Effizienzbetrachtungen zu SDAI auf OODBS[1]

Dipl.-Inform. Udo Nink
Universität Kaiserslautern, AG Datenverwaltungssysteme
Postfach 3049, 67653 Kaiserslautern
email: nink@informatik.uni-kl.de

Zusammenfassung

STEP erlaubt die elektronische Verwaltung aller während des gesamten Lebenszyklus eines Produktes anfallenden Daten. Mit SDAI als standardisierter Zugriffsschnittstelle, die von konkreten Datenbanksystemen abstrahiert werden heterogene Frameworks mit einheitlicher Oberfläche für die Entwicklung umfangreichster Anwendungssysteme unterstützt. Mittlerweile existieren Implementierungen von SDAI auf OODBS und RDBS, jedoch ist kaum etwas über deren Effizienz veröffentlicht. In diesem Aufsatz stellen wir unsere Erfahrungen und Ergebnisse durchgeführter Leistungsmessungen mit SDAI auf zwei verschiedenen OODBS vor. Wir grenzen die verwendeten Realisierungen von anderen in der Literatur veröffentlichten ab und beschreiben unsere Sicht auf ein sinnvoll einsetzbares System. Zusammenfassend erscheint uns SDAI eine weitgehend akzeptable Schnittstelle für den CAD-Bereich zu sein. Für die Realisierung von Anwendungen, die eher deskriptives Zugriffsverhalten aufweisen, ist SDAI noch keine gute Wahl.

Abstract

STEP allows for electronic management of complete product life cycles. With SDAI becoming a standardized access interface hiding concrete database technology, heterogenous frameworks with a uniform surface for megaprogramming are supported. By now some implementations of SDAI on OODBMS and RDBMS exist telling little or nothing about their performance though. We describe experiences and results of performance tests we made with SDAI on two different OODBMSs. We compare the mentioned implementations to others found in literature and show our point of view on how such a system should look like. We think SDAI to be largely acceptable for CAD-programming. For the implementation of applications with rather descriptive access patterns SDAI still is no choice.

1) Dieses Papier wurde veröffentlicht in Informatik Xpress 8, ISBN 3-00-000510-2, Tagungsband GI-Fachtagung CAD'96, Verteilte und Intelligente CAD-Systeme, Kaiserslautern, März 1996.

1. Einleitung

Computerunterstützte Anwendungsbereiche wie CAD stellen hohe Anforderungen an die zu verwendende Informationstechnologie. Neben leistungsfähiger Hardware zur Durchführung der eigentlichen Entwurfsarbeit treten in den letzten Jahren vor allem eher software-spezifische Aspekte der Modellierung und Verarbeitung in den Vordergrund, um eine optimale Grundlage für die tägliche Entwurfsarbeit zu liefern. Viele dieser Aspekte wurden und werden auch im Datenbankbereich diskutiert.

- **Repräsentation der Miniwelt**: Es werden geeignete Repräsentationen für die zu verwaltenden Anwendungsdaten benötigt. Typischerweise handelt es sich bei diesen Daten um sehr komplexe, hochgradig vernetzte Strukturen. Man nutzt nun die Ähnlichkeit vieler Objekte aus, um über ein Typkonzept ein Modell der Anwendungswelt (Miniwelt) zu schaffen. Zur Beschreibung eines Anwendungsmodelles steht heute eine Vielfalt von verschiedenen Datenmodellen unterschiedlicher Mächtigkeit zur Verfügung (Relationenmodell, objektorientierte Modelle).
- **Datenaustausch**: Der Aspekt des Datenaustausches nimmt wesentlich auf die anfallenden Kosten bei der Durchführung von Entwurfsprozessen Einfluß. Die hier auftretenden Probleme resultieren aus den unterschiedlichen Daten- und Dateiformaten, Datenbeschreibungsmöglichkeiten (Datenmodelle), sowie konkreten Datenbeschreibungen (Schemata), die jeweils auf Quell- und Zielsystem vorliegen.
- **Verarbeitungskonzept**: Gerade im Entwurfsbereich benötigt man oft Unterstützung für lang andauernde Teilprozesse (lange Transaktionen; Checkout, Work, Checkin). Die lokale Datenverwaltung kann man so von der zentralen entkoppeln, was Performance-Steigerungen und höhere Sicherheit (Fehlerisolation) für die Verarbeitung ermöglicht.
- **konkurrierender Zugriff**: Der Produktentwurf erfordert Teamarbeit. Viele Entwickler benötigen Zugriff auf einen gemeinsamen Datenbestand. Dies erfordert flexible Synchronisationsmaßnahmen. In Verbindung mit langen Transaktionen ist die Versionierung des Datenbestandes sinnvoll (Versionierung ist oft auch explizit für die Modellierung erwünscht). Dadurch lassen sich die bei gängigen Verfahren auftretenden Zugriffskonflikte abschwächen oder sogar lösen.

Erste Standardisierungen für CAD konzentrierten sich entweder auf die Datenmodellierung oder auf den Datenaustausch, also auf nur einen der beiden ersten der oben aufgeführten Aspekte. Für den Datenaustausch beschränkte man sich dabei im wesentlichen auf Daten- und Dateiformate. Heute steht mit dem STEP-Standard (Product Data Representation and Exchange, ISO 10303 (Le93)) ein Werkzeug zur Verfügung, mit dem man diese beiden Aspekte, Datenmodellierung und Datenaustausch, in den Griff bekommen kann.

Mit EXPRESS (ISO11), der objektorientierten Datenbeschreibungssprache des Standards, lassen sich Entitäten (Entities) der Miniwelt “naturgetreu” über ihren gesamten Lebenszyklus hinweg nachbilden. EXPRESS bildet den Grundstein für den angestrebten Datenaustausch, da es durch seinen Formalismus die generische Abbildung von Datenstrukturen von und zu anderen Datenmodellen ermöglicht. Setzt man EXPRESS als globales Datenmodell ein, daß von darunterliegenden Datenmodellen und deren Heterogenität abstrahiert, sprechen wir von STEP-Datenbanken. Zu deren Realisierung kann (und muß) bestehende Datenbanktechnologie eingesetzt werden, was zu STEP-Datenbanksystemen und damit Eigenschaften wie Persistenz, Konsistenz und Konkurrenz führt. Zur Kapselung der eingesetzten DBS ist eine einheitliche Zugriffsschnittstelle notwendig, die STEP mit SDAI (Standard Data Access Interface (ISO22)) in Form einer Anwendungsprogrammierschnittstelle (Application Programming Interface, API) zur Verfügung stellt. SDAI-Programme lassen sich mit geringstem Aufwand auf andere STEP-DBS portieren; STEP-Datenbanken werden austauschbar. Da seit November 1994 Transaktionen in SDAI integriert sind, werden prinzipiell auch die letzten beiden Aspekte der obigen Aufzählung abgedeckt; bislang fehlen allerdings Aussagen zu langen Transaktionen oder zu Versionierung.

