konsistente Sicht auf alle Daten gewährleistet, enorme Laufzeitkosten verursacht und wegen der strikten Vorplanung der TAPs auch gar nicht benötigt wird.

Aus diesen Gründen ist als Entwicklungsansatz für die Datenhaltung ein *Server-Modell* besonders im Rahmen von Transaktionssystemen vielversprechend. Die Idee des verteilten DBS, charakterisiert durch ein globales konzeptionelles Schema, wird aufgegeben zugunsten eigenständiger Systeme mit eigenem DB-Schema, die als Server ihre Leistungen anbieten. Von vorneherein wird kein Anspruch auf Verteilungstransparenz der Datenhaltung erhoben, wodurch ein Großteil der Verwaltungs-/Organisationsprobleme in ortsverteilten Serversystemen überhaupt nicht auftritt. Das heißt nicht, daß Verteilungsprobleme u.a. bis zum Benutzer durchschlagen. Die Schichten 1-3 unserer Aufrufhierarchie (Bild 1) bieten dafür genügend Lösungsmöglichkeiten. Beispiele für homogene Systemlösungen stellen [21, 1, 31] dar.

Im Bereich der Kooperation von TP-Monitoren sind auch heterogene Systemkonzepte denkbar und ohne allzu hohen Aufwand zu realisieren. Diese Vorgehensweise ist immer dann erforderlich, wenn Leistungen existierender Transaktionssysteme in ein neues System integriert werden sollen (z.B. gewachsene Systeme der Flugbuchung). Auch hier ist allein das Server-Konzept zur Erschließung solcher Systemdienste angemessen.

Im weiteren gehen wir davon aus, daß künftig Dienste in verteilten Transaktionssystemen von ortsverteilten Servern erbracht werden. Dabei können wir ohne zusätzliche Annahmen neben der Datenhaltungsfunktionen auch Archivierungs- und Maildienste usw. solchen Servern zuordnen. Das Verteilungswissen wird in den Schichten der Transaktionsverarbeitung (Schicht 1-3) gehalten, wobei spezielle Kommunikationsprotokolle auch heterogene Systemlösungen unterstützen sollen.

2.3 Einsatz von Workstations

In künftigen Transaktionssystemen werden leistungsfähige Arbeitsplatzrechner u.a. die Funktionen der Terminals übernehmen. Da große zentrale Rechner in der Rolle von Servern gemeinsame Aufgaben wie Datenhaltung, Archivierung, Sicherung usw. abwickeln müssen, ergibt sich zwangsläufig hardwareseitig ein verteilter Systemaufbau, wie er in Bild 2 skizziert ist.

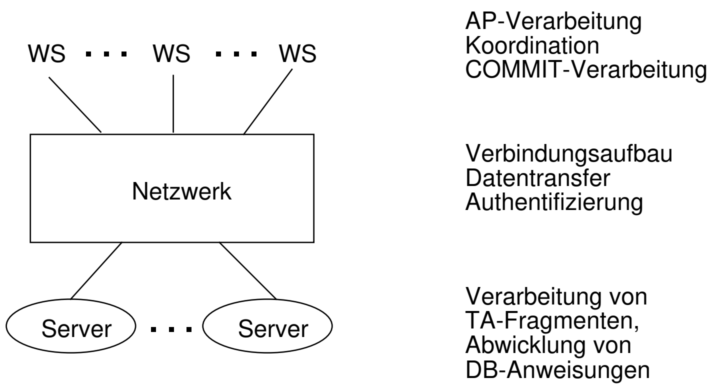

Bild 2: Aufbau eines Workstation/Server-Systems

In den Workstations steht eine enorme Verarbeitungskapazität ggf. mit speziellen Funktionen lokal für die Transaktionsverarbeitung "frei" zur Verfügung, so daß schwerlich einzusehen ist, daß sie weiterhin ausschließlich zentral wie in herkömmlichen Systemen ausgeführt wird. Zwar sind in solchen dedizierten Workstations neue benutzerfreundliche Werkzeuge wie Textverarbeitung, Geschäftsgraphik, Tabellenkalkulation (Spreadsheet) usw. verfügbar, doch bisher nicht in die Transaktionsverarbeitung integriert. Ihre Nutzung gestattet das Zusammenbinden umfangreicher Arbeitsvorgänge im Rahmen der Transaktionsverarbeitung; das impliziert jedoch viel komplexere Rechenabläufe und, will man einen optimalen Workstation-Einsatz erreichen, eine angepaßte Abbildung des Schichtenmodells (Bild 1) auf die vorgegebene Workstation/Server-Umgebung.

Eine bloße Nutzung der Workstation als "intelligentes Terminal" erlaubt etwa die Auslagerung von Funktionen wie:

- Nachrichtenaufbereitung und Maskenabbildung (dezentrale Bildschirmformatierung)
- lokale Konsistenzprüfung von Nachrichten
- Datenaufbereitung (Anpassung/Konversion der Feldinhalte)
- Behandlung sehr großer Masken (Blättern in Masken, Fenstertechnik).

Insgesamt gesehen bleibt jedoch der Verteilungseffekt auf die Transaktionsverarbeitung und damit die Entlastung der zentralen Komponenten gering. Eine "leistungsgerechte" Nutzung von Workstations setzt deshalb voraus, daß sie als selbständige Verarbeitungseinheiten weitere Aufgaben übernehmen, die lokal am effektivsten zu erledigen sind. Das bedeutet, daß dort auch TAPs ausgeführt werden, die Serverleistung in Anspruch nehmen. Nach einer solchen Neubestimmung ist zu untersuchen, wie die durch unser Schichtenmodell repräsentierte Aufrufhierarchie und ihre Komponenten auf Workstations und Server zugeordnet werden sollen.

3. Ablauf einer Transaktion

3.1 Interaktion des Benutzers mit dem System

Transaktionssysteme stellen, wie bereits erwähnt, ihren mit wenig EDV-Kenntnissen ausgestatteten Benutzern fertig ausprogrammierte Funktionen zur Verfügung, die nur noch aufgerufen und (maskengesteuert) mit aktuellen Parametern zu versehen sind. Es ist hervorzuheben, daß die Ausführung einer Funktion, die als *Vorgang* bezeichnt wird, i.a. nicht nur aus dem Aufruf mit den Parametern und einer abschließenden Antwort besteht, sondern *im Dialog* mit dem Benutzer erfolgt. Es kann also Zwischenantworten geben, auf die der Benutzer mit weiteren Eingaben reagiert, etwa zur Bestätigung eines Eintrags oder zur Auswahl aus mehreren angebotenen Möglichkeiten.

Ein einfaches Beispiel ist die Flugreservierung. Die Funktion wird zunächst aufgerufen mit Angaben über Start- und Zielflughafen, Tag und Uhrzeit des Abflugs, gewünschte Klasse, Raucher/ Nichtraucher usw. Das vom System gestartete TAP wählt einen passenden Flug aus und zeigt dem Benutzer die noch freien Plätze an. Dieser kann einen davon auswählen, worauf das System die Reservierung durchführt und mit einer bestätigenden Meldung den Vorgang abschließt. Ebenso gut kann der Benutzer den angebotenen Flug aber auch ablehnen und sich weitere Flüge mit freier Kapazität anzeigen lassen, so daß noch weitere Zwischenantworten folgen.

Die Einheit der Interaktion von Benutzer und System ist dabei jeweils eine einzelne Eingabe und die unmittelbar darauf folgende Antwort; sie wird als *Dialogschritt* bezeichnet. Ein Vorgang setzt sich also i.a. aus mehreren Dialogschritten zusammen. Auch der Sonderfall eines Vorgangs mit einem einzigen Dialogschritt kommt vor; die bereits erwähnte Kontenbuchung ist das bekannteste Beispiel.

Diese Art der Interaktion mit dem System ist vor allem deshalb für Laien so gut geeignet, weil sie einen hohen Grad an *Daten- und Programmunabhängigkeit* bietet. Der Benutzer braucht weder Dateien oder Datenbanken noch Satzstrukturen noch die auf sie zugreifenden Programme (TAPs) zu kennen. Folglich können diese sogar geändert werden, ohne daß die Benutzersicht davon betroffen ist.

Ein einzelner Dialogschritt kann intern eine ganze Reihe von Änderungen in verschiedenen Datenbeständen durchführen. Aus der Sicht des Benutzers muß er aber ununterbrechbar sein, so daß er entweder vollständig ausgeführt wird oder bei Auftreten eines Fehlers keine Wirkung hinterläßt. Ein nur teilweise ausgeführter Dialogschritt hinterließe das System in einem Zustand, der extern für den Benutzer gar nicht darstellbar und dadurch auch nicht korrigierbar wäre, sondern den (manuellen) Eingriff des Systemadministrators erforderte. Um das zu vermeiden, muß jeder Dialogschritt durch eine *Transaktion* gesichert werden [14].

Daß ein einzelner Dialogschritt unteilbar sein muß, schließt nicht aus, daß mehrere aufeinanderfolgende Dialogschritte zu einer Transaktion zusammengefaßt werden können. Scheitert eine solche Transaktion, so werden dadurch zwar im nachhinein Ausgaben des Systems und Eingaben des Benutzer ungültig gemacht. Das System nimmt jedoch wieder einen Zustand ein, den der Benutzer verstehen und von dem aus er - durch Wiederholen der gleichen Eingabe oder durch andere Eingaben - weiterarbeiten kann. Ein

Dialogschritt kann also nur zu einer Transaktion gehören, eine Transaktion aber durchaus mehrere Dialogschritte umfassen. Folglich kann ein Vorgang auch als Folge von Transaktionen angesehen werden. Bild 3 skizziert die Schachtelung der drei Konzepte.

