in: Tagungsband der GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW'2001), A. Heuer (Hrsg.), Informatik aktuell, Oldenburg, März 2001, Springer-Verlag, pp. 227-243

Leistungsuntersuchung von ORDB-gestützten objektorientierten Anwendungssystemen

Weiping Zhang, Norbert Ritter

Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
{wpzhang | ritter}@informatik.uni-kl.de

Zusammenfassung: Objektorientierte Programmiersprachen (OOPL), wie z. B. C++, Java, SmallTalk, etc., haben sich in der Entwicklung komplexer Softwaresysteme durchgesetzt. Mit der Integration objektorientierter Konzepte und Erweiterungsmechanismen verfolgen objekt-relationale Datenbankverwaltungssysteme (ORDBVS) das Ziel, komplexe Softwaresysteme der neuen Generation, die überwiegend mit OOPL entwickelt werden, besser und effektiver zu unterstützen. In diesem Beitrag berichten wir über unseren Ansatz, die Fähigkeiten eines (beliebigen) ORDBVS zur Unterstützung von OOPLs (in der Entwicklung komplexer Softwaresysteme) zu bewerten und zu quantifizieren. Zu diesem Zweck werden zunächst Modellierungs- und operationale Aspekte sowohl des objektorientierten als auch des objekt-relationalen Paradigmas betrachtet, um die noch immer zu beobachtende Lücke zwischen diesen beiden Paradigmen aufzuzeigen. Die dabei getroffenen Aussagen werden dann mit Hilfe von Leistungsmessungen belegt, die mit einem neuartigen Benchmark-Ansatz durchgeführt wurden, den wir zur Analyse des Leistungsverhaltens von ORDBVS unter besonderer Berücksichtigung ihrer Fähigkeiten bei der Unterstützung von OOPL in der Entwicklung komplexer Softwaresysteme entwickelt haben. Neben dem Ziel, das Leistungsverhalten von ORDBVS entsprechend quantifizieren zu können, setzt sich dieser Ansatz das Langzeitziel, die weitere Entwicklung der objekt-relationalen Datenbanktechnologie dahingehend zu beeinflussen, dass die Anforderungen der objektorientierten Systementwicklung angemessener als bisher erfüllt werden können.

1 Motivation

Objektorientierte Programmiersprachen (OOPL), wie z. B. C++, Java, SmallTalk, etc., haben sich in der Entwicklung komplexer Softwaresysteme durchgesetzt. Ebenso wie die Struktur dieser Systeme gestalten sich auch die Strukturen der von diesen Systemen zu verwaltenden Objekte als sehr komplex. Die von OOPL angebotenen objektorientierten Konzepte und Mechanismen eignen sich zunächst zur Manipulation dieser komplex strukturierten Objekte. Darüberhinaus bestehen jedoch weitere Anforderungen, wie z. B. Persistenz und transaktionsgeschützte Manipulation, die nur durch Integration von Datenbankverwaltungssystemen (DBVS) erreicht werden können. Somit gehört die Datenbanktechnologie zu den Kerntechnologien der modernen Systementwicklung.

Dennoch ist im Gegensatz zu objektorientierten Datenbanksystemen (OODBVS) die Nutzung von RDBVS in der objektorientierten Systementwicklung aufgrund des *impedance mismatch* [4, 11] als problematisch zu beurteilen. Dieser besagt, dass komplexe Objektstrukturen und navigierende Verarbeitung auf Seiten der OOPL nur schwerlich auf die einfachen Datenstrukturen und die mengenorientierte, deskriptive Anfragesprache (SQL) von RDBVS abgebildet werden können. Hier ist jedoch zu betonen, dass dieses Problem nicht zu einer verstärkten Nutzung von OODBVS führte. Diese wurden aufgrund anderer Probleme [4], auf die wir in diesem Papier nicht eingehen können, nicht ausreichend akzeptiert.

Mit der Integration objektorientierter Konzepte und Mechanismen verfolgen objekt-relationale Datenbankverwaltungssysteme (ORDBVS, [16]) das Ziel, komplexe Softwaresysteme der neuen Generation, die überwiegend mit OOPL entwickelt werden, besser und effektiver als bisher zu unterstützen. Das objekt-relationale Datenmodell und seine Erweiterungsmechanismen haben zunächst die Kluft zwischen DBVS und OOPL entscheidend verringert. Dennoch ist die gewünschte, direkte Zuordnung von Strukturen und Verhalten zwischen OOPL und ORDBVS (noch) nicht möglich. Dies hat seine Ursache vor allem darin, dass die DB-Schnittstelle (SQL:1999, [18, 19]) die Aufspaltung der im ORDBVS gehaltenen, objektorientierten Strukturen in traditionelle, relationale Strukturen erzwingt. Insgesamt lässt sich die Lücke zwischen OOPL und ORDBVS auf eine Reihe von Modellierungs- und operationalen Aspekten zurückführen, wie wir im nachfolgenden Kapitel näher betrachten werden.

Es existieren bisher zahlreiche kommerzielle Systeme, die objektorientierte Strukturen auf RDBVS abbilden [2, 10, 12, 13, 15, 17]. Ein solches System wird auch als “Persistent Object System built on Relation” (kurz *POS*) bezeichnet. Mit der Verfügbarkeit von ORDBVS stellt sich nun die Frage, inwieweit man durch Ausnutzung von OR-Technologie eine effektivere Abbildung objektorientierter Strukturen auf DBVS erreichen kann, um so die objektorientierte Systementwicklung besser zu unterstützen.

Die Beantwortung dieser Frage erfordert sowohl eine konzeptionelle als auch eine empirische Betrachtung. Die konzeptionelle Betrachtung soll Aussagen darüber liefern, inwieweit das mit SQL:1999 standardisierte Datenmodell mit dem von OOPL korrespondiert und welche generellen Abbildungsregeln angemessen sind. Die empirische Betrachtung soll ähnlich wie ein Benchmark-System [8] die Beurteilung eines gegebenen ORDBVS hinsichtlich seiner Fähigkeiten zur Unterstützung der objektorientierten Systementwicklung durch Vergabe von Leistungskennzahlen ermöglichen. Neben der Bewertung einzelner ORDBVS sollte ein solcher neuer Benchmark-Ansatz dann auch die angemessene(re) Berücksichtigung der Anforderungen der objektorientierten Systementwicklung bei der weiteren Entwicklung von ORDBVS fördern.

Bei der empirischen Betrachtung eines gegebenen ORDBVS hinsichtlich seiner Fähigkeiten zur Unterstützung der objektorientierten Systementwicklung müssen folgende Fragen beantwortet werden:

1. Welcher (Mindest-)Aufwand ist zur Abbildung der Objektstrukturen (Schematransformation) und zur Transformation von (POS-)Anfragen (in SQL-Anfragen) zu kalkulieren?
2. Welches Leistungsverhalten zeigt das betrachtete DBVS in der Verarbeitung typischer, durch die in 1. angesprochene Transformation (dynamisch) erzeugter SQL-Anfragen?

Während die zweite Frage (vom Ansatz her) mit herkömmlichen Benchmark-Systemen [8] beantwortet werden kann, ist zur Einbeziehung der ersten Frage ein Überdenken der traditionellen Benchmark-Ansätze unerlässlich, wie bereits in [1, 24] ausgeführt.

