in: Proc. 19. GI-Jahrestagung, Informatik-Fachberichte 222, München 1989, Band 1, S. 259-273

Aspekte der Datenbank-Anbindung in workstation-orientierten Ingenieuranwendungen

Christoph Hübel, Bernd Sutter
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, D-6750 Kaiserslautern

Überblick

Die Entwicklung in den verschiedenen Bereichen der rechnergestützten Ingenieuranwendungen wird derzeit zum einen durch eine zunehmende Verbreitung von Arbeitsplatzrechnern bestimmt, zum anderen wird immer häufiger die Forderung nach einer durchgängigen, zentralen Datenhaltung erhoben. Hierfür müssen neue Konzepte zur Datenbank(DB)-Anbindung in workstation-orientierten Ingenieurumgebungen entwickelt werden.

Dieser Beitrag beschäftigt sich mit der Struktur von workstation-orientierten DB-Anbindungen und zeigt Konzepte für die Entwicklung adäquater Verarbeitungsmodelle auf, die neben der zugrundeliegenden Hardware-Architektur (Workstation-Server-Umgebung) auch die spezifischen Anforderungen der Ingenieuranwendungen erfüllen müssen. Ein weiteres Kapitel beschäftigt sich dann mit einem Vorschlag für die Architektur von DB-gestützten Anwendungssystemen in einer workstation-orientierten Hardware-Umgebung.

1. Einleitung

In immer mehr Anwendungsbereichen ist eine zunehmende Verbreitung von Arbeitsplatzrechnern (Workstations) zu beobachten. Dies trifft besonders auf den Bereich der Ingenieuranwendungen zu (CAD/CAM, Entwurf von Elektronikschaltungen und VLSI-Bausteinen), wo typischerweise eine ganz spezifische Endgerätefunktionalität und eine hohe lokale Rechnerleistung gefordert wird. Die derzeitige Entwicklung ist durch eine massive Vernetzung und durch die Integration zentraler Dienste über Mainframe-Rechner (Server) bestimmt. Neben der auf die speziellen Anwendungs- und Benutzerbedürfnisse ***zugeschnittenen Funktionalität*** der Workstations (graphische Ein-/Ausgabe, Window-Technik, Ein-/Ausgabe von Analogsignalen usw.) rückt mit zunehmender Rechnerleistung und dem gleichzeitigen Preisverfall immer deutlicher der ***Verarbeitungsaspekt*** in den Mittelpunkt des Interesses: durch den Workstation-Einsatz am Arbeitsplatz kann der einzelne Benutzer mit ungewohnt hoher Rechnerleistung und hoher Speicherkapazität versorgt werden, bei einer gleichzeitigen Verbesserung der Verfügbarkeit. Als Konsequenz werden nun große Teile der Anwendungsalgorithmen direkt auf Workstation-Seite ausgeführt, womit eine weitgehend lokale und daher effiziente Bearbeitung der einzelnen Benutzeroperationen erreicht wird.

Insgesamt ergeben sich aus dieser Entwicklung eine Reihe neuer Möglichkeiten; es entstehen aber auch spezielle Anforderungen an integrierte Anwendungssysteme. Dies gilt in besonderem Maße für datenbankgestützte Anwendungssysteme, die ja, quasi "per Definition", eine integrierte Sichtweise auf gemeinsame Datenbestände ermöglichen. Für diese meist datenintensiven Anwendungen ist eine besonders sorgfältige Strukturierung der Software-Komponenten erforderlich, wenn die ***Workstation/Server-Schnittstelle*** nicht zu einem Engpaß werden soll, der die Akzeptanz des gesamten Anwendungssystems in Frage stellt /

DGKOW86, Re87/. Daneben muß durch die Software-Strukturierung die ***Expansion der Hardware-Umgebung*** optimal unterstützt werden, will man nicht an einen einmal gewählten Systemausbau dauerhaft gebunden sein. Vom Standpunkt des Benutzers aus sollte die Anzahl der zusammengeschlossenen Workstations idealerweise nur geringfügigen Einfluß auf das Systemverhalten (insbesondere die Reaktionszeiten der Anwendungsprogramme) ausüben.

Im folgenden Kapitel wollen wir die für die Effizienz eines Anwendungssystems in einer Workstation/Server-Umgebung entscheidenden Schnittstellenaspekte diskutieren. Dabei steht zunächst die Frage nach einer sinnvollen Aufgabenverteilung und der damit verbundenen Konzeption zur dezentralen Verarbeitung im Vordergrund. Danach stellen wir ein Verarbeitungsmodell zur Unterstützung von Ingenieuranwendungen auf Arbeitsplatzrechnern vor. Im Anschluß daran erläutern wir eine Grobarchitektur, die einen Rahmen für DB-gestützte Ingenieur-Anwendungssysteme in einer Workstation/Server-Umgebung darstellt.

2. Struktur einer workstation-orientierten DB-Anbindung

Die Anbindung der auf Workstation-Seite ablaufenden Anwendungsalgorithmen an einen gemeinsamen Datenbestand bestimmt die Gestaltung der Workstation/Server-Schnittstelle und ist damit entscheidend für die Effizienz des Gesamtsystems. Allgemein können die folgenden Kriterien zur Schnittstellengestaltung genannt werden:

- die Anzahl der Workstation/Server-Aufrufe sollte auf ein Minimum beschränkt werden,
- es sollten nur die tatsächlich benötigten Daten zur Workstation übertragen werden (minimales Datentransportvolumen),
- die Arbeitsverteilung sollte möglichst ökonomisch erfolgen, d.h., es sollte möglichst wenig "Arbeit" auf Workstation- bzw. Server-Seite wiederholt werden müssen, und
- es sollte ein möglichst hohes Maß an Unabhängigkeit zwischen Workstation und Server, aber auch zwischen verschiedenen Workstations erreicht werden (Workstation-Autonomie).

Homogene vs. heterogene DBS-Aufteilung

Insbesondere die Forderung nach einer hohen Workstation-Autonomie macht die Trennung zwischen einer globalen (server-seitigen) und lokalen (workstation-seitigen) Datenhaltungskomponente erforderlich, wobei das Zusammenspiel der beiden Komponenten durch die übrigen Kriterien bestimmt wird. Ein erster, naheliegender Ansatz zu einer konkreten Systemgestaltung besteht im Einsatz eines verteilten DBS /BEEK84/ (DDBS) (bzw. in der Verwendung von Konzepten aus dem Bereich der DDBS-Systeme). Auf Server- und auf Workstation-Seite befinden sich dabei gleichartig ***homogene Knoten*** des DDBS, die i. allg. über ***dezentrale Verwaltungsmechanismen*** eine transparente Sicht auf physisch verteilte Daten realisieren (Bild 1a). Kann nun die aktuelle Datenverteilung so organisiert werden, daß möglichst viele der auf Workstation-Seite benötigten Daten vom jeweils lokalen Knoten des DDBS verwaltet werden, so ist relativ einfach eine autonome Verarbeitung auf Workstation-Seite möglich. Allerdings ist hierbei zu bedenken, daß die Datenanforderungen auf Workstation-Seite sehr dynamisch sind und sich die Anforderungen verschiedener Workstations oftmals gegenseitig überlappen, so daß wesentlich flexiblere Verteilungsmechanismen benötigt werden, als dies in heutigen DDBS vorgesehen ist.