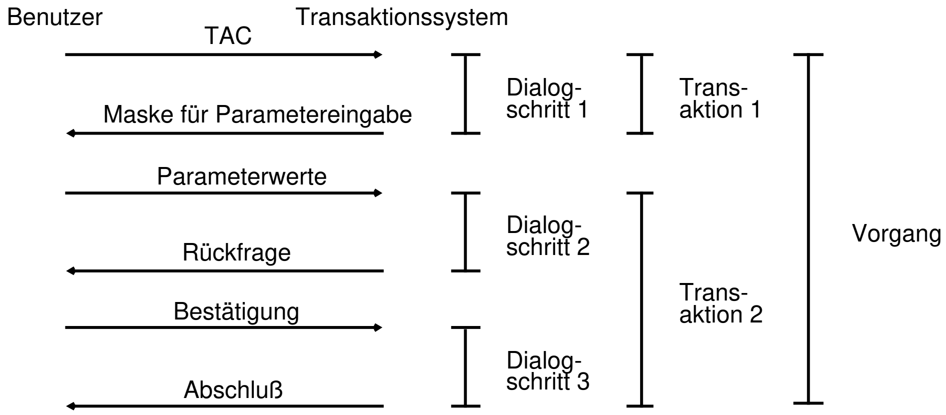

Bild 3: Schachtelung von Dialogschritt, Transaktion und Vorgang

3.2 Anwendungsprogramme auf der Workstation

Obwohl TP-Monitore und DB/DC-Systeme für Zentralrechner mit vielen Terminals entwickelt wurden, kann die von ihnen realisierte abstrakte Sicht auf die Kommunikation mit dem Benutzer einerseits und die Daten andererseits selbst auf Workstations im Ein-Benutzer-Betrieb sehr nützlich sein, wenn die zu erstellenden TAPs auf Workstations unterschiedlichen Typs ablaufen sollen. Auch ist es einfacher, neue Funktionen (TACs) und die dazugehörenden TAPs einzubringen, wenn an zentraler Stelle im TP-Monitor eine Tabelle von TACs und TAPs verwaltet wird.
Der TP-Monitor bestimmt aufgrund des vom Benutzer eingegebenen TACs das auszuführende TAP und ruft es auf. Das TAP führt den ersten Dialogschritt aus und erzeugt eine Ausgabe an den Benutzer. Direkt anschließend wartet es auf die Reaktion des Benutzers, mit der der zweite Dialogschritt beginnt. Weil diese Kombination von Ausgabe mit darauffolgender Eingabe in TAPs so oft vorkommt, gibt es in einigen Systemen dafür einen speziellen Befehl, der beide zusammenfaßt (CONVERSE bei CICS [18].

Man bezeichnet solche TAPs als *Konversations*- oder *Vorgangsprogramme*, weil sie mehrere Dialogschritte nacheinander ausführen, u.U. alle Dialogschritte eines Vorgangs [12]. Wenn in zentralisierten Transaktionssystemen eine große Zahl von Benutzern gleichzeitig solche Programme ausführt, ist es sehr ungünstig, daß während der sekundenlangen Wartezeit auf die nächste Eingabe des Benutzers der Programmzustand erhalten bleiben muß, denn es bedeutet eine unnötige Belegung von Ressourcen. Deshalb erzwingen viele TP-Monitore, daß die Anwendungsprogramme nur genau einen Dialogschritt ausführen, sich mit der Ausgabe an den Benutzer beenden, ihre Ressourcen freigeben und ein Nachfolgeprogramm benennen, das erst bei Eintreffen der nächsten Eingabe gestartet wird, um diese zu bearbeiten. Man spricht dann auch von *Dialogschrittprogrammen* [12]. Im Einbenutzer-Betrieb der Workstation gibt es jedoch keine anderen Benutzer, so daß auf diese Art der Programmierung verzichtet werden kann. Wir werden im folgenden von Konversationsprogrammen ausgehen.

3.3 Anbindung von Serverfunktionen

In der hier betrachteten Konfiguration (Bild 2) ist es möglich, dem Benutzer der Workstation auch Funktionen anzubieten, die nicht nur mit den lokal auf der Workstation verfügbaren Ressourcen (vor allem Daten) auskommen, sondern in ihrer Ausführung auch auf Ressourcen eines oder mehrerer Server zugreifen müssen. Vor allem gemeinsame Daten mit einem hohen Grad an Aktualität können nicht auf die Workstations ausgelagert werden. Dazu sind im Server je nach Anbindungskonzept bestimmte TP-Monitor-Funktionen (Schicht 1-3) und Datenhaltungsfunktionen (Schicht 4-5) vorhanden, und in Kap. 2.2 wurden bereits einige Möglichkeiten angesprochen, wie die verteilte Verarbeitung in solchen Systemen organisiert werden kann. Sie sollen hier auf das Zusammenspiel von Workstation und Server angewendet werden.

Transaction Routing

Transaction Routing bedeutet, daß ein Vorgang entweder vollständig auf der Workstation oder vollständig auf dem Server abgewickelt wird. Dadurch stehen die Funktionen des Servers auf der Workstation zur Verfügung, allerdings *neben* den lokalen Funktionen ohne jeden Zusammenhang. Für eine Server-Funktion leitet die Workstation alle Eingaben einfach weiter und bereitet nur die Ausgaben evtl. noch etwas für ihren lokalen Bildschirm auf. Dem Benutzer kann es verborgen bleiben, ob ein Vorgang lokal oder auf einem Server ausgeführt wird.

Nach dem Prinzip des Transaction Routing läßt sich die DB/DC-Verarbeitung folgendermaßen organisieren. Ein TAP, das ausschließlich lokale Berechnungen (ohne Server-Zugriff) verrichtet, kann in der Workstation angesiedelt werden. Alle TAPs, die auf Serverfunktionen zurückgreifen, werden in die jeweiligen Server ausgelagert. Dazu benötigt man in der Workstation alle Grundfunktionen eines TP-Monitors (Schicht 1-3), die sowohl die lokale TAP-Verarbeitung unterstützen als auch als Verteiler fungieren und einen Vorgang an den zuständigen Server weiterleiten. Dort läuft die Transaktionsverarbeitung wie in einem herkömmlichen zentralisierten Transaktionssystem ab, d.h., grundsätzlich sind im Server alle Systemschichten nach Bild 1 erforderlich. Es werden keine serverübergreifenden Funktionen zur Verfügung gestellt.

Function Request Shipping

Um auch Funktionen zu ermöglichen, die gleichzeitig Ressourcen der Workstation und des Servers benutzen, kann man die Ressourcen des Servers auf den Workstations zugänglich machen, als ob sie lokal wären. Auch ohne ein verteiltes DBS kann der TP-Monitor diese lokale Sicht in Schicht 3 realisieren. In CICS wurde dafür die Bezeichnung Function Request Shipping eingeführt [1, 31]. Die TAPs sind dann zwar einfach zu erstellen, weil sie scheinbar nur auf lokale Ressourcen zugreifen. Diese lokalen Ressourcen sind jedoch teilweise von verschiedenen (heterogenen) Servern "importiert" worden, so daß es sich beispielsweise um Subschemata in unterschiedlichen Datenmodellen handeln kann, auf die auch mit unterschiedlichen Operationen zugegriffen werden muß.

Eine Verallgemeinerung des Function Request Shipping im Hinblick auf die Standardisierung in offenen Netzen stellt der Remote Database Access (RDA) dar [26]. Er unterstützt den Zugriff von einem Programm aus über ein Netz auf eine Datenbank in einem anderen Rechner. Das Format der Anfrage und die Übergabe von Daten in beiden Richtungen sind dabei festgelegt. Die TAPs in der Workstation könnten also RDA benutzen, um auf die Datenbanken der Server zuzugreifen. Ob sie dabei selbst lokalen Zugriff und RDA unterscheiden müssen oder ob erst der TP-Monitor in Schicht 3 diese Unterscheidung macht, spielt eine untergeordnete Rolle.

Es bleibt die Forderung bestehen, daß ein Dialogschritt unteilbar sein muß und deshalb nur von einer einzigen Transaktion überdeckt werden darf. Folglich müssen die lokale Verarbeitung und die Datenbank-Zugriffe auf den verschiedenen Servern zu einer verteilten Transaktion [23] zusammengefaßt werden.

Beim Prinzip des Function Request Shipping sind die Schichten 1-3 in der Workstation allokiert, die alle TAPs der Anwendung verwalten. DB-Aufrufe u.a. werden an die Server weitergeleitet. In den Servern werden nur Funktionen der Schicht 3 benötigt, die die DBS-Kommunikation im Server abwickeln und das Ergebnis der Anforderung an die Workstation zurückschicken. Diese Verarbeitungsweise ist bestenfalls bei mengenorientierten DB-Sprachen tolerierbar, bei navigierenden DB-Sprachen [4] würde das Leistungsverhalten unerträglich werden. Hierzu betrachte man wieder die "Kontenbuchung" mit 9 DB-Operationen.

Obwohl mit Function Request Shipping oder RDA beliebige Funktionen auf Workstation- und Server-Ressourcen realisiert werden können, hat dieser Ansatz in der hier betrachteten Umgebung neben dem möglicherweise sehr schlechten Leistungsverhalten weitere *Nachteile*:

- Die Programme müßten das *Schema* der Datenbank auf dem Server kennen und wären von Änderungen dieses Schemas (soweit sie nicht durch Sichten - Views - verborgen werden könnten) betroffen.
- Die Einheit der Übertragung ist *jeder einzelne Datenbankaufruf.* Komplexe Aggregationen, die nicht durch eingebaute Funktionen der Anfragesprache wie MIN, MAX, SUM oder AVG abgedeckt werden, führen zu langen Folgen von DB-Zugriffen und dadurch zu vielen Nachrichten.
- Die Administration des Servers muß die *Zugriffsrechte* aller Benutzer sehr aufwendig definieren, weil diese Definition an die Operationen der Anfragesprache und deren zahlreiche Varianten gebunden ist.

Aufruf von Vorgängen im Server durch die TAPs

Aus guten Gründen (vgl. Kap. 1) hat man in zentralisierten Transaktionssystemen nicht allen angeschlossenen Benutzern das Datenbankschema zur Verfügung gestellt und sie ihre Funktionen mit Hilfe einer interaktiven Anfragesprache wie SQL ausführen lassen, obwohl das sehr viel Flexibilität geboten hätte. Statt dessen hat man Zugriffsfunktionen programmiert, die von einem TP-Monitor verwaltet werden, und die Datenbestände so eingekapselt, daß sie nur über diese Zugriffsfunktionen erreicht werden können.