Allein aufgrund des großen Einflusses objektorientierter Konzepte auf STEP liegt es nahe, objektorientierte Datenbanksysteme (OODBS) für die Realisierung von STEP-DBS einzusetzen. Während die Umsetzung von EXPRESS auf die Datenmodelle verschiedener OODBS schon gut verstanden wird, muß im Bereich von SDAI noch einiges an Arbeit geleistet werden. Da aber die Sprachmächtigkeit von SDAI größtenteils der von OODBS-APIs entspricht, sollte SDAI auf einem OODBS direkt und effizient realisierbar sein. Tatsächlich ist die Verarbeitung mit SDAI darauf zugeschnitten, Wurzeln von Objektnetzen direkt zu identifizieren und die Objekte innerhalb dieses Netzes über Navigation von Objekt zu Objekt oder durch Iteration auf einer Objektmenge zu erreichen und zu verarbeiten.

Neben der Sprachmächtigkeit ist für die Akzeptanz von SDAI insbesondere auch seine Effizienz sehr wichtig. Es liegt auf der Hand, daß mit einer zusätzlichen Software-Schicht ein Genauigkeitsverlust, was die Problembeschreibung betrifft, sowie ein Leistungsverlust einhergehen; beide Aspekte sind hochgradig abhängig vom betrachteten Anwendungsbereich. Wir wollen feststellen, wie hoch diese Verluste sind und wie sie möglichst gering gehalten werden. Für den ersten Aspekt diskutieren wir dies eher informell über die Darstellung unserer Erfahrungen. Zur Diskussion des zweiten Aspektes betrachten wir SDAI-Implementierungen auf zwei unterschiedlichen OODBS in einer Workstation/Server-Umgebung. Wir verwenden für unsere Messungen den OO7-Benchmark (CDKN94), da er für den Vergleich von OODBS entwickelt wurde und wir STEP-DBS als solche betrachten können. Insbesondere bezieht sich das Verfahren auf den CAD-Bereich. Hier zeigt sich der große Vorteil einer einheitlichen Zugriffsschnittstelle wie SDAI: Der Benchmark muß nur einmal in SDAI kodiert werden (mit minimalen Anpassungen); lediglich das verwendete STEP-DBS wird ausgetauscht.

In Kapitel 2 geben wir einen kurzen Überblick über den STEP-Standard und insbesondere über EXPRESS und SDAI. Anschließend motivieren wir in Kapitel 3 den OO7-Benchmark und beschreiben detailliert seinen Aufbau. Kapitel 4 dokumentiert die von uns durchgeführten Messungen. Wir stellen dabei die betrachteten STEP-DBS vor und erläutern unsere Meßergebnisse. Kapitel 5 enthält eine Zusammenfassung unserer Arbeit.

2. Repräsentation und Austausch von Produktdaten mit STEP

Langfristiges Ziel der intensiven Bemühungen im Bereich der rechnergestützten Produktion zielen auf eine rechnerintegrierte Produktionsumgebung ab, in der sämtliche der heute zu findenden Insellösungen in den Bereichen CAD (Computer Aided Design), CAP (Computer Aided Planning), PPS (Produktionsplanung und -steuerung), usw. integriert werden sollen (LK95). Die Motivation dahinter begründet sich aus dem Wunsch der Hersteller, in einer überaus dynamischen Marktwirtschaft zu überleben. Ein breites Produktspektrum, möglichst unverzögerte Reaktion auf Marktänderungen, hohe Lieferbereitschaft, minimale Lagerhaltung, wirtschaftlicher Entwurf und ebensolche Herstellung von Produkten hoher Qualität sind die wichtigsten Eigenschaften, die ein Betrieb dazu haben muß und deren Unterstützung und Verbesserung durch den Einsatz der Informationstechnologie möglich werden.

Die Lebensdauer von Produktdaten ist aber um vieles größer als die Lebensdauer aktueller Datenbank- oder noch allgemeiner Informationstechnologie. Aufgrund der damit verbundenen sehr teuren Migration der Daten wird die Unabhängigkeit von konkreten Datenbanksystemen bzw. deren Herstellern gefordert. Mit STEP besteht die Möglichkeit, ein neutrales Repository (Le93) aufzubauen, dessen Schnittstelle allein durch EXPRESS und SDAI gegeben ist.

2.1 Die Datenmodellierungssprache EXPRESS

Es werden einfache Datentypen und Konstrukte zum Aufbau komplexer Datentypen zur Verfügung gestellt. Wichtigster Baustein ist das *Entity*, das aus einer Menge von Attributen schon bekannter Datentypen aufgebaut wird. Da damit der Begriff Entity-Typ assoziiert ist, ersetzen wir im folgenden Entity durch Entity-Typ und bezeichnen dessen Ausprägungen als Instanzen. Neben einfachen, benannten, varianten, Enumerations- und Aggregattypen sind

```
ENTITY Assembly
   ABSTRACT SUPERTYPE OF (ONEOF
       (ComplexAssembly, BaseAssembly));
   id:              INTEGER;
   type:            STRING(TypeSize);
   buildDate:       INTEGER;
   INVERSE superAssembly:ComplexAssembly
       FOR subAssemblies;
   myType:          INTEGER;
END_ENTITY;
```

```
ENTITY ComplexAssembly
   SUBTYPE OF (Assembly);
   subAssemblies:SET OF Assembly;
END_ENTITY;
ENTITY BaseAssembly
   SUBTYPE OF (Assembly);
   components:    BAG OF CompositePart;
END_ENTITY;
```

Abb. 1: Ausschnitt aus einem EXPRESS-Schema

auch Entity-Typen als Attributtypen nutzbar. Letztere erlauben die Definition von *Beziehungen*; wir sprechen dann auch von entity-wertigen Attributen. Aggregierte entity-wertige Attribute ermöglichen die Modellierung von (1:n)- und (n:m)-Beziehungen, die zusätzlich über Kardinalitätsrestriktionen verfeinert werden können. Eine Menge von Typdefinitionen läßt sich modular zu einem *Schema* zusammenfassen. In Abbildung 1 sind der Entity-Typ "Assembly” und seine Subtypen dargestellt, die logische Abstraktionen (Datenmanipulator, ALU usw.) im Entwurfsbereich (hier VLSI) in einer Hierarchie verwalten sollen. Assembly dient als abstrakter Entity-Typ, d.h. hat selbst keine Instanzen. ComplexAssemblies setzen sich aus weiteren Assemblies zusammen (“subAssemblies”); BaseAssemblies bilden die Blätter des entstehenden Baumes und fassen Instanzen von CompositePart (nicht dargestellt) zusammen. Das Attribut “superAssembly” stellt (für beide Subtypen) eine Beziehung zum Vater in der Hierarchie her. Durch das Schlüsselwort INVERSE (bei “superAssembly”) wird eine bestehende Beziehung (“FOR subAssemblies”) weiter eingeschränkt, wodurch referentielle Integrität gewährleistet werden kann.

Da für Entity-Typen keine Methoden definiert werden können, ist EXPRESS nicht voll objektorientiert, sondern “nur” strukturell objektorientiert. Zur Verfeinerung von Entity-Typen ist (multiple) *Vererbung* einsetzbar. Eine Besonderheit ist hier die Einschränkbarkeit von Vererbungsbeziehungen über die SUPERTYPE-Klausel. Dort kann man festlegen, welche Kombinationen von Subtypen vorkommen dürfen. In der Abbildung gibt das Schlüsselwort “ONEOF” an, daß ein Assembly entweder ein ComplexAssembly oder ein BaseAssembly ist. Für unsere Messungen ist eine weitere Vertiefung ohne Relevanz.