Der sehr allgemeine Ansatz verteilter Datenbanksysteme kann in speziellen Anwendungsumgebungen, etwa in dem Bereich workstation-basierter Ingenieuranwendungen, soweit abgewandelt werden, daß man zu spezielleren und für die betrachtete Anwendungsklasse besser angepaßten Lösungen gelangt. Dies führt zu dem Ansatz einer ***heterogenen DBS-Aufteilung***, der im weiteren diskutiert werden soll. Er geht von einer zentralen Datenverwaltungskomponente aus, die Mechanismen für eine ***dezentrale Verarbeitung*** anbietet. Das Zusammenspiel zwischen der Datenhaltungskomponente auf Server- und Workstation-Seite ist demnach nicht mehr bestimmt durch den Aspekt der Datenverteilung, sondern vielmehr durch den Aspekt der ***Datenbereitstellung*** (Bild 1b), d.h., die auf Workstation-Seite benötigten Daten, die den "Gegenstand der Verarbeitung" darstellen, werden auf der Seite der globalen Datenhaltung extrahiert und zur lokalen Datenhaltung transferiert, wo sie zur weiteren Verarbeitung bereitgestellt werden. Ein entscheidendes Argument für diese differenzierende Betrachtungsweise sehen wir in den unterschiedlichen Anforderungsprofilen an die Datenhaltung auf Workstation- und Server-Seite. Die Forderungen der globalen Datenhaltung sind angelehnt an die klassischen DBS-Eigenschaften, etwa die zentrale und algorithmen-neutrale Beschreibung aller relevanten Datenstrukturen, die Unabhängigkeit vom real verfügbaren Hauptspeicherbereich, die Kontrolle des Mehrbenutzerbetriebs sowie die automatischen Sicherungsmaßnahmen (Logging/Recovery) und die Maßnahmen der Zugriffskontrolle.

Diese Punkte spielen für die lokale Datenhaltungsseite keine bzw. nur eine geringfügige Rolle; dagegen treten gerade für den Bereich interaktiver Ingenieuranwendungen Verarbeitungsaspekte in den Mittelpunkt /KWDKL89/:

- effiziente Datenanbindung und damit verbunden
- adäquate Zugriffsoperationen sowie
- individuelle Maßnahmen zur autonomen "Fehlerbehandlung".

Die dominierende Bedeutung der Datenanbindung für das resultierende Leistungsverhalten wird klar, wenn man die Zugriffscharakteristik entsprechender Anwendungssysteme betrachtet. So sind Häufigkeiten zwischen 10^3 und 10^6 DB-Referenzen im Rahmen einer einzigen Benutzerinteraktion nicht selten. Um angesichts solcher Referenzhäufigkeiten ein akzeptables Antwortzeitverhalten zu erreichen, muß eine engere Bind-

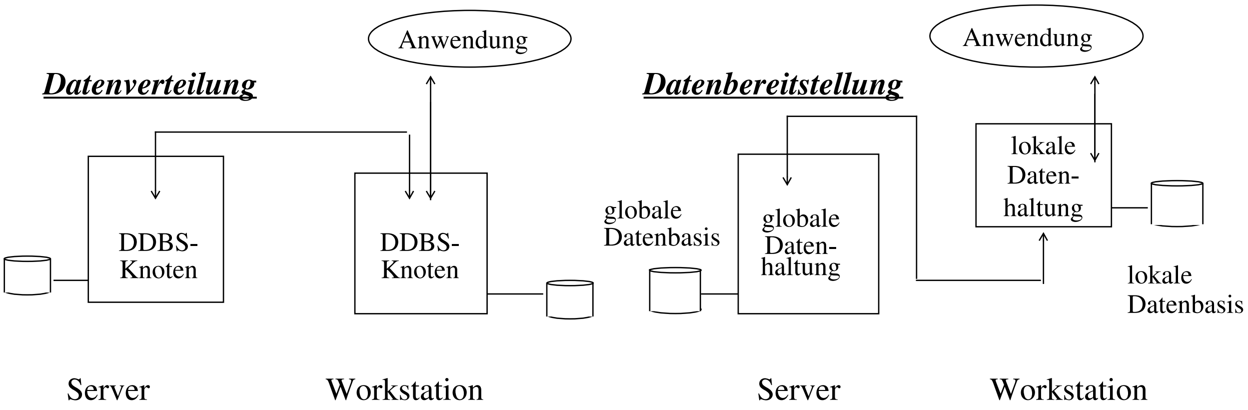


Bild 1a: homogene DDBS-Knoten realisieren eine transparente Verteilung der Daten

Bild 1b: heterogene Datenhaltungskomponenten erlauben eine angemessene Spezialisierung

ung der zu verarbeitenden Daten an die Anwendungsprogramme erfolgen, als dies in konventionellen DBS üblicherweise der Fall ist /HP89/. Aus Sicht der Anwendungsprogramme ist es daher wünschenswert, wenn nicht sogar notwendig, zwischen den Daten der Datenbank und den auf Workstation-Seite bereitgestellten Daten unterscheiden zu können. Die von verteilten DBS realisierte Ortstransparenz der Daten ist daher, zumindest in den hier unterstellten Anwendungsbereichen, nicht erwünscht.

3. Ein Verarbeitungsmodell für workstation-orientierte Anwendungen

Unsere weiteren Überlegungen beschäftigen sich mit der Entwicklung eines Verarbeitungsmodells, das von dem Ansatz einer heterogenen DBS-Aufteilung ausgehend eine adäquate Unterstützung für Ingenieuranwendungen bieten soll. Durch das Verarbeitungsmodell wird die Art und Weise des Datenzugriffs ebenso wie die Strukturierung der Verarbeitung festgelegt:

- Datenanbindung einer Anwendung

  Wichtigste Aufgabe des Verarbeitungsmodells ist eine Beschreibung der Datenanbindung, d.h., es werden die Sichtweise einer Anwendung auf die globale und lokale Datenhaltung definiert und die Frage des Datenzugriffs zur Durchführung der Verarbeitung geklärt.

- Strukturierung der Verarbeitung

  Das Verarbeitungsmodell umfaßt darüberhinaus die dynamischen Aspekte der Verarbeitung. Es werden sowohl Konzepte zur Strukturierung eines (möglicherweise einige Tage dauernden) Verarbeitungsabschnittes als auch Mechanismen zur Konsistenzsicherung angeboten.

Die hier unterstellte heterogene DBS-Aufteilung mit einer globalen und lokalen Datenhaltung erlaubt mehrere Änderungssemantiken bei der Bereitstellung der zu verarbeitenden Daten (vgl. dazu auch /JWZ87/):

- Auf Workstation-Seite wird eine lokale Kopie der Daten angelegt, d.h., Änderungen auf der Kopie werden in der globalen Datenhaltung nicht berücksichtigt; ebenso werden nachträgliche Änderungen auf den globalen Daten in der lokalen Kopie nicht sichtbar. Eine solche Vorgehensweise ist sinnvoll, wenn die Daten nur gelesen, bzw. keine dauerhaften Änderungen ausgeführt werden sollen.
- Änderungen werden sofort in der globalen Datenhaltung reflektiert, d.h., auf beiden Datenbeständen ist der gleiche Änderungsstand wiedergegeben. Dies setzt allerdings voraus, daß entweder die Änderungen auf beiden Datenbeständen gleichzeitig durchgeführt werden, was zu einem erheblichen Mehraufwand und damit einer sehr ineffizienten Verarbeitung führen kann, oder daß die zu ändernden Daten nicht in der lokalen Datenhaltung bereitgestellt wurden und somit nur eine direkte Änderung auf den globalen Daten ausgeführt wird.
- Änderungen der lokalen Daten werden verzögert (z.B. am Ende eines Verarbeitungsschrittes) in der globalen Datenhaltung nachgefahren.

Im folgenden werden wir ein Verarbeitungsmodell vorstellen, das die verschiedenen Änderungsvarianten berücksichtigt und die Strukturierung eines Verarbeitungsabschnittes erlaubt. Am Ende des Kapitels werden wir die auftretenden Konsistenzprobleme beim Datenzugriff diskutieren und einen möglichen Lösungsweg aufzeigen.

