Objektorientierte Datenintegration

über mehrere Technologiegenerationen

Wolfgang Keller, sd&m
Christian Mitterbauer, HYPO-Bank
Dr. Klaus Wagner, sd&m

Erschienen auf der ONLINE 1996, Hamburg
Beitrag C 644

München Februar 1996

sd&m
software design & management
GmbH & Co. KG
Thomas-Dehler-Straße 27
81737 München
Telefon (089) 6 38 12 - 210
Telefax (089) 6 38 12 - 410

Objektorientierte Datenintegration über mehrere Technologiegenerationen

Wolfgang Keller, Christian Mitterbauer, Klaus Wagner.

Abstract

EDV-Großanwender arbeiten heute oft mit drei und mehr Generationen von Software-Technologie simultan. Während meist einige Pilotprojekte mit Objekttechnologie arbeiten, wird die Masse der Software oft konventionell mit relationalen Datenbanken und 3GL-Sprachen entwickelt. Parallel sind in der Regel auch noch ältere Anwendungen mit hierarchischen Datenbeständen (IMS) vorhanden. Diese tragen oft die Hauptlast des EDV-Betriebs. Datenintegration über alle Technologiegenerationen ist daher unverzichtbar. Die objektorientierte Anwendungsentwicklung muß auf die Datenbestände anderer Softwaregenerationen zugreifen, ohne daß in alte Anwendungen eingegriffen werden darf. Objektorientierte Datenbanken alleine reichen hier nicht aus. Der vorliegende Artikel stellt einen Integrationsrahmen für die Lösung dieser Probleme vor und gibt einen Überblick über verfügbare Produktkategorien, mit denen dieser Rahmen zu einer konkreten Lösung gefüllt werden kann.

Als Anwendungsbeispiel wird eine Lösung des Problems durch eine objektorientierte Zugriffsschicht auf heterogenen Datenbanken bei einer großen Geschäftsbank diskutiert. Dort wurde besonderer Wert auf die Integration aller bestehenden Datenbestände und auf die Koexistenz von Client/Server- und Host-Umgebungen mit Transaktionsmonitoren gelegt. Ein besonderer Schwerpunkt des Artikels sind die typischen Randbedingungen von Großanwendern, die heute noch den problemlosen Einsatz käuflicher Standardlösungen, wie Objektdatenbanken mit Gateways, verhindern.

Über die Autoren

Dipl.-Inform. Wolfgang Keller ist Berater bei der sd&m, software design & management GmbH & Co. KG. Dipl.-Inform. (BA) Christian Mitterbauer arbeitet in der Abteilung O/EDV-VDS (Verfahrenstechnik) der Bayerischen Hypotheken- und Wechselbank AG. Dr. Klaus Wagner ist Berater bei sd&m. Die Autoren sind über das Internet unter *Wolfgang.Keller@sdm.de*, *100066.467@compuserve.com* (Christian Mitterbauer) und *Klaus.Wagner@sdm.de* zu erreichen.

Inhaltsverzeichnis

1 Einführung und Überblick

Viele EDV-Großanwender, wie zum Beispiel Banken und Versicherungen, stehen heute vor der Entscheidung, objektorientierte Software-Entwicklungsmethoden einzuführen oder haben bereits erste Pilotprojekte gestartet. Dabei ergibt sich meist das Problem der Integration alter und neuer Software-Technologie. Eine vollständige, sofortige Ablösung sämtlicher Altanwendungen durch neue, objektorientierte Anwendungen ist ein Weg, der gerade Großanwendern nicht zu empfehlen sein wird. Es ist eher zu erwarten, daß es 10 Jahre und mehr dauern wird, bis Neuentwicklungen durchgängig mit Objekttechnologie durchgeführt werden. Altanwendungen werden noch deutlich länger weiterleben.

Konventionell entwickelte Host-Anwendungen werden also mit objektorientierten Client/Server-Anwendungen lange kooperieren müssen. Es kann dabei im allgemeinen nicht davon ausgegangen werden, daß die Datenbestände für beide Klassen von Anwendungen sauber zu trennen sind. Unternehmensdatenmodelle und integrierte Datenbestände sind, professionell implementiert, von enormem Nutzen für die Unternehmen. Nur um eine neue Technologie einzuführen, kann auf das bislang Erreichte nicht verzichtet werden. Für eine reine technologische Erneuerung wird ein betriebswirtschaftlicher Nutzen selten nachzuweisen sein.

Anwender mit großen Datenbeständen stehen heute vor dem Problem, Datenbestände aus drei bis vier[1] Technologiegenerationen der Datenbanktechnologie miteinander integrieren zu müssen:

- Als erste Generation werden im folgenden hierarchische Datenbanksysteme wie IMS-DB bezeichnet. Anwendungen dieser Generation tragen heute in vielen Fällen noch die Hauptlast der operativen Datenverarbeitung.

- Die zweite Generation sind relationale Datenbanksysteme, wie zum Beispiel DB2. Bei den meisten Großanwendern hat diese zweite Generation die erste noch nicht vollständig verdrängt - oft sind hier noch Ablösungen angestrebt.

- Als dritte Generation wollen wir objektorientierte Datenbanksysteme (OODBMS) [ODMG93, Kim95, Cat94] betrachten. Hier stehen die Produkte erst am Anfang. Eingeführt sind sie bei Großanwendern allenfalls in Pilotprojekten.