Die zentrale Idee in den weiteren Überlegungen dieses Aufsatzes ist nun, den TAPs in einer Workstation an Stelle eines Function Request Shipping zum Zugriff auf die Ressourcen eines Servers auch nur noch die abstraktere Sicht von aufrufbaren Funktionen zu bieten. Dadurch können auch schon vorhandene zentralisierte Transaktionssysteme in neue, systemübergreifende Funktionen eingebunden werden, die mit Hilfe von TAPs auf den Workstations realisiert werden. Beispielsweise könnte ein Server die bereits erwähnte Kontenbuchung anbieten, und ein Vorgang zur Buchung einer Urlaubsreise auf einer Workstation im Reisebüro könnte sie benutzen, um Fahrkarten zu bezahlen (s. Bild 5).

Das dabei zugrundeliegende Verarbeitungsprinzip ist in Bild 4 skizziert. Die Vorgänge in den Servern laufen exakt so ab, als wären sie von menschlichen Benutzern an einfachen Datenstationen aufgerufen worden; was sich an Unterschieden ergibt (Verzicht auf Maskenaufbereitung bei Ein-/Ausgabenachrichten, Synchronisation am Ende der verteilten Transaktion), wird durch den TP-Monitor in Schicht 1 abgehandelt. Natürlich kann es auch notwendig sein, für einige Vorgänge in den Workstations neue Funktionen auf den Servern bereitzustellen. Deren TAPs sollen so konzipiert sein, daß sie mehrere zusammengehörige DB-Operationen und lokale Auswertungen auf dem Server ausführen und nur das Ergebnis in einer einzelnen Nachricht an die Workstation liefern. Gegenüber dem Function Request Shipping werden dadurch etliche Nachrichtenübertragungen im Netz eingespart, bei navigierenden DB-Sprachen [4] noch mehr als bei mengenorientierten (relationalen).

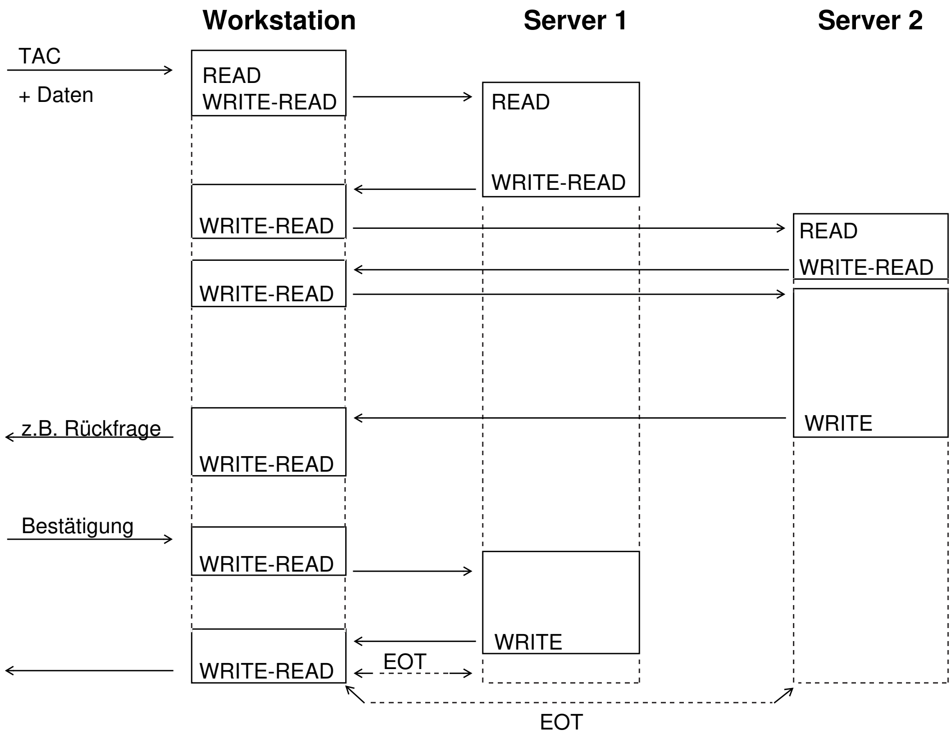

Bild 4: Ablaufbeispiel für eine Transaktion nach dem Konzept der programmierten Verteilung im TAP

Der Grenzfall tritt dann ein, wenn jeder Dialogschritt auf dem Server nur eine DB-Operation ausführen kann, bevor er wieder bei seinem Auftraggeber auf der Workstation rückfragen muß. Dadurch wird das Konzept des Aufrufs von Vorgängen auf dem Server ad absurdum geführt und auf eine Nachbildung des Function Request Shipping reduziert. Obwohl dann keine Reduktion des Nachrichtenaufkommens möglich ist, bleibt der Vorteil der Server-Kapselung weiter bestehen.

Serverseitig sind für die Organisation der TAP-Verarbeitung zwei Prinzipien denkbar:

- Jeder Server-Aufruf erzeugt einen Task, der mit der Rücklieferung des Ergebnisses gedächtnislos beendet wird (Single Call Interface). Diese Vorgehensweise entspricht der Verarbeitung mit einem Application Server Process [3]. (Die DB-Transaktion im Server bleibt natürlich bis COMMIT offen).
- Alle Server-Aufrufe der Workstation werden von einem oder mehreren TAPs (Dialogschrittprogramme) behandelt, wobei der Transaktionszustand in den Server-TAPs erhalten bleibt (Gedächtnis). In diesem Fall kann man von einem Conversational Call Interface sprechen.

Die Implementierung dieser Verarbeitungskonzepte erfordert, daß die Schichten 1-3 (nach Bild 1) prinzipiell sowohl in der Workstation als auch in jedem Server vorhanden sein müssen.

Eine Konsequenz dieser Vorgehensweise ist, daß die TAPs in der Workstation auf Ressourcen der Server anders zugreift als auch die lokalen. Wie Bild 4 zeigt, kann dafür wieder der WRITE-READ-Befehl (bzw. CONVERSE) verwendet werden, der einen (Teil-) Auftrag an den Server sendet und auf dessen Antwort wartet. Die Verteilung ist also im TAP sichtbar und im Programmcode explizit berücksichtigt. Weil die Teile eines Vorgangs, die auf einem Server ablaufen sollen, u.U. auch noch erstellt werden müssen, kann man von *programmierter Verteilung* sprechen. Diese Lösung hat dennoch gewichtige Vorteile:

- Das Workstation-Programm braucht das Datenbankschema auf dem Server nicht mehr zu kennen, sondern nur noch die Funktionen, die der Server nach außen anbietet, mit ihren Ein-und Ausgabeparametern. Änderungen des Schemas werden durch Umprogrammierung der Funktionen auf dem Server unter Beibehaltung der Schnittstelle vor den Klienten im Netz verborgen.

- Komplexe Aggregationen werden als (parametrisierte) Funktionen realisiert, die die Datensätze lokal auf dem Server lesen und nur das Ergebnis an die Workstation zurücksenden.
- Da die Administration die Server-Programme selbst schreibt und der Zugriff auf die Datenbank nur über diese Programme erfolgt, kann sie sehr viel besser kontrollieren, welche Abfragen und Änderungen von den Workstations aus überhaupt durchgeführt werden können.
- Der Mechanismus hat in langjährigem Einsatz seine Tauglichkeit bewiesen und steht unmittelbar zur Verfügung.

In der vorgeschlagenen Lösung wird der Server angesprochen über eine Art "Remote Procedure Call" (RPC), wobei aber hervorzuheben ist, daß es sich nicht nur um *ein* Nachrichtenpaar handelt (Funktion + Eingabeparameter, Ergebnis), sondern um eine Folge von Dialogschritten, also eine Folge von Nachrichtenpaaren. LU 6.2 in IBM's Systems Network Architecture unterstützt diese Art der Kommunikation zwischen Programmen [7]. In die gleiche Richtung zielen für heterogene Systeme die Normierungsvorschläge der ISO zur verteilten Transaktionsverarbeitung [20]. Damit werden aber nur Nachrichtenfolgen zwischen transaktionsverarbeitenden Systemen strukturiert und kontrolliert, nicht jedoch die weitergehenden Aspekte des DB-Zugriffs von Serverfunktionen.

Die verschiedenen Merkmale der Verfahren zur Serveranbindung sind in Tab. 1 zusammengefaßt.

	Transaction Routing	Programmierte Verteilung	Function Request Shipping
Granulat der Verteilung	Vorgang	Teil-Vorgang	TP-Monitor- bzw. DB-Aufruf
Ressourcen des TAP	lokal	verteilt	quasi-lokal
Übergreifende Funktionen	nein	ja	ja
globales DB-Schema	nein	nein	ja
verteilte Transaktionen	nein	ja	ja

Tab. 1: Merkmale der Verfahren zur Serveranbindung

3.4 Synchronisation der Transaktionen auf Workstation und Server

Wenn Vorgänge auf einer Workstation Funktionen eines Servers aufrufen, müssen die Teiltransaktionen in den beteiligten Systemen zu einer verteilten Transaktion zusammengefaßt werden. Dazu muß der Abschluß der Teiltransaktionen synchronisiert werden, damit entweder alle erfolgreich sind oder keine von ihnen. Das soll an einem Beispiel verdeutlicht werden. In einem Reisebüro soll eine Workstation stehen, die Zugriff auf eine Reihe von Servern hat: Flugreservierungssysteme verschiedener Luftfahrtgesellschaften, Hotelreservierungssysteme, das Fahrkartenverkaufs- und Platzreservierungssystem der Bundesbahn usw. Für alle diese Server wird unterstellt, daß sie ihre Dienste nach Art der Transaktionssysteme anbieten: Es gibt Funktionen, die über einen Transaktionscode (TAC) aufgerufen und mit Eingabeparametern versehen werden müssen und die einen Dialog mit dem Auftraggeber (Klienten) führen. Um zum Beispiel in der Zusammenstellung einer Reise einen Flug zu buchen, ermittelt ein Programm auf der Workstation zunächst in mehreren lokalen Dialogschritten die Wünsche des Kunden. (Dies kann recht komplex sein und sogar ein Expertensystem benutzen [28]). Dann meldet es sich bei einem Flugbuchungssystem an, ruft den TAC für eine Flugreservierung auf und übermittelt die Daten des Flugs (Start und Ziel, Datum, Klasse, Raucher/Nichtraucher usw.).

Im Flugbuchungssystem wird ein Vorgang eröffnet, und im ersten Dialogschritt wird mit den Angaben des Workstation-Programms in der Datenbank gesucht. Als Antwort wird ein Angebot mit einer spezifischen Flugverbindung zurückgeliefert. Damit wurde auch die erste Transaktion gestartet, und wenn man gewährleisten will, daß niemand anders dieselben Plätze belegen kann, muß diese Transaktion auch über das Ende des Dialogschritts hinaus geöffnet bleiben.