In diesem Artikel wollen wir sowohl die angesprochene konzeptionelle Betrachtung der Möglichkeiten einer Unterstützung der objektorientierten Systementwicklung durch ORDBVS vornehmen (Kapitel 2), als auch unseren ersten Ansatz eines konfigurierbaren Benchmark-Systems vorstellen, der versucht, die beiden oben angeführten Fragen zu beantworten (Kapitel 3). Erste Messergebnisse mit zugehörigen Analysen werden in Kapitel 4 vorgestellt, bevor wir dann mit einer Zusammenfassung und einem Ausblick schließen.

2 Konzeptionelle Betrachtung

Es existiert eine Vielzahl von Objektdatenmodellen, wie z. B. ODMG [7], UML [22], COM [2], C++ und Java. Alle diese Modelle unterstützen das Paradigma der Objektorientierung, weisen aber gewisse Unterschiede auf. Weitgehend unabhängig von der in der objektorientierten Softwareentwicklung benutzten Modellierungssprache (z. B. UML) muss SQL:1999 an eine konkrete OOPL gebunden werden. Aufgrund ihrer Relevanz und der konzeptionellen Nähe ihrer Objektmodelle orientieren wir uns an C++ und ODMG und vergleichen diese mit dem SQL:1999-Standard [9, 18, 19].

2.1 Modellierungsaspekte

Objektorientierung in OOPL

Das Konzept des Objektes stellt die Grundlage des objektorientierten Paradigmas dar. Ein ***Objekt*** ist die ***Kapselung*** von Daten, die eine reale Einheit in ihren Strukturen/Werten und ihrem Verhalten repräsentieren [14]. Eine ***Klasse*** implementiert einen Objekttyp (***Klassifizierung***), der durch einen Namen sowie eine Menge von Attributen und Methoden charakterisiert ist. Jedes Attribut hat einen bestimmten Datentyp, der wiederum einwertig oder mehrwertig (***Kollektionstypen***) sein kann. Darüberhinaus können Datentypen skalar (z. B. Integer, Boolean, String, etc.) oder komplex sein. Im letzteren Fall kann es sich um Referenzen (***Assoziation***) oder Objekte anderer Klassen (***Aggregation***) handeln, so dass komplexe Strukturen modelliert werden können. Eine Klasse kann Methoden (***Verhalten***) implementieren, die Objektzustände verändern können. Klassen können in einer ***Generalisierung***shierarchie angeordnet werden, entlang derer

Struktur und Verhalten vererbt wird bzw. verfeinert (***Spezialisierung***) werden kann. Auf eine detailliertere Betrachtung der Konzepte des objektorientierten Paradigmas wollen wir an dieser Stelle verzichten. Im folgenden soll vielmehr herausgearbeitet werden, inwieweit das (O)R-Datenmodell diese objektorientierten Konzepte umfasst.

Objektorientierung in SQL:1999

Nachdem im relationalen Datenmodell (SQL2) semantische Modellierungskonzepte nicht ausreichend unterstützt wurden, besteht die fundamentale Erweiterung des SQL:1999-Standards in der Unterstützung von strukturierten benutzer-definierten Datentypen (***user-defined types, UDT***, [9]), die als Objekttypen betrachtet und darüberhinaus in der gleichen Weise wie vordefinierte Datentypen (***built-in types***, Basistypen) benutzt werden können. Somit ist das Typsystem von SQL:1999, ähnlich wie das Typsystem von OOPL, ***erweiterbar***. UDTs können komplex strukturiert sein und somit nicht nur vordefinierte Datentypen sondern auch mehrwertige Attribute (***Kollektionstypen***) und sogar auch andere UDTs (***Aggregation***) bzw. Referenzen (***Assoziation***) enthalten. Offenbar sind UDTs mit den Klassen des objektorientierten Paradigmas vergleichbar, sie müssen nach dem SQL:1999-Standard jedoch mit Tabellen assoziiert werden. Der Mechanismus der ***typisierten Tabelle*** (auch als ***Objekttabelle*** bezeichnet) erlaubt, Instanzen von einem bestimmten UDT in einer Tabelle zu verwalten. Jedes Tupel einer solchen Tabelle repräsentiert eine Instanz (***Objekt***) des entsprechenden UDT's und bekommt einen (benutzer- oder systemvergebenen) Identifikator (OID, ***Objektidentität***). Neben instanziierbaren UDTs sieht der SQL:1999-Standard auch nicht-instanziierbare UDTs vor, die den abstrakten Klassen der OOPLs entsprechen. Es sei an dieser Stelle noch einmal betont, dass sich UDTs nicht explizit von Basistypen unterscheiden, d. h., UDTs können genauso wie Basistypen als Datentypen von Attributen definiert werden. Darüberhinaus können UDTs mit Methoden ausgestattet werden, die ihr Verhalten repräsentieren, und sie können in einer Vererbungshierarchie angeordnet werden. Somit unterstützt SQL:1999 Polymorphie und Substituierbarkeit; Mehrfachvererbung ist jedoch nicht vorgesehen. Im wesentlichen aufgrund der Assoziation von UDTs mit Tabellen sieht der SQL:1999-Standard keine Kapselung im Sinne des objektorientierten Paradigmas vor. Folglich gibt es auch für die von OOPLs her bekannten Kapselungsgrade (***public, protected, private***) kein Pendant.

Beziehungen zwischen UDTs können durch Referenzen abgebildet werden (***Assoziation***). Durch die Nutzung von mehrwertigen Referenzattributen können auf einfache Weise (n:m)-Beziehungen dargestellt werden. Während der SQL:1999-Standard fordert, dass Referenzen die Eigenschaft "***scoped***" aufweisen müssen, d. h., ausschließlich Objekte genau einer vorgegebenen Objekttabelle referenzieren dürfen, erlauben manche kommerziell verfügbaren ORDBVS auch freie ("***unscoped***") Referenzierung. Laut SQL:1999-Standard sind Referenzen unidirektional, obwohl bidirektionale Referenzen die Erhaltung der Datenintegrität und die Erreichbarkeit von Objekten während der Verarbeitung erleichtern würden.

2.2 Allgemeine Verarbeitungsaspekte

Neben den grundlegenden, oben diskutierten Modellierungsmöglichkeiten sind auch operationale Aspekte zu untersuchen, um den “konzeptionellen Abstand” zwischen OOPLs und SQL:1999 zu erfassen. Zu diesem Zweck halten wir die nachfolgend angeführten Aspekte für relevant.

Navigation vs. deskriptive Anfragen

Während in OOPLs navigierende Verarbeitung unterstützt wird, bietet SQL mengenorientierte, deskriptive Verarbeitung an der Schnittstelle des ORDBVS. Eine direkte Unterstützung der Navigation besteht in der Bereitstellung eines über seine OID angesprochenen DB-Objektes als Instanz einer in der jeweiligen OOPL definierten Klasse. Die meisten ORDBVS unterstützen die Navigation nicht direkt. Eine naive Kopplung einer OOPL mit SQL, die vorsieht, zum Zeitpunkt einer Dereferenzierung *GetObject(**Ref**)* das benötigte Objekt mit einer entsprechenden SQL-Anfrage (siehe Abb. 1) vom DB-Server anzufordern, führt unweigerlich zu einem schlechten Laufzeitverhalten, da die Kosten der Transformation zu einer Anfrage, der Evaluation dieser Anfrage durch das DBS und der Client/Server-Kommunikation sehr hoch sein können. Offensichtlich müssen geeignetere Kooperationsmechanismen zwischen OOPL und DBVS gefunden werden, die zumindest einen mengenorientierten DB-Zugriff vorsehen. Letzteres ist insbesondere bei Ingenieuranwendungen, in denen vorab bekannte, geschlossene Verarbeitungskontexte auftreten, möglich, aber auch andere Anwendungen können von einem geeigneten Prefetching selbstverständlich profitieren.

