Workstation/Server-Architekturen für datenbankbasierte Ingenieuranwendungen

T. Härder, B. Mitschang, U. Nink, N. Ritter

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
e-mail: { haerder / mitsch / nink / ritter }@informatik.uni-kl.de

Zusammenfassung

Werkzeugorientierte Verarbeitung in datenbankbasierten Ingenieursystemen geschieht typischerweise in einer Workstation/Server-Umgebung. Dabei sind die werkzeugrelevanten Entwurfsdaten für die Dauer der Verarbeitung aus Effizienzgründen (Referenzlokalität) im Hauptspeicher der Workstation zu puffern. Die Rolle der langfristigen Datenspeicherung sowie der Datenversorgung der beteiligten Workstations übernimmt der Datenbank-Server.

In diesem Artikel werden verschiedene Workstation/Server-Architekturen vorgestellt und hinsichtlich ihrer Tauglichkeit für datenbankbasierte Ingenieuranwendungen bewertet. Im Brennpunkt dieser Betrachtungen stehen Datenextraktion, Datenbereitstellung und -verarbeitung sowie Integration der geänderten Daten, Datensicherung und Synchronisation. Das Verarbeitungsmodell haben wir detailliert und dabei wichtige Merkmale herausgearbeitet sowie die Grundformen bekannter Architekturen verfeinert. Zudem wird eine Entscheidungshilfe zur Auswahl der geeigneten Systemarchitektur für eine gegebene Anwendung erarbeitet. Unsere Untersuchungen zeigen, daß keine global optimale Architektur existiert und daß für die Bewertung eines Ansatzes die spezifischen Charakteristika der Zielanwendung entscheidend sind. Hier ist insbesondere ausschlaggebend, ob deskriptiv formulierte Datenanforderungen und vorab bekannte Verarbeitungskontexte der Anwendung effizient und auf ihre Bedürfnisse zugeschnitten verfügbar gemacht werden können.

Schlüsselwörter: Workstation/Server, Datenbanken, Objektorientierung, Ingenieuranwendungen, Verarbeitungskonzept

Abstract

Engineering and design systems typically employ their engineering and design tools in a workstation/server environment. Usually it is the task of the database server to manage the persistent design data, whereas the tools run at the workstation site. Due to efficiency considerations it is overly important that the application's current design data are cached in the main memory at the workstation, thereby exploiting near-by-application locality.

In this paper, we introduce some important workstation/server architectures and we discuss their suitability for engineering applications that build upon database systems. The discussion is focussed on data extraction from the server, data representation and processing at the workstation, integration of committed design data into the global database as well as on synchroni-

veröffentlicht in Informatik Forschung und Entwicklung (1995) 10: 55-72, Springer

zation and recovery issues. Furthermore, the processing model is being detailled whilst refining the basic architectures. Our investigations reveal that there is no single optimal architecture; in contrast, the specific characteristics of the application at hand determine the suitability of the various architectures. Therefore, we give some decision guidelines on how to choose the best system architecture for a given application scenario.

Key words: Workstation/server, databases, object orientation, engineering applications, processing model

CR Subject Classification: D.4.4, E.m, H.2.4, H.2.8, J.6

1. Einleitung

Der Einsatz *integrierter rechnergestützter Entwurfsumgebungen* (engl. "integrated computer-aided design environments") stellt einen wesentlichen Schritt zur Beherrschung der Komplexität des Entwurfs (etwa VLSI-Entwurf oder CAD in den Bereichen Maschinenbau, Architektur und Bau- und Raumwesen) dar. Diese Entwurfsumgebungen bieten für einen konkreten Anwendungsbereich eine zugeschnittene Infrastruktur für die Anwendung entsprechender CAD-Werkzeuge und unterstützen alle Phasen des Entwurfsprozesses in lückenloser und kontinuierlicher Weise. Eine Basis zur Erstellung solch angepaßter Umgebungen bilden die sog. *CAD-Frameworks* [14, 32], die eine möglichst allgemeine und damit wiederverwendbare Betriebsumgebung für CAD-Werkzeuge bereitstellen. In diesem Sinne ist es eine zentrale Aufgabe eines CAD-Frameworks, die folgenden Basisdienste anzubieten:

- Langfristige Verwaltung aller *entwurfs- und werkzeugrelevanten* Daten, sowie Maßnahmen zur Erhaltung der semantischen Integrität dieser Daten.
- Integration von Werkzeugen in die Entwurfsumgebung unter besonderer Berücksichtigung der vorherrschenden Objekt- und Entwurfsstrukturen.
- Datenversorgung der Werkzeuge und Bereitstellung von operationalen Schnittstellen für die werkzeugorientierte Verarbeitung.
- Kontrolle von Ablauffolgen (Transaktionsdienste [13], workflow management [18]) bei der werkzeugorientierten Verarbeitung und Mechanismen zur Fehlerbehandlung.

Als wesentliche Systemarchitektur hat sich hier die Workstation/Server-Architektur herausgebildet. Hierbei finden die einzelnen Werkzeuganwendungen auf den Workstations statt, und die zentrale Datenverwaltung geschieht auf dem Server. Zur Optimierung der Workstation/Server-Kommunikation sowie zur Ausnutzung der Verarbeitungslokalität der Werkzeuge ist der Einsatz eines Hauptspeicherpuffers auf der Workstation-Seite (“Objektpuffer”) notwendig [17]. Diese Schlüsseleigenschaft soll durch die explizite Verwendung des Begriffs *Workstation* hervorgehoben werden; dadurch unterscheiden sich diese Architekturen erheblich von solchen, die sich durch die bloße Funktionalität des Client/Server-Ansatzes charakterisieren lassen.

Ausgehend von dieser Überlegung kristallisiert sich ein Zyklus der Werkzeugverarbeitung (Abb. 1) heraus, für den folgende Aspekte von besonderer Relevanz sind:

(1) *Extraktion der werkzeugrelevanten Daten (Check-out)*
Die für den Verarbeitungsschritt notwendigen Daten sind aus den auf dem Server verwalteten Entwurfsdaten zu selektieren und mit minimalem Kommunikationsaufwand zur Workstation (WS) zu transportieren. Wir bezeichnen diese Menge der für die Werkzeuganwendung relevanten Daten als *Verarbeitungskontext*.

(2) *Bereitstellung/Repräsentation der Daten im Objektpuffer*
Die gelieferten Daten sind dem Werkzeug in geeigneten Datenstrukturen zur Repräsentation der komplexen Entwurfsobjekte und ihrer verschiedenartigen inter- und intrastrukturellen Beziehungen zur Verfügung zu stellen, die idealerweise dem Zugriffsverhalten und den Verarbeitungscharakteristika des Werkzeuges flexibel angepaßt werden können.

(3) *Unterstützung der Werkzeug-Verarbeitung*
Bei den Operationsfolgen (Suche, Änderung) der Werkzeuganwendungen dominiert eine navigierende Verarbeitungsweise, die durch ein flexibles Cursorkonzept gesteuert wird. Zusätzlich kann es sinnvoll sein, inhaltsorientierte Suchvorgänge zu unterstützen. Beispielsweise kann dies durch eine deskriptive Anfragemöglichkeit und mengenorientierte Verarbeitungsweise effektiv unterstützt werden. Bereits während dieser Objektpuffer-Verarbeitung ist es sinnvoll, die Einhaltung bestimmter Integritätsbedingungen zu überwachen.

(4) *Integration der geänderten Daten (Check-in)*
Am Ende eines Verarbeitungsschrittes ist das Ergebnis der Werkzeuganwendung in die zentrale Datenbank zurückzuschreiben, was wiederum mit minimalem Kommunikationsaufwand geschehen sollte. Nach erfolgreicher Integration der geänderten Daten muß Schemakonsistenz gewährleistet sein. Dies bedeutet insbesondere, daß am Ende des Verarbeitungsschrittes weitere Integritätsbedingungen zu überprüfen sind.

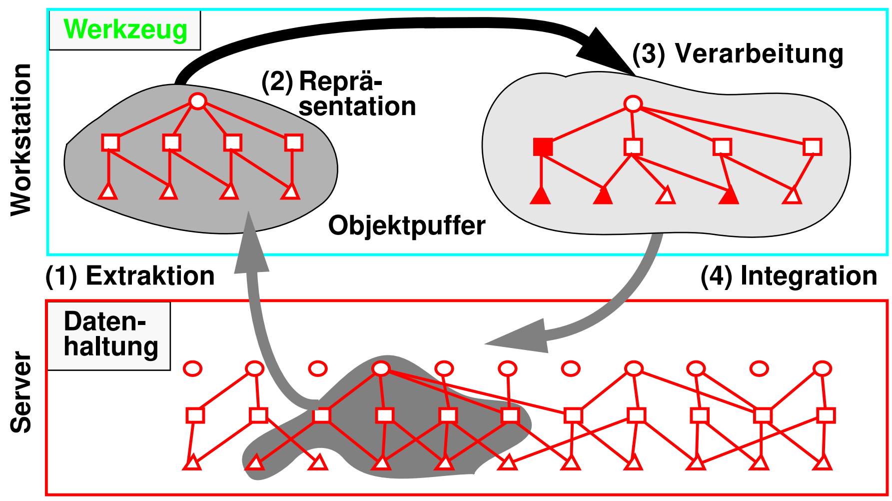

Abb. 1. Werkzeugbasiertes Verarbeitungsmodell

Mit dieser Skizze des Verarbeitungszyklus einer Werkzeuganwendung sind bereits wesentliche Anforderungen an ein unterstützendes Architekturkonzept angesprochen. Aus Sicht des Werkzeugs hat das Leistungsverhalten oberste Priorität, da es oft entscheidend für die Akzeptanz des gesamten Verarbeitungskonzepts ist. Typischerweise sind die innerhalb eines Verarbeitungskontextes zu bearbeitenden Datenmengen sehr groß und hochgradig vernetzt. Bei komplexen Berechnungen des Werkzeuges, die viele Datenelemente miteinander in Beziehung setzen, fallen deshalb enorm viele Datenzugriffe (navigierender Art) an. Aus diesem Grund ist es eine oft genannte Forderung, bei solchen Operationsfolgen etwa in der Größenordnung von Hauptspeicher-Zugriffsoperationen zu liegen (was der Möglichkeit, etwa 10^5 Objektreferenzen pro Sekunde auswerten zu können, entspricht). Um diese Zielvorgabe erreichen zu können, erscheint es notwendig, die Datenrepräsentation und auch die Datenstrukturen im Objektpuffer an das Zugriffsverhalten des Werkzeuges anzupassen und zudem effiziente Zugriffsfunktionen mit variablen Verarbeitungsgranulaten (Attribut, Objekt, Menge von Objekten) und einer einheitlichen Schnittstelle (Datenunabhängigkeit) bereitzustellen.

Neben der Vorstellung der Grundformen von Workstation/Server-Architekturen liegt die Hauptintention dieses Artikels in der Diskussion möglicher Optimierungen und in der Bewertung der Eignung dieser Ansätze für bestimmte Anwendungsfälle. Daher wendet sich dieser Artikel sowohl an den DBMS-Implementierer als auch an den Anwendungsentwickler. Während sich die Diskussion der vorgeschlagenen Architekturerweiterungen eher an den Implementierer richtet, wird den Interessen des Anwendungsentwicklers insbesondere durch bestimmte Abstraktionen und die getrennte Diskussion der Architekturgrundformen und der zugehörigen Optimierungen Rechnung getragen. Für den an weiteren Details interessierten Leser werden weiterführende Spezialliteraturreferenzen gegeben. Damit soll insbesondere eine leicht zugängliche Einführung in die Problematik unserer Themas gegeben werden, die die sehr umfangreiche und z.T. sehr undurchsichtige Spezial- und Orginalliteratur verständlich aufbereitet.

Zunächst betrachten wir in Kapitel 2 ein typisches Verarbeitungsszenario. Dort wird am Beispiel des VLSI-Entwurfs ein Mengengerüst für einen typischen Verarbeitungskontext im Chip-Planning vorgestellt. Die Kapitel 3 und 4 bilden den Hauptteil dieser Arbeit. Während hier aus Anwendersicht eher die Beschreibung der einzelnen Architekturen (Abschnitt 3.2) und deren anwendungsbezogene Bewertung (Abschnitt 4.2) von Interesse sind, richten sich die Diskussion der vorgestellten Erweiterungen (Abschnitt 3.3) und die aufgestellten Grundprinzipien für den Systementwurf (Abschnitt 4.1) in erster Linie an den Implementierer. Ein kurzes Resümee und ein Ausblick auf weitere Arbeiten schließen die Betrachtungen ab.

2. Charakteristika von Verarbeitungskontexten

Ziel dieses Kapitels ist es, genauere Informationen über den Aufbau und die Größe von Verarbeitungskontexten bereitzustellen. Beispielhaft werden für den VLSI-Entwurf die dort vorherrschenden/typischen Verarbeitungskontexte bestimmt. Die in diesem Kapitel erarbeiteten Mengengerüste und allgemeinen Charakteristika von Verarbeitungskontexten werden in den nachfolgenden Kapiteln zur System- bzw. Architekturbewertung herangezogen.

2.1 Anwendungsszenario

Chip-Planning. Der VLSI-Entwurf ist ein komplexer Anwendungsbereich, in dem eine Vielzahl von Entwurfswerkzeugen zum Einsatz kommen. Wir wollen an dieser Stelle jedoch nicht auf die Methodik des Gesamtentwurfs eingehen [37], vielmehr sollen anhand eines Entwurfswerkzeugs die Charakteristika von Werkzeuganwendungen illustriert und die typischen Objektstrukturen aufgezeigt werden. Durch das in [1] beschriebene Werkzeug *Chip-Planner* wird die Topographie einer Zelle bestimmt. Da typischerweise die Anzahl der Standardzellen für eine zu entwerfende Zelle sehr groß ist (~ 10^6 Standardzellen), entwirft man *hierarchisch* aufgebaute Zellen, die jeweils aus einer begrenzten Anzahl von Subzellen bestehen. Der Chip-Planner betrachtet in einer konkreten Anwendung eine Zelle dieser Hierarchie zusammen mit ihren direkten Subzellen. Die Eingabe besteht aus der Schnittstellenbeschreibung der zu planenden Zelle (Rahmen und Anschlußbereiche/Pin-Intervalle) sowie Informationen über die Subzellen (mögliche Formen und Vernetzungen). Davon ausgehend werden Floorplan-Informationen berechnet, d. h. insbesondere die Anordnung der Subzellen innerhalb der Zelle und die Schnittstellen-Beschreibungen der Subzellen. Letztere dienen als Eingabe für nachfolgende Chip-Planner-Anwendungen zur Berechnung der Subzellen-Topographien. In diesem Sinne wird die Gesamt-Topographie des zu entwerfenden Chips durch sukzessive Chip-Planner-Anwendungen in einer Top-Down-Vorgehensweise ermittelt.