Das Workstation-Programm prüft das Angebot des Servers und gleicht es mit den Wünschen des Kunden ab. Falls das Angebot zu stark von den Wünschen abweicht, kann es direkt eine Ablehnung an den Server schicken. Andernfalls bereitet es das Angebot des Servers für die Ausgabe auf und kann dabei die speziellen Darstellungseigenschaften des lokalen Bildschirms (z.B. Farbe) nutzen. Mit der Ausgabe wird ein Dialogschritt auf der Workstation abgeschlossen, in den ein Dialogschritt auf einem Server eingebettet war. Da die Transaktion in diesem Server noch nicht abgeschlossen ist, kann auch die Transaktion auf der Workstation noch nicht abgeschlossen werden.

Lehnt der Kunde den angebotenen Flug ab oder möchte er noch alternative Angebote prüfen, so leitet die Workstation dies mit einer Aufforderung, eine andere Verbindung zu suchen, an den Server weiter, der erneut auf die Datenbank zugreifen muß. Die nächste Flugverbindung wird wiederum wie beim ersten Dialogschritt zuerst an die Workstation zurückgeliefert und dann an den Benutzer. Wieder bleiben beide Transaktionen offen.

Wenn jetzt der Kunde den Flug akzeptiert, gibt das Workstation-Programm "Reservierung durchführen" an den Server weiter, der daraufhin den Sitzplatz in diesem Flugzeug als reserviert kennzeichnet und vermutlich auch schon eine Rechnung an das Reisebüro mit der Workstation schreibt. Damit ist der Vorgang im Server beendet, und auch die Transaktion könnte abgeschlossen werden. Die Transaktion in der Workstation ist jedoch noch offen, und es könnte durch einen lokalen Fehler dort zum Zurücksetzen kommen, was das Zurücksetzen im Server umfassen muß. Deshalb wird die Transaktion auf dem Server nur vorläufig beendet. Wenn die Workstation die Bestätigung der Buchung erhält, vermerkt sie diese in ihren lokalen Daten und stößt einen Überweisungsauftrag auf einem anderen Server (dem einer Bank) an, der ähnlich wie die oben gezeigte Kontenbuchung abläuft. Erst dann kann sie die Bestätigung auf dem Bildschirm vermelden, z.B. auch das Ticket drucken und damit nun ihre Transaktion beenden. Über ein 2-Phasen-Commit stimmt sie sich mit den Servern ab, so daß auch dort die Transaktion abgeschlossen werden kann.

Auf diese Weise kann garantiert werden, daß die drei Transaktionen in Workstation und Server entweder beide zum Abschluß kommen oder alle zurückgesetzt werden. Das muß so sein, weil es sich aus globaler Sicht um eine einzige (verteilte) Transaktion handelt, die nur durch die Arbeitsteilung zwischen Workstation und Server in drei Teiltransaktionen zerfällt. Bild 5 faßt das Beispiel noch einmal graphisch zusammen. Mit Hilfe des Konzeptes der programmierten Verteilung (siehe Bild 4) läßt es sich in einfacher Weise realisieren.

Auf der Workstation waren damit zwar Transaktion und Dialogschritt beendet, aber noch nicht der Vorgang. Der kann sich durchaus noch fortsetzen mit einer Hotelreservierung, die mit einem ganz ähnlichen Ablauf auf einen anderen Server zugreift. Dabei kann es zu recht komplizierten Fällen kommen, bei denen der schon gebuchte Flug wieder hinfällig wird, weil zu der geplanten Zeit kein Hotelzimmer mehr verfügbar ist. Das Flugbuchungssystem muß ohnehin eine Funktion zum Stornieren von Reservierungen anbieten, und die hätte das Workstation-Programm in diesem Fall aufzurufen. Um solche Fälle von vorneherein zu vermeiden, könnte das Workstation-Programm gleichzeitig die Reservierungsfunktion im Flugbuchungs- wie im Hotelbuchungssystem starten und die beiden Angebote miteinander vergleichen. Erst wenn beide Angebote zueinander passen, wird

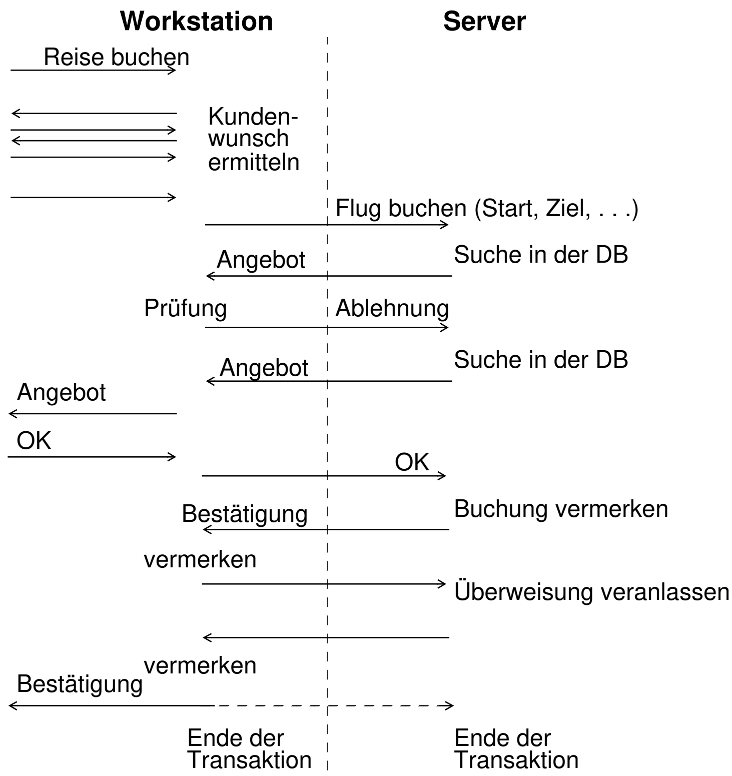

Bild 5: Ablauf einer Transaktion auf Workstation und Servern

ein kombiniertes Angebot ausgegeben. Nach Annahme dieses Angebotes wird wie vorher ein Überweisungauftrag im Bankserver angestoßen. Bei diesem Ablauf verknüpft die Transaktion Teilvorgänge in der Workstation und in drei Servern (nach dem in Bild 4 gezeigten Interaktionsmuster). Die Teilvorgänge in den Servern werden dabei einerseits über ein Single Call Interface (Bank) und andererseits über ein Conversational Call Interface (Flug-, Hotelbuchung) angesprochen. Da es nicht zu vertreten ist, daß überall auf diesen Servern Transaktionen über längere Zeit geöffnet bleiben und dadurch andere Benutzer (Workstations) zum Warten zwingen, wird man evtl. vorläufige Reservierungen durchführen, die man wieder rückgängig macht, wenn man etwas Besseres findet.

Diese Beispiele sollten verdeutlichen, daß die Workstation in einem solchen Netz mit etlichen Servern die Rolle eines *Vermittlers* spielt, der die verschiedenen Informationsquellen kennt und sie je nach Bedarf in die vom Benutzer gewünschte Gesamtfunktion einbindet. Der Programmierer einer Anwendung auf der Workstation muß die Server und ihre Funktionen kennen. Er muß zunächst für eine Verbindungsaufnahme zum Server sorgen, in der der Name des Servers eine wichtige Rolle spielt. In vielen Netzarchitekturen muß der Rechner bekannt sein, auf dem der Server (als Transaktionssystem) läuft, und der Name der "Anwendung", also der Name des Transaktionssystems. Solche Adressen ändern sich natürlich, wenn der Server auf einen anderen Rechner (z.B. den Stand-by-Rechner) verlagert wird. Deshalb wäre es besser, einen netzweit eindeutigen Namen für die Server festzulegen, der mit Hilfe einer Tabelle auf Rechner- und Anwendungsname abgebildet wird. Hinter einem *logischen Server* (LS) könnten sich dann sogar mehrere Rechner verbergen, von denen jeder die gewünschte Leistung erbringen kann und mit denen lastabhängig Verbindung aufgenommen wird.

Wenn die Verbindung aufgebaut ist, kann die Workstation TACs aufrufen. Der Programmierer muß diese TACs und das genaue Format der Ein- und Ausgabeparameter kennen. Aus einer Antwort des Servers geht hervor, ob damit der Vorgang auf dem Server beendet ist (also ein anderer TAC folgen könnte) oder mit einer weiteren Eingabe fortgesetzt werden muß. Gleiches wird für die Transaktion signalisiert. Wurde die Transaktion auf dem Server beendet, so muß der Programmierer auch für die Beendigung der Transaktion auf der Workstation sorgen, bevor er den Vorgang mit dem Server (jetzt in einer neuen Transaktion) fortsetzt. Die Abwicklung der 2-Phasen-Freigabe sollte ihm allerdings durch geeignete Systemsoftware (z.B. einen abgeschwächten TP-Monitor) abgenommen werden. Spätestens bei Ende des Workstation-Programms, ggf. auch schon vorher, wird die Verbindung zum logischen Server wieder aufgelöst.

4. Realisierungsprobleme

Im Gegensatz zu einem VDBS bieten eine Menge von (heterogenen) Server-Systemen keine netzweiten Transparenzeigenschaften für den Benutzer. Jeder Server ist nur für seinen Service verantwortlich; im Falle einer Datenhaltung sind deshalb nur solche Transparenzeigenschaften zu erwarten, wie sie von zentralisierten DBS geboten werden. Der Server-Ansatz, der ja die Idee der *Kapselung* einzelner Funktionen verkörpert, führt also auf eine Dichotomie zur herkömmlichen Forderung eines "single system image" für den Benutzer.

Datenverteilung, -fragmentierung und -replikation sind prinzipiell für die Anwendung sichtbar, wenn sie über den Bereich einzelner Server hinausgehen. (Ihre Auswirkungen auf die TAPs können lediglich durch geeignete Maßnahmen im TP-Monitor gemildert werden, um beispielsweise bei Veränderungen Umprogrammierungen zu begrenzen oder gar zu vermeiden). Auf die Konsequenzen für den TAP-Entwurf soll hier nicht genauer eingegangen werden.