2.2 Die Datenzugriffsschnittstelle SDAI

Ein weiterer wichtiger Punkt für die Unterstützung des Produktdatenaustausches ist neben einem einheitlichen Datenmodell eine einheitliche Zugriffsschnittstelle, die vom darunterliegenden DBS abstrahiert. Während EXPRESS das Datenmodell in STEP festlegt, beschreibt SDAI diese Schnittstelle. Für ihre Spracheinbettung werden verschiedene Einbettungsvorschriften (Language Bindings) standardisiert. Anbindungen an Sprachen wie C oder Fortran weisen aufgrund fehlender objektorientierter Konzepte einen deutlichen Reibungsverlust auf. Die beste Unterstützung für SDAI bietet in unseren Augen die Anbindung an C^{++}, auf die wir uns hier beschränken. Es wird zwischen Early- und Late-Binding unterschieden, was nicht mit den Begriffen frühes und spätes Binden in objektorientierten Programmiersprachen verwechselt werden sollte. Im Late-Binding sind alle SDAI-Operationen, die dem Programmierer zur Verfügung stehen, schemaunabhängig spezifiziert; alle Hinweise auf beteiligte schemaabhängige Strukturen erhält eine solche Operation beispw. durch Parameter vom Typ Zeichenkette. Das Early-Binding hingegen ist sehr viel spezieller; es stellt nur eine Teilmenge der Operationen zur Verfügung; die bei einer Operation beteiligten schemaabhängigen Strukturen bestimmen weitgehend die Namen der Operationen. So gibt es z.B. für das Attribut “id” einer Instanz des Entity-Typs Assembly u.a. folgende Methoden zum Auslesen: “anAssembly->id()” und “anAssembly->GetAttr(“id”)”. Die zweite Methode, die nur im Late-Binding definiert ist, erlaubt im Gegensatz zur ersten auch den Zugriff auf erst zur Laufzeit bekannte Attribute (anstelle der Zeichenkettenkonstanten darf auch eine Variable als Parameter verwendet werden).

SDAI führt nun folgende Abstraktionen für Datengranulate und -operationen ein. Instanzen werden in gröberen Datenbehältern, *Models* genannt, gesammelt (Abbildung 2). Ein Model ist einem *Schema* zugeordnet, das Metainformationen für Instanzen bereitstellt. Eine Menge von

Models, die sich auf das gleiche Schema beziehen, kann man logisch zu einer *Schema Instance* zusammengefassen, die als Gültigkeitsbereich von Beziehungen und Regeln verwendet wird. Ein *Repository* beinhaltet eine Menge von Models und die Schemata, auf die sie sich beziehen. Auf *SDAI-Queries* (erlauben assoziative Extraktion einer Teilmenge eines Aggregats) konnten wir nicht eingehen, da sie in den betrachteten Systemen nicht realisiert waren.

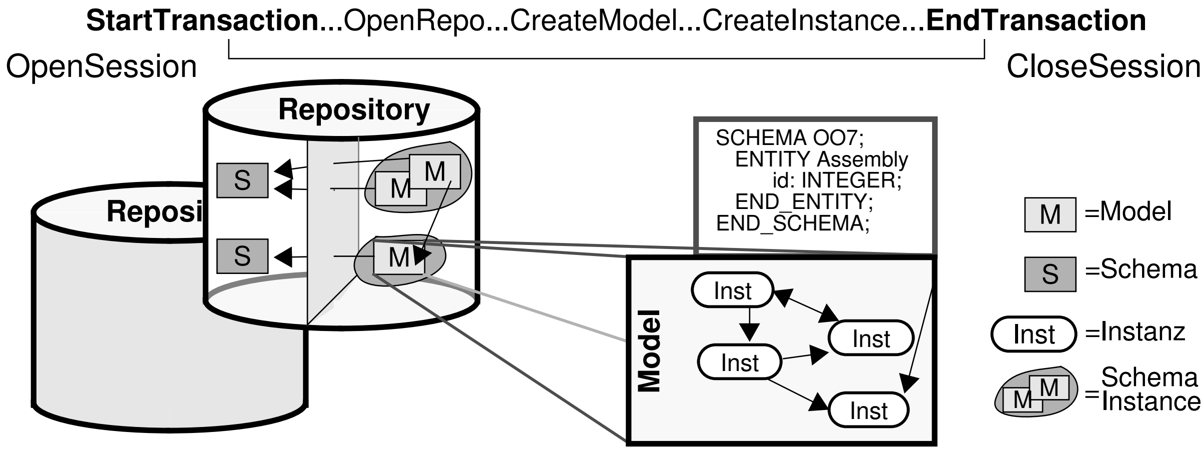

Abb. 2: Datenorganisation und -verarbeitung in SDAI

Nach dieser Einführung in die Modellierungssprache EXPRESS und die zugehörige Zugriffsschnittstelle SDAI gehen wir nun zur Diskussion der Leistungsanalyse von SDAI über. Im darauf anschließenden Kapitel stellen wir die Ergebnisse unserer Messungen vor.

3. Leistungsanalyse mit Benchmarks

Ist die Leistung eines (Kern-)Systems schwach, so gilt dies meist auch für darauf aufgesetzte Software-Schichten[2]. Dies ist ein wesentlicher Grund für die Notwendigkeit von Leistungsmessungen. Benutzer erhalten zusätzliche Argumente für den Vergleich von und die Auswahl zwischen verschiedenen Systemen und Hersteller wertvolle Hinweise auf Schwachstellen ihres Produktes (Gr91).

Leistungsmessungen werden in verschiedenen Varianten betrieben. Da der jeweils betrachtete Anwendungsbereich, wie z.B. CAD, wesentlichen Einfluß auf die Effizienz eines Systems hat, sollte dies auch im verwendeten Meßverfahren zum Ausdruck kommen. Insbesondere die Messung realer Anwendungen oder Anwendungssysteme, die auf dem zu untersuchenden System laufen sollen, liefert die zuverlässigsten Aussagen für dessen späteren Einsatz. Dies kann system- oder einsatzbezogen geschehen. Während bei systembezogenen Messungen die Leistung der einzelnen Komponenten oder auch Funktionen eines Anwendungssystems im Vordergrund stehen, zielen einsatzbezogene Messungen darauf ab, auch z.B. das zeitliche Verhalten der Endbenutzer (Denkzeitverhalten (Di94)) im täglichen Betrieb zu be-

2) Durch Einführung höherer Abstraktionsebenen, wie beim Übergang von einer satzorientierten zu einer mengenorientierten Schnittstelle für ein DBS läßt sich die Leistung zumindest für bestimmte Operationen (eben solche, die wesentlich mehr als ein Objekt bearbeiten) erhöhen.

rücksichtigen, um dem tatsächlichen Einsatz möglichst nahe zu kommen. Beispw. wird ein Entwerfer bei längeren Berechnungen seines CAD-Tools parallel andere Arbeiten erledigen wollen und damit die für das CAD-Tool zur Verfügung stehende Systemleistung wesentlich beeinflussen. Der große Nachteil solcher Vorgehensweisen sind die enormen Kosten (Zeit, Geld, Implementierungen), die bei ihrer Durchführung entstehen.