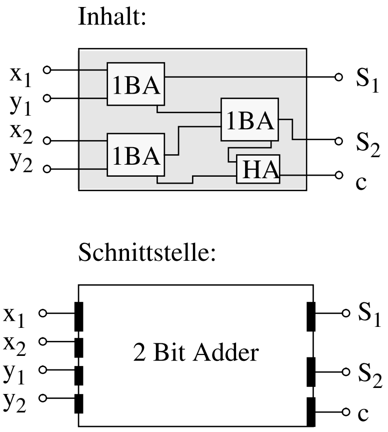

Abb. 2. Zellhierarchie eines 2-Bit-Addierers

Abb. 2 visualisiert die Zellhierarchie eines 2-Bit-Addierers. Im unteren Teil des Bildes ist ausschließlich die Zellschnittstelle dargestellt, während im oberen Teil auch der Zellaufbau zu sehen ist. Die Hierarchisierung ist dabei gut zu erkennen: Die Beschreibung der Zelle des 2-Bit-Addierers setzt sich direkt in der Beschreibung der Subzellen (drei 1-Bit-Addierern und einem Halbaddierer) fort. Weiter wird dort auch die Anordnung der Subzellen und der Verbindungsleitungen angedeutet.

Datenmodellierung. Eine ausführliche Betrachtung der Informations- und Datenmodellierung für das angesprochene Anwendungsbeispiel ist im Rahmen dieses Artikels nicht angebracht [22]. Trotzdem sei an dieser Stelle darauf hingewiesen, daß bspw. Zellschnittstellen, Zellen und Zellinhalte sinnvollerweise als *komplexe Objekte* zu modellieren sind, die sich aus elementaren Objekten, wie z.B. Verdrahtungen, Anschluß-Pins etc., zusammensetzen. Um die fortschreitende Entwicklung und Vervollständigung der Entwurfsobjekte zu reflektieren, wird eine *Versionierung* der komplexen Objekte durchgeführt. Da jetzt für komplexe Objekte mehrere Versionen auftreten können, muß festgelegt werden, welche Versionen zu einer größeren Verarbeitungseinheit zusammenzufassen sind. Dieser Auswahlvorgang heißt *Konfigurierung*[1]. Damit wird ein Verarbeitungskontext aus Versionen von zueinander in Beziehungen stehenden komplexen Objekten durch Konfigurierung gewonnen. Das Konfigura-

1) Im mechanischen CAD entspricht der Konfigurierung die Baugruppenerstellung, wobei Versionen von Bauteilen ausgewählt werden. Verschiedene Konfigurationen führen hier zu verschiedenen Baugruppen-Varianten.

tionsobjekt dient dabei als eine Art Anker für den gesamten Verarbeitungskontext. Von diesem Anker aus können alle werkzeugrelevanten Daten durch Traversieren von Objektbeziehungen erreicht werden.

2.2 Eigenschaften von Verarbeitungskontexten

Allgemeine Eigenschaften. Die Verarbeitungskontexte verschiedener Werkzeuganwendungen können natürlich überlappen; man spricht dann von gemeinsamen Teilobjekten ("object sharing"). Weiterhin können auch rekursive Objektstrukturen auftreten (vgl. rekursive Zellhierarchie des Beispiels). Der Chip-Planner benötigt als Eingabe nicht die vollständigen Objekte, sondern es genügt eine *Sicht* auf diese. Dabei ist sowohl einfache Projektion einzelner Attribute als auch das Verdichten von Information (Aggregation) sowie eine Komponentenprojektion innerhalb der Komplexobjekte vorzusehen.

Quantitative Aspekte von Verarbeitungskontexten. Im folgenden wird eine grobe, quantitative Abschätzung typischer Verarbeitungskontexte unserer Beispielwelt gegeben. In einer typischen Zell-Hierarchie nach [37] besteht der zu entwerfende Chip aus ca. 50 -100 Modulen, die sich selbst jeweils aus ebenfalls ca. 50 - 100 Blöcken zusammensetzen, die wiederum jeweils in ca. 1000 Standardzellen zerlegt werden können. Tab. 1 beinhaltet charakteristische Schätzgrößen für die Eingabedaten einer Chip-Planner-Anwendung. Dabei unterscheiden wir einen Hierarchieübergang mit ca. 60 Subzellen (Verarbeitungskontext VK 1) und einen mit ca. 1000 Standardzellen (Verarbeitungskontext VK 2). Für diese beiden Situationen wurden die folgenden Größen berechnet:

- #CO: Anzahl der zu lesenden Komplexobjekte
- #EO: Anzahl zugehöriger Elementar-Objekte
- DVol: Datenvolumen bei ausschließlicher Betrachtung relevanter Daten
- #P(a): Anzahl belegter DB-Seiten bei maximaler Clusterbildung, d. h. bei Ablage des gesamten Datenvolumens DVol in aufeinanderfolgende Seiten.
- #P(b): Anzahl belegter DB-Seiten bei fehlender Clusterbildung; in diesem Fall liegt jedes Elementarobjekt auf einer eigenen Seite.
- #P(c): Anzahl belegter DB-Seiten bei 25% Nutzdaten pro Seite.

Die Anzahl der Komplexobjekte (#CO in Tab. 1) setzt sich im wesentlichen zusammen aus den 60 (1000) Komplexobjekten vom Typ Zelle, den zugehörigen Schnittstellen-Objekten (im Schnitt besitzen je vier Zellen die gleiche Schnittstelle) und den Objekten zur Beschreibung der Superzelle und der Konfiguration. Die jeweiligen Anzahlen von Elementarobjekten (#EO) innerhalb eines Komplexobjektes wurden aufgrund von Erfahrungswerten und abhängig vom Typ des Verarbeitungskontextes abgeschätzt. Unsere Verarbeitungskontexte bestehen aus 20 Elementarobjekttypen, die im Mittel 10 Attribute von durchschnittlich 10 Byte aufweisen. Daraus ergeben sich eine Elementarobjektgröße von 100 Byte und die in Tab. 1 enthaltenen Datenvolumina (DVol). Für die Ermittlung der Seitenbelegungen wurde eine Seitengröße von 4 KB angenommen.

Man erkennt aus Tab. 1 sehr deutlich, daß für die Ein-/Ausgabe auf dem Server und die Übertragung eines Verarbeitungskontextes zwischen Server und Workstation die Clusterbildung ein sehr entscheidender Leistungsfaktor ist. So sind in jedem Fall wesentliche Effizienzeinbußen zu erwarten, wenn für die Planung einer Zelle mit 60 Subzellen eine Datenmenge von 20 MB zwischen Server und Workstation transferiert werden muß und die reinen Nutzdaten nur 1/40 davon (500KB) betragen.

In der Realität wird die Clusterbildung von Verarbeitungskontexten durch die zeitliche Trennung von Erzeugung der Objektversionen und Konfigurierung der Verarbeitungskontexte erschwert. Wird erst unmittelbar vor der Werkzeuganwendung konfiguriert (dies ist die Regel), so wird ein kostspieliges Umspeichern der Objekte zur Erreichung optimaler Clusterbildung notwendig. In ähnlicher Weise steht "object sharing" der Clusterbildung entgegen. Überlappende Verarbeitungskontexte können nur unter Einführung von Redundanz vollständig zusammenhängend gespeichert werden.

	#CO	#EO	DVol	#P(a)	#P(b)	#P(c)
VK 1	80	5000	500 KB	125 (500 KB)	5000 (20 MB)	500 (2 MB)
VK 2	1250	20000	2MB	500 (2 MB)	20000 (80 MB)	2000 (8 MB)

Tab. 1. Quantitative Beschreibung der Verarbeitungskontexte

Zusätzlich zum Datenvolumen macht Tab. 1 auch deutlich, daß typische Verarbeitungskontexte aus sehr vielen einzelnen Objektinstanzen (und auch Objektreferenzen bzw. Objektbeziehungen) aufgebaut sind. Damit werden auch Anforderungen an die Repräsentation eines Verarbeitungskontextes im Objektpuffer der Workstation gestellt. Diese Größenordnungen legen nahe, daß im Objektpuffer entsprechende Maßnahmen für die effiziente Navigation und auch zugeschnittene Zugriffspfade für wertabhängige Suche vorzusehen sind.

Abgeschlossener vs. offener Verarbeitungskontext. Entwurfsdaten von CA*-Anwendungen werden in der Regel in versionierter Form verwaltet. Es gibt jedoch auch komplexe Anwendungen (z. B. wissensbasierte Systeme), die auf eine Versionierung verzichten und ihre Datenänderungen direkt einbringen. Der Begriff des Verarbeitungskontextes wird deshalb dahingehend erweitert, daß seine Komponenten entweder über eine Konfigurierung oder direkt bestimmt werden können.

Viele Anwendungen aus Entwurfsbereichen zeichnen sich dadurch aus, daß ein Verarbeitungskontext bereits vor der Werkzeuganwendung inhaltlich fixiert (bzw. entsprechend zusammengestellt und konfiguriert) werden kann. Damit kann der Verarbeitungskontext zu Beginn des Verarbeitungsschrittes angefordert und im Objektpuffer installiert werden, und ein Nachfordern bzw. Nachladen von zusätzlich benötigten Daten ist unnötig. Wir bezeichnen solche Verarbeitungskontexte als *abgeschlossen*.

Die Abgeschlossenheit der Verarbeitungskontexte ist jedoch keine generelle Eigenschaft *aller* hier diskutierter Anwendungsbereiche. Daher ist es wichtig, auch *offene* Verarbeitungskontexte zu berücksichtigen. Diese sind dadurch gekennzeichnet, daß sie nicht vollständig zu Beginn der Werkzeuganwendung spezifiziert werden können und ein Nachladen von Daten während des Werkzeuglaufs (oder der Entwurfsarbeit) notwendig werden kann.

3. Workstation/Server-Architekturen

Werkzeugorientierte Verarbeitung in Ingenieursystemen geschieht typischerweise in einer Workstation, wobei die Entwurfsdaten wegen der Nutzung der Referenzlokalität und der graphischen Ein-/Ausgabe in der "Nähe der Anwendung" gehalten werden. Ein Server übernimmt dabei die Rolle der langfristigen Datenspeicherung und der Datenversorgung für die beteiligten Workstations. Je mehr Aufgaben bei dieser Rollenverteilung in die Workstation verlagert werden können, umso mehr Workstations lassen sich von einem Server effizient unterstützen. Da große Mengen an Daten zu übertragen sind, ist auch die Organisation der Kommunikation von großer Wichtigkeit. Einerseits gilt es hier, das Verbindungsnetz zu entlasten, und andererseits soll durch gebündelte Datenübertragung und zusätzliche Sicherungsmaßnahmen (siehe transaktionsgestützter RPC in Encina [13]) ein möglichst großes Maß an Fehlerisolation erzielt werden. Diese Isolation, die durch eine (weitgehende) gegenseitige Maskierung der Fehler in Workstation und Server erreicht werden kann, ist vor allem deshalb wichtig, weil Entwurfstransaktionen sehr lange dauern und interaktive Entwurfsarbeit nach Möglichkeit nicht unterbrochen bzw. zurückgesetzt werden sollte.

3.1 Charakteristika der verschiedenen Architekturtypen

Workstation/Server-Architekturen lassen sich danach unterscheiden, wie die Datenversorgung und damit maßgeblich die Aufgabenverteilung organisiert ist. Die drei wichtigsten Grundformen (Page-Server, Object-Server und Query-Server) sind in Abb. 3 veranschaulicht. Oft wird zusätzlich der File-Server-Ansatz (z. B. mit NFS) als eigenständiger Architekturtyp diskutiert. Dieser läßt sich als spezieller Page-Server auffassen, der NFS als Transportmechanismus für die zu übertragenden Seiten benutzt. Er weist allerdings ein schlechteres Leistungsverhalten auf [11]. Referenzsysteme sind die File-Server von Objectivity [28] und Gemstone [3].

Wie Abb. 3 zeigt, stellt jeder Architekturansatz durch den Objekt-Manager den verschiedenen Anwendungen die gleiche navigierende und objektorientierte Schnittstelle zur Verfüung, und auch jeder Server besitzt als minimale Funktionalität (siehe grau unterlegte Komponenten) ein Speichersystem zur Verwaltung von Dateien auf Externspeicher (DB) und einen Seitenpuffer zur Minimierung der physischen Ein/Ausgabe auf die DB. Weiterhin muß der Server als zentrale Komponente die Synchronisation der Workstation-Anforderungen übernehmen, was wegen der langen Entwurfstransaktionen mit Hilfe von persistenten Datenstrukturen (persistente Sperren) zu geschehen hat. Vorsorgemaßnahmen für DB-Fehler und Server-Crash (Logging) und die Behandlung dieser Situationen (Recovery) gehören ebenfalls zu den zentralen Aufgaben. In Anlehnung an die Originalliteratur [11] erhalten die einzelnen Architekturtypen ihren Namen vom Granulat der Datenanforderung durch die Workstation.

Beim *Page-Server* wird jede Seitenanforderung workstation-seitig durch eine Fehlseitenbedingung (Page-Fault) ausgelöst; daraufhin wird die betreffende Seite vom Server zur Verfügung gestellt. Der Server hat dabei keine Kenntnis des Objektbegriffs: er verwaltet nur Seiten, die (normalerweise) auch das Granulat der Synchronisation und des Logging darstellen. Die Pufferung der Objekte in der Workstation kann nur über die Pufferung der zugehörigen Seiten geschehen, in denen auch die DB-bezogene Verarbeitung (Objekt-Manager) stattfindet. Alle objektbezogenen Zugriffe und Auswertungen erfolgen demnach in der Workstation. Der Anwendung wird im wesentlichen eine navigierende, objektorientierte Schnittstelle durch den Objekt-Manager geboten.