4.1 Forderungen des Transaktionskonzeptes

Grundsätzlich schlagen beim Server-Ansatz auch alle Auswirkungen des Mehrbenutzer-Betriebs und des Fehlerfalls "zwischen Servern" auf die Anwendung durch. Andererseits garantiert jedoch das Transaktionskonzept, wie es ursprünglich definiert wurde [5], als seine Schlüsseleigenschaften "failure isolation" und "concurrency isolation" für das Anwendungsprogramm. Deshalb müssen für Workstation/Server-Systeme technische Lösungen entwickelt werden, die im Zusammenspiel von TP-Monitor und Server die Isolation des TAP bei Fehlerfällen und bei Nutzung von gemeinsamen Daten (über mehrere Server verstreut) gewährleisten.

Zunächst gehen wir davon aus, daß die Eigenschaften des Transaktionskonzeptes für die Anwendung in der Workstation auch bei Nutzung mehrerer (heterogener) Server (so weit wie möglich) erhalten bleiben. Das bedeutet, daß das ACID-Prinzip [14], das ursprünglich für ein zentrales DBS definiert wurde, so umgesetzt werden muß, daß seine Zusicherungen auch in einer heterogenen Serverwelt gelten.

Atomicity: Sie bezieht sich auf die Zugriffe der Transaktion auf die Server-Ressourcen (ununterbrechbar) und auf die Folge der Aktionen (Server-Aufrufe), die aus der Sicht anderer Transaktionen unteilbar ist. Der Zusammenschluß aller Aktionen zu einer atomaren Einheit wird in der Workstation durch ein Zwei-Phasen-Commit-Protokoll (2PC) erreicht. Falls ein Dialog mit dem Benutzer geführt wird, bleibt ihm i.a. der Fehlerfall nicht verborgen.

Consistency: Aus Gründen der Server-Kapselung gibt es keine serverübergreifenden (definierten) Integritätszusicherungen. Jeder Server gewährleistet im Rahmen der Transaktion die für seinen Bereich spezifizierten Zusicherungen.

Isolation: Die Unabhängigkeit der Server impliziert auch lokale und eigenständige Synchronisationsmaßnahmen. Die globale Serialisierbarkeit muß im Rahmen des 2PC-Protokolls hergestellt werden.

Durability: Jeder Server muß für seinen Bereich Recoveryfähigkeiten im Fehlerfall garantieren. Nach erfolgreichem COMMIT liegt deshalb die Verantwortung für die Persistenz seiner geänderten Daten bei jedem Server.

Die zu erwartende Größe von Transaktionen wird sich im Vergleich zu solchen in zentralen DB/ DC-Systemen, die typischerweise nur wenige DB-Zugriffe (0-30) ausführen und außerdem sehr kurz sind (Millisek. oder wenige Sek.), drastisch verändern. Zum einen sind mehrfach "lange" Wege durchs Netz zu den Servern in Kauf zu nehmen, zum anderen aber ist die interaktive Rechnernutzung (Aufruf leistungsfähiger Werkzeuge, Denkzeiten) explizites Ziel des Workstation-Einsatzes. Bei solchen Mehrschritt-Dialogen ergeben sich wesentlich längere Bearbeitungszeiten (und damit Bindungsdauern für Ressourcen) als bei den bisher dominierenden Einschritt-Transaktionen; eine Zeitbegrenzung ist prinzipiell nicht mehr möglich (the human is in the loop). Es sind sogar Transaktionsabläufe denkbar, die Charakteristika langer Transaktionen, wie sie aus dem Ingenieurbereich bekannt sind, aufweisen. Beispielsweise könnte der Benutzer recht lange mit Hilfe extrahierter DB-Daten Kostenmodelle (zu Vergleichs- oder Angebotszwecken) berechnen oder Graphiken aufbauen. Solche Betriebsmerkmale werden einen wesentlichen Einfluß auf die Realisierung der Transaktionsverwaltung und ihrer Komponenten ausüben.

4.2 Synchronisation in einer Server-Umgebung

Aus der Sicht der Synchronisation verwaltet jeder Server eine disjunkte Menge von Betriebsmitteln und regelt alle Synchronisationsprobleme (auch wegen der Server-Kapselung) ausschließlich lokal. Im Prinzip kann dabei jeder Server sein eigenes Verfahren einsetzen, wobei nur die Serialisierbarkeit aller Transaktionsanforderungen an einen Server zu garantieren ist. Wendet sich eine Transaktion nur an einen Server, so ergeben sich im Vergleich zum zentralisierten Fall keine neuen Probleme. Insbesondere können für die Anforderungen solcher Transaktionen alle bekannten Strategien der (lokalen) Konfliktvermeidung oder -erkennung (Wartegraph, . . .) angewendet werden.

Bei Synchronisationsanforderungen über mehrere Server kann, falls erforderlich, ohne großen Aufwand globale Serialisierbarkeit erzielt werden. Es läßt sich nämlich bei COMMIT überprüfen, ob Serialisierbarkeit vorliegt. Im Falle von blockierenden Verfahren (Sperren) in allen Servern kann eine Transaktion unmittelbar "als serialisierbar" akzeptiert werden, wenn sie EOT erreicht hat (d.h., sie wartet nicht mehr auf die Gewährung von Sperren). Bei optimistischen Verfahren ist der Aufwand zur Validierung sicher wesentlich höher, da nicht ohne weiteres von einem gemeinsamen Schema zur Vergabe von Transaktionsnummern oder zur Bestimmung der globalen Zeit ausgegangen werden kann. Solche Fragen sollen jedoch hier wegen der geringen praktischen Relevanz von optimistischen (und anderen) Verfahren nicht erörtert werden.

Anforderungen an Sperrverfahren

Das *Standardverfahren* zur Synchronisation ist das Sperrkonzept und zwar in seiner hierarchischen Verfeinerung [10]. Bei herkömmlichen Transaktionssystemen konnte man mit zunehmender Größe des DB-Puffers, dem Einsatz von sehr schnellen Zwischenspeichern (Global Storage, Extended Storage) und damit einhergehend der Reduktion von Anzahl und Dauer von E/As tendenziell eine Entschärfung des Synchronisationsproblems beobachten, weil durch solche Architekturmaßnahmen die Sperrdauer der Objekte einer Transaktion (und damit das Blockierungspotential für andere Transaktionen) deutlich verringert wurde. Trotz dieser Verbesserungen ist auf folgende wichtigen Aspekte beim Entwurf eines Sperrverfahrens zu achten. Als kleinstes verfügbares Sperrgranulat sollten Einträge verwendet werden, da es sonst selbst bei herkömmlichen Anwendungen z.B. auf Katalogen und Indexstrukturen zu langen Blockaden kommen kann. Weiterhin erfordert die Behandlung von Hot Spots spezielle Protokolle. Die

Idee, einen "Vermittler oder Notar" einzusetzen (escrow paradigm), scheint in solchen Situationen besonders hilfreich [24].

Das Synchronisationsproblem (insbesondere mit den diskutierten Aspekten) bleibt selbst in herkömmlichen Transaktionssystemen unter der Annahme ständig wachsender Transaktionslasten "hartnäckig" erhalten; es verkörpert die Ursache des fortdauernden "Performance-Problems". In zentralen Systemen läßt sich einiges durch organisatorische Maßnahmen erreichen, da die Transaktionsanwendungen (im Gegensatz zu Ad-hoc-Fragen) zumindest in Grenzen vorgeplant werden können.

Verschärfung der Synchronisationsproblematik

Mit der größeren Selbständigkeit von Workstations und der Nutzung von unabhängigen Servern fällt der Aspekt der Planbarkeit des Transaktionsablaufs weitgehend weg. Zugleich ist eine enorme Verschärfung der Synchronisationskonflikte zu erwarten, da lange Transaktionsdauern und größere Objektmengen wiederum ein höheres Konflikt- und Behinderungspotential aufbauen. Wartende Transaktionen verlängern ihre Laufzeit und halten dadurch ihre Sperren entsprechend länger - im Kern ist hier eine Konfliktspirale angelegt.

Dieser Zusammenhang unterstreicht, daß ein Sperrkonzept mit kleinen Granulaten (auf variabel langen Sätzen) nicht nur für Systemdaten, sondern auch für normale Benutzerdaten immer größere Bedeutung gewinnt, da längere Sperrdauern auch bei Daten mit geringerer Zugriffsfrequenz die Konfliktwahrscheinlichkeit erhöhen. Die konfliktvermeidende Behandlung von Hot Spots erhält dadurch offensichtlich einen noch viel höheren Stellenwert.

Auflösung von Deadlocks

Ein neues Problem stellen Mehrserver-Transaktionen (Ti) hinsichtlich von Verklemmungen dar. Wegen der Unabängigkeit der Server können sich beispielsweise in einem Server die Serialisierungsreihenfolge T1 > T2 und zur gleichen Zeit in einem anderen die Folge T2 > T1 herausbilden, was (bei Sperrverfahren) zu einem klassischen Deadlock über Servergrenzen hinweg führt. Globale Deadlockvermeidung oder -erkennung kann nun aber aus folgenden Gründen nicht eingesetzt werden:

- Heterogene Serversysteme sind nicht für die Kooperation entworfen; sie besitzen keine gemeinsame Basis und deshalb keine Abstimmungsmöglichkeit zur Konfliktregelung.
- Die Idee der Serverkapselung steht solchen übergreifenden Aufgaben entgegen; auch im homogenen Fall ist keine Kooperation vorgesehen.

Als einfachstes Verfahren der Konfliktauflösung kann hier bei Einsatz von blockierenden Synchronisationsmethoden ein Timeout-Mechanismus vorgeschlagen werden.

4.3 Logging und Recovery

Das Transaktionskonzept impliziert weitreichende Maßnahmen zur Behandlung von (erwarteten) Fehlerfällen. Es setzt jedoch keine Ausfallsicherheit oder permanente Verfügbarkeit des Systems voraus. Fehlerisolation des Benutzers betrifft die Aspekte "Atomarität und Dauerhaftigkeit", die aus seiner Sicht zu gewährleisten sind, und bedeutet lediglich Maskierung, nicht jedoch Verhinderung von Fehlern. Es sind also Vorkehrungen zur Fehlerbehandlung zu treffen, damit stets der "jüngste transaktionskonsistente DB-Zustand" [14] nach Behandlung eines auftretenden Fehlers (als Zielzustand erfolgreicher Recovery) erreicht wird. Handelt es sich bei einem aufgetretenen Fehler um einen nicht-erwarteten Fehler oder versagt die vorgesehene Recovery (Katastrophe), so muß eine manuelle Fehlerbehandlung erfolgen.