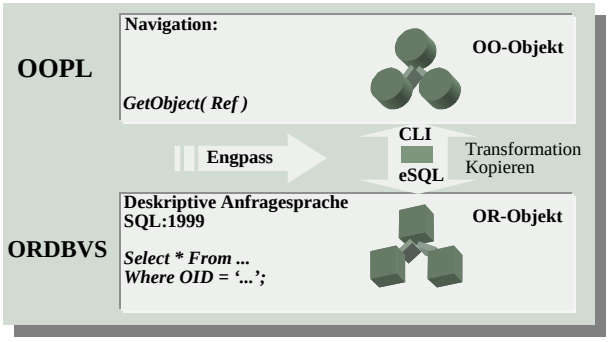

Abb. 1: Engpass zwischen OOPL und ORDBVS

Strukturierte Anfrageergebnisse

Wie oben bereits mehrfach motiviert, unterstützen OOPL komplex strukturierte Objekttypen, insbesondere durch die Möglichkeiten der Schachtelung von Typen sowie der Nutzung von Kollektionstypen und Referenzen. Wir haben ebenfalls bereits an früherer Stelle angesprochen, dass diese Konzepte der Komplex-Objekt-Bildung auch im SQL:1999-Standard berücksichtigt worden sind. Leider gehen aufgrund der traditionellen Grundprinzipien von SQL komplexe Strukturen an der Schnittstelle des DBVS verloren, da SQL nur flache Tupelmengen als Ergebnisse liefert. Daher ist es im Falle der Kopplung von OOPL und (O)RDBVS häufig notwendig, einfach strukturierte Bestandteile von komplexen Objekten durch mehrere SQL-Anfragen vom DB-Server anzufor-

dern, und die komplexe Struktur dann im Programm wieder aufzubauen (siehe Abb. 1). Das angesprochene Problem verschlimmert sich, wenn nicht einzelne, komplex strukturierte Objekte sondern komplexe Strukturen (Objektgraphen) bestehend aus mehreren, durch Referenzen verbundenen Objekten als Einheiten selektiert werden sollen.

Caching

Im Abschnitt “Navigation vs. deskriptive Anfragen” haben wir bereits angesprochen, dass insbesondere Ingenieuranwendungen geschlossene Verarbeitungskontexte aufweisen, die i. allg. bereits zu Beginn der Verarbeitung bekannt sind. Darüberhinaus zeichnen sich solche Anwendungen durch eine sehr hohe Lokalität der Verarbeitung aus, so dass die applikationsseitige Pufferung der Daten für die Dauer der Verarbeitung sehr sinnvoll ist. Leider unterstützen die zur Zeit verfügbaren ORDBVS nicht über applikationsseitige Cache-Komponenten, die eine anwendungsgesteuerte, explizite Einlagerung, Verarbeitung und Propagierung von Verarbeitungskontexten ermöglichen. Auch der SQL:1999-Standard macht dazu keinerlei Aussagen.

2.3 (DB-)Anwendungsspezifische Aspekte

Während wir in den vorangegangenen Abschnitten den “Abstand” zwischen OOPL und SQL:1999 anhand der auf beiden Seiten vorherrschenden, allgemeinen (d. h., von einer bestimmten, zu betrachtenden Miniwelt unabhängigen) Modellierungs- und Verarbeitungsaspekte untersucht haben, wollen wir in diesem Abschnitt herausfinden, inwieweit die vom (Daten-)Modellierer DB-seitig erfasste Anwendungssemantik bei einer Verarbeitung von DB-Daten in der OOPL genutzt bzw. gesichert werden kann. So umfassen die operationalen Aspekte natürlich auch die Ausnutzung von DB-seitig realisiertem Objektverhalten. Diese Methoden (UDRs) sind oft bereits aufgrund der Art ihrer Realisierung nur Server-seitig aufrufbar bzw. zeigen unter Umständen andere, möglicherweise unerwünschte Seiteneffekte, wenn sie (oder spezielle Pendants) in der OOPL aufgerufen werden. Darüber hinaus wird die Anwendungssemantik zu einem nicht unerheblichen Teil DB-seitig durch verschiedenartige Integritätsbedingungen erfasst, deren Prüfung in die SQL-basierte, DB-seitige Verarbeitung automatisch durch das DBVS eingeflochten werden kann, die aber OOPL-seitig nur schwer oder sogar aufgrund ihrer Reichweite überhaupt nicht gesichert werden können. Den auf diese Art und Weise beim Übergang vom ORDBS zur OOPL entstehenden Verlust an Anwendungssemantik bezeichnen wir als *semantische Lücke*. Dies ist relevant, da oft der mit der OOPL arbeitende Programmierer diese semantische Lücke genau kennen muss, um mit dem Gesamtsystem geeignet arbeiten zu können. Im Wissen um die Anwendungssemantik muss er bereits im Programm Vorkehrungen treffen, die eine integritätserhaltende Propagierung der im Programm manipulierten DB-Daten am Ende der Verarbeitung anstreben.

2.4 Weitere Aspekte

Wir möchten an dieser Stelle darauf hinweisen, dass in diesem Papier nicht alle Aspekte von ORDBVS hinsichtlich eines potentiellen Nutzen in der objektorientierten Software betrachtet werden können. Mit den oben beschriebenen meinen wir die wichtigsten erfasst zu haben. Natürlich können auch weitere Mechanismen nützlich sein, wie z. B. die Möglichkeiten der Integration heterogener Daten, die es erlauben, (aus Sicht des DBVS) extern liegende Daten in die DB-Verarbeitung miteinzubeziehen. Damit können dem (OOPL-)Programmierer sinnvolle, weniger aufwendige Alternativen zum direkten Zugriff auf externen Daten von der OOPL aus angeboten werden. Aus Platzgründen verzichten wir auf eine Diskussion solcher Mechanismen, die indirekt die Kopplung von OOPL und ORDBVS beeinflussen können.