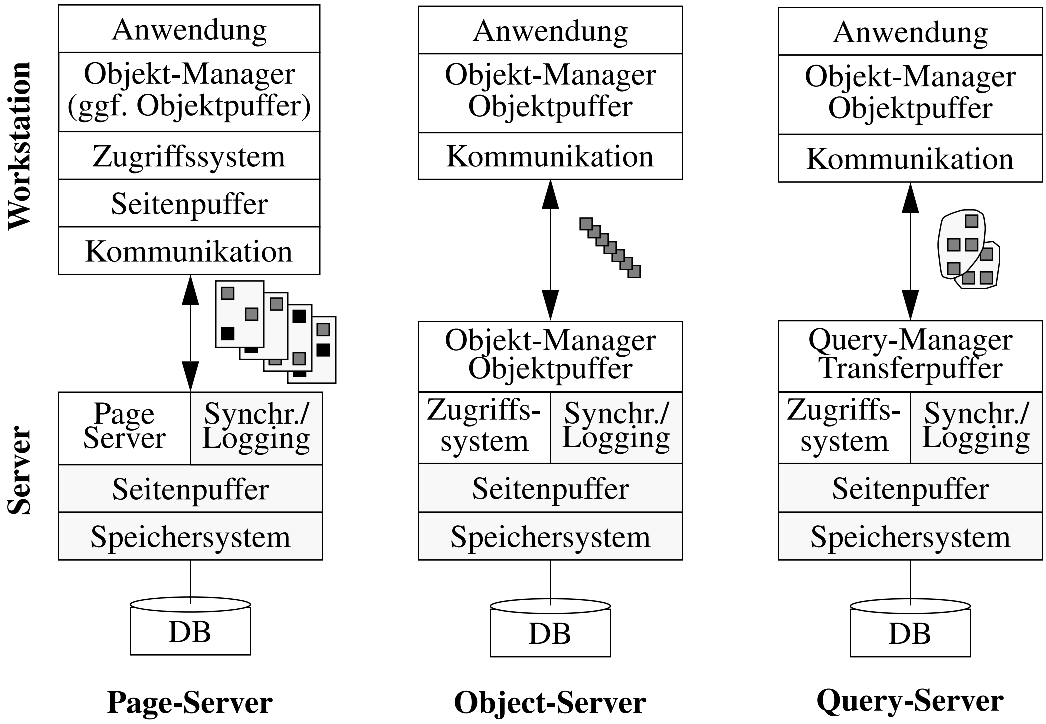

Abb. 3. Workstation/Server-Architekturen

Ein *Object-Server* liefert in der Grundform auf Anforderung (über eine OID) ein einzelnes Objekt. Erweiterungen dieses Ansatzes erlauben eingeschränkte Anfragemöglichkeiten über inhaltsorientierte Suche mit Hilfe einfacher Suchausdrücke (SSA: simple search arguments). Auch bei diesen erweiterten Anforderungsmöglichkeiten wird ein objektweiser Transport zur Workstation unterstellt. Die übertragenen Objekte können in den Objektpuffer so eingelagert werden, daß ihre Speicherungs- und Zugriffsstrukturen an die Bedürfnisse der navigierenden Anwendung angepaßt sind. Workstation wie auch Server kennen "Objekte", besitzen die entsprechende Verarbeitungsfunktionalität und sind in der Lage, objektbezogene Manipulationen und Auswertungen (Selektion, Projektion, Methoden) durchzuführen. Im Gegensatz zum Page-Server kann hier (wie auch beim nachfolgend beschriebenen Query-Server) das Objekt sowohl als Synchronisations- als auch als Logginggranulat herangezogen werden.

Query-Server beliefern die Anwendung auf eine Anfrage hin mit einer Menge von (komplexen) Objekten, die auf einmal übertragen und auf die Verarbeitungsanforderungen zugeschnitten im Objektpuffer gespeichert wird. Voraussetzung ist eine mächtige Anfrageschnittstelle, die der Objekt-Manager der Anwendung verfügbar macht. Die Verarbeitung im Objektpuffer geschieht dann über eine navigierende Schnittstelle.

Im folgenden werden einige wichtige Merkmale und grundlegende Prinzipien hinsichtlich Datenrepräsentation und Verarbeitungskonzept angesprochen, die wesentlich zum allgemeinen Verständnis von Workstation/Server-Architekturen beitragen.

Speicherungsmodelle. Hinsichtlich Datenrepräsentation läßt sich das *Server-Speicherungsmodell* von dem *WS-Speicherungsmodell* unterscheiden. Wird das Speicherungsmodell des Servers auch als Speicherungsmodell workstation-seitig übernommen, so spricht man von einem *Server-Speicherungsmodell* (homogenes Speicherungsmodell). Dies bedeutet, daß die Objektstrukturen und Zugriffspfade im wesentlichen unverändert bleiben, d.h. weder Konversion der Datentypen noch Attributprojektion möglich sind. Lediglich eine Anpassung der Objektreferenzen kann aus Effizienzgründen durchgeführt werden. Im Gegensatz dazu geht der Ansatz des *WS-Speicherungsmodells* von zwei unterschiedlichen Datenrepräsentationen aus. Dies äußert sich auch dadurch, daß nun zu dem Seitenpuffer im Server noch ein Objektpuffer in der Workstation hinzukommt. Die Datenrepräsentation im Objektpuffer ermöglicht dabei eine Konversion der Datentypen zusammen mit Attributprojektion und Anpassung der Objektreferenzen. Zur besseren Unterstützung der Verarbeitung ist es oft erforderlich, temporäre Zugriffspfade, die auch Collections genannt werden, für homogene Objektmengen anzulegen. Diese Collections können als Listen, Bäume, Hash-Strukturen u.a. ausgeprägt sein und dadurch an das Zugriffsverhalten der Anwendung angepaßt werden. Solche Collections werden vor allem dazu eingesetzt, spezielle Referenzbeziehungen zwischen Objekten zu verkörpern und effizient auszuwerten. Insgesamt läßt sich mit diesem Konzept eine Anpassung von Speicherungs- und Zugriffspfadstrukturen an die Erfordernisse der Anwendungsschnittstelle durchführen.

Swizzling. In DBS ist es erforderlich, die Schlüssel der Objekte (OID) indirekt durch Verweisstrukturen (z. B. Datenbanknr., Segmentnr., Offset) zu implementieren, die ein Aufsuchen und Verschieben der Objekte auf Externspeicher gestatten. Für die hauptspeicherbasierte Verarbeitung in der Workstation ist diese durch Indirektion gewonnene Flexibilität jedoch weniger dringlich. Vielmehr belastet jede Indirektion das Leistungsverhalten und das macht sich bei der vorherrschenden navigierenden Verarbeitung besonders negativ bemerkbar, da die Auswertung einer OID-Referenz selbst bei einer effizienten Hash-Implementierung mindestens 50 Instruktionen kostet. Eine wesentliche Verbesserung erzielt man durch eine Klasse von Techniken - *Pointer-Swizzling* [27] genannt -, die externspeicherbasierte OID-Referenzen während der Verarbeitung auf hauptspeicherbasierte Verweise umstellen. Ihr Lösungsspektrum baut auf drei orthogonalen Kriterien auf: Ort, Zeitpunkt und Art des Swizzling [20].

Wenn die Objektformate und Seitenstrukturen beibehalten werden müssen (z. B. beim Server-Speicherungsmodell), kann *In-Place-Swizzling* gewählt werden. *Copy-Swizzling* arbeitet auf Kopien der Objekte und stellt die OID-Referenzen in der Kopie auf Verweise im Hauptspeicher (oder im virtuellen Speicher) um; dies würde beim WS-Speicherungsmodell angewendet werden. Beim ersten Verfahren erfolgt ein *Unswizzling* der Verweise (auf die OID-Referenzen) beim Zurückschreiben der geänderten Seiten, während beim zweiten die Rückstellung der Verweise nur für geänderte oder neue Objekte erforderlich ist. Das zweite Klassifikationskriterium charakterisiert den Zeitpunkt der Umstellung, wobei *sofortiges Swizzling* (eager swizzling) eine Umstellung der OID-Referenzen bei allen Objekten vornimmt, sobald sie in den Hauptspeicher gebracht werden. Hierbei ist die Auswertung der Verweise besonders schnell, da keine Laufzeitprüfung mehr erforderlich ist. Beim *Swizzling auf Anforderung* (lazy swizzling) werden die OID-Referenzen bei Erstreferenz oder später (d. h. nach anwendungsspezifischen Kriterien) umgestellt, was bei jeder Referenzauswertung eine zusätzliche

Prüfung impliziert. Weiterhin läßt sich als Art *direktes und indirektes Swizzling* unterscheiden, d. h., jeder Verweis zeigt direkt auf das Objekt oder nur auf einen Deskriptor des Objektes. Indirektes Swizzling kostet wiederum mehr, da ein zweiter Verweis zu verfolgen ist, jedoch gestattet es auch größere Flexibilität, da es das spätere Einladen oder die Ersetzung von Objekten während der Verarbeitung ermöglicht.

Hier können wir nicht alle möglichen 2^3 Swizzling-Kombinationen diskutieren und bewerten. Wegen der Vielzahl der Verfahrensvarianten mit unterschiedlichen Kosten dürfte es schwierig sein, allgemeingültige Regeln aufzustellen, bei welcher Referenzhäufigkeit sich Swizzling zu lohnen beginnt. Jedenfalls sind Faustregeln, die ein Swizzling empfehlen, wenn die mittlere Referenzhäufigkeit eines Objektes bei drei liegt, kritisch zu hinterfragen. Hervorheben wollen wir nur das sofortige, direkte Swizzling, das offensichtlich die schnellste Referenzauswertung gestattet. Es bietet sich insbesondere bei geschlossenen Kontexten an. In seiner Variante auf Objektkopien können zusätzlich noch spezielle Verarbeitungsanforderungen (durch das WS-Speicherungsmodell) berücksichtigt werden.

Versionsspeicherung. Verarbeitungskontexte werden durch Konfigurierung von Versionen von komplexen Objekten gebildet; sie dienen als Eingabe für die Anwendung (Werkzeugläufe). Ihr Ergebnis sind neue Versionen, die persistent zu speichern sind. Bei dieser Versionserzeugung werden in der Regel nur ein Teil der eingegebenen (Elementar-) Objekte aktualisiert und nur einige Objekte eingefügt bzw. gelöscht. Als Speicherrepräsentationsformen lassen sich dabei die *redundante* und die *inkrementelle* Versionsspeicherung unterscheiden. Die redundante Speicherungsform weist allen Objekten einer neuen Version neue Speicherplätze zu, die inkrementelle nur den geänderten und neu hinzugekommenen Objekten.

In allen Architekturen ermöglicht die redundante Versionsspeicherung eine optimale Clusterbildung, da bei der DB-Integration für alle an der neuen Version beteiligten Objekte gezielt ein Cluster aufgebaut werden kann (Seiten oder Objekte im Objektpuffer werden in ein entsprechendes Speichersegment kopiert). Im Vergleich zur versionsfreien Repräsentation der geänderten Objekte (update-in-place) verringert sich hierbei eher der Schreibaufwand. Es sind aber enorme Speicherungskosten in Kauf zu nehmen. Inkrementelle Versionsspeicherung dagegen gewährleistet Speicherökonomie, verschlechtert jedoch schrittweise eine evtl. vorhandene Clusterbildung.

Auch Synchronisationsaspekte werden wesentlich durch das Führen von Versionen beeinflußt. Bei einer versionsfreien Repräsentation führen überlappende Änderungsanforderungen von Transaktionen zu Zugriffskonflikten oder Blockierungen, während eine Versionsbildung die konfliktfreie Zuordnung einer Version zu einer Transaktion erlaubt. Selbst die nebenläufige Ableitung von neuen Versionen aus einer gemeinsamen Objektversion ist aus Sicht der Synchronisation problemlos möglich. Die Synchronisation wird besonders effektiv und effizient, wenn sie Versionen von Komplexobjekten als Einheiten (z. B. für das Sperren) nutzen kann.

Verarbeitungsformen. Verschiedene Formen der Verarbeitung von Anwendungsoperationen sind in Workstation/Server-Architekturen möglich. *Einstufige Verarbeitung* bedeutet, daß es nur einen Ort für die (DB-bezogene) Verarbeitung der Anwendungsoperationen gibt. Das impliziert, daß dort der aktuelle DB-Zustand vorliegen muß. In unserem Fall ist dieser Verarbeitungsort die Workstation. Bei kurzen Transaktionen würde sich hierfür auch der Server anbieten, der mit Hilfe von “function request shipping” oder “programmierter Verteilung” an die Workstation angebunden werden kann [16]. Demnach bedeutet *zweistufige Verarbeitung*, daß die Abwicklung der Anwendungsoperationen auf zwei verschiedene Orte aufgeteilt sein kann. Dazu ist wechselweise der aktuelle DB-Zustand (soweit er benötigt wird) herzustellen. In unserem Fall erstreckt sich die Verarbeitung auf Workstation und Server.

Verarbeitungsschnittstelle. In allen Architekturen ist die Anwendung von den Einzelheiten der Objektdarstellung und des Objektzugriffs isoliert. Wegen der Komplexität der Objekte und ihrer Verknüpfungen würde einerseits die Anwendungsentwicklung erschwert und andererseits wäre bei direkter Objektmanipulation durch die Anwendung ihre Integrität gefährdet. Deshalb findet die gesamte DB-bezogene Verarbeitung unter Kontrolle des Objekt-Managers (siehe Abb. 3) statt; er bietet der Anwendung eine objektorientierte Schnittstelle mit objektweiser Verarbeitung, wobei die Objektmanipulation durch Methodeneinsatz erfolgt. Optional kann mit einer Query-Schnittstelle zusätzlich eine mengenorientierte Verarbeitungsweise eingeführt werden.