3.1 Ein hybrides Verarbeitungskonzept

In der beschriebenen Systemumgebung ist eine Verarbeitung auf der globalen und der lokalen Datenhaltungseite möglich. Während für die Verarbeitung auf globaler Seite die Algorithmen (in Form von DB-Anweisungen) zu den Daten gebracht werden, werden bei der lokalen Verarbeitung die Daten zu den Verarbeitungsalgorithmen gebracht.

Die lokale Verarbeitung erfordert ein ***zweistufiges Verarbeitungskonzept***. Die Idee hinter diesem Vorgehen ist einfach: können im Vorfeld der eigentlichen Verarbeitung ein bzw. mehrere "Verarbeitungsgegenstände" bereits ermittelt und zur lokalen Datenhaltung auf die Workstation-Seite transferiert werden (CHECKOUT), so sind die folgenden Verarbeitungszugriffe um so direkter und damit kostengünstiger zu realisieren. Nach erfolgter Verarbeitung müssen dann die veränderten Daten zurück zur globalen Datenhaltung übermittelt werden (verzögertes Einbringen der Änderung, CHECKIN) (Bild 2a). Ein solches Vorgehen erscheint sinnvoll und auch praktikabel, wenn die Verarbeitungsgegenstände nicht zu klein und dennoch gut beschreibbar sind, wenn eine große Anzahl von Folgezugriffen erwartet wird und wenn die bereitgestellten Daten für einen längeren Zeitraum auf Workstation-Seite verbleiben können (was i. allg. die Möglichkeit einer lokal persistenten Speicherung voraussetzt).

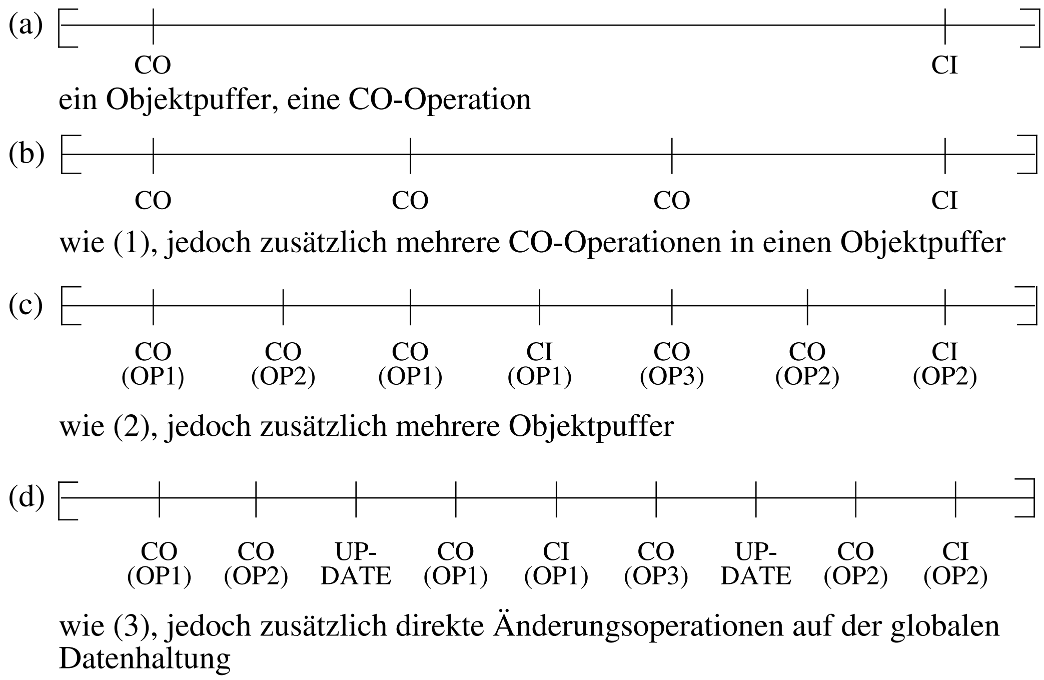
In den hier betrachteten Ingenieuranwendungen kann die erste Stufe, nämlich eine adäquate Beschreibung der bereitzustellenden Daten, zu einem eigenständigen Problem werden. Dieses Problem hängt wesentlich vom Grad der Objektunterstützung des an der globalen Datenhaltungs-Schnittstelle angebotenen Datenmodells ab. Hier vereinfachen strukturell-objektorientierte DBS /Di86/, die eine ganzheitliche Handhabung von komplexen Objektstrukturen erlauben, wesentlich die Beschreibung und Extraktion der Datenmenge. Für den vorliegenden Beitrag gehen wir davon aus, daß die meist umfangreichen und komplexen Strukturen der Anwendungsobjekte mehr oder weniger gut prädikativ beschreibbar sind. In der Regel wird durch eine Reihe von DB-Anfragen, ausgehend von elementaren "Datenbausteinen" (Tupeln im relationalen Datenmodell, Records im Netzwerk-Datenmodell) und unter Ausnutzen von Beziehungsinformationen eine aktuelle Objektstruktur ermittelt. Dies geschieht auf der Server-Seite. Die ausgewählten Daten werden auf die Workstation-Seite transferiert und in einer Hauptspeicherstruktur aufgebaut, die zusammen mit den zu ihrer Verwaltung notwendigen Strukturen im folgenden als ***Objektpuffer*** bezeichnet wird.

In der zweiten Stufe erfogt dann die eigentliche Verarbeitung. Hier halten wir eine mehr prozedurale oder navigierende Vorgehensweise für angebracht, da so eine einfache, sehr direkte und effiziente Datenversorgung der Anwendungsalgorithmen erreicht werden kann /HS89/.

Bisher haben wir die Ausführung einer CHECKOUT-Operation und die Abspeicherung der Daten in einem Objektpuffer betrachtet. Für die hier unterstellten Ingenieuranwendungen sind flexiblere Datenbereitstellungsmechanismen wünschenswert. Häufig soll oder kann der Verarbeitungsgegenstand nicht vollständig zu Beginn des Verarbeitungsvorganges aus der globalen Datenhaltung extrahiert werden, vielmehr ist eine ***flexible und inkrementelle Erweiterung*** des in einem Objektpuffer repräsentierten Verarbeitungskontextes während einer Verarbeitung wünschenswert. Damit können gezielt die Daten in die lokale Datenhaltung transferiert werden, die zur Durchführung eines nächsten Verarbeitungsschrittes benötigt werden. Die Ausführung mehrerer, unabhängiger CHECKOUT-Operationen, die die Daten im gleichen Objektpuffer ablegen, erlaubt eine derartige dynamische Datenbereitstellung (Bild 2b). Dabei tritt i.allg. das Problem auf, daß eine CHECKOUT-Operation Daten extrahiert, die bereits im Objektpuffer eingelagert sind. Durch solche nachfolgenden CHECKOUT-Operationen dürfen die bereits im Objektpuffer eingelagerten Daten nicht überschreiben werden, da ansonsten die auf den Daten eventuell durchgeführten Änderungen rückgängig gemacht würden. Nach erfolgter Verarbeitung werden die Daten durch eine CHECKIN-Operation in die globale Datenhaltung zurückgeschrieben. Sie sind danach in der lokalen Datenhaltung nicht mehr enthalten. Die Ausführung von inkrementellen CHECKIN-Operationen scheint nicht sinnvoll, da der Inhalt eines Objektpuffers als eine logische Einheit zu betrachten ist.