3 Leistungsuntersuchungen

Im vorangegangenen Kapitel haben wir festgestellt, dass bzgl. der Datenmodellierung nur kleinere, aber bzgl. der Verarbeitungsmechanismen und der Applikationssemantik (semantische Lücke) deutliche Diskrepanzen zwischen den Welten der OOPL und der ORDBVS anzutreffen sind. Um diese Distanz bzw. den zu ihrer Überbrückung notwendigen Aufwand quantitativ zu erfassen, verfolgen wir einen konfigurierbaren Benchmark-Ansatz [16, 24]. Vor der Erläuterung dieses Ansatzes sei jedoch noch einmal betont, dass es nicht Ziel der vorliegenden Arbeit ist, ORDBVS mit OODBVS in ihren Fähigkeiten zur Unterstützung von OOPL zu vergleichen. Wir wollen vielmehr der Beobachtung Rechnung tragen, dass ORDBVS verstärkt in diesem Bereich eingesetzt werden, aber zugehörige Leistungskennzahlen noch nicht erhoben werden können. Daher ist auch der OO7-Benchmark [6], der auf die Beurteilung objektorientierter Systeme ausgelegt ist, für unsere Zwecke nicht relevant. Er ist ‘lediglich’ ein Beispiel eines Standard-Benchmarks [8], mit deren Hilfe bisher das Leistungsverhalten verschiedenartiger DBVS evaluiert werden konnte. Weitere Vertreter sind der Wisconsin-Benchmark [3], die TPC-Benchmarks [21] und der Bucky-Benchmark [5]. Diese Benchmark-Ansätze erlauben es jedoch nicht, die durch das Potential des DBVS bestimmte (zwischen OOPL und DBS benötigte) Abbildungsschicht bzw. deren Leistungsverhalten in die Gesamtbetrachtung miteinzubeziehen. Außerdem ist zu beachten, dass die zu verwaltenden Datentypen und die typischen Operationen in den Zielanwendungen von ORDBVS stark variieren (double-edged sword [5, 20]), so dass *ein* Standard-Benchmark das gesamte Spektrum nicht abdecken kann. Um, wie bereits angesprochen, das Gesamtsystem (inkl. Abbildungsschicht) in seinem Leistungsverhalten hinsichtlich seiner typischen Anwendungen angemessen beurteilen zu können, halten wir ein offenes, konfigurierbares Benchmark-System für notwendig. Dies wird uns auch helfen, zukünftig spezielle Aspekte, wie z. B. die Unterstützung navigierender Operationen, genauer zu untersuchen, da ein Benchmark-System, das zumindest hinsichtlich der Generierung von Benchmark-Daten und zu betrachtenden -Anfragen (Last) offen und konfigurierbar ist, den allgemeinen Untersuchungsaufwand reduzieren sollte. Unsere

ersten Ansätze werden im folgenden kurz erläutert. Details finden sich in [1, 24].

3.1 Benchmark-System

Ein offenes, konfigurierbares Benchmark-System muss nicht unbedingt schwer zu handhaben sein, wie wir an unserem Ansatz verdeutlichen wollen. Unser gegenwärtiger Prototyp bietet 3 Standardkonfigurationen (*small*, *medium* und *large*), um ausreichend skalierbar zu sein. Strukturierungsmöglichkeiten und Datenmengen der 3 Standardkonfigurationen wurden in einer Kooperation mit einem der führenden Hersteller betrieblicher Standardsoftware ermittelt und sollen ein breites Spektrum der typischen Anwendungsdomainen repräsentieren. Das Benchmark-System kann hinsichtlich der speziellen Anforderungen einer konkreten Anwendung einfach konfiguriert werden. Abb. 2 gibt einen Überblick über die Architektur des gegenwärtigen Systems. Bei der Generierung der Benchmark-Datenbank beispielsweise kann der Benutzer unter anderem Einfluss nehmen auf die Datenmenge, die Komplexität der Klassenhierarchie und die Komplexität der einzelnen Objekte, um so die Anforderungen seiner Zielanwendung zu berücksichtigen. Nach der Generierung der Benchmark-Daten kann der Benutzer verschiedene Benchmark-Anfragen aus einer vordefinierten Menge von typischen Anfrage-Templates [1, 24] wählen. Darüber hinaus können weitere Anfrage-Templates hinsichtlich einer geeigneten Berücksichtigung der typischen Operationen der Zielanwendung hinzugefügt werden. Weiterhin kann durch entsprechende Parametrisierung auf die Generierung der eigentlichen Last (siehe Lastgenerator in Abb. 2) Einfluss genommen werden. Der Messtreiber setzt zur Durchführung der Messungen die in der Last enthaltenen Anfragen an das Klassensystem ab. Zur Erhebung von Messdaten können vielfältige Messpunkte gesetzt werden; in jedem Fall werden die zur Bearbeitung einer Anfrage im DBVS sowie in der Abbildungsschicht benötigten Zeiten festgehalten und nach Beendigung des Messlaufs in der Datenbank zu Analysezwecken gespeichert.

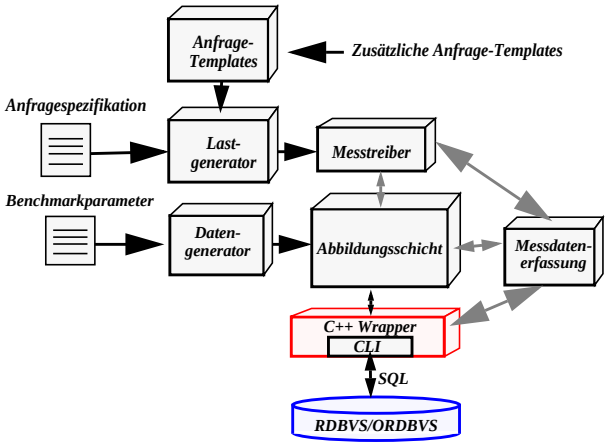

Abb. 2: Architektur des Benchmark-Systems

Wie in [1, 24] näher erläutert, besteht die besondere Herausforderung dieses Benchmark-Ansatzes einerseits darin, ein bzgl. seiner Anwendbarkeit in der objektorientierten Systementwicklung geeignet gestaltetes POS zu berücksichtigen, und andererseits eine im Hinblick auf die Möglichkeiten eines bestimmten, zu evaluierenden ORDBVS

optimale Abbildung (d. h. Realisierung des POS) zu garantieren. Die Erarbeitung entsprechender Regeln ist Gegenstand aktueller Arbeiten.

3.2 Benchmark und Messungen

Um angemessene Aussagen über das Leistungsverhalten von ORDBVS hinsichtlich der Unterstützung von OOPL zu erhalten, konzentrieren wir uns auf folgende Fragenstellungen, die in diesem Abschnitt näher motiviert und im nachfolgenden Kapitel anhand von empirischen Untersuchungen beantwortet werden sollen:

1. Welchen (zusätzlichen) Aufwand verursacht die Abbildungsschicht und wie verhält sich der Zusatzaufwand bei unterschiedlichen Anfragen?
2. Welche Leistungsvorteile bieten ORDBVS im Vergleich zu RDBVS hinsichtlich ihrer Nutzung in der objektorientierten Softwareentwicklung?
3. Wie wirken sich die Fähigkeiten der DB-Programmierschnittstelle aus?