Die Art der Integritätssicherung ist weitgehend durch die objektorientierte Schnittstelle vorgegeben. Einfache Integritätsbedingungen wie die Typkorrektheit oder Einhaltung von Wertebereichen können durch Nutzung abstrakter Datentypen für Objektattribute gewährleistet werden. Komplexere Bedingungen wie attributübergreifende Zusicherungen innerhalb eines Objekts lassen sich innerhalb der Objekt-Kapselung, z.B. durch entsprechende Methoden, realisieren. Für objektübergreifende Bedingungen (z.B. symmetrische Referenzbeziehungen) stellt der Objekt-Manager klassenspezifische oder gar globale Methoden zur Verfügung. Bei der Überprüfung kann es erforderlich sein, Objekte (Seiten) aus der DB nachzuladen.

Persistenz. Aus Sicht der Anwendung ist es wünschenswert, transiente und persistente Daten nebeneinander und gleichförmig benutzen zu können. Wenn Anwendung und DBS dasselbe Datenmodell verwenden (seamless interface), ist dies auf einfache Weise möglich. Dazu ist es sinnvoll, die Persistenz von Objekten typunabhängig zu realisieren. Erreichen läßt sich dies z.B. durch einen speziellen Allokieroperator (wie bei ObjectStore [24]), durch den die “Lebensdauer” eines Objektes zum Zeitpunkt seiner Erzeugung festgelegt wird. Eine andere Möglichkeit ist, Persistenz über die Erreichbarkeit (wie bei O_2 [10]) zu definieren, d. h., ein Objekt wird persistent, sobald es von einem bereits persistenten Objekt referenziert wird.

3.2 Grundformen der Workstation/Server-Architekturen

Nach diesem kurzen Überblick über die Workstation/Server-Architekturen soll ihre Eignung für die werkzeugorientierte Verarbeitung im Detail diskutiert werden. Dabei beziehen wir uns auf die Ausführungen in Abschnitt 3.1, insbesondere Abb. 3, und die Aufgaben, die bereits in Kapitel 1 identifiziert worden sind. Während in diesem Abschnitt zunächst die Grundformen der Architekturen diskutiert werden, gehen wir im anschließenden Abschnitt 3.3 auf Verbesserungen und Erweiterungen ein.

3.2.1 Page-Server

Bei der Grundform des Page-Servers bleibt die Einrichtung eines Objektpuffers in der Workstation außer Acht. Die DB-bezogenen Operationen erfolgen direkt auf den im Seitenpuffer eingelagerten Seiten. Wegen ihrer Einfachheit wurde die Page-Server-Architektur bisher in vielen Systemimplementierungen bevorzugt. Bekannte Vertreter sind ObjectStore [24] und O_2 [10].

Datenversorgung. Wie bereits eingeführt, erfolgt die Datenversorgung über den Austausch einzelner Seiten, gesteuert über Fehlseitenbedingungen, die beim Objektzugriff entstehen. Bei der Anforderung wird die voraussichtliche Verwendung der Seite (Lesen, Schreiben) spezifiziert, so daß der Lock-Server die entsprechenden Sperren anlegen kann. Eine Nutzungsänderung (Schreiben) verlangt deshalb eine explizite Sperrkonversion.

Der Kommunikationsaufwand zwischen Server und Workstation ist bestimmt durch die Anzahl und Größe der zu übertragenden Seiten (z. B. 4 KByte). Die Übertragung einer ganzen Seite benötigt nur etwa doppelt soviel Zeit wie die eines einzelnen Objektes, dessen Größe vielleicht 100 Byte beträgt, ist also pro Speichereinheit günstiger. Um diesen Effekt gewinnbringend einsetzen zu können, sollten im Schnitt mindestens zwei Objekte in jeder Seite von der Anwendung benötigt werden. Im Sinne der Optimierung ist es also zwingend geboten, möglichst viele von der Anwendung benötigten Objekte in möglichst wenigen Seiten unterzubringen. Diese Clusterbildung geschieht aber in der Regel zum Zeitpunkt der Objekterzeugung, so daß sie aus der Sicht nachfolgender Anwendungen unveränderbar ist und auch sehr ungünstig sein kann. Mangelnde Clusterbildung (entsteht z.B. bei Anwendungen mit überlappenden Objektmengen) zeigt eine Reihe von schwerwiegenden negativen Auswirkungen. Je weniger angeforderte Objekte in einer Seite sind, umso mehr "andere" Objekte sind normalerweise in ihr gespeichert. Dadurch ist die referenzierte Seitenmenge größer, was für die gleiche Objektmenge einen höheren Übertragungsaufwand und größere Pufferbereiche impliziert. Das bedeutet weiterhin, je geringer die Clusterbildung ist, umso störender wird die Wirkung der Seitensperren auf parallele Transaktionen. Die Spannweite der dabei entstehenden Kosten lassen sich bei unserer Referenzanwendung leicht aus Tab. 1 in Abschnitt 2.2 ableiten.

Der ständige Austausch von Seiten wirkt einer starken Fehlerisolation zwischen Workstation und Server entgegen. Wegen der häufigen Eintragungen ist es zu teuer, die aktuelle Sperrtabelle persistent zu halten. Ein Server-Crash impliziert aber dann den Abbruch der Verarbeitung in allen Workstations, da nach Wiederanlauf keine konsistente Fortsetzung der Datenversorgung garantiert werden kann. Seitenanforderungen während des Ausfalls würden ohnehin zu Verzögerungen oder zum Abbruch der Verarbeitung führen.

Datenrepräsentation in der Workstation. Da die Konvertierung der Seiten vom Externspeicherformat des Servers in das Hauptspeicherformat der Workstation möglichst geringen Aufwand verursachen soll, werden sowohl im Server als auch in der Workstation in der Regel gleiche Seitenformate benutzt (homogenes Speicherungsmodell). Durch die Wahl der zu unterstützenden Programmiersprache sind so Modell und Formate auf dem Server festgelegt. Damit ist der Ansatz (eher) *monolingual*, und für die Integration in weitere Sprachen sind umfangreiche Veränderungen (auch im DBS) erforderlich.

Da die Seiteninhalte ungeschützt in den Seitenpuffer der Workstation eingelagert werden, ergibt sich prinzipiell ein Problem bei der objektbezogenen Zugriffskontrolle, d.h., es gibt keine besonderen Schutzmaßnahmen für die mitgelieferten, aber nicht benötigten Objekte, für die möglicherweise gar keine Zugriffsrechte vorliegen. Es ist deshalb zu fordern, daß Seitenpuffer und Systemsoftware (Objekt-Manager u.a.) durch spezielle Maßnahmen (getrennte Adreßräume) hinreichend stark von der Anwendung isoliert werden.

Wegen der sprachbezogenen Repräsentation der Objekte in den Seiten findet auch keine besondere Anpassung an die Bedürfnisse der Anwendung statt. Insbesondere fehlt die Möglichkeit der Attributprojektion, da die Seiteninhalte vom Server übernommen und in der Workstation unverändert bleiben; es sind also immer alle Attribute eines Objektes sichtbar, obwohl in der Regel nur eine Teilmenge davon benötigt wird.

Sind zur inhaltsorientierten Suche spezielle Zugriffspfade des DBS erforderlich, so werden diese seitenweise zur Workstation übertragen und dort im Seitenpuffer verarbeitet. Auf diese Weise müssen alle Suchvorgänge und Datenmanipulationen in der Workstation abgewickelt werden, was einer einstufigen Verarbeitung entspricht. Dies bedeutet aber auch, daß bei einem Objekttyp-Scan alle Seiten des betroffenen Speichersegments zur Workstation zu transportieren sind, und zwar unabhängig von der Selektivität des Sucharguments.

Verarbeitung der Anwendungsobjekte. Wie bereits erwähnt bietet der Objekt-Manager der Anwendung eine objektorientierte Schnittstelle, durch die die Objektmanipulation und die Art der Integritätssicherung weitgehend vorgegeben sind.

Integration. Am Ende der Transaktion sind verzögerte Integritätsbedingungen zu überprüfen, was ebenfalls möglicherweise unter Nachladen von Seiten in der Workstation zu geschehen hat. Alle zum Schreiben angeforderten Seiten müssen zum Commit an den Server zurückgeschickt werden (erzwungenes Zurückschreiben, engl. FORCE). Da der Server nur Seiten und keine Objekte kennt, muß er als Log-Information zwangsläufig jeweils ganze Seiten schreiben.

3.2.2 Object-Server

In der Grundform dieses Ansatzes werden die Objekte durch die Anwendung ausschließlich über Navigation referenziert. Der Zugriff auf ein im Objektpuffer fehlendes Objekt produziert dabei genau eine Objektfehlbedingung, die zur Anforderung genau dieses Objektes an den Server führt. Dieser lädt die zugehörige Seite oder Seitenmenge in seinen Seitenpuffer, kopiert aus ihr das gewünschte Objekt in seinen Objektpuffer, sperrt es und liefert es der Workstation. Im Vergleich zum Page-Server weist der Object-Server eine deutlich höhere Systemkomplexität auf. Als Workstation/Server-Systeme, die diesen Ansatz verfolgen, sind zu nennen: Ontos [29], Versant [36] und Itasca (Orion) [21].

Wir betrachten hier nur kleine Objekte, die in einer Seite oder in wenigen Seiten gespeichert werden können. Wirklich große Objekte, etwa Multimedia-Objekte, erfordern spezielle Lösungsverfahren. Es liegt nahe, die Objekte selbst als Einheit von Synchronisation und Recovery auf dem Server zu verwenden, zumal man damit eine sehr gute Unterstützung der Parallelität zwischen Anwendungen und ein effektives Logging erzielt. Die beim Page-Server geschilderten Probleme hinsichtlich der Fehlerisolation gelten wegen der Häufigkeit der Objektanforderungen auch hier.

Datenversorgung. Die Objekte selbst bilden das Austauschgranulat zwischen Server und Workstation. Der Kommunikationsaufwand ist bestimmt durch die Anzahl und Größe der zu übertragenden Objekte (vgl. Tab. 1) und ist insensitiv gegenüber Clusterbildung zusammengehöriger Objekte in gemeinsamen Seiten. Die Clusterbildung wirkt sich hierbei nur kostensparend bei der Ein-/Ausgabe von der DB zum Seitenpuffer des Servers aus. Betrifft die sukzessive Objektanforderung mehrere Objekte aus einer Seite, so werden unabhängige Übertragungsvorgänge initiiert; dabei summieren sich die Kosten für alle erforderlichen Aktionen (RPC, Kopieren, Konvertieren, ...).

Bei diesem Ansatz erhält die Workstation alle Objekte, die sie referenziert, und zwar mit sämtlichen Attributen, obwohl in der Regel nur ein Teil der Attribute für die Verarbeitung notwendig ist (keine Projektionsmöglichkeit). Zudem wird jedes Attribut, wie groß auch immer, vollständig im Objektpuffer abgelegt. Weiterhin kann bei wertabhängiger Objektverarbeitung erst im Objektpuffer überprüft werden, ob ein Objekt die Suchbedingung erfüllt. Beispielsweise erfordert ein Objekttyp-Scan das aufeinanderfolgende Holen aller betroffenen Objekte in die Workstation und ihre Überprüfung im Objektpuffer. Besonders bei hoher Scan-Selektivität impliziert diese Verarbeitungsweise ein hohes Maß an nutzlosen Datenübertragungen.

Datenrepräsentation in der Workstation. Auf Server und Workstation können unterschiedliche Satzformate innerhalb der Objektpuffer existieren. Die Formate im DBS und in der Anwendungssprache sind zwar nicht zwingend abhängig voneinander, jedoch ist bei Sprachen mit ähnlicheren Formaten eine effizientere und einfachere Unterstützung möglich, da hier der Konvertierungsaufwand geringer ist.

Verarbeitung der Anwendungsobjekte. Genau wie beim Page-Server sind Objektmanipulation und auch die Art der Integritätssicherung durch die objektorientierte Schnittstelle weitgehend vorgegeben.

Integration. Nach der Verarbeitung durch die Anwendung erfolgt das Unswizzling aller geänderten Objekte, die dann zum Server zurückgeschickt werden (FORCE bei Check-in). Dort müssen alle Objektänderungen in die betreffenden Seiten eingebracht werden. Für die Log-Funktion bietet sich in diesem Fall Eintrags-Logging (Objekt-Logging) an.

3.2.3 Query-Server

Die wichtigste Erweiterung im Vergleich zum Object-Server ist an der Schnittstelle zur Anwendung zu finden. Diese Schnittstelle mit ihrer objektorientierten, navigierenden Verarbeitungsmöglichkeit wird durch eine allgemeine Anfragesprache ergänzt, mit der die Anwendung ihre Datenanforderungen formulieren kann. Die Datenbereitstellung kann dabei vom Query-Server mengenorientiert erfolgen, die Datenverarbeitung geschieht wiederum navigierend.

Query-Server-Architekturen wurden bisher nur als Prototyp-Systeme erprobt. Das Spektrum ihrer Realisierungsvarianten wird durch folgende Implementierungen illustriert: PRIMA [15], AIM-P [23], XNF [26], Starburst's Coexistence Approach [2], KRISYS [35], Persistence [19], PENGUIN [25], and ADMS-EWS [31].

Bei der Diskussion des Verarbeitungszyklus gehen wir zunächst in der Diskussion der Grundform des Query-Servers von einem abgeschlossenen Kontext aus, bei dem alle Operationen (außer der Extraktion und Integration der Daten) einstufig, d.h. nur in der Workstation, abgewickelt werden können. Dabei wird durch eine oder mehrere aufeinanderfolgende Anfragen der benötigte Kontext geholt (Check-out); nach ausschließlich lokalen Manipulationen und Integritätsprüfungen werden die geänderten Daten bei Commit zurück zum Server propagiert und in die DB integriert (Check-in).

Datenversorgung. Vor Beginn der Verarbeitung wird der gesamte Kontext durch eine oder mehrere Anfragen spezifiziert. Typischerweise wird dieser eine Menge von komplexen Objekten (netzwerkartig verknüpfte Objektmengen heterogener Art) umfassen, die man beispielsweise mithilfe von MQL- oder SQL/XNF-Anweisungen [26] beschreiben kann. Bei SQL müßte man zur Ableitung einer Menge von komplexen Objekten Anfragen zulassen, die sich aus mehreren Anweisungen zusammensetzen.