Aus diesem Grund zieht man oft Testverfahren (Benchmarks) vor, die mehr oder weniger abstrakte Last- und DB-Modelle bereitstellen und deshalb auch weniger aussagekräftig sind. Dennoch liefern sie zumindest Indizien für die Leistungsfähigkeit oder mögliche Schwachstellen eines Systems. Wir unterscheiden hier zwischen Verfahren, die reale Anwendungen simulieren, und solchen, die ein von konkreten Anwendungen unabhängiges, abstrakteres Modell (oft nur für einen bestimmten Anwendungsbereich) bereitstellen. Zu ersteren kann man z.B. den Debit/Credit-Benchmark (Gr91) zählen. Da wir keine bestimmte Anwendung ins Auge gefaßt haben, wählen wir die zuletzt genannte Meßmethode. Insbesondere für den Anwendungsbereich CAD und den Einsatz von OODBS ist der **OO7-Benchmark** (CDKN94) für uns der geeignetste Kandidat.

3.1 Der OO7-Benchmark

Speziell für den Vergleich und die Bewertung verschiedener OODBS entwickelt, sind die Messungen darauf ausgelegt, verschiedene Traversierungen (mit kaltem oder heißem Cache, weit gestreut, dicht), Aktualisierungen (auf indizierten und nicht indizierten Attributen, wie-

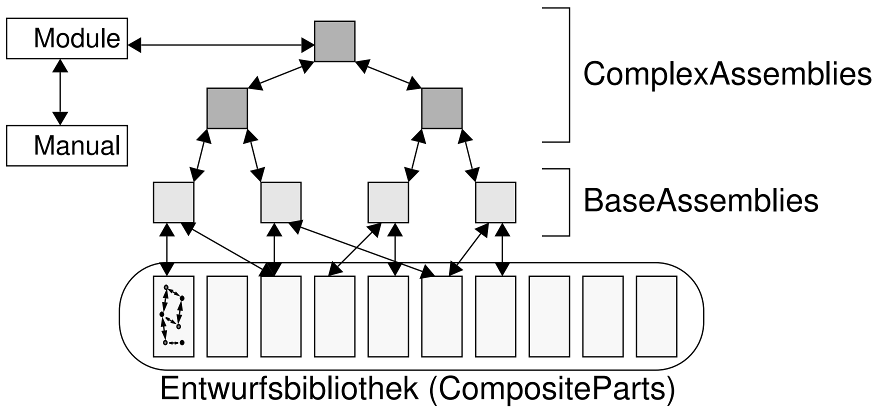

Abb. 3: Vereinfachte Struktur der Ausprägungen einer OO7-Datenbank

derholte Änderungen) und Anfragen (über Anfragesprache oder programmiert) zu untersuchen. Insgesamt werden 20 Operationen vorgegeben. Durch Variation verschiedener Parameter (siehe Tabelle 1) sind 105 Messungen durchzuführen. Für verschiedene OODBS ist der gesamte Quellcode elektronisch und frei verfügbar.

Das vorgegebene Schema bildet eine Art Entwurfsbibliothek nach und beschreibt verschiedene Entity-Typen, die aus einfachen Attributen aufgebaut sind (id, buildDate, type usw.) und Beziehungen zu anderen Entity-Typen haben. Zunächst ist da der Entity-Typ **Module** (etwa ein Chip im VLSI-Entwurf; siehe Abbildung 3). Ein Module besitzt bidirektionale Beziehungen zu einem **Manual**, einem größeren Textobjekt, das als Beschreibung des Modules zu verstehen ist, und zu der Wurzel einer **Assembly**-Hierarchie.

Assemblies werden unterschieden in Complex- und BaseAssemblies. Ein **ComplexAssembly** aggregiert drei weitere Assemblies (wie einige andere Details ist auch dieses in der Abbildung der Übersichtlichkeit wegen vereinfacht dargestellt). **BaseAssemblies** bilden die Blätter des so entstehenden Baumes und aggregieren drei Instanzen aus einer großen Menge von **CompositeParts**. BaseAssemblies dürfen gleiche CompositeParts verwenden; sie dürfen sich also überlappen. Ein CompositePart (z.B. ein Registersatz) faßt eine Menge von miteinander verbundenen **AtomicParts** (etwa einzelne Register) zusammen, womit eine Form der Lokalität durch komplexe Objekte eingeführt wird, und steht in Beziehung zu einem **Document**. Es ist hilfreich, sich unter der Menge aller CompositeParts eine für den Entwurf zur Verfügung stehende Entwurfsbibliothek vorzustellen. Pro CompositePart ist ein AtomicPart als Wurzel ausgezeichnet. AtomicParts sind über **Connection**-Objekte mit weiteren AtomicParts verbunden, damit werden attributierte Beziehungen nachgebildet. Um zu garantieren, daß alle AtomicParts eines CompositeParts miteinander verbunden sind, werden sie bei ihrer Erstellung in einem Ring aufgebaut; anschließend werden zufällig weitere Verbindungen erzeugt.

Tabelle 1: Datenbankparameter für den OO7-Benchmark

Parameter	klein	mittel	groß
Dokumentgröße	2 KB	20 KB	20 KB
Größe eines Manuals	100 KB	1 MB	1 MB
Höhe des Assembly-Baumes	7	7	7
# AtomicParts pro CompositePart	20	200	200
# CompositePart pro Module	500	500	500
# Assemblies pro Assembly	3	3	3
# CompositeParts pro BaseAssembly	3	3	3
# Modules	1	1	10
# Connections pro AtomicPart (FanOut)	3, 6, 9	3, 6, 9	3, 6, 9

Benchmark-Parameter: Die Größe der DB wird durch eine Reihe von Parametern festgelegt (siehe Tabelle 1). Grundsätzlich werden drei verschiedene Größenordnungen (klein, mittel und groß) unterschieden, die jeweils durch einen Satz von Parametern bestimmt sind, die Größen und Anzahlen von Instanzen angeben. Der letzte Eintrag soll für jede Größenordnung variiert werden; wir werden diesen Wert im folgenden mit “FanOut” ansprechen.

Die einzelnen Operationen, die für das Verfahren vorgegeben sind und deren Art schon weiter vorne geschildert wurde, werden wir nicht alle detailliert vorstellen. Lediglich die für die im anschließenden Kapitel erläuterten Messungen betrachteten Operationen beschreiben wir dort genauer. Die Klassendefinitionen des OO7 wurden möglichst eins zu eins in EXPRESS nachgebildet. Dies ist aufgrund der hohen Ähnlichkeit der Datenmodelle von C++ und EXPRESS relativ einfach. Unterschiede liegen darin begründet, daß die vorgegebenen Methoden (swapXY etwa) nicht direkt in EXPRESS modelliert werden können, sondern nachträglich in der Zielsprache implementiert werden.