Um die erste und die zweite Frage beantworten zu können, haben wir in Zusammenarbeit mit einem der führenden Hersteller betrieblicher Standard-Software eine Reihe typischer Anfragen (auf objektorientierten Klassenhierarchien) ausgewählt. Darauf aufbauend haben wir unter Nutzung eines der führenden, gegenwärtig kommerziell verfügbaren ORDBVS eine rein relationale mit einer objekt-relationalen Abbildung (Nutzung der objektorientierten Modellierungskonzepte des objekt-relationalen Modells) verglichen. Derartige Untersuchungen sollen quantitative Aussagen darüber liefern, inwieweit die objektorientierte Softwareentwicklung durch Ausnutzung objekt-relationaler Modellierungskonzepte (strukturierte UDTs, Referenzen, etc.) profitieren kann. Die Anfragen sind in folgenden Kategorien gruppiert:

a) Navigierende Operationen: Operationen wie *GetObjekt(OID)* werden in der Regel nicht direkt bzw. effizient von (O)RDBVS unterstützt. Die Betrachtung solcher Operationen hilft, das Leistungsverhalten bei der Unterstützung von Navigation zu erfassen.

b) Anfragen mit einfachen Prädikaten unterschiedlicher Selektivität: Diese Gruppe von Anfragen dient hauptsächlich dazu, Abhängigkeiten des Leistungsverhaltens von der Kardinalität der jeweiligen Treffermenge zu bestimmen.

c) Anfragen auf UDTs: Durch Betrachtung dieser Art von Anfragen, kann die Effizienz der Abbildung von UDTs auf das (O)RDBVS untersucht werden. UDTs können im ORDBVS auf unterschiedliche Weise (*named row type, opaque type*) direkt abgebildet werden. Da im rein relationalen Fall keine direkte Abbildung möglich ist, wurden spezielle Methoden [2, 10, 12, 13, 15, 17] zur Abbildung von objektorientierten Strukturen auf das relationale Datenmodell entwickelt. Die von den meisten POS-Anbietern [2, 10, 12, 13, 15] gewählte Methode sieht vor, alle einfach strukturierten, sowohl originären als auch ererbten Attribute einer Klasse auf genau eine Tabelle abzubilden und für komplexe Attribute eigene Tabellen vorzusehen. Unsere Voruntersuchungen [1] zeigen, dass diese Methode i. allg. den leistungsfähigsten Weg darstellt.

d) Anfragen auf mengenwertigen Datentypen: Diese (und auch die nachfolgend beschriebene) Gruppe von Anfragen erlaubt die Untersuchung der Auswirkungen komplexer Attributtypen auf das Leistungsverhalten bei der Anfrageauswertung. ORDBVS unterstützen wiederum die direkte Abbildung mengenwertiger Datentypen, wohingegen im relationalen Fall weitere, über Primär-/Fremdschlüssel-Konstrukte zu integrierende Tabellen angelegt werden müssen.

e) Anfragen auf Referenztypen: Mit der Betrachtung dieser Gruppe von Anfragen werden ähnliche Ziele verfolgt wie bei der vorgenannten. Hier geht es um die Auswirkungen der verschiedenen Methoden der Abbildung von Referenzen im objekt-relationalen und im rein relationen Fall.

f) Anfragen mit komplexen Prädikaten: Anhand dieser Gruppe von Anfragen soll das Optimierungspotential des (O)RDBVS untersucht werden.

g) Anfragen auf der Klassenhierarchie (auch transitive Instanzen): Anhand dieser Anfragen kann die Effizienz der Abbildung einer Generalisierungshierarchie in die Leistungsuntersuchung miteinbezogen werden.

Die dritte, zu Beginn dieses Abschnitts angeführte Frage befasst sich mit der Thematik, wie sich die Fähigkeiten der DB-Programmierschnittstelle auswirken. Hier geht es uns insbesondere darum, inwieweit neben dem (traditionellen) mengenorientierten Zugriff auf einfach strukturierte Objekte auch komplex strukturierte Objekte zurückgeliefert werden können bzw. inwieweit Navigation adäquat unterstützt werden kann. Insbesondere um solche Fragestellungen empirisch untersuchen zu können, haben wir Messungen auf zwei verschiedenen (kommerziell erfolgreichen) ORDBVS durchgeführt, wobei eines eher die traditionelle SQL-Schnittstelle anbietet, während das andere bereits über eine rudimentäre Unterstützung von komplex strukturierten Objekten verfügt.

Alle Messungen wurden mit einer Benchmark-Datenbank mit 100 Klassen und insgesamt 250000 Instanzen (Konfiguration *Medium*) vorgenommen. Dabei haben wir die DB-Zeit und die Gesamt-Systemzeit festgehalten. Die DB-Zeit einer SQL-Anfrage ist die Zeit zwischen dem Absenden der Anfrage an die Datenbank und dem Erhalt der Ergebnismenge (Cursor öffnen, Iterator durchlaufen). Sie beinhaltet die Client/Server-Kommunikationszeit zwischen dem Datenbankserver und der Abbildungsschicht (POS) und die Zeit zur Evaluation der Anfrage durch das DBS. Die Gesamt-Systemzeit ergibt sich damit zur Summe von DB-Zeit und der Zeit, die zusätzlich von der Abbildungsschicht benötigt wird.

Insgesamt sind wir der Ansicht, dass die zu Beginn dieses Abschnitts angeführten Fragen zu beantworten sind, bevor darüber nachgedacht werden kann, wie ORDBVS-Technologie weiterentwickelt werden sollte, um eine bessere Unterstützung von OOPL zu erreichen. Im nachfolgenden Kapitel berichten wir über die Ergebnisse der von uns durchgeführten Messungen.

4 Ergebnisanalyse

4.1 Leistungscharakteristika der Abbildungsschicht

Um das Leistungsverhalten der Abbildungsschicht zu charakterisieren, haben wir zunächst einfache Anfragen mit verschiedener Selektivität betrachtet. Die zugehörigen Ergebnisse sind in Abb. 3 dargestellt. Wie bereits in Kapitel 2 erwähnt, unterstützen Programmierschnittstellen von (O)RDBVS navigierenden Zugriff nicht direkt, so dass eine navigierende Operation wie *GetObject(**Ref**)* von der Abbildungsschicht zu einer DB-Anfrage "*Select * From ... Where OID = **Ref** "* transformiert werden muss. Eine solche Vorgehensweise führt bei intensiver Nutzung von Navigationsoperationen unweigerlich zu hohen Verarbeitungskosten durch das Datenbanksystem sowie ebenfalls sehr hohen Kommunikationskosten (über 70% der gesamten Systemzeit; siehe Abb. 3, links). Betrachten wir den mengenorientierten Zugriff, so ist mit steigender Treffermenge nur ein leichter Anstieg der DB-Kosten zu verzeichnen, während der zusätzliche Aufwand der Abbildungsschicht rasant anwächst. Bei einer Treffmenge von 1250 Objekten kann man sogar feststellen, dass die von der Abbildungsschicht benötigte Zeit knapp 86% der Gesamt-Systemzeit ausmacht (siehe Abb. 3, rechts). Diese Beobachtung kann darauf zurückgeführt werden, dass die von RDBVS an ORDBVS "vererbte", traditionelle DB-Programmierschnittstelle keine komplexen Objekte unterstützen kann, so dass jedes angefragte komplexe Objekt zuerst in skalare Werte zerlegt werden muss, und anschließend in der Abbildungsschicht wieder zusammengesetzt werden muss. Die Kosten dieses Vorgangs steigern sich mit zunehmender Kardinalität der Treffermenge und beeinträchtigen so die gesamte Systemeffizienz sehr stark.