Die Auswertung der Anfrage findet im Server statt, wobei die komplexen Objekte (je nach Datenmodell unterschiedlich) abgeleitet und in Transferpuffern gesammelt werden. Durch Operationen wie Projektion, Selektion oder Verbund können die Objekte genau an die Sicht der Anwendung angepaßt werden. Bei der Wahl des Sperrkonzeptes und -granulats bleiben alle Freiheitsgrade erhalten. Idealerweise sollten die komplexen Objekte als Ableitungseinheit direkt gesperrt werden. Dies ist jedoch dann nicht möglich, wenn sie als dynamisch definierte und netzwerkartig verknüpfte Strukturen keinen Repräsentanten besitzen, der als Sperrhalter benutzt werden kann. Deshalb sind in einem solchen Fall alle Objekte der Menge einzeln zu sperren. Liegt ein geeignetes Versionskonzept für den Kontext vor [22], so läßt sich der Versionsrepräsentant als Sperrhalter heranziehen.

Als Abschluß der Check-out-Operation wird die gesamte Ergebnismenge auf einmal zur Workstation transferiert und im Objektpuffer allokiert. Ausgehend von der Annahme, daß im Schnitt nur etwa 50 % der Daten eines für die Werkzeugverarbeitung relevanten Objektes projiziert werden, entspricht hier das tatsächliche Volumen der zu transportierenden und in der Workstation einzulagernden Daten der Hälfte des in Tab. 1 aufgeführten Gesamtdatenvolumens (DVol). Da bis zum Check-in keine weiteren Kontakte mit dem Server erforderlich sind, resultiert aus dieser Art der Datenversorgung eine maximale Isolation im Fehlerfall (z.B. Crash von Server oder Workstation).

Datenrepräsentation in der Workstation. Es wurde bereits unterstrichen, daß die Anwendung ihre Sicht auf die Objekte spezifizieren kann. Zur effizienteren Referenzierung lassen sich alle Beziehungen zwischen Objekten als Hauptspeicheradressen ausprägen. Da die Speicherungsstrukturen der komplexen Objekte explizit angelegt werden, können sie (bereits im Transferpuffer) optimal auf das Zugriffsverhalten der Anwendung zugeschnitten werden. Falls spezielle Scans oder inhaltliche Suchvorgänge im Objektpuffer zu unterstützen sind, ist es in einfacher Weise möglich, passende Hauptspeicher-Zugriffspfade zu erzeugen. Der Query-Server-Ansatz bietet also maximale Flexibilität für Objektrepräsentation und Zugriffsunterstützung, welche wegen der objektweisen Anforderung bei den anderen Ansätzen nicht gegeben ist.

Verarbeitung der Anwendungsobjekte. Die dominierende Verarbeitungsart ist navigierend, wobei die Wahl der Speicherungsstrukturen ihre Leistung verbessern sollte. Als bewährtes Cursor-Konzept für komplexe Objekte lassen sich hierarchische Cursor [17] einführen, deren strukturelle Abhängigkeiten der Objekt-Manager kontrollieren kann.

Integration. Bei der Änderung von Objekten sind diese zu markieren, so daß bei Commit das Check-in selektiv erfolgen kann. So ist es möglich, daß nur die geänderten Attribute der Objekte (Netto-Änderungen) auf einmal zum Server zurücktransferiert werden. Als Zusatzaufwand fällt jetzt die Wiederholung der Änderungen auf Serverseite an. Dazu sind alle betroffenen Objekte mit ihren Seiten aufzusuchen und, falls noch nicht vorhanden, in den Serverpuffer zu laden. Die Integration der Änderungen betrifft die Objekte und ihre abhängigen Strukturen wie Zugriffspfade usw. Auch hier bietet sich das Eintrags-Logging für die Log-Funktion an.

3.3 Weiterentwicklungen der Architektur-Grundformen

In den vorangegangenen Abschnitten sind einige Schwachstellen der Architekturen angesprochen worden. Im folgenden werden Weiterentwicklungen vorgestellt, die diese Schwachstellen vermeiden sollen.

3.3.1 Erweiterungen des Page-Servers

Workstation-seitige Optimierung. Wichtig für eine Steigerung der Effizienz ist die Berücksichtigung der Verarbeitungslokalität und des Zugriffsverhaltens der Anwendung. Dazu ist es sinnvoll, die Speicherungsstruktur der Objektmengen beeinflussen zu können. Wie bereits in Abb. 3 angedeutet, läßt sich die Page-Server-Architektur durch Einführung eines Objektpuffers entsprechend erweitern. Damit wird gleichzeitig die Integration anderer Sprachen in das System sehr viel einfacher. Die Objekte werden dabei aus den übertragenen Seiten extrahiert und in einer der Anwendungsverarbeitung angepaßten Form in einem Objektpuffer eingelagert. Falls diese Anpassung Projektion von Attributen, Typkonversion und Pointer-Swizzling erlaubt, können viele Nachteile der direkten Übernahme des Server-Speicherungsmodells vermieden werden.

In der Workstation kann man nun entweder den Seitenpuffer durch einen Objektpuffer ersetzen oder aber zusätzlich zu dem Seitenpuffer einen Objektpuffer halten. In jedem Fall ergeben sich zusätzliche Kosten, da einerseits die Objekte aus den Seiten in die entsprechenden Speicherungsstrukturen des Objektpuffers kopiert, und andererseits alle Objektänderungen in den betreffenden Seiten nachgeführt werden müssen. Werden Seiten redundant zu Objekten gehalten, wird im Vergleich zur anderen Variante (ausschließliche Verwendung eines Objektpuffers) das Nachladen von Seiten zum Zwecke der Einlagerung der zum Server zu propagierenden, geänderten Objekte vermieden; jedoch verschlechtert sich die Speicherplatzausnutzung in der Workstation.

Server-seitige Optimierung. Bisher wurde unterstellt, daß der Server nur Seiten kennt, also nur diese als Einheiten des Sperrens und des Logging wählen kann. Dadurch wird das Logging enorm speicherintensiv, und (schreibgeschützte) Seiten können nicht vor Commit der Transaktion, der sie zugeteilt wurden, einer anderen Transaktion in einer anderen Workstation verfügbar gemacht werden.

Sobald der Server Objekte kennt, kann er die globale Synchronisation auf der Basis von Objektsperren bewerkstelligen. Die Objektanforderung der Anwendung (über OIDs) wird, falls das Objekt nicht in der Workstation gefunden wird, an den Server weitergereicht. Bei Sperrgewährung wird das Objekt O_i zusammen mit seiner Hausseite zur Workstation übertragen, wobei die Objektsperre als "lang" (bis Commit) und die Seitensperre als "kurz" eingetragen wird, d.h., eine Seite kann während einer Transaktion vom Server zurückgefordert und einer anderen Workstation zugeordnet werden, wenn dort ein Objekt O_k bearbeitet werden soll oder wenn die neue Sperranforderung verträglich zur Sperre von O_i ist. Dieser Ansatz, der viele Aspekte von DB-Sharing [30] aufweist und eine gemeinsame Nutzung von Seiten in parallelen Transaktionen gestattet, besitzt folgende Verarbeitungseigenschaften:

- Jede Objektsperre wird einzeln beantragt. Bei Clusterbildung werden zwar viele zusammengehörige Objekte auf einmal übertragen, bei Erstreferenz müssen die Sperren jedoch objektweise vom Server angefordert werden.
- Falls in den Seiten direkt gearbeitet wird (Server-Speicherungsmodell), verursacht die Zurückforderung oder der Austausch von Seiten zusätzliche Kosten, wenn Swizzling angewendet wird. Außerdem können Objektverschiebungen und Split-Vorgänge auftreten.
- Ist ein Objektpuffer vorhanden (WS-Speicherungsmodell), so sind Objektänderungen verschiedener Transaktionen bei deren Commit in eine Seite einzubringen, was sehr aufwendig sein kann (z.B. bei B*-Baumseiten).
- Es ist ein Caching von Seiten über Commit-Grenzen hinaus möglich. Damit kann ein Verarbeitungskontext (inklusive der Sperrren) direkt an den nachfolgenden Verarbeitungsschritt weitergegeben und so eine erneute Ladephase eingespart werden. Eine Seitenübertragung erfolgt dann beispielsweise erst auf Veranlassung einer anderen Workstation. Diese Vorgehensweise erfordert die Berücksichtigung von sog. Kohärenz-Protokollen [4], die die Aktualität der lokalen Daten garantieren.
- Die Workstation kann Log-Aufgaben übernehmen und eintragsbezogenes oder logisches Logging durchführen und die Log-Information dem Server geblockt zur Verfügung stellen. Auf diese Weise kann der globale Log wesentlich platz- und E-/A-sparender organisiert werden.

In [5] werden weitere Möglichkeiten der Erweiterung des Page-Server-Ansatzes hinsichtlich feinerer Synchronisationsgranulate im Detail diskutiert.

3.3.2 Erweiterungen des Object-Servers

Die bisher skizzierte Grundform des Object-Servers kann in mehreren Aspekten verbessert werden, um vor allem Kommunikationskosten und Speicherplatzbedarf zu reduzieren. Besonders wichtig erscheint eine Erweiterung bei der Objektanforderung, die neben der Anforderung über OIDs zumindest eine inhaltsorientierte Suche mit einfachen Suchausdrücken erlauben sollte. Voraussetzung ist jedoch eine entsprechend große Ausdrucksmächtigkeit der Anwendungsschnittstelle.

Einfache Suchbedingungen. Durch diese scheinbar “geringfügige” Erweiterung erzwingt man jedoch einen Übergang zur zweistufigen Verarbeitung. Dies führt im allgemeinen eine hohe Zusatzkomplexität ein, da von einer potentiell replizierten DB ausgegangen werden muß. Durch die Objektänderungen im WS-Objektpuffer sind manche Objekte in zwei Versionen im System (veraltete Version im Server) vorhanden. In Abschnitt 3.3.3 wird gezeigt, daß bei allgemeinen Anfragen der aktuelle DB-Zustand erst auf Server- oder Workstationseite hergestellt sein muß, bevor die Anfrage in konsistenter Weise evaluiert werden kann. Das erfordert aber beispielsweise eine Propagierung von Änderungen zum Server oder ein Nachladen des Auswertungskontextes zur Workstation.

Die Beschränkung auf einfache Suchausdrücke gewährleistet die Überprüfbarkeit des Suchprädikats auf jeweils einem Objekt, was hier zur Vereinfachung des Suchvorgangs ausgenutzt werden kann. Wegen des speziellen Suchprädikates kann mit Hilfe der folgenden Vorgehensweise auf den im allgemeinen erforderlichen Auswertungs-Overhead verzichtet werden. Ein Objekttyp-Scan mit dem Suchprädikat *SP* liefert auf dem Server im allgemeinen eine Menge potentiell veralteter Objekte. Diese Treffermenge kann außerdem zu groß oder unvollständig sein, da in der Workstation entsprechende Objekte gelöscht, geändert oder neu erzeugt worden sein können. Eine Suche mit dem gleichen Suchprädikat SP in der Workstation stellt eine Menge aktueller Objekte bereit. Diese Treffermenge kann ebenfalls unvollständig sein, da nicht alle Objekte, die sich für SP qualifizieren, in der Workstation gepuffert sein müssen. Zur Konstruktion der aktuellen Treffermenge kommt erschwerend hinzu, daß auch gelöschte oder geänderte Objekte im WS-Objektpuffer, die SP erfüllt haben, zu berücksichtigen sind. Diese Objekte sind nur als gelöscht oder geändert markiert, da die betreffenden Änderungen noch zum Server propagiert werden müssen.

Die aktuelle Objektmenge, die sich für SP qualifiziert, muß durch Vergleich der OIDs im WS-Objektpuffer konstruiert werden. Dazu führen wir folgende Bezeichnungen ein:

- OID_{Server} : Menge der OIDs der Objekte, die vom Server angeliefert werden
- OID_{qual} : Menge der OIDs der Objekte, die im WS-Objektpuffer SP erfüllen (unveränderte Objekte, neu eingefügte Objekte und geänderte Objekte, so daß SP nach Änderung erfüllt)
- OID_{mod} : Menge der OIDs der Objekte, die im WS-Objektpuffer SP nicht mehr erfüllen (gelöschte Objekte, geänderte Objekte, so daß SP nach Änderung nicht mehr erfüllt)

OID_{qual} und OID_{mod} sind disjunkt; $OID_{qual} \cup OID_{mod}$ ergibt OID_{WS} als Menge der OIDs aller betroffenen Objekte in der Workstation. Somit läßt sich die aktuelle Ergebnismenge OID_{Erg} für SP wie folgt ableiten:

$$OID_{Erg} = OID_{qual} \cup (OID_{Server} - OID_{WS})$$

Besonders einfach wird diese Ergebnisbereitstellung, wenn keine betroffenen Objekte im WS-Objektpuffer gespeichert sind:

$OID_{Erg} = OID_{Server}$

Projektion. Es bleibt aber auch bei dieser Erweiterung das Problem, daß alle Attribute eines Objektes, die sehr groß sein können (Text oder Grafik), vollständig in der Workstation abgelegt werden. Wenn im voraus bekannt ist, welche Attribute (oder Attributwertintervalle) bearbeitet werden, ist eine Projektionsmöglichkeit auf Typ- oder Wertebene vorteilhaft. Das führt allerdings zu neuen Problemen, da durch das Wegfallen eines Attributes evtl. einige Methoden nicht mehr anwendbar sein können.

Delegation von Integritätsprüfungen. Um den Übertragungsaufwand bei einmaligen Objektreferenzen zu begrenzen, können in einer Anwendung Aufträge zur Integritätsüberprüfung an den Server delegiert werden. Dabei sind Vorkehrungen erforderlich, um für die Integritätsprüfung im Server den aktuellen DB-Zustand zu gewährleisten.

Komplexe Objekte. Eine Erweiterung dieses Ansatzes, bei der dem Object-Server die Strukturbeziehungen zwischen den Elementarobjekten bekannt gemacht werden, zielt auf eine beträchtliche Verminderung der Kommunikation zwischen Server und Workstation ab. Dabei kann der Object-Server bei Anforderung eines Wurzelobjektes abhängige Objekte bestimmen und als komplexes Objekt in einem Vorgang zur Workstation übertragen. Dadurch läßt sich außer dem Kommunikationsaufwand auch die Last auf dem Server, der für jedes Objekt mehrmals die zwischendurch vielleicht sogar verdrängte Seite abarbeiten muß, reduzieren. Bei dieser Vorgehensweise hängt jetzt die Anzahl der Datentransfer-Vorgänge von der Anzahl der benötigten komplexen Objekte ab (siehe Tab. 1). Jedoch bleiben bei diesem Ansatz die komplexen Objekte für die Anwendungen statisch, d. h. fest vorgegeben, und es ist nicht möglich, dynamisch beliebige komplexe Objekte zusammenzustellen.