4. Messungen

In (HL95) werden drei Architekturen für die Realisierung von SDAI auf objektorientierten und relationalen Systemen unterschieden: **Upload/Download**, **Cached** und **Direct**. Ersteres bezeichnet Realisierungen, die über Export-Routinen im voraus festgelegte Datenmengen aus einer DB in eine Datei (nach STEP Part 21) herausschreiben, welche dann über eine Hauptspeicherimplementierung (SDAI-Working-Form) geladen und bearbeitet wird. Die Verarbeitung schließt mit dem Zurückschreiben in die Datei und das Importieren in die DB. Die zwei-

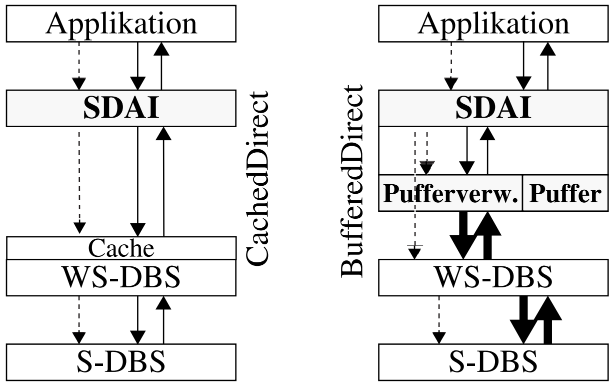

Abb. 4: Architektur von SDAI-Schnittstellen auf Workstation/Server-DBS

te Architektur (Cached) vermeidet den Zwischenschritt über die Datei und lädt bzw. entlädt den Working-Form-Puffer direkt aus der bzw. in die DB. Die letzte Architektur, Direct, setzt direkt auf der Schnittstelle des DBS auf und vermeidet vollständig die Zwischenspeicherung der Daten (es sei denn, das DBS darunter tut dies unsichtbar). Aus unserer Sicht sind die beiden ersten Architekturen, die einander von der Vorgehensweise sehr ähneln, bestenfalls Übergangslösungen, da sie jeder Form von Mehrbenutzerbetrieb entgegenwirken. Nur der letzte der drei Ansätze ist auf lange Sicht sinnvoll. Von den vier konkreten Systemen, die in (HL95) vorgestellt werden, genügt eines dem letzten Ansatz, bietet aber nur “SDAI-like operations” an. Veröffentlichte Ergebnisse von Leistungsmessungen existieren unseres Wissens zu diesen und für in (KNS94) und (LK95) beschriebene Systeme keine.

Wir verfeinern Direct weiter in **CachedDirect** und **BufferedDirect** (siehe Abbildung 4). Die erste Form ist insbesondere geeignet für die Implementierung auf satzorientierten OODBS mit impliziten Caching-Mechanismen (das System sorgt hier für die Lieferung der vom Programmierer besuchten Objekte), während die zweite eher mit anfragebasierten Systemen (Ni95) harmoniert, die solche Mechanismen nicht anbieten. Bei letzterem ist demnach ein explizit anwendbarer Puffer Teil der SDAI-Implementierung.

Einschränkungen bei den Messungen:

• Die Überprüfung von Regeln, die in EXPRESS definiert werden können, haben wir bei den Messungen nicht berücksichtigt. Sowohl die von uns betrachteten Systeme als auch der OO7 stellen hierfür keine Unterstützung bereit.

• Weiter haben wir lediglich Einbenutzerbetrieb gemessen, da der OO7 zunächst für Mehrbenutzerbetrieb weiterentwickelt werden muß und auch wird (CDKN94). Da der STEP-Standard gerade in großen Anwendungssystemen (z.B. Entwurfssystemen mit einer Vielzahl von kooperierenden Entwerfern; concurrent engineering) zum Einsatz kommen soll, ist es unerläßlich sich in Zukunft mit dieser Problematik zu befassen.

• Zudem wird mit OO7 lediglich die Ablaufgeschwindigkeit eines Systems gemessen, nicht jedoch Einfachheit der Programmentwicklung und -pflege oder Aufwand für einwandfreien Betrieb (Archivierung, Recovery, Systemverwaltung (CR94)).

• Bislang haben wir lediglich CachedDirect-Realisierungen betrachtet.

4.1 Betrachtete Systeme

Die OODBS der betrachteten Realisierungen sind Pageserver (DFMV90), (HMNR95). Diese erhalten ihre Bezeichnung durch das Granulat, das zwecks Datenaustausch zwischen Server und Workstation, zur Speicherung in workstation-seitigen Caches und für mögliche Rückrufe des Servers (Callbacks) herangezogen wird. Page-Server verwenden dazu Seiten oder Seitenmengen ohne Kenntnis der eigentlich geforderten Objekte, wogegen Object- oder Query-Server als weitere bekannte Alternativen einzelne Objekte bzw. dynamisch zusammengestellte Objektmengen versenden. Da ein Pageserver lediglich für das Liefern der exakt bestimmten Daten und für die Verwaltung von Sperr- und Log-Informationen zuständig ist (data shipping approach), ist er sehr gut skalierbar. Die eigentliche Arbeit, die eine Anwendung verrichtet, geschieht auf der Workstation. Zur besseren Darstellung vergeben wir Namen für die betrachteten Systeme:

• **Oodbs1** und **Oodbs2** sind kommerzielle Pageserver mit C^{++}-Interface.

• **SdaiProto** ist eine Prototyprealisierung (He94) von SDAI (Early- und Late-Binding) auf dem Page-Server Oodbs1 basierend auf dem fedex-plus Parser, der EXPRESS nach C^{++} umsetzt. Der generierte Kode wird an Oodbs1 angepaßt. SDAI-Transaktionen werden auf das Transaktionskonzept von Oodbs1 abgebildet.

• **SdaiKomm**: Dieses kommerzielle SDAI-System stellt das Early-Binding auf dem Page-Server Oodbs2 zur Verfügung. Hier wird ebenfalls das Transaktionskonzept des darunterliegenden OODBS eingesetzt.

Für die Messungen wurden zwei über ein Ethernet-LAN (10 MBit/s Übertragungsrate) gekoppelte Sun-Rechner eingesetzt. Die Server der OODBS liefen auf Sun SparcStation 10/402 mit 128 MB Hauptspeicher, 4 GB Plattenkapazität und 550 MB Swapspace unter Solaris 2.2. Als Workstation-Maschine diente eine Sun Sparc ELC 4-25 mit 64 MB Hauptspeicher und 230 MB Swapspace unter SunOS 4.1.3.

4.2 Messungen und Folgerungen zum Prototyp

Zunächst ist zu bemerken, daß bei SdaiProto das Volumen einer DB um einen akzeptablen Faktor 1,7 größer ist. Für die kleine DB wurden Durchläufe für kalte und heiße Client-Caches durchgeführt, für die mittlere DB liefern wir nur Zahlen für kalte Durchläufe: heiße Durchläufe der Queries auf der mittleren DB liefern keine neuen Informationen über die SDAI-Operationen, da die gesuchten Objekte meist komplett in den Cache passen; bei den Traversals ist dagegen die Menge der Objekte oft viel zu groß für den Cache, so daß die Kommunikation zwischen Server und Workstation den Löwenanteil eines Meßwertes ausmacht und wir ebenfalls keine neuen Informationen über die SDAI-Operationen erhalten.