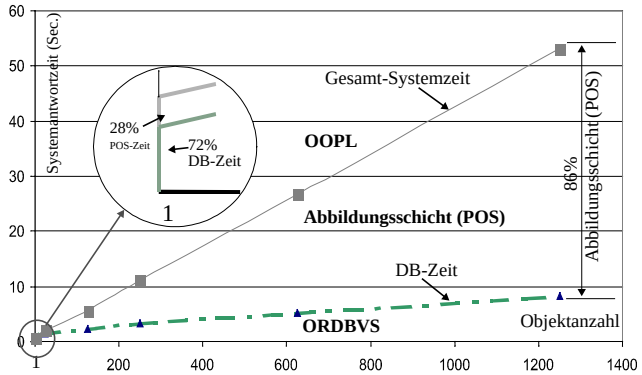

Abb. 3: Messergebnisse I

Aus den angesprochenen Messungen kann man folgende Schlussfolgerung ziehen. Um navigierenden Zugriff besser unterstützen zu können, sollte die (OR)DB-Programmierschnittstelle navigierende Operationen wie *GetObject(OID)* direkt unterstützen, so dass die Transformation einer navigierenden Operation zu einer Select-Anfrage durch die Abbildungsschicht und die Evaluationskosten dieser Select-Anfrage durch das DBS vermieden werden können. Weiterhin sollte die DB-Programmierschnittstelle Möglichkeiten anbieten, komplexe Objekte als Einheiten zu selektieren. Derartige Verbesserun-

gen lassen nach auf unseren Messergebnissen beruhenden Schätzungen eine Steigerung der gesamten Systemeffizienz um bis zu ca. 400% zu.

4.2 ORDBVS vs. RDBVS

In einer zweiten Messreihe haben wir unter Nutzung eines der führenden, gegenwärtig kommerziell verfügbaren ORDBVS eine rein relationale mit einer objekt-relationalen Abbildung verglichen. Dabei wurde natürlich das Ziel verfolgt, den Nutzen der objektorientierten Erweiterungen des objekt-relationalen Modells näher zu quantifizieren.

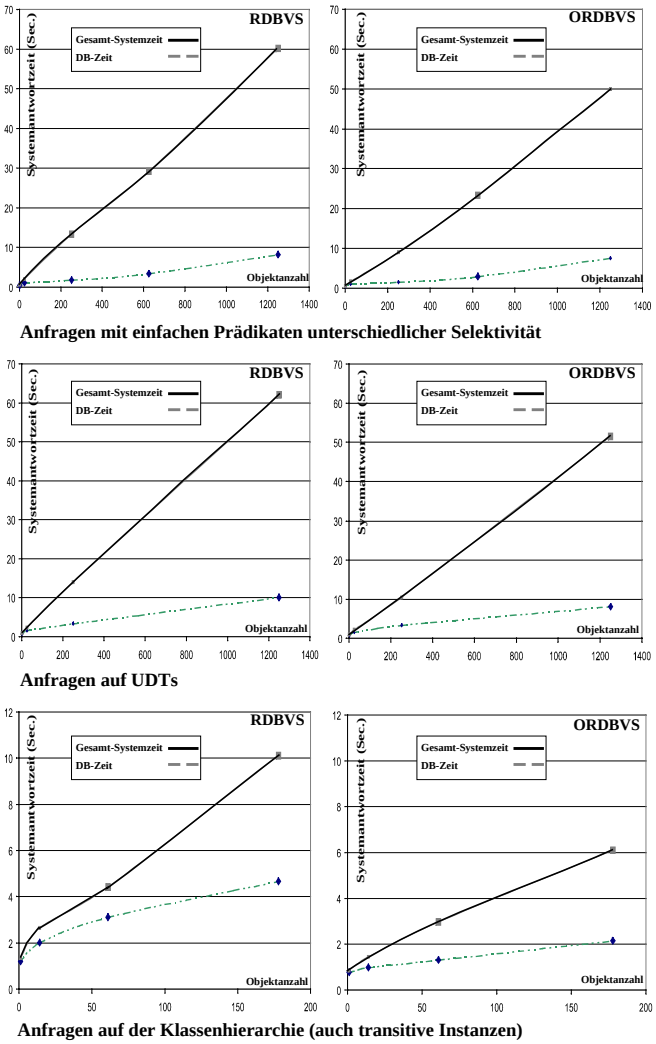

Abb. 4: Messergebnisse II

Abb. 4 stellt die zugehörigen Messergebnisse in groben Zügen dar. Leider ist es uns aus Platzgründen nicht möglich, unsere Ergebnisse in allen Details anzuführen. Wir konnten bei der Untersuchung beobachten, dass die objekt-relationale Ab-bildung in allen Anfragekategorien ein besseres Leistungsverhalten zeigt (siehe Abb. 4). Bei kleineren Treffermengen ist jedoch nur ein sehr kleiner Vorsprung der objekt-relationalen Abbildung festzustellen. Insbesondere bei Anfragen, die mehrere Klassen einer Klassenhierarchie betreffen und größere Treffermengen haben, ist ein Leistungsgewinn von bis zu 40% zu verzeichnen. Diese Beobachtungen lassen sich wohl darauf zurück-

führen, dass die um objektorientierte Konzepte erweiterten Datenmodellierungs- und Anfragemöglichkeiten von ORDBVS dazu beitragen, die Komplexität der Abbildungsschicht (und den zu deren Entwicklung nötigen Aufwand) und die Client/Server-Kommunikation gering halten.

Im rein relationalen Fall ist es nicht möglich, eine (komplexe) Klasse auf genau eine Tabelle abzubilden. Um mehrwertige Attribute, Aggregationen und (m:n)-Beziehungstypen abzubilden, ist es im Gegensatz zur objekt-relationalen Abbildung notwendig, mehrere, über Primär-/Fremdschlüssel-Beziehungen miteinander verbundene Tabellen vorzusehen. Dies führt dazu, dass zur Selektion der Instanzen *einer* Klasse mehrere Anfragen notwendig sind, was wiederum höhere DB-Zeiten sowie höhere Kommunikationskosten impliziert. Darüber hinaus wirkt sich auch der in der Abbildungsschicht zur Anfragetransformation und zur Rekonstruktion der Objektstrukturen anfallende Aufwand insgesamt leistungsmindernd aus (im Vergleich mit einer semantisch äquivalenten objekt-relationalen Abbildung). Dieser Effekt wird außerdem dadurch verstärkt, dass weitere, in ORDBVS vorhandene Dienste, wie Indexstrukturen, auf die objektorientierten Erweiterungen zugeschnitten sind und so eine insgesamt bessere Abbildung unterstützen.

Da in unseren Messungen Client und Server auf derselben Maschinen liefen, sind die Kommunikationskosten zwischen dem DB-Server und der Abbildungsschicht relativ gering. Es ist zu erwarten, dass bei einer Verteilung von Client und Server eine weitere Vergrößerung der Unterschiede in den Messergebnissen entsteht und so die oben getroffenen Aussagen weiter verstärkt werden. Während in früheren Messungen [5, 23] objekt-relationale Systeme in vielen Kategorien noch schlechter abschnitten als relationale, scheint die objekt-relationale Technologie mittlerweile ausgereifter zu sein, zumindest was das in unseren Messungen betrachtete kommerzielle ORDBVS betrifft.