Diese Erweiterung zum *Complex-Object-Server* und die inhaltsorientierte Suche sind erste Schritte in Richtung *Query-Server*, der auf eine Anforderung hin eine Menge von komplexen Objekten auf einmal zurückliefert. Die dabei auftretende Verarbeitungskomplexität wird im nachfolgenden Kapitel diskutiert.

3.3.3 Erweiterungen des Query-Servers

Der allgemeine Fall bezieht sich auf einen offenen Kontext und wird benötigt, wenn der Verarbeitungskontext nicht vollständig im voraus bekannt ist; er schließt mehrere Check-out sowie die Delegation von Prüfaufträgen an den Server oder das Nachladen von Daten zum Zwecke der Integritätsprüfung ein. Diese Verarbeitungsform wurde als zweistufig bezeichnet.

Da Anfragen verschickt werden, besitzt die Grundform schon die volle funktionale Mächtigkeit. Die Erweiterungen beziehen sich deshalb hier auf die Unterstützung flexiblerer Anwendungsanforderungen (offener Kontext) und die Optimierung der Anfrageverarbeitung (Delegation von Teilanfragen).

Offener Kontext. Im Verlauf des Verarbeitungszyklus wird der Kontext in der Workstation ständig aktualisiert. Dabei können die Änderungen (nicht-freigegebene Daten) beim abgeschlossenen Kontext bis zum Check-in im Objektpuffer gehalten werden, bis sie dann auf einmal zum Server propagiert werden. In diesem Fall ist der Teil der aktuellen DB, der von der Anwendung referenziert wird, ausschließlich im Objektpuffer. Beim offenen Kontext dagegen wird vom Prinzip her eine zweistufige Verarbeitung erzwungen, für die jeweils der aktuelle DB-Zustand, der aus der Sicht der Anwendung alle ihre bisherigen Änderungen einzuschließen hat, verfügbar sein muß. Die wesentliche Schwierigkeit besteht dabei darin, daß es sich bei der Workstation/Server-Datenverteilung um eine teilweise replizierte DB handelt, da Änderungen in Datenkopien im Objektpuffer erfolgen und diese nicht sofort in die (globale) Server-DB propagiert werden. Das unmittelbare Nachführen jeder Änderung auf der Serverseite würde jedoch die Nützlichkeit des gesamten Ansatzes in Frage stellen.

Um überhaupt nach tragfähigen Lösungskonzepten suchen zu können, treffen wir die Annahme, daß es sich bei den Check-out- oder anderen Nachladeforderungen um relativ seltene Ereignisse handelt, weil sonst das Gesamtsystem fast nur noch mit der Herstellung des aktuellen DB-Zustandes beschäftigt sein dürfte. Folgende Vorgehensweisen sind möglich:

- **Nicht-überlappende Nachforderung.** Die Anwendung garantiert, daß die nachzuladenden Objekte disjunkt zu den Objekten, die sich momentan im Objektpuffer befinden, sind. Die Objekte können ohne Zusatzmaßnahme in der Server-DB bestimmt und in den Objektpuffer integriert werden.
- **Propagierung des relevanten Objektpufferinhalts.** Die Anwendung stellt eine deskriptive Anfrage als Nachforderung. Der Objektmanager stellt fest, welche geänderten Teile des Objektpuffers von Anfrageprädikat oder der Ausgabemenge betroffen sind, und propagiert diese in die Server-DB. Falls die betroffene Objektmenge nicht eingeschränkt werden kann, sind alle geänderten Objekte zu propagieren. Dabei müssen die Objekte erst in die Seiten, Zugriffspfade usw. integriert werden, bevor sie sich in die Anfrageevaluation einbeziehen lassen. Weiterhin impliziert diese Wiederholung der Änderung auf Serverseite das Schreiben der erforderlichen Log-Information. Die Ergebnisobjekte werden dann zur Workstation transferiert und redundanzfrei in den Objektpuffer eingelagert, was über die OIDs kontrolliert werden kann.
- **Evaluierung der Nachforderung im Objektpuffer.** Die Idee dieser Vorgehensweise ist es, den aktuellen DB-Zustand, soweit er für die Auswertung der Nachforderung gebraucht wird, im Objektpuffer aufzubauen. Dazu ist es erforderlich, Auswertungsmöglichkeiten für die volle Anfragemächtigkeit im Objektpuffer bereitzustellen. Dieser Ansatz impliziert, daß in der Regel sehr viele Daten zu übertragen (z. B. Objekttyp-Scan) und geeignete Zugriffspfade im Objektpuffer anzulegen sind. Die Hilfsdaten, die nur der Auswertung der Nachforderung dienen, müssen dann wieder entfernt werden.

Die zuletzt angesprochene Vorgehensweise wird i. a. viel zu komplex und teuer sein, weil zunächst die Auswertungsumgebung in die Workstation zu übertragen ist. So bleibt - von Sonderfällen abgesehen - die “Propagierung des relevanten Objektpufferinhalts” als einzige gangbare Lösung. Da die Änderungen ohnehin propagiert werden müssen, wird lediglich ein Vorziehen dieser Arbeit erzwungen; nur bei Mehrfachänderungen eines Objekts kann so Mehrarbeit entstehen.

Bei offenen Kontexten ist die Datenversorgung wesentlich komplexer geworden. Bei der Verarbeitung dagegen ergeben sich keine Unterschiede. Lediglich bei der Integritätsprüfung ist eine differenziertere Vorgehensweise möglich. Ist eine Überprüfung erforderlich, die Daten/Referenzen außerhalb des verfügbaren Kontextes benötigt, so kann entweder eine Nachforderung veranlaßt oder ein Prüfauftrag (z.B. für die Einhaltung der referentiellen Integrität) an den Server gestellt werden. Natürlich muß auch hier vor Ausführung der Operationen die Konsistenz des aktuellen DB-Zustandes gewährleistet sein.

Check-in liefert keine neuen Aspekte. Es ist die Integration aller Änderungen zu bewerkstelligen, wobei viele Änderungen schon während der Verarbeitung bei Nachforderungen propagiert worden sein können.

Delegation von Teilanfragen. Die Diskussion der Nachforderungen bei offenen Kontexten, die mit einer deskriptiven Anfragesprache spezifiziert werden können, hat im wesentlichen zwei Optionen erbracht: die vollständige Ausführung der Anfrage auf dem Server(-puffer) oder ihre vollständige Ausführung im Objektpuffer. Dabei wird der Aufwand, die gesamte Auswertungsumgebung im Objektpuffer einrichten zu müssen, diese zweite Möglichkeit im allgemeinen disqualifizieren. Wenn aber andererseits Objektpuffer sehr groß werden (> 100 MBytes), dann kann das Propagieren der Änderungen (deren Anzahl proportional zur Puffergröße angenommen wird) auch sehr teuer werden. Deshalb kann es sinnvoll sein, für die Query-Server-Architektur nach Optimierungsmöglichkeiten für die Aufgabe, wo und wie Anfragen ausgewertet werden, zu suchen.

Die Delegation von Teilanfragen an den Server, die Auswertung der komplementären Teilanfragen im Objektpuffer sowie das Mischen/Vervollständigen des Gesamtergebnisses auf der Workstationseite ist ein solcher Ansatz. Er impliziert, daß sowohl in der Workstation als auch im Server alle Funktionen der Anfrageverarbeitung ausgeführt werden können.

Die wichtigste und am schwierigsten zu realisierende Voraussetzung auf der Workstationseite ist das genaue Wissen, was alles (welche Objekte und ob vollzählig) im Objektpuffer ist, um entscheiden zu können, ob sich eine Teilanfrage konsistent und vollständig auswerten läßt. In KRISYS [12] übernimmt diese Aufgabe der Kontext-Manager, der mit Hilfe von Prädikaten die gepufferten Objektmengen charakterisiert und Informationen über ihre Anwesenheit/Vollzähligkeit verwaltet. Im Verlauf von Änderungen der Objekte können sich erhebliche Komplexitäten bei der Buchführung ergeben, da Objekte von m Kontextmengen (Prädikate) in n andere Kontextmengen migrieren können.

Abb. 4 zeigt auf der linken Seite ein Beispiel einer Anfrage in SQL-Notation mit dem zugehörigen Operatorbaum. Sie dient dazu, zu einer im Objektpuffer vorhandenen Menge von Zellen (<oid-set>) mittels eines Joins die zugehörigen Schnittstellen nachzufordern. Bei der Übersetzung/Optimierung einer Anfrage erkennt der Kontext-Manager, welche Teilanfragen sich im Objektpuffer beantworten lassen. Die restlichen Teilanfragen werden zur Abwicklung an den Server geschickt. Der optimierte Operatorbaum beinhaltet als Blätter zwei Teilanfragen, von denen die rechte an den Server zu delegieren ist und die linke direkt auf dem Objektpuffer ausgeführt wird. Die Qualifikation der Schnittstellen geschieht mittels der Menge <ref-set>, die der Kontext-Manager aus den im Objektpuffer vorhandenen Zellen ermittelt. Bei dieser Auswertung muß ggf. durch Propagation von Änderungen garantiert werden, daß jede Teilanfrage als konsistente Sicht auf demselben DB-Zustand evaluiert wird. Die serverseitigen Ergebnisobjekte werden an den Objektpuffer geliefert, wo die Frageauswertung dann komplettiert wird. Dieses Verarbeitungskonzept wurde in ganz ähnlicher Weise im Projekt

ADMS [31] aufgegriffen. Dort werden Anfrageergebnisse in einem Anfrage-Cache, der auf der Workstation liegt, zwischengespeichert und, soweit möglich, zur Beantwortung von nachfolgenden Anfragen verwendet [7]. Allerdings liegen dazu bisher nur vereinfachende Simulationsstudien vor.

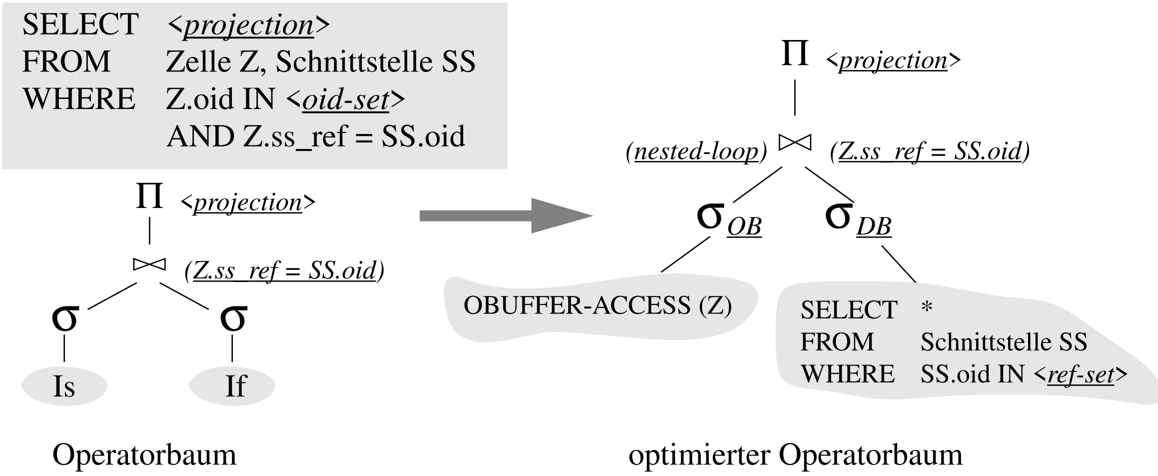

Abb. 4. Beispiel für die Delegation von Teilanfragen

Mit dem Ansatz der Delegation von Teilanfragen werden der Anwendung deskriptive Anfragen hoher Auswahlmächtigkeit ermöglicht, die, falls der Kontext ausreicht, ausschließlich im Objektpuffer bewältigt werden. Bei Referenzen auf den Server strebt die Auswertung eine Aufteilung der Teilanfragen an, so daß die Verarbeitungskosten minimal werden. Das Ergebnis ergänzt dann den offenen Kontext im Objektpuffer.

4. Bewertung der Architekturansätze

Nachdem die verschiedenen Workstation/Server-Architekturen diskutiert worden sind, sollen diese nun anhand der sie charakterisierenden Kriterien bewertet werden. Darauf aufbauend und ausgehend von den spezifischen Eigenschaften eines Anwendungsbereiches wird eine Entscheidungshilfe zur Auswahl einer geeigneten Zielarchitektur skizziert.

4.1 Bewertungskriterien für die Workstation-Server-Kooperation

Im folgenden werden allgemeine "Faustregeln" aufgestellt, die Leistungsfähigkeit und Flexibilität einer Workstation/Server-Verarbeitung in hohem Maße bestimmen. Für unsere Fragestellung werden diese als Bewertungskriterien für die verschiedenen Ansätze benutzt.

(1) **Minimierung der Kommunikation zwischen Server und Workstation.** Die Anzahl der Datenübertragungen und die Größe des Kommunikationsvolumens haben offensichtlich wesentlichen Einfluß auf die Systemeffizienz. Beim Page-Server hängt der Kommunikationsaufwand stark vom Grad der Clusterbildung ab. Der Object-Server genügt dieser Regel aufgrund der Einzelanforderung und -übertragung von Objekten weniger, während der Query-Server diese Forderung am ehesten erfüllt, da mengenorientierte Anforderungen mit der Bereitstellung der Ergebnismenge beantwortet werden. Darüber hinaus fördert das Konzept der Delegation von Teilaufträgen eine weitere

Reduktion der Kommunikation. Allerdings wird an der Query-Server-Architektur deutlich, daß die Verringerung des Kommunikationsvolumens durch eine höhere Systemkomplexität erkauft wird.