In der nachfolgenden Tabelle 2 stellen wir die Meßergebnisse für SdaiProto relativ zu Oodbs1 dar. Diese relativen Werte sind Quotienten der Meßergebnisse also Verlustfaktoren. Ein Verlustfaktor von fünf bedeutet, daß das Meßergebnis für SdaiProto fünf mal so groß ist wie das Ergebnis für Oodbs1. Aus Platzgründen stellen wir nicht alle unsere Meßergebnisse vor, sondern interpretieren einige ausgewählte. Die Spalte "Messung" in Tabelle 2 bezeichnet die einzelnen Meßvorgänge (Operation, DB, Durchlaufart). Die meisten Felder der Spalte "SdaiProto" enthalten drei Meßwerte, die sich von links nach rechts auf FanOut 3, 6 und 9 beziehen. Liefert die Berücksichtigung des FanOuts keine Information, so ist nur der gemittelte Wert eingetragen.

Tabelle 2: Meßergebnisse zu SdaiProto

Messung	SdaiProto			Messung	SdaiProto		
T1, klein, kalt	31,9	22,2	16,3	T9, klein, kalt		5,2	
T1, klein, heiß	120,0	92,9	70,3	T9, klein, heiß		41,1	
T1, mittel, kalt	9,1	5,4	4,7	T9, mittel, kalt		16,6	
T2B, klein, kalt	22,7	16,2	12,9	Q1, klein, kalt	13,0	18,9	17,8
T2B, klein, heiß				Q1, klein, heiß	815,1	783,0	899,1
T2B, mittel, kalt	16,2	9,2	8,0	Q1, mittel, kalt	151,7	196,9	199,3
T6, klein, kalt	2,5	2,3	2,0	Q3, klein, kalt	2,0	2,0	1,6
T6, klein, heiß	22,0	22,8	22,6	Q3, klein, heiß	3,3	3,0	3,0
T6, mittel, kalt	2,4	2,2	2,0	Q3, mittel, kalt	3,3	2,8	2,5
T8, klein, kalt		1,8		Q5, klein, kalt	1,4	1,9	1,5
T8, klein, heiß		2,0		Q5, klein, heiß	146,3	217,8	217,9
T8, mittel, kalt		1,4		Q5, mittel, kalt	1,6	1,5	1,4

Die Operation T1 (**Traversal T1**) dient zur Ermittlung der reinen Traversierungsgeschwindigkeit. Rekursiv werden der Assembly-Baum durchwandert und in allen CompositeParts eine Tiefensuche so durchgeführt, daß alle AtomicParts besucht werden (Zykel werden durch eine Markierungsliste vermieden).

Es wurden sehr hohe Verlustfaktoren ermittelt, was zunächst darauf schließen läßt, daß Navigation sehr ineffizient implementiert ist. Vermutlich liegt dies daran, daß hier der Realisierung mit Oodbs1, wo DB-Pointer genauso wie Hauptspeicher-Pointer gehandhabt werden können, ein teurer Prozeduraufruf gegenübersteht. Gerade die besonders hohen "heißen" Wer-

te untermauern dies. Versuche, diese Prozeduren als “inline” zu verwenden (entsprechen in etwa vom Compiler unterstützten Makros), scheiterten jedoch aus (compiler-)technischen Gründen. Sicher sind dadurch die Werte wesentlich zu verbessern. Weiter läßt sich beobachten, daß sehr häufig temporäre Objekte angelegt werden (z.B. für SdaiBoolean). Dies impliziert dynamische (und teure) Speicherplatzanforderung und -freigabe. Das Late-Binding ist nur unwesentlich langsamer (20 - 30%), was zunächst ein überraschendes Ergebnis darstellt. Verständlich wird dies, wenn man berücksichtigt, daß der Abarbeitungsaufwand für einen Navigationsschritt absolut zwar deutlich höher wird, aber relativ (zur ohnehin schon teuren Operation) eben nur geringfügig. Mit höherem FanOut sinkt übrigens der Verlustfaktor, was wir folgendermaßen erklären: Die Zahl der zu betrachtenden Objekte steigt (mehr Connections), aber die Zahl der durchzuführenden Navigationsoperationen nimmt relativ dazu nur wenig zu, da bei der Tiefensuche AtomicParts nicht mehrfach besucht werden.
Die besseren Werte für die mittlere DB täuschen ein wenig und bestätigen bei genauer Betrachtung nur obige Vermutungen: Der Cache kann nicht alle Daten fassen, was zu Verdrängung und verstärkter Kommunikation mit dem Server führt, die auf allen betrachteten Systemen ähnlich hohe Kosten verursacht. Da diese Kosten sehr viel höher sind als die der auf der Workstation verrichteten Arbeit, werden die Ergebnisse einander angeglichen. Da aber gerade im CAD-Bereich beobachtet werden kann, daß der Löwenanteil der Arbeit lokal auf Workstations verrichtet wird, ist es wichtig, sehr gute “heiße” Zeiten zu liefern.

Die Operation T2 (**Traversal T2**) führt zusätzlich während der Navigation Update-Operationen aus (T2A: auf dem Wurzel-AtomicPart; T2B: auf allen AtomicParts; T2C: auf allen AtomicParts viermal). Hiermit sollen Rückschlüsse auf Logging und Aktualisierung der Daten (inkl. Indizes) ermöglicht werden.
Die Unterscheidung der drei Varianten für T2 liefert kaum Neuigkeiten, deshalb ist in der Tabelle nur T2B dargestellt. Der Übergang von Variante A zu B führt zu einer größeren Erhöhung der Laufzeiten als der Übergang von B zu C, welche aber jeweils gering ist. Ein größerer Unterschied besteht da schon zu T1, d.h., wichtiger als die Anzahl der Update-Operationen ist, ob überhaupt Aktualisierung erfolgt oder nur gelesen wird. Die Verbesserung der Verlustfaktoren gegenüber T1 erklären wir wie folgt: SdaiProto reagiert aus schon bekannten Gründen auf das Hinzunehmen von Update-Operationen sehr viel weniger sensibel als Oodbs1; der Overhead der SDAI-Operationen verbirgt andere Einflußfaktoren. Daraus folgt, daß der Prototyp unbedingt für Navigation und wie wir später noch sehen werden Iteration und einfachen Attributzugriff optimiert werden muß. Der Einfluß anderer Faktoren wie Cluster-Bildung kann erst dann gerecht beurteilt werden.

Die Operation T6 (**Traversal T6**) traversiert den Assembly-Baum, besucht sämtliche CompositeParts und die von dort erreichbaren Wurzel-AtomicParts auf. Es wird somit geringe Lokalität erzielt, so daß nur wenige Cache-Treffer zu erwarten sind.
Bei den kalten Durchläufen werden geradezu gute Werte erreicht, hier macht sich erneut der Einfluß der Kommunikation mit dem Server bemerkbar. Der heiße Durchlauf erhöht die Lokalität, so daß die Ineffizienz der Schnittstelle wieder zu Tage tritt.