Erweitertes Fehlermodell

Welche Fehlertypen sollten nun erwartet und automatisch durch das System behandelt werden? In zentralisierten DB/DC-Systemen beschränkt man sich hierbei auf das Standard-Fehlermodell:

- Transaktionsfehler, der zum nebenwirkungsfreien Rücksetzen der gesamten Transaktion führt.
- Systemfehler, bei dem nach Restart alle Spuren nicht-beendeter Transaktionen beseitigt und die Ergebnisse aller erfolgreichen Transaktionen wiederhergestellt sind.
- Gerätefehler, der die Rekonstruktion von (Teilen von) Externspeichern impliziert.
- Kommunikationsfehler, der aufgrund von Störungen von Nachrichtenverbindungen auftritt und (im einfachsten Fall) durch Protokollierung und wiederholte Übertragung der Nachrichten behoben werden kann.

In offenen Netzen, die oft heterogen und recht unzuverlässig sind, sind noch eine Reihe von Fehlerszenarios denkbar:

- Ein Ausfall (Aussetzen) von Knoten- oder Prozeßaktivitäten ist vorübergehend; aufgrund zu knapp bemessener Timeout-Zeiten nehmen andere Knoten ein Fehlverhalten (Crash) an (transient crash failures)
- Es gehen manchmal Nachrichten verloren (Omission failures)
- Es werden willkürliche, d.h. syntaktisch korrekte, jedoch inhaltlich verfälschte Nachrichten gesendet (Byzantine failures)

Solche Fälle sollen hier nicht in unser Fehlermodell aufgenommen werden. Dagegen ergeben sich aus der Art des Workstation-Einsatzes sowie der Dauer und Verteilung der Transaktionsverarbeitung spezifische Fehlerfälle, für die eine automatisierte Behandlung vorgesehen werden sollte. Da Transaktionen typischerweise interaktiv und recht lang sind, sollten systemgestützte Lösungen für folgende Fehlerfälle angestrebt werden:

- Server-Ausfall (Fail-Stop-Verhalten): Nach einem Server-Crash ist (wie bei einem zentralisierten DBS) der aktuelle Server-Zustand verlorengegangen; es ist ein lokaler Restart des Servers erforderlich. Nach Möglichkeit sollte das Auftreten solcher Fehler den beteiligten Workstations/Transaktionen verborgen werden. Anderenfalls sind ihre Auswirkungen auf laufende Transaktionen zu minimieren.
- Workstation-Ausfall: Der aktuelle Zustand der Transaktion ist workstationseitig verlorengegangen. Wurden in der Workstation keine Vorkehrungen zum Wiederanlauf (Checkpointing) getroffen, muß die gesamte Transaktion vollständig zurückgesetzt werden. Insbesondere für lange Transaktionen sollten jedoch systemgestützte Maßnahmen (ggf. Rollforward) vorgesehen werden, die den Verlust der gesamten Transaktion verhindern.

Zwischen den Aktivitäten der Transaktion in der Workstation und den Servern gibt es eine Menge von Abhängigkeiten, die im Fehlerfall zu berücksichtigen sind. Beim Entwurf des Transaktionssystems sind deshalb geeignete Strukturierungsmaßnahmen zu entwickeln (z.B. geschachtelte Transaktionen), die im Fehlerfall eine *größtmögliche Isolation von Workstation und Servern* gewährleisten, um im Betrieb als Maximalziel eine gegenseitige *Fehlermaskierung* anstreben zu können.

Ein mit der Behandlung des Transaktionsfehlers verwandter Aspekt ist das partielle Rücksetzen einer Transaktion. Ähnlich wie bei Ingenieurtransaktionen oder beim Arbeiten mit einem Editor sollte es dem Benutzer möglich sein, (fehlerhafte) Aktionen oder Arbeitsabschnitte auf einfache Weise rückgängig zu machen. Beispielsweise könnten die letzten Operationen durch eine wiederholte Ausführung einer UNDO-Anweisung "zurückgerollt" werden. Für das Rückgängigmachen größerer Verarbeitungsabschnitte bietet sich ein benutzerkontrolliertes Sicherungspunktkonzept (mit SAVE(n) und RESTORE(n) [11]) an. Mit der Dauer einer Transaktion wächst auch die Notwendigkeit, systemseitig Maßnahmen zum partiellen Zurücksetzen (und ggf. zum partiellen Wiederholen) verfügbar zu machen. Im allgemeinen Fall ist ein solches Konzept mit einer Menge von Implikationen, die hauptsächlich Systemstruktur und Fehlerfall betreffen, behaftet. Solange das partielle Rücksetzen nur den Zustand der Transaktion in der Workstation berührt, läßt es sich lokal bewerkstelligen. Sind jedoch auch Server-Operationen rückgängig zu machen, sind nicht beliebige einseitig definierte Zustände (nämlich von der Workstation) erreichbar. Auch Server müssen dieses Konzept unterstützen und die Spezifikation von abgestimmten Sicherungspunkten erlauben. Bei einem partiellen Rücksetzen sind also im allgemeinen k (von n) Server involviert.

Sicherungsmaßnahmen im Normalbetrieb

Um die diskutierten Fehlerfälle behandeln zu können, muß geeignete Redundanz entweder im Systemablauf oder als Sicherungsinformation verhanden sein. Im Rahmen unserer Betrachtungen wollen wir uns auf Log-Maßnahmen, die typischerweise in DB/DC-Systemen eingesetzt werden, beschränken und auch nur überblickartig die Problemstellung skizzieren.

Jeder Server hat für Änderungen, die er durchführt, Log-Informationen zu sammeln. Im Vergleich zu zentralisierten DBS treten für die Behandlung der Standard-Fehlerfälle keine neuen Aspekte auf. Es ist genügend UNDO/REDO-Information gemäß WAL-Prinzip und COMMIT-Regel [14] auf einen sicheren Platz zu schreiben. Dabei können pro Server unterschiedliche Techniken angewendet werden, solange die Zusicherungen der transaktionsorientierten Recovery eingehalten werden. Alle Optimierungsüberlegungen (z.B. Schreiben von Checkpoints) gelten unabhängig für jeden Server [14]. Im Zusammenhang mit der verschärften Synchronisationsproblematik soll hier lediglich darauf hingewiesen werden, daß das Sperrgranulat immer größer oder gleich dem Loggranulat sein muß. Das bedeutet, daß das in vielen Systemen übliche Seitenlogging für das Erzielen höherer Parallelitätsgrade nicht akzeptabel ist. Es ist vielmehr zwingend ein flexibles Eintragslogging erforderlich.

Die Möglichkeit partiellen Rücksetzens setzt zusätzlich Log-Maßnahmen in der Workstation voraus. Eine erforderliche Behandlung von selektivem Workstation- oder Server-Ausfall erzwingt auch ein häufiges Schreiben von Sicherungspunkten. Gegebenenfalls sind auch Rollforward-Strategien bei bestimmten Verarbeitungsmodellen anwendbar. Je nach Grad der Workstation/Server-Isolation ist außerdem für eine mehr oder weniger komplexe Abstimmung von Rücksetz- und Sicherungspunkten zu sorgen, die an dieser Stelle nicht diskutiert werden kann. Diese eben angesprochene Problematik wird nur dann sehr einfach, wenn während der Transaktionsverarbeitung eine lose Kopplung zwischen Workstation und Servern durch Einsatz spezieller CHECKOUT/CHECKIN-Protokolle [11] erzielt wird. Jedoch dürfte die lokale Verarbeitung komplexer Objekte - wie bei CAD-Anwendungen - nicht der Regelfall in Transaktionssystemen werden.

4.4 Zwei-Phasen-Commit

Es wurde bereits festgestellt, daß alle Aktivitäten einer Transaktion durch ein 2PC-Protokoll zu einer atomaren Einheit zusammenzubinden sind. Dabei muß davon ausgegangen werden, daß kein globales Wissen über alle an einer Transaktion beteiligten Komponenten existiert. Das trifft auch für die Workstation als Initiator der Transaktion zu, denn der Allgemeinheit halber sollte angenommen werden, daß Server andere Server (im Auftrag der Transaktion) aufrufen können, um beispielsweise Standarddienstleistungen zu erfragen. Aus solchen Gründen müssen alle beteiligten Server in der Lage sein, an einem hierarchischen 2PC [23] teilzunehmen. Die Workstation übernimmt dabei in natürlicher Weise die Rolle des Koordinators.

Ein gravierendes Problem ergibt sich offensichtlich durch die Vielzahl der Workstations unter Benutzeradministration. Ein Workstation-Ausfall während der Abwicklung des 2PC kann zu einer sehr langen Blockierung der belegten Betriebsmittel führen. Wegen der großen Anzahl der zum Transaktionssystem gehörenden Workstations resultiert daraus eine nicht zu vernachlässigende Blockierungsgefahr für große Systemteile. In [25] wurden deshalb eine Verschiebung der Commit-Verantwortung von den "unzuverlässigen" Workstations hin zu einem sicheren Knoten (Server) vorgeschlagen und entsprechende Algorithmen skizziert.

4.5 Anpassung des Transaktionskonzeptes

Die Forderung nach Serialisierbarkeit schafft bei Mehrschritt-Transaktionen ein gewisses Behinderungspotential für parallel laufende Transaktionen, das sich mit ihrer Lebensdauer (langlebige Transaktionen mit langen Sperren und vielen gesperrten Objekten) verschärft. Es ist abzusehen, daß dadurch bei vielen Anwendungen kein vernünftiger Transaktionsbetrieb mehr möglich wäre (lange Blockierungen, höhere Rücksetzraten). Der Kern des Problems liegt vor allem darin begründet, daß das Transaktionskonzept vor COMMIT die vollständige Rücknahme der Transaktion gestattet und zwar so, als wäre sie nie gestartet worden. Erst ein Aufweichen dieser Zusicherung verspricht flexible Lösungsmöglichkeiten.