(2) **Minimierung der Anzahl der Kopiervorgänge.** Die Einführung eines eigenen WS-Speicherungsmodells impliziert Extra-Kopiervorgänge durch die Extraktion der Objekte und das Wiedereinbringen der Änderungen. Deshalb genügt die Übernahme des Server-Speicherungsmodells dieser Regel am besten; der Page-Server bietet also in dieser Hinsicht die optimale Lösung.

(3) **Unterstützung größtmöglicher Transaktionsparallelität.** Gegenseitige Behinderung oder gar Blockierung von Transaktionen (Werkzeugläufen) werden dann minimal, wenn die Objekte als Einheiten der Verarbeitung auch als Einheiten des Sperrens verwendet und mit angemessenen Sperrmodi belegt werden können. Wenn der Page-Server als Sperrgranulat Seiten verwendet, kann er dieser Regel nur nachkommen, falls alle gelieferten Seiten ausschließlich für die jeweilige Anwendung relevante Daten enthalten. Ist dies nicht der Fall, so werden implizit mit der Seitensperre auch andere Objekte isoliert, was dann zur Blockierung parallel laufender Anwendungen führen kann. Object- und Query-Server besitzen hier größere Freiheitsgrade zur Optimierung, wobei bei letzterem durch die Verwendung komplexer Objekte oder eines geeigneten Versionskonzeptes der Synchronisationsaufwand drastisch reduziert werden kann.

(4) **Reduktion der E/A durch platzsparendes Logging.** Logisches Logging oder Eintragslogging läßt sich bei allen Architekturen anwenden, bei denen der Server Objekte kennt. Nur der Page-Server in seiner Grundform verstößt gegen diese Regel.

(5) **Effizienter Objektzugriff während der Verarbeitung.** Eine Zugriffsoptimierung kann nicht nur über Pointer-Swizzling, sondern auch über zusätzliche Zugriffspfade erfolgen, wenn spezielle Zugriffsfolgen häufig auftreten. Dies ist in allen Ansätzen möglich, aber von der Anwendung abhängig: Pointer-Swizzling lohnt sich immer dann, wenn Objekte mehrmals referenziert werden, und Zugriffspfade können nur gewinnbringend eingesetzt werden, wenn das Zugriffsverhalten der Anwendung bekannt ist.

(6) **Maximierung der Fehlerisolation.** Mit der Minimierung der Anzahl der Datenanforderungen an den Server wird eine höhere Fehlerisolation dadurch erreicht, daß nunmehr längere Verarbeitungsphasen in der Workstation bei Ausfall des Servers ohne Verzögerung abgearbeitet werden können. Im Falle nicht persistenter Sperren wird jedoch die Fehlerisolation eingeschränkt, da ein Ausfall des Servers dann das Rücksetzen aller laufenden Transaktionen impliziert.

Die Diskussion verdeutlicht bereits, daß keine global optimale Architektur existiert. Dies wird noch deutlicher angesichts der Existenz gegenläufiger Abhängigkeiten zwischen den Regeln sowie der Notwendigkeit der Einbeziehung von Anwendungskriterien in die Beurteilung. So ist die Flexibilität der Anwendungsanbindung und die Auswahlmächtigkeit bei der Datenanforderung durch die Gesichtspunkte Sprachintegration, Datenrepräsentation und Datenversorgung bestimmt. Hierbei ist zu beachten, daß multi-linguale Sprachintegration [8], [34] dem Konzept der nahtlosen Schnittstelle (seamless interface) widerspricht und zugeschnittene Datenrepräsentation ein unabhängiges WS-Speicherungsmodell fordert. Damit zeigt sich ein Konflikt zu Regel 2. Weiterhin besteht zwischen Verarbeitungsart und der geforderten Flexibilität bzgl. der Datenversorgung folgende Abhängigkeit: Einstufige Verarbeitung erzwingt die Beschränkung auf genau eine deskriptive, mengenorientierte Anforderung (Check-out) oder verschiedene Einzelanforderungen (über OID), während Zweistufigkeit beliebige deskriptive, mengenorientierte Nachforderungen erlaubt. Der Einfluß anwendungsspezifischer Gesichtspunkte wird im folgenden Abschnitt weiter ausgeführt.

4.2 Kriterien zur Architekturauswahl

In Kapitel 2 wurde zwischen abgeschlossenen und offenen Kontexten unterschieden. Allgemein kann eine Architektur, die sich zur Verarbeitung offener Kontexte eignet (dies gilt für die Grundform aller angesprochenen Ansätze), auch bei abgeschlossenen Kontexten eingesetzt werden. Während bei abgeschlossenem Kontext einstufige Verarbeitung genügt, ist bei offenen Kontexten und deskriptiven Nachforderungen Zweistufigkeit erforderlich. Um nun für eine gegebene Anwendung eine Wahl zwischen den Architekturen zu treffen, ist es jedoch sinnvoll, eine mögliche Ausnutzung von Kontextwissen zu berücksichtigen. Diese kann auf zwei Arten geschehen. Explizite Nutzung bedeutet, daß der Kontext in einer oder mehreren Anfragen an- bzw. nachgefordert wird. Werden die Objekte dagegen über Objekt/Seiten-Fehlbedingungen angefordert, konnte aber bereits vorher eine Clusterbildung als Kontext erzielt werden (insbesondere für Page-Server wichtig), so liegt eine implizite Nutzung vor. Die explizite Nutzung durch den Query-Server ist flexibler, da der Kontext zum Zeitpunkt der Anforderung beschrieben werden kann und nicht im voraus durch eher starre Anordnung der Objekte in entsprechenden Speicherungsstrukturen manifestiert sein muß. Weitere Vorteile der expliziten Nutzung sind weniger Kommunikation, Möglichkeit der Query-Optimierung und weniger Wartungsaufwand (Reorganisation) für Pufferstrukturen.

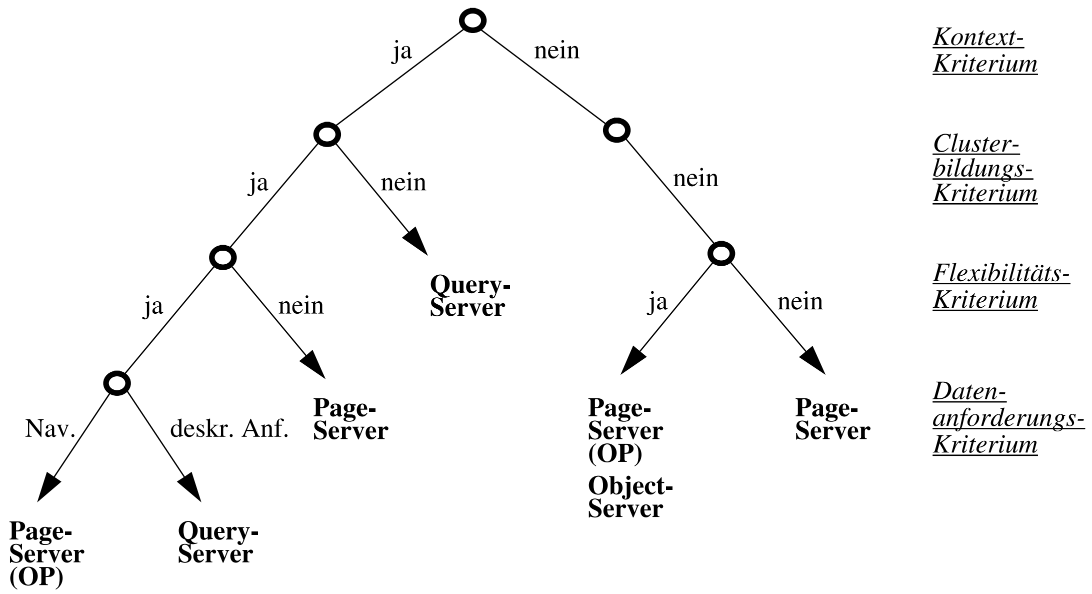

Abb. 5. Architekturauswahl nach Anforderungen der Anwendung

Der in Abb. 5 dargestellte Baum gibt eine Hilfestellung für die Auswahl eines Architekturansatzes ausgehend von folgenden anwendungsspezifischen Fragestellungen:

- Liegt Kontextwissen vor? (Kontextkriterium)
- Ist die Clusterbildung von Verarbeitungskontexten möglich? (Clusterbildungskriterium)
- Ist Flexibilität bzgl. der Datenrepräsentation erforderlich? (Flexibilitätskriterium)
- Sind deskriptive Anfragemöglichkeiten gefordert oder soll implizit beim Navigieren angefordert werden? (Datenanforderungskriterium)

Ist kein Kontextwissen vorhanden (rechter Teilbaum), so kann auch davon ausgegangen werden, daß keine sinnvolle Clusterbildung von Verarbeitungskontexten existiert. Unter Berücksichtigung des Flexibilitätskriteriums können hier Page- und Object-Server empfohlen werden, wobei Effizienzgründe insbesondere für den Page-Server sprechen (weniger Kommunikation, geringere Anzahl von Kopiervorgängen). Sind flexible Datenrepräsentation und/oder mehrsprachige Anwendungsanbindung erforderlich, so können die Page-Server-Erweiterung oder der Object-Server herangezogen werden. Die Erweiterung des Object-Servers um eingeschränkte, mengenorientierte Verarbeitung (SSA) läßt sich in beschränktem Maße effizienzsteigernd einsetzen, bedient aber die gleiche Anwendungsklasse wie die Grundform. Der Query-Server erscheint in einem solchen Szenario (kein Kontextwissen) ungeeignet, da eine seiner wesentlichen Fähigkeiten, die deskriptive Anforderung ganzer Kontexte/Kontextteile via Server-Anfrage, nicht sinnvoll ausgenutzt werden kann.

Ist andererseits Kontextnutzung - eine Unterscheidung zwischen abgeschlossenem und offenem Kontext ist hier unerheblich - möglich (linker Teilbaum), stellt sich zunächst die Frage, ob Clusterbildung auf die Kontexte anwendbar ist. Ist dies nicht der Fall, so ist der Query-Server zu empfehlen, da die Auswertung einer deskriptiven Anfrage serverseitig optimiert werden kann. Hierbei ist zu beachten, daß bei abgeschlossenem Kontext nicht die volle Systemkomplexität zum Tragen kommt, da zwischenzeitig kein aktueller DB-Zustand auf dem Server zu erzeugen ist und keine nachgeforderten Daten in den Objektpuffer integriert werden müssen. Ist Clusterbildung möglich, sollte weiter beachtet werden, inwieweit eine flexible Datenrepräsentation notwendig ist. Besteht hierzu keine Notwendigkeit, dann ist dem Page-Server aus Effizienzgesichtspunkten der Vorzug zu geben. Andernfalls (flexible Repräsentation) sollten die Page-Server-Erweiterung oder der Query-Server abhängig von der gewünschten Art der Datenanforderung eingesetzt werden. In dieser Situation scheint der Object-Server seinen Konkurrenten zu unterliegen: deskriptive Anforderung votiert eindeutig für den Query-Server, und aufgrund der Clusterbildung ist das Anforderungsgranulat Seite beim Page-Server effektiver als die objektweise Anforderung beim Object-Server. Insgesamt ist bei Ausnutzbarkeit von Kontextwissen insbesondere der Query-Server zu empfehlen. Sollen deskriptive Anfragen gestellt werden können, so kommt nur er in Frage. Ein Ansatz mit geringerer Systemkomplexität (Page-Server-Varianten) ist nur dann sinnvoll, wenn eine geeignete Clusterbildung dieser Kontexte garantiert werden kann.

Diese anwendungsbezogene Architekturauswahl greift im wesentlichen auf die Grundformen (Abschnitt 3.2) zurück, gilt aber auch für die daraus abgeleiteten Optimierungen, die in Abschnitt 3.3 vorgestellt wurden. Um zwischen den verschiedenen Optimierungen auswählen zu können, sind detailliertere Anwendungskenntnisse hinsichtlich Mehrbenutzerbetrieb, Kontext und Clusterbildung sowie quantitative Analysen erforderlich.

4.3 Konsequenzen für Entwurfsanwendungen

In der in Kapitel 2 angesprochenen VLSI-Entwurfsanwendung liegen abgeschlossene Kontexte vor, da die Eingangsdaten für einen Werkzeuglauf ausschließlich durch in vorangegangenen Verarbeitungsschritten erzeugte Versionen bestimmt sind. Durch die Versionierung der komplexen Entwurfsobjekte aufgrund der schrittweisen Verarbeitung wird eine Konfigurierung der Eingabedaten notwendig. Da diese von der Erzeugung der einzelnen Objektversionen zeitlich getrennt ist, kann im allgemeinen keine Clusterbildung erzielt werden. Deshalb ist das Kontextwissen nur über deskriptive Anfragen nutzbar. Weiter erfordern die verschiedenen Werkzeug-Anwendungen aufgrund unterschiedlichen Zugriffsverhaltens eine möglichst flexible Anpassung der Datenrepräsentation. Letzteres ist möglich, weil das Zugriffsverhalten eines Werkzeugs vorab bekannt ist. Ähnliche Kontexteigenschaften liegen unserer Meinung nach in vielen Entwurfsanwendungen (z.B. Software-Entwurf, CAD im Maschinenbau) vor.

Bei solchen Anwendungen kann nach unserem Auswahlbaum der Query-Server in seiner einstufigen und deshalb einfacheren Version herangezogen werden. Der abgeschlossene Kontext läßt sich zu Beginn der Verarbeitung mit einer entsprechend mächtigen Anfragesprache (Mengenorientierung, Projektion) spezifizieren und laden (Check-out). Dadurch liegt beim Aufbau der Speicherungsstrukturen in der Workstation bereits der gesamte Kontext vor, so daß kein Nachfordern notwendig wird und die Reorganisation der Pufferstrukturen auf ein Minimum beschränkt wird. Zur Begrenzung des Synchronisationsaufwandes sind Sperren auf Repräsentanten (Komplexobjekt oder Version) zu empfehlen. Eine Begrenzung des Logging-Aufwands erreicht man mit Eintrags-Logging bzgl. der Versionsobjekte.

Page- oder Object-Server sind in diesem Anwendungbereich weniger geeignet, da der Verarbeitungskontext in der Regel sehr groß (Größenordnung MB) ist und sehr viele Elementarobjekte enthält, was zu einer hohen Zahl von Anforderungen führen würde. Durch die beschriebene fehlende Möglichkeit zur Clusterbildung bedeutet dies nicht nur für den Object-, sondern auch für den Page-Server einen unverhältnismäßig hohen Kommunikationsaufwand.