Eine nächste Erweiterung ist die Einführung mehrerer, unabhängiger Objektpuffer in der lokalen Datenhaltung. Dies erlaubt eine logische Zuordnung von unabhängigen Verarbeitungsgegenständen zu je einem Puffer (Bild 2c). Somit können die einzelnen Verarbeitungsgegenstände unabhängig voneinander (also auch vor Beendigung des Verarbeitungsabschnittes) durch eine CHECKIN-Operation in die globale Datenhaltung eingebracht werden. Hierdurch wird zum einen eine Strukturierung der lokal verfügbaren Daten erreicht und zum anderen eine Voraussetzung zur Realisierung von Gruppenarbeit geschaffen: z.B. können andere Mitglieder einer Arbeitsgruppe auf vorzeitig in den globalen Datenbestand eingebrachte und bedingt freigegebene Daten zugreifen /Di88/. Eine Aufteilung der lokalen Datenhaltung in mehrere Puffer unterstützt zudem die Ausführung von parallelen Aktivitäten in der Anwendung (in CAD-Anwendungen ist etwa die Durchführung von zeitintensiven Berechnungsalgorithmen und das gleichzeitige Aktivieren von Recherchefunktionen wünschenswert). Ein weiteres Argument liegt in der differenzierenden Behandlung der eingelagerten Daten aufgrund unterschiedlicher Zugriffsrechte. Eine Anwendung fordert die in der lokalen Datenhaltung bereitzustellenden Daten i.allg. teils mit lesendem, teils mit änderndem Zugriffsrecht an. Durch eine Abspeicherung der angeforderten Daten in verschiedenen Puffern wird eine Trennung nach unterschiedlichen Zugriffsrechten gewährleistet.

Darüber hinaus werden in Ingenieuranwendungen häufig Operationen ausgeführt, die keine Lokalität aufweisen und die sich einfach mit den (deskriptiven) Manipulationsoperationen der globalen Datenhaltungskomponente ausdrücken lassen. Diese ***Operationen*** werden sinnvoller Weise unmittelbar ***auf Server-Seite ausgeführt*** (UPDATE), da ansonsten häufig große Datenmengen zur Ausführung einer einzigen, kurzen Operation auf Workstation-Seite transferiert werden müßten (Bild 2d).



CO: CHECKOUT ; CI: CHECKIN ; OP: Objektpuffer

Bild 2: Stufenweise Erweiterung des Verarbeitungsmodells

Da in dem von uns vorgeschlagenen Verarbeitungsmodell sowohl direkte Änderungen auf der globalen Datenhaltung als auch lokale Änderungen, die verzögert in den globalen Datenbestand eingebracht werden (deferred update), erlaubt sind, bezeichnen wir diesen Ansatz als ein ***hybrides Verarbeitungsmodell***.

Beispiel

Wir wollen nun an einem Beispiel aus dem Konstruktionsbereich im Maschinenbau die Verwendung des vorgestellten Verarbeitungsmodells verdeutlichen. Wir betrachten zunächst den Prozeß des Baugruppen-Entwurfs /HPS89/. Baugruppen sind Komponenten einer Maschine bzw. Anlage, die eine Teilfunktion erfüllen. Eine Baugruppe kann wieder aus Baugruppen oder Einzelteilen bestehen. Einzelteile sind Teile einer Baugruppe, die entweder vorgefertigt sind oder in einer späteren Entwurfsphase, der Einzelteildetaillierung, vom Konstrukteur entworfen werden.

Im Baugruppen-Entwurf ist nun der (hierarchische) Aufbau einer Maschine aus Baugruppen bis hin zur Festlegung der Aufgabenspezifikation von eingehenden Einzelteilen durchzuführen. Soll beispielsweise ein Motor mit umschließendem Gehäuse entworfen werden, wobei für den Motor und das Gehäuse selbst bereits ein Baugruppen-Entwurf durchgeführt wurde, so ist folgende Vorgehensweise denkbar: Zunächst wird die Baugruppen-Struktur des Motors in einen Objektpuffer der lokalen Datenhaltung der Workstation gebracht (erste CHECKOUT-Operation). Danach wird die Gehäuse-Struktur in den gleichen Objektpuffer geladen (zweite CHECKOUT-Operation). Will der Konstrukteur während der Bearbeitung der Baugruppen-Struktur auf andere Informationen zugreifen, so daß er sich z.B. die Form des (bereits entworfenen) Gehäuses anschauen möchte, so wird die geometrische Information in einen zweiten Objektpuffer geladen und das Objekt visualisiert. Danach können die Daten im zweiten Objektpuffer freigegeben werden. Nach erfolgreich durchgeführtem Baugruppen-Entwurf werden schließlich die modifizierten Daten des ersten Objektpuffers durch eine CHECKIN-Operation zurück in die globale Datenhaltung gebracht.

Die Möglichkeiten der Direktänderungen hängen wesentlich von der Mächtigkeit des von der globalen Datenhaltungskomponente angebotenen Datenmodells ab. In einfachen Fällen werden auf Workstation-Seite berechnete Datenwerte, etwa das Volumen oder die Masse eines Einzelteils, direkt in den globalen Daten geändert. Erlaubt das Datenmodell komplexere Operationen (z.B. rekursive Stücklistenauflösung beim Baugruppen-Entwurf), so können diese Operationen auf Server-Seite ausgeführt und das Ergebnis (z.B. Zahl der benutzten Einzelteile) ebenfalls direkt in den globalen Daten abgelegt werden.

Strukturierung der Verarbeitung

Eine wichtige Forderung an die workstation-lokale Datenhaltung betrifft die individuelle Reaktion auf Fehlersituationen. Das automatische Rücksetzen bei System- oder Benutzerfehlern erscheint im Fall einer dialog- und verarbeitungsintensiven Workstation-Nutzung und vor dem Hintergrund lang andauernder Verarbeitungsabschnitte nur in Ausnahmefällen sinnvoll. Dagegen halten wir benutzergesteuerte Logging- und (vor allem aber auch) Recovery-Maßnahmen für angebracht. Hierzu werden Operationen bereitgestellt, die eine lokale Sicherung des Verarbeitungszustandes (dieser umfaßt neben den selektierten Verarbeitungsgegenständen auch temporäre Anwendungsdaten) bewirken (SAVE), bzw. einen identifizierten Zustand wieder herstellen (RESTORE). In ähnlicher Weise kann eine Anwendung unterbrochen (SUSPEND) und zu einem späteren Zeitpunkt wieder aufgenommen werden (RESUME) /HHMM88/. Bild 3 zeigt eine mögliche Ablaufstruktur eines Verarbeitungsabschnittes auf Workstation-Seite.