Gemessen an den Vorgängen im betrieblichen Geschehen bedeutet die "Alles-oder-Nichts"-Eigenschaft von Transaktionen oft eine idealisierte Kapselung, die sich im Modell zwar erreichen läßt, jedoch in der Realität weder gefordert noch gegeben ist. Bei vorzeitiger oder irrtümlicher Abwicklung von Teilvorgängen kann die Stornierung (siehe Kap. 3.2) als Möglichkeit der "sichtbaren" Korrektur herangezogen werden. Für Transaktionen gibt es eigentlich ein solches Konzept nicht, da sie stets definitionsgemäß "die DB von einem konsistenten in einen konsistenten Zustand überführen". Die Beobachtung, daß gelegentlich auch eine falsch programmierte oder mit fehlerhaften Daten versorgte Transaktion ausgeführt wird und ihre Ergebnisse freigibt, erzwingt eine Reaktionsmöglichkeit; es muß eine Gegentransaktion entworfen und ausgeführt werden, mit der die Fehler repariert und ihre Auswirkungen (nach Möglichkeit) vollständig kompensiert werden. Eine solche Kompensation ist nicht gleichzusetzen mit einem bloßen UNDO der Transaktion; es muß vielmehr anwendungsbezogenes Wissen berücksichtigt werden.

Bei vielen Anwendungen bietet es sich nun an, eine Mehrschritt-Transaktion T in mehrere Teiltransaktionen T_i zu zerlegen, um das durch lange Transaktionen verursachte Blockierungs- und Leistungsproblem zu lindern. Die einzelnen Transaktionen T_i geben ihre Ressourcen bei EOT(T_i) frei. Damit ist T nicht mehr atomar und folglich nicht mehr serialisierbar. Bei sorgfältiger Planung der Zerlegung von T bedeutet dies jedoch keine Preisgabe der DB-Konistenz. Was explizit in Kauf genommen wird, ist eine Kompensation der bereits freigegebenen T_i im Fehlerfall. Dazu ist eine koordinierte Fehlerbehandlung erforderlich.

Eine entsprechende Vorgehensweise wird durch das Konzept der Sagas [6] oder der Transaktionsketten [2] vorgeschlagen. Eine Transaktion T wird danach in eine Sammlung von Teiltransaktionen $T = (T_1, T_2, T_3, \ldots, T_n)$ aufgeteilt und kann als spezielle Art einer zweistufig geschachtelten Transaktion angesehen werden. Dabei werden folgende Eigenschaften garantiert:

- Jedes T_i bewahrt für sich das ACID-Prinzip.
- Alle T_i gehören zusammen; es gibt also keine teilweise Ausführung von T.

Da die T_i ihre Ressourcen freigeben, ist eine Verzahnung mit T_j anderer Transaktionen möglich, d.h., für die gesamte Transaktionskette T kann keine Serialisierbarkeit gewährleistet werden. Der Fehlerfall ist derart einzuplanen, daß für jedes T_i eine Kompensationstransaktion C_i bereitzuhalten ist. Wie das Beispiel der Hotelbuchung in 3.4 zeigte, ist das in vielen Transaktionssystemen ohnehin schon der Fall. Dadurch kann die Transaktionsverwaltung bei Auftreten eines Fehlers automatisch eine Kompensation aller freigegebenen Änderungen der T_i vornehmen. Diese Kompensation als integraler Bestandteil des Konzeptes ist nur für nicht-abgeschlossene Transaktionsketten T vorgesehen und erfolgt in umgekehrter Ausführungsreihenfolge ($T_1, T_2, \ldots T_i, C_i, \ldots C_2, C_1$). Einzelheiten und Optimierungen dieses Konzeptes finden sich in [6]. Es sei nur noch einmal darauf hingewiesen, daß im allgemeinen Fall

$$T_i, C_i \neq \text{UNDO}\ (T_i)$$

ist. Die Anwendungssemantik hat in jedem Einzelfall bereits beim TAP-Entwurf über die Art der Kompensation zu entscheiden. Beispielsweise kann die Stornierung einer Hotelbuchung mit einer Gebühr belegt werden.

Eine Transaktionskette kann das angemessene Ausführungsmodell darstellen, wenn alle Vorgaben eines Vorganges ganz oder überhaupt nicht erreicht werden sollen. Es sind jedoch insbesondere in unserer heterogenen Server-Umgebung noch ganz andere Modelle einer Transaktionsabwicklung denkbar. Bei Planungsvorgängen ist es oft erforderlich, gleichzeitig Alternativen zu verfolgen, bei denen sich am Ende ein Gewinner und n Verlierer ergeben. Da solche Planungen sehr langlebig sein können, müssen sie ggf. für andere sichtbar gemacht werden. Das erfolgreiche Ende einer Transaktion hat nun die ausgeschiedenen Planungsalternativen automatisch zu kompensieren. Systemkontrolle ist hier besonders wichtig, da bei manueller Handhabung der einzelnen Planungen einzelne Aktivitäten vergessen werden könnten. Als Beispiel ziehen wir wieder die Flugreservierung heran. Bestimmte Flugrouten sind so überbucht, daß man sich oft bei mehreren Fluggesellschaften (Servern) in die Warteschlange für einen entsprechenden Flug eintragen läßt. Wochen und Monate können solche Transaktionen natürlich nicht offen gehalten werden. (Da viele Reisende sich mehrfach eintragen lassen, hat man gute Chancen, irgendwo einen Flugplatz zu ergattern). Sobald eine Reservierung akzeptiert ist, können die restlichen Reservierungswünsche wieder ausgetragen werden.

Es sind sicher noch andere Aktivitätsmuster von Transaktionen möglich. Durch Anpassung des Transaktionskonzeptes läßt sich in vielen Fällen durch Nutzung von Anwendungswissen ein Ausgleich finden, der dem Konsistenzanspruch und dem Leistungsbedarf der Anwendung gerecht wird. Als formaler Rahmen für solcher Ansätze wurde in jüngster Zeit auch das Konzept der ConTracts [27] entwickelt.

4.6 Schutzmechanismen

In zentralisierten Transaktionssystemen muß kontrolliert werden, wer von welchem Terminal aus welche Funktionen (TACs) aufrufen darf, ggf. sogar noch mit welchen Parametern (Eingabedaten). Solange "dumme" Terminals verwendet werden, kann der Benutzer nur relativ wenig Einfluß auf die Nachrichten nehmen, die an den Server übertragen werden. Oft sind die Formate fest durch die Hardware vorgegeben. Dagegen ist es auf frei programmierbaren Workstations nicht allzu schwierig, beliebige Nachrichtenformate zusammenzustellen und dem Server den einen oder anderen Gerätetyp vorzugaukeln. Das ist bei der sog. Terminalemulation oft geübte Praxis. Hier ist der Server noch stärker auf die Netzsoftware angewiesen, um so viel wie möglich über den tatsächlichen Standort der Workstation ("in der Personalabteilung") zu erfahren. Bei hohen Sicherheitsanforderungen hat es sich bewährt, die Verbindung zunächst aufzulösen und dann vom Server aus neu aufzubauen. Danach muß sich nicht nur die Workstation identifizieren, sondern auch der Benutzer, der gerade an ihr arbeitet (Paßwortverfahren).

Ein Workstation-Programm sollte Benutzernamen und Paßwort als Eingabe anfordern und an den Server weiterreichen. Die Folge davon ist, daß der Programmierer den Fall vorsehen muß, daß einige Server-Funktionen durch den aktuellen Benutzer gar nicht ausgeführt werden dürfen. Es wären auch kompliziertere Mischformen denkbar, bei denen der Server eine Authentifizierung vom Benutzer und vom Programm verlangt. Dadurch ließe sich sicherstellen, daß Benutzer x die Serverfunktion y nur ausführen darf, indem er sich des Workstation-Programms z bedient.

Umgekehrt muß eine Workstation sicher sein können, daß sie wirklich mit dem logischen Server x Verbindung aufgenommen hat, und nicht etwa mit einer anderen bösartigen Workstation, die ihr den Server x nur vorspielt, um z.B. das Paßwort oder andere vertrauliche Daten zu erfahren. Im Prinzip muß sich also auch der Server der Workstation gegenüber authentifizieren. Da allerdings die Workstation von sich aus die Verbindung aufbaut, müßte eine bösartige Workstation sich schon im Netz hinter dem Namen oder gar der Netzadresse des Servers verbergen. Es ist Aufgabe der Netzsoftware und der Netzadministration, so etwas zu verhindern. Eine Maßnahme, die auch allgemein das "Anzapfen" einer Verbindung erschwert, ist die Verschlüsselung der Nachrichten. Sie bildet eine weitere Barriere, selbst wenn ein eigentlich unzulässiger Verbindungsaufbau erzwungen werden konnte. Für die Verschlüsselung stehen zahlreiche Verfahren zur Verfügung [29].

5. Schlußfolgerungen und Ausblick

In dieser Arbeit wurde untersucht, welche neuen Möglichkeiten sich im Bereich der klassischen Transaktionsverarbeitung ergeben, wenn anstelle der "dummen" Terminals Workstations mit dezentraler Verarbeitungskapazität eingesetzt werden. Ein Schichtenmodell bot den Rahmen für die Diskussion verschiedener Arten von Verteilung:

- Nutzung existierender Funktionen (Vorgänge) auf den zentralen Rechnern neben den lokal auf der Workstation verfügbaren (Transaction Routing),
- Nachrichtenaustausch zwischen TAPs auf der Workstation und dem Server (programmierte Verteilung),
- für den Programmierer unsichtbare Weiterleitung von Datei- und Datenbankzugriffen an den Server, der die Datei oder Datenbank verwaltet (Function Request Shipping).

Der Dialog des Benutzers mit einem Programm auf der Workstation wurde mit einer Struktur unterlegt, die eine Schachtelung von Funktionsausführungen (Vorgängen), Transaktionen und Dialogschritten vorsieht. Ein Vorgang kann dabei im allgemeinen mehrere Transaktionen enthalten, die sich selbst wieder aus mehreren Dialogschritten zusammensetzen. Diese Dialogstruktur prägt auch die Programmstruktur.

Transaction Routing läßt es ohnehin nicht zu, daß ein Vorgang auf der Workstation Ressourcen auf dem Server benutzt, so daß nur verteilte Programmierung und Function Request Shipping dafür genutzt werden können. Weil Function Request Shipping das Anwendungswissen nicht ausnutzt, zu sehr hohen Nachrichtenraten führt und außerdem auch noch als Spezialfall der programmierten Verteilung betrachtet werden kann, wurde nur die programmierte Verteilung genauer untersucht.