Bei interaktiven CAD-Anwendungen, bei denen der Entwerfer ad-hoc DB-Anfragen stellt, ergibt sich ein Zugriffsverhalten, das sich als offener Kontext beschreiben läßt. In ähnlicher Weise können wissensbasierte Anwendungen [12] charakterisiert werden. Dabei entstehen vorzugsweise durch sukzessive Konsultationen offene Kontexte, deren Bereitstellung insbesondere durch “Delegation von Teilfragen” effektiv erfolgen kann [35], so daß auch hier der Query-Server empfohlen werden kann.

5. Resümee und Ausblick

In dieser Arbeit wurden verschiedene Ansätze zur Realisierung von datenbankbasierten Ingenieuranwendungen in Workstation/Server-Umgebungen beschrieben und bewertet. Dabei bezog sich der Fokus unserer Betrachtungen auf die Problematik der Datenversorgung einer workstation-basierten Anwendung durch einen Datenbank-Server. Im einzelnen wurden folgende Schwerpunkte gesetzt:

- Extraktion der benötigten Daten vom Datenbank-Server

- Datensicherung und Synchronisation
- Datenbereitstellung und -verarbeitung im workstation-seitigen Objektpuffer
- Integration der geänderten Daten.

Unsere Untersuchungen der verschiedenen Workstation/Server-Architekturen haben gezeigt, daß keine global optimale Architektur existiert und daß für die Bewertung eines Ansatzes die spezifischen Charakteristika der Zielanwendung entscheidend sind. Hier wurden speziell die Auswirkungen eines Verarbeitungskontextes (offen oder geschlossen), der Einfluß von Clusterbildung sowie die Rolle eines WS-Speicherungsmodells untersucht und deren Konsequenzen für die Datenversorgung in Workstation/Server-Umgebungen analysiert. Im Gegensatz zu bisherigen Untersuchungen [11, 9] haben wir das Verarbeitungsmodell detailliert und dabei wichtige Merkmale herausgearbeitet, die sich in den betrachteten Architekturtypen und deren Verfeinerungen widerspiegeln. Insgesamt kristallisierten sich folgende generellen Bewertungen heraus:

- Der Page-Server vereinigt eine recht einfache Systemkonzeption (einstufige Verarbeitung, direkte Übernahme des Server-Speicherungsmodells für die Objektverarbeitung in der Workstation) mit einem, abhängig vom Grad der Clusterbildung, guten Leistungsverhalten.
- Der Object-Server scheint seinen Konkurrenten zu unterliegen: Verarbeitungskontexte votieren für den Query-Server, und bei Clusterbildung ist das Anforderungsgranulat Seite beim Page-Server effektiver als die objektweise Anforderung beim Object-Server.
- Der Query-Server zeigt eine hohe Systemkomplexität aufgrund des sehr flexiblen und mächtigen Verarbeitungsmodells (zweistufige Verarbeitung (falls notwendig), flexibles WS-Speicherungsmodell, Nutzung definierter Verarbeitungskontexte).

Diese qualitativen Untersuchungen zur prinzipiellen Eignung von Workstation/Server-Architekturen für ein gegebenes Anwendungsszenario sollten mit quantitativen Analysen kombiniert werden, um eine endgültige Architekturauswahl zu ermöglichen. Hierbei kann auf die existierenden Benchmarks, die in [6] charakterisiert und klassifiziert sind, nur bedingt verwiesen werden. Es fehlen nämlich sowohl der direkte Vergleich von Query-Server-Architekturen zu den anderen beiden Architekturansätzen als auch allgemeingültige Aussagen über die Auswirkungen eines Mehrbenutzerbetriebs (bei gleichartigen und gemischten Anwendungen). Erste vergleichende Untersuchungen zu Page-Server- und Object-Server-Ansatz unter Mehrbenutzerbetrieb finden sich in [5].

Durch eine Pufferung der Verarbeitungskontexte über Commit-Grenzen hinaus kann der aktuelle Verarbeitungskontext (inklusive der zugehörigen Sperren) direkt und ohne extra Ladephase an den nachfolgenden Verarbeitungsschritt weitergegeben werden. Dies scheint insbesondere für den Bereich der Ingenieuranwendungen eine sehr lohnende Sache, da dort oftmals das Ergebnis einer Werkzeuganwendung den Verarbeitungskontext für die darauffolgende Werkzeuganwendung darstellt. Erste Leistungsmessungen [4] hinsichtlich der dann notwendigen Kohärenz-Protokolle sind auf alle Architekturansätze zu erweitern, um vergleichende Bewertungen durchführen zu können.

Kommunikation bleibt bei den hier diskutierten Architekturen ein wichtiges Thema, auch wenn FDDI (100Mb/s) bzw. künftig ATM (Asynchronous Transfer Mode) als Standard-Technologien bei LANs eingesetzt werden. ATM erlaubt die Integration von Multimedia-Daten und soll als Trägersystem für Breitbandnetze in einigen Jahren noch höhere Transferkapazitäten bereitstellen (ATM 1 mit 155Mb/s und ATM 2 mit 622 Mb/s). Diese lassen sich jedoch zunehmend schwieriger ausschöpfen, da die beteiligten CPUs die Kommunikationsdatenraten begrenzen. Bei jeweils 20000 Instr./Transfer verkörpert bei hohen Raten nicht das Übertragungsmedium den begrenzenden Faktor, sondern die CPU-Leistung des Servers. Dieser Sachverhalt macht sich beim Page-Server und besonders beim Object-Server wegen der großen Anzahl der Kommunikationsvorgänge nachteilig bemerkbar. Er dürfte die Skalierbarkeit dieser Architekturen beschränken, da ja neben der Unterbrechungs- und Kommunikationsabwicklung zusätzlich immer noch die eigentliche Ausführung des DBS-Codes anfallen.

Aktuelle Forschungsarbeiten im Bereich integrierter rechnergestützter (Ingenieur-)Anwendungen (insbesondere im Bereich CAD-Frameworks [32, 14]) erweitern das Einsatzgebiet eines DB-Servers auf eine Welt heterogener Datenhaltungssysteme (Dateisysteme und verschiedenartige DBS), welche die bekannten Standardisierungs- und Austauschformate unterstützen müssen und auch eine Erweiterbarkeit hinsichtlich weiterer Datenhaltungssysteme zeigen. Im Lichte dieser zukünftigen Anforderungen erscheint ein Architekturansatz auf der Abstraktionsebene und mit der Flexibilität des Query-Servers angebracht. Aus diesem Grund zielen unsere weiteren Arbeiten auf einen modularen und auf einen Anwendungsfall konfigurierbaren Query-Server. Einen ersten Schritt in diese Richtung stellen unsere praktischen Arbeiten zur Query-Komponente dar: in [35] untersuchen wir die Optimierung von Objektpufferanfragen sowie die Delegation von Teilanfragen an den DB-Server. Entwicklungsziel ist die Integration eines Query-Servers in unser Framework-System [33].

6. Literatur

1. Altmeyer, J., Schürmann, B., Zimmermann, G.: Three-phase chip planning. Proc. Int. Conf. on Computer Aided Design, Santa Clara, Ca., 1992
2. Ananthanarayanan, R., Gottemukkala, V., Käfer, W., Lehman, T.J., Pirahesh, H.: Using the coexistence approach to achieve combined functionality of object-oriented and relational systems. Proc. ACM SIGMOD'93, Washington, DC, May 1993, pp. 109-118
3. Butterworth, P., Otis, A., Stein, J.: The Gemstone object database management system. Comm. of the ACM 34:10, 1991, pp. 64-77
4. Carey, M., Franklin, M., Livny, M., Shekita, E.: Data caching tradeoffs in client-server DBMS architectures. Proc. ACM SIGMOD'91, Denver, June 1991, pp. 12-21
5. Carey, M., Franklin, M., Zaharioudakis, M.: Fine-grained sharing in a page server OODBMS. Proc. ACM SIGMOD'94, Minneapolis, June 1994, pp. 359-370
6. Chaudhri, A., Revell, N.: Benchmarking object databases: past, present & future. UNICOM-Seminar on Object-Oriented Databases: Realising their Potential and Interoperability with RDBMS, London, May, 1994
7. Chen C., Roussopoulos, N.: The implementation and performance evaluation of the ADMS query optimizer: integrating query result caching and matching. Advances in Database Technology - EDBT'94, LNCS 779, pp. 322-336, Springer 1994

8.. DeMichiel, L., Chamberlin, D., Lindsay, B., Agrawal, R., Arya, M.: Polyglot: extensions to relational databases for sharable types and functions in a multi-language environment. Proc. of 9th Int. Conf. on Data Engineering, Vienna, April 1993, pp. 651-660
9. Deppisch, U., Obermeit, V.: Tight database cooperation in a server-workstation environment. Proc. 7th Int. Conf. on Distrib. Computer Systems, Berlin, 1987
10. Deux, O., et al.: The O_2 system. Comm. of the ACM 34:10, 1991, pp. 34-49
11. DeWitt, D.J., Futtersack, P., Maier, D., Velez, F.: A study of three alternative workstation-server architectures for object-oriented database systems. Proc. VLDB'90, Brisbane, Australia, pp. 107-121
12. Deßloch, S., Leick, F.J., Mattos, N.M., Thomas, J.: The KRISYS project: a summary of what we have learned so far. Tagungsband der GI-Fachtagung 'Datenbanksysteme für Büro, Technik und Wissenschaft', Informatik aktuell, S. 124-143, Springer 1993
13. Gray, J., Reuter, A.: Transaction processing - concepts and techniques. San Mateo, Ca., Morgan Kaufmann 1993
14. Harrison, D.S., Newton, A.R., Spickelmier, R.L., Barnes, T.J.: Electronic CAD-frameworks. Proc. of the IEEE 78:2, 1990, pp. 393-417.
15. Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: Processing and transaction concepts for cooperation of engineering workstations and a database server. Data and Knowledge Engineering 3, 1988, pp. 87-107
16. Härder, T., Meyer-Wegener, K.: Transaktionssysteme in Workstation/Server-Umgebungen. Informatik - Forschung und Entwicklung, 1990, S. 127-143
17. Hübel, C., Sutter, B.: DB-Integration von Ingenieuranwendungen - Modelle, Werkzeuge, Kontrolle. Reihe Datenbanksysteme, Vieweg 1993
18. Jablonski, S.: Transaction support for activity management. Proc. Workshop on High Performance Transaction Processing Systems, Asilomar, Ca., Sept. 1993
19. Keller, A., Jensen, R., Agrawal, S.: Persistence software: bridging object-oriented programming and relational database. Proc. ACM SIGMOD'93, Washington, DC, May 1993, pp. 523-528
20. Kemper, A., Kossmann, D.: Adaptable pointer swizzling strategies in object bases. Proc. 9th Int. Conf. on Data Engineering, Vienna, April 1993, pp. 155-162
21. Kim, W.: Introduction to object-oriented databases. Computer System Series, Cambridge, Ma., MIT Press 1991
22. Käfer, W., Mitschang, B.: Flexible Entwurfsdatenverwaltung für CAD-Frameworks: Konzept, Realisierung und Bewertung. Tagungsband der GI-Fachtagung 'Datenbanksysteme für Büro, Technik und Wissenschaft', Informatik aktuell, S. 144-163, Springer 1993
23. Küspert, K., Dadam, P., Günauer, J.: Cooperative object buffer management in the advanced information management prototype. Proc. 13th VLDB Conf., Brighton, Sept. 1987, pp. 483-492
24. Lamb, C., Landis, G., Orenstein, J., Weinreb, D.: The ObjectStore database system. Comm. of the ACM 34:10, 1991, pp. 50-63
25. Lee, B., Wiederhold, G.: Outer joins and filters for instantiating objects from relational databases through views. IEEE Knowledge and Data Engineering 6:1, 1994, pp. 108-119
26. Mitschang, B., Pirahesh, H., Pistor, P., Lindsay, B., Südkamp, S.: SQL/XNF - processing composite objects as abstractions over relational data. Proc. 9th Int. Conf. on Data Engineering, Vienna, April 1993, pp. 272-282
27. Moss, B., Eliot, J.: Working with persistent objects: to swizzle or not to swizzle. IEEE Trans. Software Engineering 18:8, 1992, pp. 657-673

28. Objectivity Inc.: Objectivity database system overview, Menlo Park, Ca., 1990
29. Ontologic Inc.: ONTOS developer’s guide, Billerica, MA, 1991
30. Rahm, E.: Der Database-Sharing-Ansatz zur Realisierung von Hochleistungs-Transaktionssystemen. Informatik-Spektrum 12:2, 1989, S. 65-81
31. Roussopoulos, N. Delis, A.: Modern client-server DBMS architectures. ACM SIGMOD RECORD 20:3, Sept. 1991, pp. 52-61
32. Rammig, F.J., Steinmüller, B.: Frameworks und Entwurfsumgebungen. Informatik-Spektrum 15:1, 1992, S. 33-43
33. Ritter, N., Mitschang, B., Härder, T., Gesmann, M., Schöning, H.: Capturing design dynamics - the CONCORD approach. Proc. 10th Int. Conf. on Data Engineering, Houston, Texas, Feb. 1994, pp. 440-451
34. Stonebraker, M.: Architectures of future data base systems. IEEE Data Engineering 13:4, Dec. 1990
35. Thomas, J., Mitschang, B., Mattos, N.M., Deßloch, S.: Enhancing knowledge processing in client/server environments. Proc. 2nd Int. Conf. on Information and Knowledge Management, Washington, Nov. 1993, pp. 324-334
36. Versant Object Technologies Inc: VERSANT technical overview. Menlo Park, Ca., 1990.
37. Zimmermann, G.: PLAYOUT - a hierarchical layout system. Tagungsband 18. GI Fachtagung, Hamburg, Informatik-Fachberichte 188, S. 31-51, Springer 1988

Danksagung

Wir danken unseren Kollegen S. Deßloch, M. Gesmann, J. Thomas und E. Rahm für die kritische Durchsicht einer ersten Fassung. Weiterhin bedanken wir uns bei den Gutachtern für die wertvollen Hinweise zur Verbesserung der Darstellung.