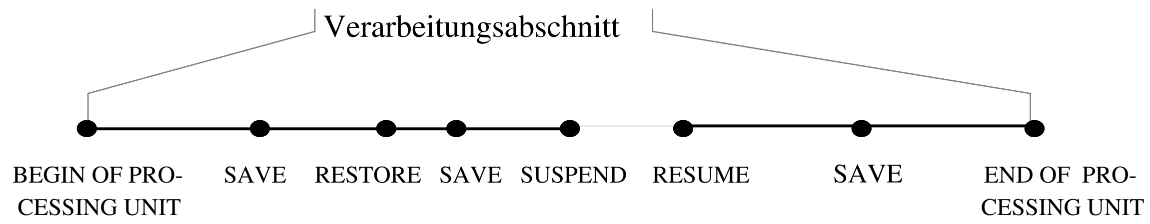


Bild 3: Ablaufstruktur eines Workstation-Verarbeitungsabschnittes

3.2 Durch das Verarbeitungsmodell bedingte Konsistenzprobleme

Das vorgestellte hybride Verarbeitungsmodell beruht im wesentlichen auf Nützlichkeitsüberlegungen. Besonders die zweistufige Datensicht verspricht eine entscheidende Leistungsverbesserung. Auf der anderen Seite wirft diese Art der Verarbeitung eine Reihe von Problempunkten auf, die sich durch die Existenz mehrerer Pufferungsbereiche auf Workstation-Seite und durch die rechnerübergreifende Schnittstelle zwischen Workstation und DB-Server weiter verstärken. Vor allem die ***Konsistenz*** zwischen lokalem und globalem Datenbestand ist hiervon betroffen. Bild 4 macht die Problemstellung an einem Beispiel deutlich. Das mit X bezeichnete Datenelement befindet sich nach einem entsprechenden CHECKOUT (1) auf Workstation-Seite in einem Objektpuffer und kann dort unmittelbar durch ein Anwendungsprogramm manipuliert werden. Gleichzeitig ist aber auch eine Modifikation des "Originalelementes" auf Server-Seite durch eine Direktänderung (2) innerhalb des gleichen Anwendungsprogramms denkbar. Ein späteres CHECKIN (4) würde diese Änderung "überschreiben", d.h., es würde ein "lost-update"-Fehler auftreten. Entsprechendes gilt für den Fall, daß ein Datenelement gleichzeitig in mehreren Pufferbereichen auftritt (3), sich die Pufferinhalte also überlappen. Neben dem direkten Verlust von Modifikationen kann es indirekt zu einer Invalidierung von Beziehungen kommen, wenn die Beziehungsinformation inhärent mit den in Beziehung stehenden Elementen verbunden ist (beispielsweise beinhalten Tupel im Relationenmodell Beziehungsinformation in Form von Fremdschlüsselattributen). Die Beziehung der in Bild 4 gezeigten Elemente X und

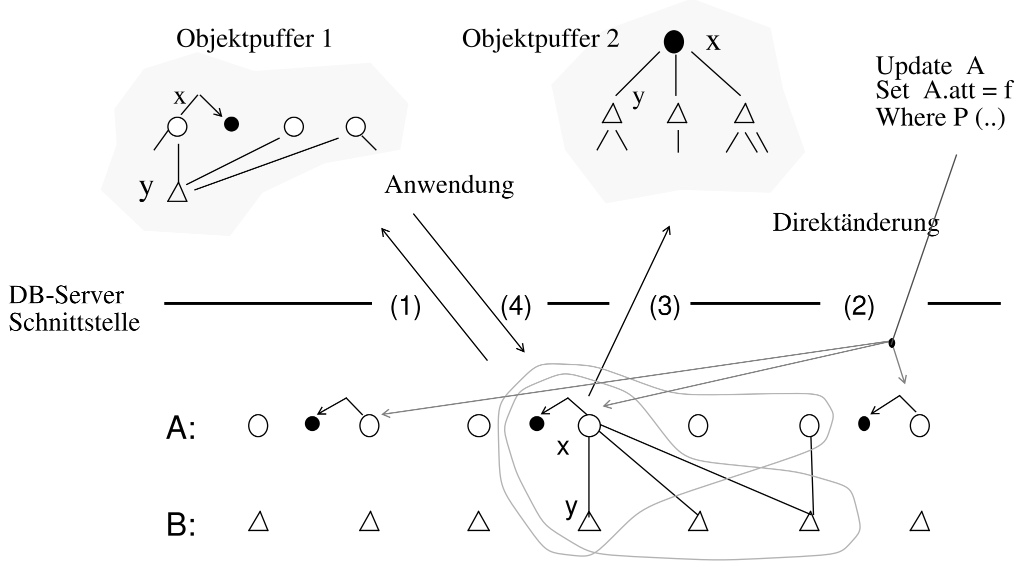
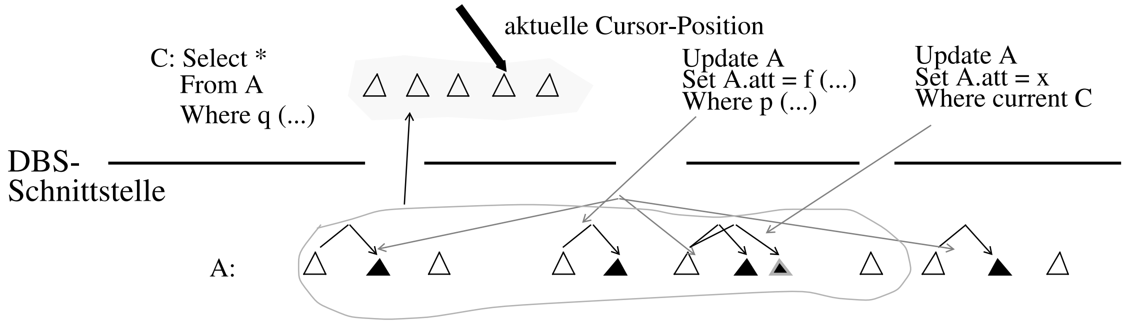


Bild 4: Durch Direktänderungen verursachte "lost-update"-Fehler

Y könnte z.B. in einem der Objektpuffer gelöscht werden. Die referentielle Integrität geht damit zumindest aus Sicht der Anwendung verloren.

Ein weiterer, hiermit zusammenhängender Integritätsaspekt ergibt sich aufgrund der Tatsache, daß auf Workstation-Seite stets nur ein Ausschnitt des gesamten Datenbestandes vorliegt und bestehende Integritätsbedingungen lediglich auf diesem Ausschnitt überprüft werden können. Es ist daher sinnvoll, einen ***abgestuften Konsistenzbegriff*** einzuführen, wobei unterschieden wird zwischen Integritätsbedingungen, die einen "lokal-konsistenten" Zustand eines Objektpuffers bestimmen (Konsistenz ***eines*** Verarbeitungsgegenstandes), und denen, die einen "umfassend-konsistenten" Zustand festlegen. Manche Bedingungen lassen sich überhaupt nicht lokal prüfen (wie z.B. Kardinalitätsrestriktionen) und müssen daher generell am Ende eines Verarbeitungsabschnittes validiert werden, also zu einem Zeitpunkt, zu dem alle Modifikationen im globalen Datenbestand des DB-Servers reflektiert sind. Im Fall einer Konsistenzverletzung erscheint uns eine "konstruktive" und im Dialog mit dem Anwendungsprogramm durchgeführte Fehlerbeseitigung unbedingt erforderlich, da ein Rücksetzen eines gegebenenfalls lang andauernden Verarbeitungsabschnittes nur in Ausnahmefällen akzeptiert werden kann. Die dabei auftretenden Probleme der Kommunikation und Umsetzung von Fehlerbedingungen innerhalb des Datenmodells in die Anwendungsumgebung sollen hier allerdings nicht weiter diskutiert werden.

Die oben angesprochenen Problempunkte wurzeln keineswegs ausschließlich in der physischen Trennung zwischen Workstation und DB-Server; sie sind vielmehr Bestandteil der allgemeinen Anbindungsproblematik von Anwendungsprogrammen an DBS. Dies wird deutlich am Beispiel relationaler Systeme, die in der Regel homogene Tupelmengen über ein Cursor-Konzept an Anwendungsprogramme binden (vgl. Bild 5). Der Zeitpunkt der tatsächlichen Verarbeitung weicht vom Zeitpunkt der Datenbereitstellung durch das DBS ab und kann nur sehr eingeschränkt kontrolliert werden. Zwar sind in der Regel lokale Änderungen auf den bereitgestellten Daten nicht möglich, allerdings werden überlappende Ergebnismengen nicht ausgeschlossen und Direktänderungen stets erlaubt, so daß im Prinzip alle oben skizzierten Seiteneffekte auftreten können. Z.B. ist die Semantik von Einfüge-, Lösch- und Änderungsoperationen, die anhand von Cursor-Positionen in zuvor bereitgestellten Ergebnismengen durchgeführt werden, relativ unklar, da sie i. allg. stark von der vom DBS aktuell verwendeten Auswertungsstrategie (vollständige vs. schrittweise Anfrageauswertung) abhängt. Die Direktänderung in Bild 5 ist bei einer schrittweisen Auswertung für das Anwen-



<u>Bild5:</u> Änderungsproblematik in relationalen Systemen

dungsprogramm wahrnehmbar, falls die aktuelle Cursor-Position "vor" dem geänderten Tupel liegt. Hat der Cursor das geänderte Tupel bereits erfaßt oder liegt eine vollständige Auswertung vor, so wird dagegen die Änderung für das Anwendungsprogramm nicht sichtbar.