Es ist dabei vorteilhaft, zwischen dem Vorgang auf der Workstation und dem Vorgang auf dem Server beim Nachrichtenaustausch die gleiche Dialogstruktur zu verwenden wie zwischen dem Endbenutzer und dem Vorgang auf der Workstation. Das entspricht genau der Dialogstruktur, die existierende Transaktionssysteme ohnehin schon unterstützen. Die einzige Änderung, die bei ihnen dann noch vorgenommen werden muß, betrifft die Synchronisation von Transaktionen auf Workstation und Server: Nach Abschluß einer Transaktion auf dem Server darf dieser noch nicht die Ressourcen freigeben, sondern muß warten, bis auch die Transaktion auf der Workstation erfolgreich abgeschlossen werden konnte (2-Phasen-Commit).

Die Workstation kann also Funktionen eines oder mehrerer Server aufrufen mit einer Art von erweitertem Remote Procedure Call, der Zwischenantworten zuläßt. Sie spielt dann die Rolle eines Vermittlers oder Koordinators, der die verschiedenen Server-Funktionen ("Flugreservierung", "Mietwagenreservierung", "Hotelbuchung" usw.) genau kennt und sie mit zusätzlicher lokaler Verarbeitung zu mächtigen, übergreifenden Funktionen ("Reise planen und buchen") integriert. Der Endbenutzer braucht nicht zu wissen, auf welche Server zur Ausführung einer Funktion zugegriffen werden muß.

Der Programmierer eines TAPs auf der WS kennt nur die Funktion der Server, nicht aber deren Datenbestände. Die können sehr verschiedenartig sein und auch reorganisiert werden, ohne daß dies die nach außen angebotenen Funktionen betrifft. Für dieses Prinzip wurde der Begriff der "Server-Kapselung" eingeführt.

Es wurden einige der Probleme angesprochen, die sich bei der Realisierung von Transaktionssystemen in einer solchen Workstation/Server-Umgebung ergeben. Insbesondere kann man bei der Implementierung des Transaktionskonzepts nicht mehr nur von sehr kurzen Transaktionen ausgehen, die nur wenige Daten berühren. Die Forderung nach Serialisierbarkeit läßt sich mit Hilfe des 2-Phasen-Commit prinzipiell erfüllen, wobei es allerdings zu sehr langen Blockierungszeiten kommen kann. Bei einer längeren Dauer der Sperren ist es um so wichtiger, daß die Sperrgranulate so klein wie möglich gewählt werden. Datenelemente mit hoher Änderungsrate (hot spots) sollten durch geeigneten DB- und Transaktionsentwurf vermieden werden. Als Alternative zur strikten Realisierung von verteilten Transaktionen über alle Server wurden Transaktionsketten vorgeschlagen, die zwar nicht mehr serialisierbar sind, sich dafür aber durch wesentlich kürzere Blockierungszeiten auszeichnen. Diese Lösung wie auch noch andere *Transaktionsmodelle* sollen in Folgearbeiten noch genauer untersucht werden. Das Ziel wird dabei immer sein, ein Maximum an wechselseitiger Fehlermaskierung zu erreichen.

Weiterhin wurden Schutzmaßnahmen angesprochen, die sich ebenfalls gegenüber klassischen Transaktionssystemen ändern müssen. Ausgeklammert wurden aus Platzgründen die sicher auch sehr interessanten Fragen der *Administration* eines solchen Transaktionssystems. Sie ist sehr aufwendig und sollte dadurch erleichtert werden, daß man die Workstations und Server so weit wie möglich entkoppelt. Die in diesem Beitrag vorgeschlagene Art der Kooperation kommt dem sehr viel weiter entgegen als etwa der Einsatz eines verteilten Datenbanksystems.

Danksagung:

Diese Arbeit entstand im Rahmen eines von der Siemens AG geförderten Projekts. Wir danken den Referenten, die uns nach gründlicher Lektüre der ersten Fassung wertvolle Hinweise zur Verbesserung gegeben haben.

6. Literatur

1. Acker, R.D., Seaman, P.H., "Modelling distributed processing across multiple CICS sites", in: *IBM Systems Journal,* Vol. 21, No. 4, 1982, pp. 471-489.
2. Bohn, V., und Wagner, Th., "Transaktionsketten - Konzepte und Implementierung," in: *Proc. 4. Int. GI/ITG/GMA-Fachtagung, Fehlertolerierende Rechensysteme/Fault-Tolerant Computing Systems,* Karlsruhe, 1989.
3. Borr, A., "Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing," in: *Proc. 7th VLDB,* 1981, pp. 155-165.
4. "Report of the CODASYL Date Description Language Committee," in: *Information Systems,* Vol. 3, No. 4, 1978, pp. 247-320.
5. Eswaran, K.P., Gray, J., Lorie, R., Traiger, I., "The Notion of Consistency and Predicate Locks in a Database System", in: *Comm. of the ACM,* Vol. 19, No. 11, Nov. 1976.
6. Garcia-Molina, H., Salem, K., "Sagas", in: *Proc. SIGMOD*, San Francisco 1987, S. 249-259.
7. Gray, J.P., "Advanced program-to-program communication in SNA", in: *IBM Systems Journal*, Vol. 22, No. 4, 1983, S. 298-318.
8. Gray, J.N., "An Approach to Decentralized Data Management Systems," in: *IEEE Transactions on Software Engineering,* Vol. 12, No. 6, 1986, pp. 684-692.

9. Gray, J.N., Good, B., Gawlick, D., Homann, P., Sammer, H., "One Thousand Transactions Per Second," in: *Proc. IEEE CompCon Spring '85,* San Francisco, Febr. 1985, pp. 3-12.
10. Gray, J., Lorie, R., Putzolu, F., Traiger, I., "Granularity of Locks and Degrees of Consistency in a Shared Data Base", in: *Proc. IFIP Working Conf. on Modelling of Database Management Systems*, Freudenstadt, Jan. 1976.
11. Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B., "Processing and transaction concepts for cooperation of engineering workstations and a database server", in: *Data and Knowledge Engineering* 3, 1988, pp. 87-107.
12. Härder, T., Meyer-Wegener, K., "Transaktionssysteme und TP-Monitore - Eine Systematik ihrer Aufgabenstellung und Implementierung," in: *Informatik - Forschung und Entwicklung,* Bd. 1, H. 1, 1986, S. 3-25.
13. Härder, T., Meyer-Wegener, K., "Die Zusammenarbeit von TP-Monitoren und Datenbanksystemen in DB/DC-Systemen - Existierende Systeme und zukünftige Entwicklungen," in: *Informatik - Forschung und Entwicklung,* Bd. 1, H. 3, 1986, S. 101-122.
14. Härder, T., Reuter, A., Principles of Transaction-Oriented Database Recovery, in: *ACM Computing Surveys*, Vol. 15, No. 4, Dec. 1983, S. 287-317
15. Häussermann, F., Viererbl, P., Universeller Transaktionsmonitor UTM - Das Transaktionssystem des BS2000, in: *Siemens telcom report* 2, H. 6, 1979, S. 387-393.
16. Häussermann, F., "Verteilte Transaktionsverarbeitung und verteilte Datenhaltung - eine Übersicht und Lösungen", in: *Elektr. Rechenanlagen,* Bd. 27 (1985), S. 15-22.
17. Helland, P., "Transaction Monitoring Facility (TMF)," in: *IEEE Database Engineering,* Vol. 8, No. 2, 1985, pp. 11-18.
18. "Customer Information Control System / Virtual Storage (CICS/VS) - Application Programmer's Reference Manual (Command Level)," *IBM Corporation*, 3rd. edition (Version 1.5), Order No. SC33-0077-2, May 1980.
19. "Customer Information Control System / Virtual Storage (CICS/VS) - General Information," *IBM Corporation*, Order No. GC33-0155-1, 1982.
20. ISO, "Information Processing Systems, Open Systems Interconnection, Distributed Transaction Processing", Part 1-3, *Draft Proposal ISO* 10026/1-3.
21. Kugler, H., "Konzept der verteilten Transaktionsverarbeitung mit UTM," in: *Siemens telcom report* 7 (1984), H. 4, S. 249-253.
22. Meyer-Wegener, K., "Transaktionssysteme - verteilte Verarbeitung und verteilte Datenhaltung," in: *Informationstechnik,* 29. Jg., H. 3, 1987, pp. 120-126.
23. Mohan, C., Lindsay, B., "Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions", in: *Proc. 2nd ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, Montreal, Canada, August 1983.
24. O'Neil, P., "The Escrow Transactional Method", in: *ACM Transactions on Database Systems*, Vol. 11, No. 4, December 1986.
25. Pappe, S., "Datenbankzugriff in offenen Rechnernetzen", *Dissertation*, Universität Kaiserslautern, März 1990.
26. Pappe, S., Heil, H.-L., Effelsberg, W., Lamersdorf, W., "Datenbankzugriff in offenen Rechnernetzen," in: *Tagungsband GI-Fachtagung "Datenbanksysteme für Büro, Technik und Wissenschaft"*, IFB 136, Springer-Verlag, April 1987, S. 375-390.
27. Reuter, A. et al., "ConTracts - a Means of Extending Control Beyond Transaction Boundaries", in: *Proc. Int. Workshop on High Performance Transaction Systems*, Asilomar, CA, Sept. 1989.
28. Rheinberger, B., "Ein Reiseberatungs-Expertensystem als Anwendung des KBMS KRISYS", Diplomarbeit, Univ. Kaiserslautern, Fachbereich Informatik, April 1989.
29. Voydock, V.L., Kent, S.T., "Security Mechanisms in High-Level Network Protocols", in: *ACM Computing Surveys*, Vol. 15, No. 2, June 1983, pp. 135-171.
30. Wedekind, H., "Die Problematik des Computer Integrated Manufacturing (CIM) - Zu den Grundlagen eines "strapazierten" Begriffes -", in: *Informatik-Spektrum*, Vol. 11, 1988, S. 22-39.
31. Yelowich, B.M., "Customer Information Control System - An evolving system facility," in: *IBM Systems Journal,* Vol. 24, Nos. 3/4, 1985, pp. 264-278.