4.3 Unterstützung komplexer Objekte

Wie bereits an früherer Stelle angesprochen, wirkt sich der Mangel an direkter Unterstützung von komplexen Objekten und navigierendem Zugriff durch die DB-Programmierschnittstelle äußerst negativ hinsichtlich der Unterstützung von OOPL aus. Dies gilt insbesondere für RDBVS, wie aus Abschnitt 4.1 hervorgeht, aber auch für ORDBVS, wie die Diskussion in Abschnitt 4.2 zeigt. Es gibt diesbezüglich jedoch Unterschiede zwischen den verfügbaren ORDBVS. Einer der führenden ORDBVS-Hersteller bietet bereits eine erweiterte Aufrufschnittstelle, die Navigation und darüberhinaus, wie wir weiter unten sehen werden, auch die Selektion von Objektgraphen unterstützt. Navigation wird dadurch ermöglicht, dass ein möglicherweise komplex strukturiertes Objekt automatisch als Instanz einer C-Struktur zu Verfügung gestellt werden kann, was die Abbildung auf OOPL, wie z. B. C++, erheblich vereinfacht. Dies ist zweifellos ein Schritt in die richtige Richtung, obwohl auch dieser Mechanismus noch nicht die gewünschte nahtlose Kopplung (transparente Transformation eines DB-Objektes in eine Instanz einer OOPL-Klasse) unterstützt. Die angesprochene Unterstüt-

zung komplexer Objekte an der DB-Programmierschnittstelle ermöglicht es, Objekte über ihre OID anzusprechen und ihre (Attribut-)Werte (ausschließlich strukturelle Information) aus der Datenbank in den Hauptspeicher zu laden. Dadurch wird die bei den in Abschnitt 4.1 beschriebenen Messungen betrachtete aufwendige Vorgehensweise vermieden. Hierbei ist zu betonen, dass bei den in Abschnitt 4.1 beschriebenen Messungen ein ORDBVS genutzt wurde, das nicht über die hier angesprochene, direkte Unterstützung komplexer Objekte verfügt.

Zum Vergleich und zur Untermauerung unserer Forderung nach einer geeigneten Unterstützung komplexer Objekte durch ORDBVS haben wir die in Abschnitt 4.1 beschriebenen Messungen auf dem ORDBVS, das komplexe Objekte in der beschriebenen Art und Weise unterstützt, wiederholt. Abb. 5 illustriert die Messergebnisse. Es kann eindeutig festgestellt werden, dass der durch die Abbildungsschicht entstehende Aufwand nun unabhängig von der Kardinalität der Treffermenge der Anfrage ist. Daraus ergibt sich ein deutlicher Leistungsgewinn durch die Unterstützung komplexer Objekte.

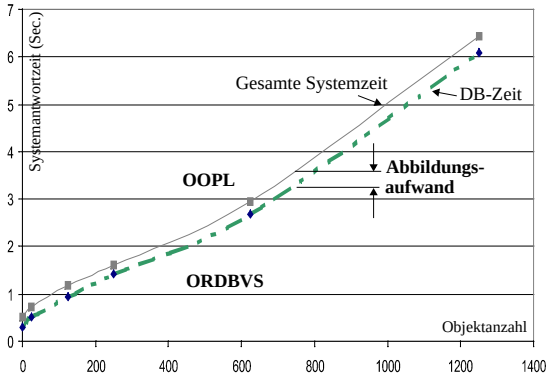

Abb. 5: Messergebnisse III

Die angesprochene Unterstützung von navigierendem Zugriff an der DB-Programmierschnittstelle, d. h., über OID(s) ist ein(e Reihe von) Objekt(en) direkt ansprechbar ("*GetObject*"), vermeidet die durch die Programmierschnittstelle anderer (O)RDBVS erzwungene Vorgehensweise (Anfragetransformation, Datenkonvertierung und Objektrekonstruktion) und trägt so dazu bei, das Leistungsverhalten zu verbessern. Abb. 6a zeigt einen relativen Vergleich zwischen einer Anfragestrategie, die auf der Nutzung von Anweisungen der Form "*Select * From ... Where OID = '...';*" beruht, und der Navigationsstrategie, die Operationen der Form

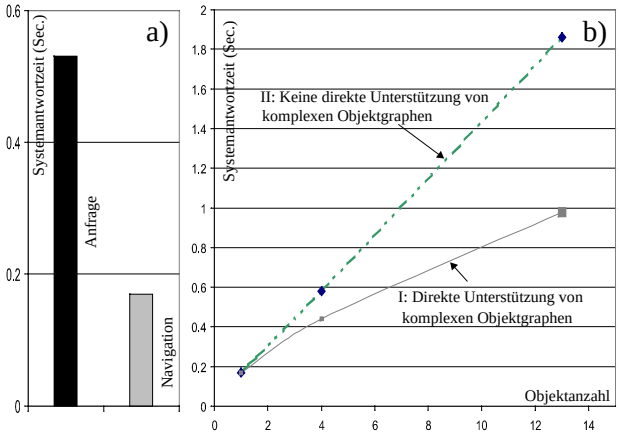

Abb. 6: Messergebnisse IV

"*GetObject(OID)*" nutzt. Der Unterschied ist deutlich zu erkennen. Durch direkte Unterstützung von navigierendem Zugriff steigt die gesamte Systemeffizienz um etwa 200%.

In objektorientierten Anwendungen ist es darüber hinaus oft wünschenswert, dass eine Menge von durch Referenzen verbundenen Objekten (Objektgraph) mit einer einzigen DB-Interaktion als Einheit geladen werden kann. Abb. 6b zeigt die Ergebnisse eines Vergleichs zweier Strategien der Selektion komplexer Objektgraphen. In diesen Messungen wurde auch wieder das bereits oben angesprochene ORDBVS verwendet, das sowohl Navigation als auch komplexe Objektgraphen direkt unterstützt. Es ist deutlich zu erkennen, dass bereits bei einer Größe der Treffermenge von etwa 13 Objekten Strategie I, die die Fähigkeiten der Selektion von Objektgraphen nutzt, ein um 100% besseres Leistungsverhalten aufweist.

Natürlich ist das Design einer neuen DB-Programmierschnittstelle, die komplex strukturierte Objekte unterstützt, nicht einfach und erfordert generische Methoden, denn benutzerdefinierte Datentypen können willkürlich sein und z. B. andere komplexe Datentypen, wie weitere UDTs, Referenzen und mengenwertige Attribute, enthalten. Auch aufgrund der geforderten Multi-Lingualität (alle gängigen OOPLs sollen unterstützt werden) wird es immer schwierig sein, das Beste beider Welten (ORDBVS und OOPL) anzubieten.

5 Zusammenfassung und Ausblick

In diesem Beitrag haben wir die Notwendigkeit der Untersuchung von ORDBVS hinsichtlich ihrer Fähigkeiten der Unterstützung von OOPL motiviert. Dazu haben wir ORDBVS und OOPL zunächst hinsichtlich Modellierungs- und operationalen Aspekten qualitativ betrachtet. Während sich das OR-Datenmodell (SQL:1999) und das objektorientierte Datenmodell wesentlich näher gekommen sind, ist die operationale Distanz zwischen den beiden Welten nach wie vor gravierend, so dass nicht nur eine zusätzliche Abbildungsschicht notwendig ist, sondern auch die gesamte Systemeffizienz dadurch erheblich (negativ) beeinträchtigt werden kann. Zusätzlich haben wir versucht, anhand von quantitativen Untersuchungen das Leistungsverhalten von ORDBVS hinsichtlich der Unterstützung von OOPL zu beurteilen und damit den durch den Paradigmenwechsel in Kauf zu nehmenden Verlust zu beziffern. Mittelbar sollen diese Messungen dazu beitragen, die optimale Nutzung gegenwärtig verfügbarer ORDBVS in der objektorientierten Systementwicklung zu erleichtern und die weitere Entwicklung von ORDBVS so zu beeinflussen, dass die Unterstützung von OOPL weiter verbessert werden kann.