Die geschilderten Probleme scheinen in einfachen Fällen durchaus überschaubar, so daß dem Anwendungsprogramm die volle Verantwortung für eventuelle Seiteneffekte übertragen werden kann. Allerdings sind die Effekte in komplexen Fällen für den Anwender nicht mehr oder nur sehr schwer zu durchschauen. Hier sind unterstützende Mechanismen wünschenswert, die eine eindeutige Sicht eines Datenelements innerhalb einer Anwendung sicherstellen.

Ein Ansatz zu einem solchen Mechanismus sehen wir in dem Konzept der geschachtelten Transaktionen /Mo81/. Die grundlegende Idee hierbei besteht darin, sowohl die Verarbeitung einer bereitgestellten Ergebnismenge als auch Direktänderungen jeweils im Rahmen einer Subtransaktion durchzuführen und damit eine gegenseitige Synchronisation der Zugriffe innerhalb einer Anwendung bzw. eines Verarbeitungsabschnittes zu erreichen. Übertragen auf die beschriebenen Gegebenheiten in einer Workstation-Server-Umgebung könnte ein Transaktionsbaum, wie in Bild 6 gezeigt, entstehen. Dem gesamten Verarbeitungsabschnitt ist die oberste Transaktionsebene zugeordnet. Hierdurch wird automatisch die Synchronisation zwischen mehreren Benutzern gewährleistet. Die lokale Objektpuffer-Verarbeitung (einschließlich der CHECKOUT/CHECKIN-Operationen) erfolgt dann im Rahmen spezieller Subtransaktionen (Objektpuffer-Transaktionen). Direktänderungen werden ebenfalls als Subtransaktionen aufgefaßt, wodurch insgesamt eine Isolation der einzelnen Zugriffe erreicht wird. Im Prinzip sind parallele CHECKOUT/CHECKIN-Operationen innerhalb verschiedener Objektpuffer-Transaktionen sowie parallele Direktänderungs-Transaktionen denkbar (in Bild 6 sind die bereits abgeschlossenen Operationen schattiert dargestellt). Sie erfordern allerdings einen erheblich höheren Realisierungsaufwand und stellen sehr spezifische Anforderungen an die Anwendungsprogrammierung. So muß z.B. die Programmierumgebung für die Anwendung Konzepte zur Handhabung asynchroner Aktivitäten (Initialisierung, Antwortabruf usw.) vorsehen.

Ein anderer Vorteil der an geschachtelten Transaktionen orientierten Betrachtungsweise liegt darin, daß der oben geforderte "abgestufte" Konsistenzbegriff in natürlicher Weise berücksichtigt werden kann, indem

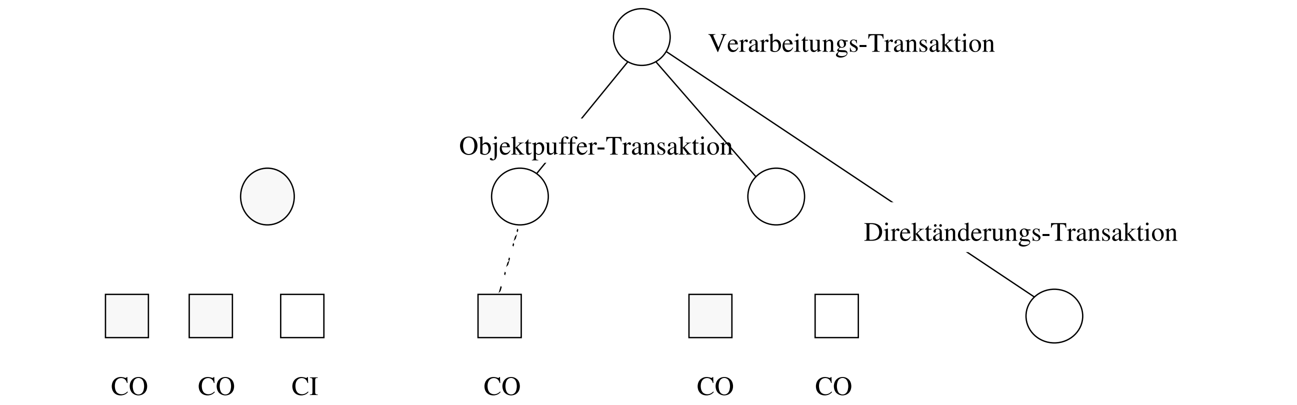


Bild 6: Transaktionsbaum zur Synchronisation der Datenzugriffe innerhalb einer Anwendung

den einzelnen Transaktionsebenen (Verarbeitungsabschnitt, Objektpuffer, CHECKOUT/CHECKIN bzw. Direktänderung) unterschiedliche Integritätsbedingungen zugeordnet werden.

4. Realisierungsaspekte

4.1 Grobarchitektur

Die in Bild 7 dargestellte Software-Architektur geht von einem konventionellen Datenbanksystem aus, auf dem aufbauend eine weitere Systemebene (Aggregationskomponente) realisiert ist, durch die auf Server-Seite eine anwendungsspezifische, strukturell-objektorientierte Schnittstelle /Di86/ angeboten werden kann. Diese Schnittstelle ermöglicht die ganzheitliche Handhabung der komplex-strukturierten Anwendungsobjekte; sie reduziert damit die Anzahl der erforderlichen Server-Aufrufe und erleichtert eine kompakte Übertragung der auf Workstation-Seite erforderlichen Daten. Durch den Einsatz neuerer, sog. Nicht-Standard-Datenbank-Kernsysteme (NDBS-Kerne) /PSSWD87, Da86, Hä88/, die eine solche Schnittstelle anwendungsneutral zur Verfügung stellen, würde die Aufgabe der Aggregationskomponente auf die Verwaltung des Datentransfers beschränkt. Allerdings existieren diese Systeme bislang nur als Prototypen, deren praxisgerechte Umsetzung noch aussteht. Auf Workstation-Seite zeigt Bild 7 außer der eigentlichen An-