Die Operation T8 (**Traversal T8**) durchsucht das Manual und zählt, wie oft der Buchstabe "I" vorkommt, während **Traversal T9** die Gleichheit des ersten und letzten Buchstabens überprüft. Dieser Test untersucht die Verarbeitung großer unstrukturierter Datenfelder.
Die Verlustfaktoren sind bei T8 sehr gut, da zum einen auf der Workstation wenig getan werden muß (also überwiegt die Kommunikation) und zum anderen im wesentlichen nur ein Attributzugriff über SDAI und die eigentliche Verarbeitung des Attributes über die in C und C^{++} üblichen "char*" geschieht. Bei T9 ist der "heiße" Verlustfaktor sehr schlecht (40). Da die Kommunikation hier weitgehend entfällt und auch kein Durchsuchen der Zeichenkette erfolgt, sondern gezielter Zugriff auf den ersten und auf den letzten Buchstaben, schlägt der Aufwand für den Zugriff über SDAI wieder voll zu Buche.

Die Operation Q1 (**Query Q1**) sucht AtomicParts zu zehn zufällig erzeugten Identifikatoren (id). Optimale Laufzeiten lassen sich durch Indizes erreichen.
Die Verlustfaktoren sprechen ein klares Wort für die Notwendigkeit einer Anfrageschnittstelle. Mittlerweile ist eine solche (in Form von logischen Ausdrücken auf Aggregaten) in SDAI aufgenommen worden, der untersuchte Prototyp verfügt jedoch nicht über dieses Feature. Die relativ geringen Faktoren beim kalten Durchlauf über die kleine DB spiegeln nur den Einfluß der Kommunikation wieder, denn auch Oodbs1 erzwingt zumindest die Übertragung der erforderlichen Indexseiten, die es verwendet.

Die Operation Q3 (**Query Q3**) wählt einen Bereich für das Datum, so daß sich 10% AtomicParts qualifizieren. Während Oodbs1 Indizes verwendet, durchläuft SdaiProto den gesamten Extent der AtomicParts (Scan) und prüft jeweils das Datum.
Q3 ist insofern interessant, als daß deutlich wird, daß die Verwendung von Indizes auch nachteilig sein kann, bzw. sich hier so äußert, daß für das SDAI-System akzeptable Faktoren erzielt werden. für OODBS gilt allgemein, daß der Einsatz von Indizes erst ab etwa einem Selektivitätsfaktor von 10% lohnt. Qualifizieren sich mehr Objekte, ist ein Scan meist günstiger.

Die Operation Q5 (**Query Q5**) durchsucht alle BaseAssemblies und vergleicht deren Datum mit dem Datum des jeweils zugehörigen CompositeParts. Sowohl SdaiProto als auch Oodbs1 implementieren diese Operation über einen ausprogrammierten Nested Loop Join, d.h., führen einen verschachtelten Scan auf den Extents beider Entity-Typen aus.
Kostet bei den kalten Durchläufen die Lieferung der benötigten Objekte (oder Seiten) soviel, daß die eigentliche Verarbeitung in den Hintergrund tritt, veranschaulichen die Zahlen bei heißem Cache den Verlust bei Iteration über SDAI-Aggregate und den Attributzugriff über SDAI. Da die Iteration selbst Attributzugriff über SDAI verwendet, erreicht man mit der Optimierung des Attributzugriffs auch gleich die Optimierung der Iteration.

4.3 Messungen und Folgerungen zum kommerziellen SDAI-System

Aufgrund technischer Probleme mit der Durchführung der Messungen für SdaiKomm, entschieden wir uns für eine Vereinfachung des OO7-Benchmarks, um kurzfristig zumindest tendenzielle Aussagen zu erhalten. Dazu haben wir das Schema auf die zwei Entity-Typen AtomicPart und Connection reduziert. Zur Steuerung der Größe der DB verwenden wir die Anzahl aller AtomicParts (siehe Spalte “Messung” in Tabelle 3) und wie bisher den FanOut (hier jedoch: 1, 2, 3, 4, 6 und 9). Fehlende Einträge weisen auf nicht erfolgreiche Messungen hin. Bei der Erzeugung der DB wird ein Zykel von AtomicParts analog zur Vorgehensweise innerhalb eines CompositeParts beim OO7-Benchmark angelegt. Das Volumen einer DB vergrößert sich bei SdaiKomm um einen Faktor von 2.

Tabelle 3: Meßergebnisse zu SdaiKomm

Messung	SdaiKomm						Messung	SdaiKomm					
Txr, 1500, kalt	1,4	1,5	1,7	1,8		10	Txi, 1500, kalt	1,2	1,2	1,9	1,5	2,8	
Txr, 1500, heiß	3,2	4,8	2,5		6,3	8,9	Txi, 1500, heiß	2,7	3,2	5,8	2,9	4,6	
Txr, 3000, kalt		2,2	4,1	6,1			Txi, 3000, kalt	1,1	2,3	3,0	4,6		
Txr, 3000, heiß	0,8	0,4	1,0	0,5			Txi, 3000, heiß	2,5	3,4	3,7	3,9		
Txr, 5000, kalt		3,2	4,0	4,9			Txi, 5000, kalt		3,1		1,5		
Txr, 5000, heiß		0,6	0,7	0,3			Txi, 5000, heiß		5,1		4,5		

Traversal Txr entspricht der (rekursiven) Tiefensuche innerhalb CompositePart der Operation T1 des OO7-Benchmarks. Verglichen mit den heißen Durchläufen von T1 mit SdaiProto auf der kleinen DB, die ebenfalls vollständig in den Cache paßt, sind die Werte von SdaiKomm um ein bis zwei Größenordnungen kleiner. Die Faktoren wachsen jedoch mit dem FanOut; allgemein hatten wir Probleme mit Oodbs1 bei hohem FanOut. Interessanterweise ist SdaiKomm bei heißen Durchläufen oft bis zu 2 mal schneller als Oodbs2 (siehe Zeilen 4 und 6); möglicherweise ist bei Oodbs2 der Compiler nicht in der Lage, die Rekursion aufzulösen. Alternativ haben wir **Traversal Txi** programmiert, das den Zykel von AtomicParts iterativ abwandert. Die Werte sind weit weniger gestreut und liegen in der gleichen Größenordnung wie bei Txr. Auch hier läßt sich ein Anstieg der Verlustfaktoren bei steigendem FanOut feststellen. Zusammenfassend bestärken uns diese Ergebnisse in unserer Auffassung, daß SDAI effizient umgesetzt werden kann, zeigen jedoch auch, daß zur Unterstützung von Anwendungen mit hohem algorithmischem Aufwand weitere Optimierungen notwendig sind.