Hinsichtlich der Leistungsuntersuchungen haben wir die Notwendigkeit eines offenen, konfigurierbaren Benchmark-Ansatzes motiviert, denn nicht nur die Performanz des ORDBVS selbst sondern auch der zusätzliche Aufwand, der zur Überbrückung der konzeptionellen und operationalen Distanz zwischen ORDBVS und OOPL nötig ist, müs-

sen in die Untersuchung miteinbezogen und charakterisiert werden. Durch die Messungen wurde deutlich, dass objekt-relationale Datenbanktechnologie mittlerweile nicht nur konzeptionell (Datenmodell, Anfragesprache), sondern auch unter Leistungsgesichtspunkten auf dem Vormarsch ist. Die Fähigkeit, komplexe Objekte direkt im ORDBVS zu modellieren, trägt dazu bei, die Abbildungsschicht im Gegensatz zu RDBVS „dünn“ zu halten. Dies reduziert einerseits den Implementierungsaufwand, andererseits kann ein besseres Leistungsverhalten erreicht werden. Trotz der sich positiv auswirkenden objektorientierten Erweiterungen, die einen eindeutigen Gewinn durch ORDBVS im Vergleich zu RDBVS mit sich bringen, ist das gesamte Potential der OR-Technologie unserer Ansicht nach noch nicht ausgeschöpft, da die DB-Programmirschnittstelle bisher zu wenig in der Lage ist, die objektorientierten Prinzipien nach aussen nutzbar zu machen. Die DB-Programmierschnittstelle der meisten ORDBVS kann nach wie vor navigierende Operationen und komplexe Objektstrukturen nicht direkt unterstützen, so dass Softwaresysteme der neuen Generation nicht in optimaler Weise von der OR-Technologie profitieren können.

Durch unsere Untersuchungen wurde deutlich, dass eine neue DB-Programmierschnittstelle, die navigierende Operationen und komplexe Objektstrukturen direkt unterstützt, dringend notwendig ist. Natürlich ist die Ausstattung von ORDBVS mit einer solchen Schnittstelle nicht trivial. Beispielsweise muss eine solche Schnittstelle multi-lingual sein. Zur Beantwortung der Frage, wie benutzerdefinierte Datentypen am effektivsten auf der OOPL-Ebene repräsentiert werden können, sind noch weitere Forschungsarbeiten nötig. Insgesamt ist die Frage zu beantworten, wie man SQL:1999 nahtlos und effektiv an OOPL wie C++ anbinden kann. Unsere zukünftige Arbeit wird sich hauptsächlich auf die Beantwortung dieser Frage beziehen. Weiterhin planen wir, ORDBVS zur Unterstützung von navigierendem Zugriff verstärkt zu untersuchen, denn im Gegensatz zu OODBVS weisen ORDBVS hier deutliche Leistungsnachteile auf. Auch hierbei besteht das Ziel, nach einer genaueren Charakterisierung Konzepte vorzuschlagen, die zur der Steigerung des Leistungsverhaltens von ORDBVS beitragen können.

Literatur

[1] R.Bernhard, M. Flehmig, A. Mahdoui, N. Ritter, H-P. Steiert, W.P. Zhang: “*Abbildung des Klassensystems auf (O)RDBMS - Konzepte und Bewertung*”, Interner Bericht, 1999.

[2] P. A. Bernstein, B. Harry, P.J. Sanders, D. Shutt, J. Zander, “*The Microsoft Repository*”, Proc. 23th. VLDB Conf., 1997, pp. 3-12.

[3] D. Bitton, D. DeWitt, and C. Turbyfill: “*Benchmarking database systems: A systematic approach*”, Proc. VLDB, 1983.

[4] M. J. Carey, D. J. DeWitt: “*Of Objects and Databases: A Decade of Turmoil*”, Proc. 22th. VLDB Conference Mumbai(Bombay), India, 1996.

[5] M. J. Carey, D.J. DeWitt, J.F. Naughton, M. Asgarian, P. Brown, J.E. Gehrke, D.N. Shah: "*The Bucky Object-Relational Benchmark*", Proc. 1996 VLDB Conf., pp. 135-146.

[6] M. J. Carey, D. J. DeWitt, C. Kant, J. F. Naughton: "*A status report on the OO7 OODBMS benchmarking effort*", Proc. ACM OOPSLA, Portland, OR, USA, 1994, pp. 414-426.

[7] R. G. G. Cattell, D. Barry, D. Bartels, et al: "*The Object Database Standard:ODMG 2.0*", Morgan-Kaufman Publishers, San Mateo, 1997.

[8] J. Gray: "*The Benchmark Handbook for Database and Transaction Processing Systems*", Morgen Kaufmann Publishers, San Mateo, CA, USA, 2nd Ed., 1993.

[9] P. Gulutzan, T. Pelzer: "*SQL-99 Complete, Really*", R&D Publications, 1999.

[10] A. Keller, R. Jensen and S. Agrawal: "*Persistence Software: Bridging Object-Oriented Programming and Relational Database*", Proc. 1993 ACM SIGMOND Conf., pp. 523-528.

[11] W. Mahnke, H-P. Steiert: "*Zum Einsatzpotential von ORDBVS in Entwurfsumgebunge*n", Proc. CAD 2000, Berlin.

[12] Ontos Business Data Server, *http://www.ontos.com*.

[13] Poet Object Server,POET Software, POET SQL Object Factory, *http://poet.com/*.

[14] B. R. Rao: "*Object-oriented Databases: Technology, Applications, and Products*", McGraw-Hill, New York, 1994.

[15] RogueWave Software, DBTools.h++, *http://www.roguewav3e.com/products/dbtools/*.

[16] H. Schreiber: "*JUSTITIA: A Generic Benchmark for the OODBMS Selection*", Intl. Conference on Data and Knowledge Systems in Manufacturing and Engineering, Tokyio, 1994.

[17] T. Scheller: "*Funktionalität des Klassensystems im System R/3*", Funktionsbeschreibung, Version 0.9, Dez. 1997.

[18] SQL99: ANSI/ISO/IEC 9075-1-1999 Database Launguages SQL Part 1 Framework.

[19] SQL99: ANSI/ISO/IEC 9075-2-1999 Database Languages SQL Part 2 Foundation

[20] M. Stonebraker, P. Brown, D. Moor: "*Object-relational DBMSs - The Next Wave*", Morgan Kaufmann, 2nd Ed., 1998.

[21] TPC: Transaction Processing Performance Council, Standard Specification 1.0, May 1995, *http://www.tpc.org*.

[22] UML, Rational Software Corp. Unified Modeling Language, *http://www.rational.com/*.

[23] W.P. Zhang: "*Evaluation of the First Generation ORDBMSs by Using Bucky Benchmark*" Interner Bericht, 1998.

[24] W.P. Zhang, N. Ritter: "*Measuring the Contribution of (O)RDBMS to Object-Oriented Software Development*", Proc. IDEAS 2000, Yakohama, Japan.