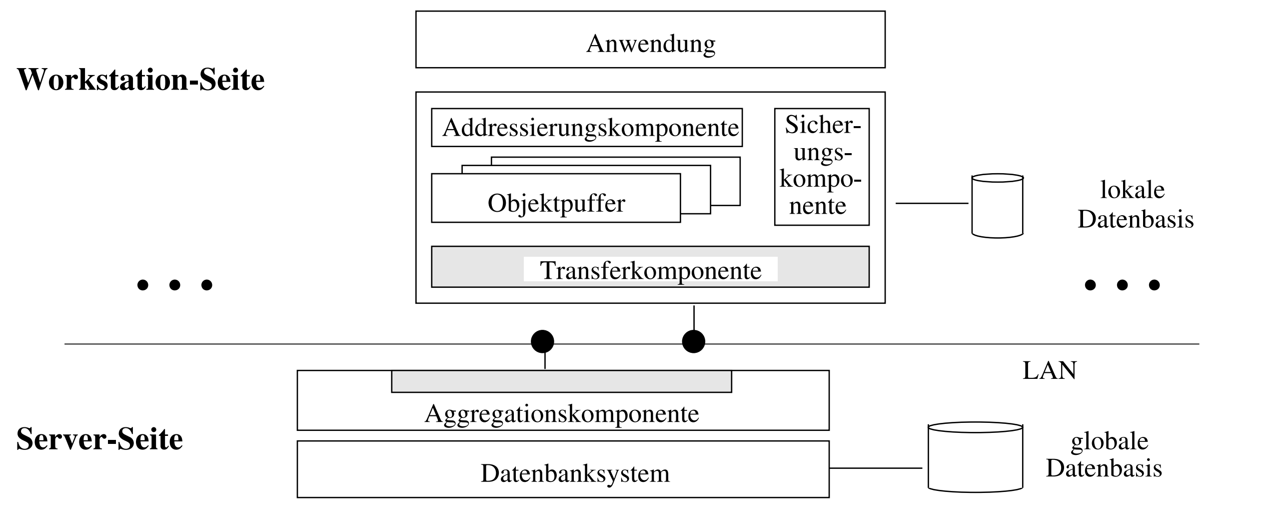


Bild 7: Grobarchitektur workstation-orientierter DB-gestützter Anwendungssysteme

wendung eine lokale Datenhaltungskomponente, die zum einen für die Kooperation mit dem Server, zum anderen aber auch für die Verwaltung und den Zugriff auf lokal bereitgestellte Daten verantwortlich ist. Die interne Struktur der lokalen Zugriffskomponente sowie die zentrale Rolle des Objektpuffers werden ebenfalls in Bild 7 verdeutlicht. Außer den Verarbeitungsgegenständen und entsprechenden Verwaltungsdaten enthält der Objektpuffer Cursor-Strukturen, die über die Addressierungskomponente den Anwendungsalgorithmen die Möglichkeit bieten, navigierend, pointer-ähnlich auf Objektpuffer-Daten zuzugreifen /HS89/ . Eine weitere Komponente erlaubt die lokale persistente Speicherung von Verarbeitungszuständen und realisiert die im vorangehenden Kapitel eingeführten Funktionen SAVE, RESTORE, SUSPEND und RESUME. Aufgrund der gewählten Objektpufferorganisation kann die persistente Speicherung relativ einfach und effizient durch Aus- bzw. Einlagern ganzer Speicherbereiche erreicht werden.

Bild 8 skizziert die in der Literatur oftmals vorgeschlagene Architektur für Nicht-Standard DBS und stellt sie der hier diskutierten Grobarchitektur workstation-orientierter, DB-gestützter Ingenieursysteme gegenüber. Hinter dem Architekturkonzept für NDBS verbirgt sich die Idee der Zweiteilung in einen anwendungsunabhängigen NDBS-Kern und in eine anwendungsorientierte Systemebene, Modellabbildung genannt. Die Vorteile dieses Ansatzes liegen vor allem darin, daß einerseits durch die Modellabbildung eine anwendungsbezogene Schnittstelle (Anwendungsmodell-Schnittstelle) mit den von der jeweiligen Anwendungsklasse benötigten Objekten und Operationen bereitgestellt werden kann und andererseits alle geeigneten, allgemein verwendbaren Darstellungs-und Zugriffstechniken sich im NDBS-Kern redundanzfrei und effizient implementieren lassen. Der Kern realisiert damit ein allgemeines, meist strukturell-objektorientiertes Datenmodell, auf dem die mit mehr Semantik ausgestatteten speziellen Modelle verschiedener Anwendungsklassen aufbauen. Aufgrund seiner Funktionalität übernimmt der NDBS-Kern die Rolle des DB-Servers. Die Modellabbildung und die eigentliche Ingenieuranwendung werden auf die Workstation-Seite verlagert, so daß das objektpuffer-basierte Verarbeitungskonzept in der lokalen Datenhaltung für die aufwendige Verarbeitung innerhalb der Modellabbildung genutzt werden kann.

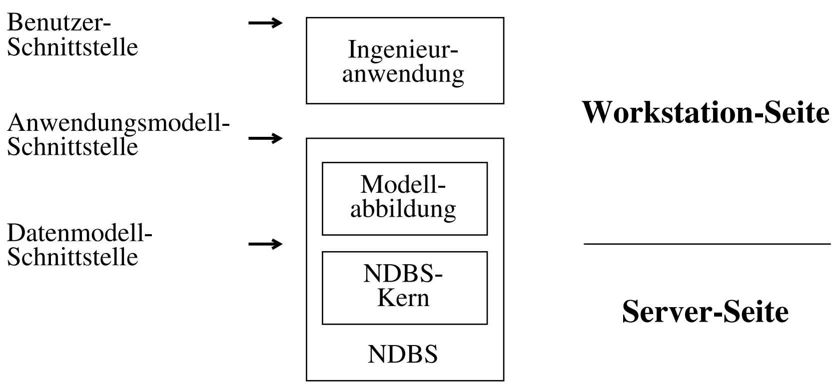


Bild 8: Systemarchitektur NDBS-basierter Ingenieuranwendungen

4.2 Informationsaustausch zwischen Server und Workstation

Der rechnerübergreifende DB-Zugriff wird durch eine spezielle Transferkomponente realisiert. Ihre Aufgabe besteht u.a. in der Abwicklung des notwendigen Datenaustausches. Die erforderlichen Informationen gehen über die der unmittelbaren Verarbeitungsobjekte hinaus. Neben den "Rohdaten" müssen Anweisungs- und Metainformationen transferiert werden. Die folgende Zusammenstellung gibt eine Charakterisierung der zu übertragenden Informationen:

(1) Anweisungen

Die einfachste Möglichkeit zur Übertragung von DB-Anweisungen besteht in dem Transfer von Zeichenketten. Auf der Seite des DB-Servers sind diese dann zu interpretieren bzw. zu übersetzen. Zu einer optimierten Transfer-Darstellung von DB-Anweisungen gelangt man durch eine syntaktische Überprüfung und Voranalyse auf der Workstation-Seite. Hierzu ist allerdings dort ein entsprechender Parser vorzusehen. Die semantische Analyse von DB-Anweisungen kann auf Workstation-Seite nur unter hohem Aufwand realisiert werden, da zunächst die betreffenden Metadaten vom DB-Server zu beschaffen wären. Um dennoch einen weiteren und wesentlichen Performance-Gewinn zu erzielen, halten wir eine möglichst vollständige Anweisungs-Vorübersetzung auf Seiten des DB-Servers für angebracht. Das Übersetzungsergebnis kann dann als Ausführungsmodul durch den DB-Server verwaltet werden. Eine längerfristige Speicherung der Ausführungsmodule ist, speziell in dem NDBS-Szenario, durchaus sinnvoll, da mit einer hohen Stabilität der einmal etablierten Modellabbildungskomponenten zu rechnen ist. Dies gilt insbesondere, wenn das zugrundeliegende Datenmodell eine Anweisungsparametrisierung erlaubt. Bei einer späteren Anweisungsaktivierung durch Programme der Modellabbildung muß dann lediglich eine Kennung des entsprechenden Ausführungsmoduls zusammen mit den aktuellen Parameterwerten übertragen werden.

(2) Daten

Die CHECKOUT/CHECKIN-Operationen, meist aber auch die DB-Anweisungen zur direkten Datenmodifikation (INSERT, DELETE, UPDATE), sehen eine "unmittelbare" Datenspezifikation vor, d.h., die Daten, die gelesen, eingefügt, geändert und gelöscht werden sollen, werden unmittelbar im Anschluß an die Ausführung einer entsprechenden Anweisung übergeben. Daher sind Antwort- (bzw. Einfüge-, Änderungs- und Löschinformationen) vom DB-Server zur Workstation (bzw. in umgekehrter Richtung) zu übertragen. Die Antwortinformation beschreibt die Menge derjenigen Datenelemente, die zur Bildung der Verarbeitungsobjekte, die sich bzgl. einer Anfrage qualifizieren konnten (CHECKOUT), erforderlich sind. Die "lokalen" Einfüge-, Änderungs- und Löschoperationen führen schrittweise zu einer Modifikation der Verarbeitungsobjekte. Die bei CHECKIN zu übertragende Information ist sinnvollerweise als "Delta-Information" organisiert, die lediglich die Differenz zwischen modifiziertem und ursprünglichem Verarbeitungsobjekt beschreibt.