5. Zusammenfassende Bewertung und Ausblick

Die Umsetzung des OO7 in SDAI gestaltete sich recht einfach, was prinzipiell ein Indiz für die Eignung der Schnittstelle zur Programmierung von CAD-Anwendungen (und anderen) ist. Für das rein navigierende SDAI hat sich gezeigt, daß man bei der Realisierung einer SDAI-Schnittstelle vor allem Iteration, Navigation (Objektzugriff) und Attributzugriff optimal umsetzen muß. Optimierungen an anderer Stelle wirken sich so noch stärker aus. Experimente mit Cluster-Bildung beispw. haben bei unseren Messungen Verbesserungen der Laufzeiten bis zu einem Faktor drei ergeben; einige wenige Meßergebnisse von SdaiProto lagen sogar unter denen von Oodbs1. Eine optimale Umsetzung der SDAI-Operationen verspricht hier noch größere Vorteile. Zu beachten ist, daß bei vielen der Messungen, die Kommunikation mit dem Server der ausschlaggebende Faktor ist. Das bedeutet, daß der Prototyp für Anwendungen mit einem geeigneten Profil auch ohne Optimierung vernünftig eingesetzt werden kann (Ni95). Die relativ zum Early-Binding sehr guten Zahlen des Late-Bindings kommen aus zwei Gründen zustande. Zum einen ist das Late-Binding letztendlich doch schemaabhängig, da die Anwendung, die es verwendet, mit den entsprechenden Schemainformationen gebunden werden muß. Der Wechsel auf eine andere DB mit einem anderen Schema erzwingt das Neubinden der Anwendung für diese DB. Im Gegenzug kann man den Zugriff direkter und damit effizienter gestalten. Zum anderen bewirkt das Late-Binding in der gegenwärtigen Umsetzung des Prototyps zwar eine deutliche absolute Pfadverlängerung beim Zugriff, die jedoch relativ zur schon bestehenden Pfadlänge eher gering ist. Dennoch vermuten wir, daß auch bei einer effizienten Umsetzung des Prototyps das Late-Binding akzeptable Werte (Verlustfaktor zwei bis drei bezogen auf Early-Binding) liefern wird. Daß effizientere Umsetzungen möglich sind, zeigen die ersten Ergebnisse für SdaiKomm, die wir noch weiter ausbauen werden.

Das Zieleinsatzgebiet von STEP und auch SDAI sind große heterogene Anwendungssysteme (heterogen im Sinne von sehr unterschiedlichen Anwendungsprofilen). Damit kommt zwangsläufig der Wunsch nach Unterstützung für mengenorientierte Verarbeitung, um die Anwendungsprogrammierung zu vereinfachen und ein hohes Optimierungspotential (siehe Query-Optimierer in relationalen Systemen) nutzbar zu machen. Erst mengenorientierte Verarbeitung ermöglicht dem darunterliegenden System unter Ausnutzung von Zugriffspfaden alternative Zugriffspläne zu erstellen und zu analysieren und damit die Anwendung zu optimieren. Zur Zeit sind RDBS und OODBS die wichtigsten Kontrahenten für die Realisierung eines STEP-DBS. Man hat also die Wahl, das satzorientierte SDAI entweder auf ein ebenfalls satzorientiertes oder aber ein mengenorientiertes DBS aufzusetzen (siehe Abbildung 5). Ist SDAI mengenorientiert (und in diese Richtung bewegt sich der Standard), dann kommen zu den obigen Spielarten noch zwei hinzu: die Realisierung eines mengenorientierten API auf ein ebenfalls mengenorientiertes oder auf ein satzorientiertes DBS.

DBS \ API	satzorientiert	mengenorientiert
satzorientiert	S->S	M->S
mengenorientiert	S->M	M->M

Abb. 5: Kombination satz- oder mengenorientierter APIs und DBS

Die Kombination “**S->S**” liegt sowohl beim Prototyp als auch beim kommerziellen Produkt vor. Hier liegt es nahe, jede SDAI-Operation mit einer oder wenigen Operationen des DBS zu implementieren; diese Architektur bezeichnen wir mit CachedDirect. “**S->M**” bedeutet aus logischer Sicht einen Rückschritt, die mengenorientierten Fähigkeiten werden nicht dem Anwender durchgereicht. Ein Ansatz hierfür sollte der BufferedDirect-Architektur genügen. Es ist leider kaum möglich, aus dem gegebenen SDAI-Code Informationen zur Optimierung zu gewinnen; man kann sich lediglich auf Methoden wie die Cluster-Bildung zurückziehen, falls das DBS dies unterstützt; (HMNR95) diskutiert die mit Cluster-Bildung verbundene Problematik detailliert. Die Variante “**M->S**” zieht natürlich einen nicht unerheblichen Aufwand (Anfrageübersetzer und -auswerter, Einbettung in Programmiersprache) für die Implementierung von SDAI mit sich. Zumindest aber trifft man auf ein Problemfeld, das in der Datenbankwelt sehr gut verstanden wird; anfragebasierte Systeme sind so aufgebaut. Sie versuchen im wesentlichen, die Zugriffspfadunabhängigkeit ihrer mengenorientierten Schnittstelle optimal auszunutzen und in effiziente Zugriffspläne umzusetzen. Auch die mit "**M->M**" verbundene Anfragetransformation ist weithin bekannt. Durch Analyse der gestellten Anfragen lassen sich adaptive Umsetzungen auf die darunterliegende Anfragesprache erzielen. Dieser Ansatz ist vom Aufwand her sehr viel einfacher durchzuführen als "M->S".

6. Literatur

(CDKN94) M. J. Carey, D. J. DeWitt, C. Kant, J. F. Naughton: A Status Report on the OO7 OODBMS Benchmarking Effort. Report, Computer Science Department, University of Wisconsin-Madison 1994.

(CR94) A. B. Chaudri, N. Revell: Benchmarking Object Databases: Past, Present & Future. Report, Dept. of Business Computing, The City University, London 1994.

(DFMV90) D. J. DeWitt, P. Futtersack, D. Maier, F. Velez: A Study of Three Alternative Workstation-Server Architectures for Object Oriented Database Systems. In VLDB 1990.

(Di94) W. Dirlewanger: Messung und Bewertung der DV-Leistung - auf Basis der Norm DIN 66273. Hüthig 1994, ISBN 3-7785-2147-0.

(Gr91) J. Gray (Ed.): The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann 1991, ISBN 1-55860-159-7.

(HMNR95) T. Härder, B. Mitschang, U. Nink, N. Ritter: Workstation/Server-Architekturen für datenbankbasierte Ingenieuranwendungen. In IFE, 10/1995, S. 55-72.

(HL95) M. Hardwick, D. Loffredo: Efficient Database Implementation of EXPRESS Information Models. Report, Design and Manufacturing Institute, RPI and STEP Tools Inc., Rensselaer Technology Park, Troy, New York 12180, 1995.

(He94) A. Herbst: Long-Term Database Support for EXPRESS Data. In Proc. Conf. on Scientific and Statistical Database Management, Charlottesville, Virginia 1994.

(KNS94) M. Koethe, A. Nieva, F. Schönefeld: Product Data Exchange in Open Systems: The PISA Approach. STAK'94, Ilmenau, S. 101-119.

(Le93) F. Leymann: Towards The STEP Neutral Repository. In Proc. Conf. on CALS and Information Management in Europe, Berlin, Deutschland 1993.

(LK95) H. Lührsen, T. Krebs: STEP Databases as Integration Platform for Concurrent Engineering. Technischer Bericht IMMD VI und FAPS, Univ. Erlangen 1995.

(Ni95) U. Nink: SDAI auf DBS implementieren und anwenden. In Informatik aktuell: Datenbanken für Büro, Technik und Wissenschaft (BTW), Dresden 1995.

(ISO11) ISO TC184/SC4/WG5: Product Data Representation and Exchange - Part 11: EXPRESS Language Reference Manual, N55 IS, Januar 1994.

(ISO22) ISO TC184/SC4/WG7: Product Data Representation and Exchange - Part 22: Standard Data Access Interface, N392 DIS, July 1995.