(3) Metadaten

Zwischen DB-Server und Workstation werden äußerst komplexe Datenstrukturen ausgetauscht. Die Verarbeitung dieser Daten erfordert eine Absprache über deren konkrete Strukturierung. Diese ist i.allg. anweisungsspezifisch. Daher muß eine anweisungsbezogene Übertragung von Metainformationen erfolgen. Hierbei sind zwei Varianten möglich: zum einen können die Metainformationen so eng mit den Daten verbunden sein, daß sie einen integralen Bestandteil bilden. Die Daten sind dann "selbstbeschreibend". Zum anderen kann der Teil der Metadaten, der anweisungsspezifisch, aber ausprägungsunabhängig ist, als Typ-Information von den "eigentlichen" Daten separiert und eigenständig übertragen werden. Hierdurch kann der wiederholte Austausch gleicher Metadaten weitestgehend vermieden werden. Diese Typ-Information kann bereits nach Übersetzung einer DB-Anweisung durch den DB-Server erstellt und zur Workstation transferiert werden. Gelingt es nun, die Typ-Information direkt an das

Anwendungsprogramm zu binden, so müssen zur Laufzeit der Anwendung lediglich die ausprägungsabhängigen Metadaten (z.B. aktuelle Länge eines String-Wertes, einer Wiederholungsgruppe etc.) übermittelt werden.

5. Zusammenfassung

Arbeitsplatzrechner prägen die zukünftige Hardware-Umgebung in fast allen ingenieurwissenschaftlichen Anwendungsbereichen. Integrierte, datenintensive Anwendungen erfordern in einer solchen Umgebung neue und angepaßte Konzepte für eine effiziente Datenhaltung und -verarbeitung.

Es wurde ein Ansatz vorgeschlagen, der, auf einer logisch zentralen Datenhaltung beruhend, eine auf Workstations verlagerte Verarbeitung erlaubt. Anhand der zugrundeliegenden Verarbeitungsvorstellung und der Grobarchitektur eines workstation-orientierten DB-gestützten Anwendungssystems konnten die prinzipiellen Ideen angesprochen werden:

- Unterschiedliche Anforderungsprofile für die globale und lokale Datenhaltungskomponenten führten zu einer heterogenen DBS-Aufteilung in Workstation/Server-Umgebungen.
- Ein hybrides Verarbeitungsmodell erlaubt Direktänderungen auf den globalen Daten und die Bereitstellung und Verarbeitung der lokalen Daten durch ein zweistufiges Konzept. Die mit dem Verarbeitungsmodell verbundenen Probleme zur Synchronisation der Verarbeitung wurden aufgezeigt und ein geschachteltes Transaktionskonzept als Mechanismus zur Behebung der Problempunkte vorgeschlagen.

Das beschriebene Verarbeitungsmodell wird derzeit realisiert und setzt für die globale Datenhaltung das NDBS-Kernsystem PRIMA, ein strukturell-objektorientiertes DBS, ein /Hä88/. Weiterführende Arbeiten werden sich mit einer Verfeinerung des Transaktionsmodells beschäftigen, das neben der Zugriffskontrolle auch die autonome Fehlerbehandlung auf Workstation-Seite und damit ein lokales Sicherungskonzept umfassen soll.

Danksagung

Wir danken Herrn Prof. T. Härder sowie unserem Kollegen Herrn Dr. K. Meyer-Wegenen für die hilfreichen Anmerkungen während der Enstehungsphase dieser Arbeit.

6. Literatur

BEEK84 R. Bayer, K. Elhardt, W. Kießling, D. Killar: Verteilte Datenbanksysteme - Eine Übersicht über den heutigen Entwicklungsstand, in: Informatik-Spektrum, Bd. 7, 1984, S. 1 - 19.

Da86 P. Dadam, et al.: A DBMS Prototype to Support Extended NF2-Relations: An Integrated View on Flat Tables and Hierarchies; in Proc. ACM SIGMOD Conf. on Management of Data, pp. 376 - 387, 1986.

Di86 K.R. Dittrich: Object-Oriented Database Systems: The Notion and the Issues; in Proc. Int. Workshop on Object-Oriented Database Systems, pp. 2 - 6, Pacific Grove, Ca., 1986.

Di88 Rehm, S., Raupp, T., Ranft, M., Längle, R., Härtig, M., Gotthard, W., Dittrich, K. R., Abramowicz, K.: Support for Design Processes in a Structurally Object-Oriented Database System; in Proc. 2nd int. Workshop on Object-Oriented Database Systems, 1988.

DGKOW86 U. Deppich, J. Günauer, K. Küspert, V. Obermeit, G. Walch : Überlegungen zur Datenbank-Kooperation zwischen Server und Workstation, Proc. der 16. GI-Jahrestagung, Informatik-Fachberichte126, Springer Verlag, S. 564 - 580, 1986.

Hä88 T. Härder (ed.): The PRIMA-Project - Design and Implementation of a Non-Standard Database System, Forschungsbericht 26/88 des SFB 124, Universität Kaiserslautern, 1988.

HHMM88 T.Härder, Ch. Hübel, K. Meyer-Wegener, B. Mitschang: Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server; in: Data and Knowledge Engineering, 3(1988), pp.87-107, 1988.

HP89 Ch. Hübel, M. Pick: Anwendungsnahe Pufferung komplex-strukturierter DB-Objekte, in Proc. der GI-Fachtagung Datenbanken in Büro, Technik und Wissenschaften (BTW), Zürich, Informatik-Fachberichte, Springer Verlag, S. 355 - 360, 1989.

HPS89 Hübel, Ch., Paul, R., Sutter, B.: Technische Modellierung und DB-gestützte Datenhaltung - ein Ansatz für ein durchgängiges integriertes Produktmodell, Bericht des Zentrums für Rechnergestützte Ingenieursysteme (ZRI), Nr. 6/89, Kaiserslautern, 1989.

HS89 Hübel, Ch., Sutter, B.: Verarbeitung komplexer DB-Objekte in Ingenieuranwendungen, Bericht des Zentrums für Rechnergestützte Ingenieursysteme (ZRI), Nr. 5/89, Kaiserslautern, 1989.

JWZ87 Jablonski, S., Wedekind, H., Zörntlein, G.: Data Distribution in Manufacturing Systems, Bericht Nr. 87/1 des SFB 182, Erlangen, 1987.

KWDKL89 A. Kemper, M. Wallrath, M. Dürr, K.Küspert, V. Linnemann: An Object Cache Interface for Complex Object Engineering Databases, Forschungsbericht TR 89.03.005, Wiss. Zentrum Heidelberg der IBM Deutschland, Heidelberg, 1989.

Mo81 Moss, J. E. B.: Nested Transactions: An Approach to Reliable Distributed Computing, PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, April 1981.

PSSWD87 H.B. Paul, H.J. Schek, M. H. Scholl, G. Weikum, U. Deppisch: Architecture and Implementation of the Darmstadt Kernel System, in Proc. SIGMOD87, San Fransisco, 1987.

Re87 A. Reuter et al.: Anforderungen an ein arbeitsplatzorientiertes Datenhaltungssystem, Proc. der GI-Fachtagung Datenbanken in Büro, Technik und Wissenschaften (BTW), Darmstadt, Informatik-Fachberichte 136, Springer Verlag, S. 391 - 404, 1987.