[1] Bei den meisten Großanwendern sind heute auch noch VSAM Datenbestände in nicht unerheblicher Menge im Einsatz. Man könnte hier von der „nullten" Generation sprechen - wodurch man zu vier zu integrierenden Generationen kommt.

Dabei sollte sich für die Anwendungsentwickler, egal ob C++-, Smalltalk-, COBOL- oder PL/I-Programmierer, jeweils ein bruchlos integriertes Bild der Datenbestände aus der Sicht jeweils ihrer Technologiegeneration ergeben. Die Vorteile integrierter Unternehmensdatenmodelle werden weiter genutzt. Ferner müssen die Datenbestände gleichzeitig von weiter konventionell entwickelten Host-Anwendungen und objektorientiert entwickelten Anwendungen lesend *und schreibend* genutzt werden können. Speziell an der letzten Anforderung scheitern noch viele Produkte. Die Lösung dieses Problems ist alles andere als trivial.

Der Markt für objektorientierte Datenbanktechnologien ist noch klein und wenig stabil. Die durchschnittliche Zeit, nach der am Markt wieder wesentliche Neuerungen auftreten, liegt derzeit bei ca. 6 Monaten. Das erklärt auch, warum viele Experten gar nicht mehr versuchen, Bücher zu dem Thema zu schreiben - sie wären beim Erscheinen veraltet. Wer ein Projekt aufsetzt, weiß fast sicher, daß die Lösung am Projektende technisch nicht mehr dem letzten Stand entspricht.

Die verwendeten Komponenten für eine Lösung des Datenintegrationsproblems müssen möglichst durch saubere Schnittstellen gegeneinander abgegrenzt werden. Damit erhält man die Möglichkeit, bei Marktbewegungen verbesserte oder kostengünstigere Komponenten nachträglich einzubauen.

Es gibt nicht *die eine* Lösung für die Integration von Datenbeständen über mehrere Technologiegenerationen. Es kann aber ein Integrationsmodell beschrieben werden, das für viele spezifische Lösungen gültig ist. In diesen Rahmen werden käufliche Komponenten und selbsterstellte Integrationsschnittstellen als Klebstoff für die gekauften Komponenten integriert. Welche Komponenten in einem konkreten Projekt gewählt werden, hängt primär von den eigenen, bisher verwendeten Datenbankprodukten, Transaktionsmonitoren und Middlewarekomponenten ab. Aus dem erwarteten Projektnutzen ergibt sich ferner eine Obergrenze für das mögliche Investitionsvolumen. Die Wahl ist auch wesentlich von den jeweils „tagesaktuell“ verfügbaren objektorientierten Datenbank- und Integrationsprodukten bestimmt.

In Abschnitt 2 wird ein praxiserprobtes Modell für die Integration von Datenbeständen über mehrere Technologiegenerationen vorgestellt. Wie schon erwähnt, sind Produktübersichten meist an dem Tag veraltet, an dem sie erscheinen. Abschnitt 3 gibt deshalb nur einen Überblick über Kategorien von Produkten, die in das vorgestellte Modell integrierbar sind. In Abschnitt 4 wird gezeigt, wie das konkrete Modell bei einer Großbank umgesetzt wurde.

Das Modell war nicht vor der Lösung vorhanden. Vielmehr sind Erfahrungen aus Projektlösungen über einen Zeitraum von ca. 5 Jahren zu dem hier vorgestellten Integrationsmodell zusammengefaßt worden.

2 Integrationsmodell

Das Integrationsmodell beschreibt eine Softwarearchitektur, mit deren Hilfe Objekte persistent[2] gemacht werden und mit der Altdatenbestände für eine Verwendung in einer objektorientierten Programmierumgebung erschlossen werden. Es handelt sich um eine Architektur für ein eingeschränktes[3], objektorientiertes Datenbanksystem. Spezialität ist, daß zur Ablage der Objekte objektorientierte, relationale oder hierarchische Datenbanken benutzt und in einer Architektur integriert werden. Damit werden mehrere Generationen von Datenbanktechnologie zu einem Bild vereinigt. Auch die Integration verteilter Datenbestände zu einer Sicht wird geleistet.

Ziel von objektorientierten Datenhaltungssystemen ist es, den Programmierer von (programmiersprachlichen) Objekten möglichst von Datenbankdetails zu entlasten. Es wird eine „persistente Programmiersprache" zur Verfügung gestellt.

Im folgenden wird zunächst beschrieben, wie ein Programmierer die Außensicht des Integrationsmodells wahrnehmen soll. Anschließend diskutieren wir den inneren Aufbau eines Modells, das verschiedene Datenbankparadigmen in einer einheitlichen Sicht integrieren kann.

2.1 Außensicht

Die Außensicht wird durch ein Beispiel für eine Einbindung persistenter Objekte in C++ illustriert. Diese orientiert sich am ODMG-Standard für objektorientierte Datenbanken [ODMG93]. Das Beispiel enthält bei weitem nicht alle Konstrukte der ODMG-Spezifikation - es hat sich für Projekte als ausreichend erwiesen, eine Teilmenge des Standards zu implementieren und besonders teure Konstrukte[4] zu umgehen. Das Beispiel ist nicht formal vollständig. Hier soll lediglich der Grundgedanke einer „persistenten Programmiersprache" illustriert werden.

[2] Persistenz ist die Eigenschaft von Objekten, ein Prozeßende zu überleben und anderen Prozessen wieder zur Verfügung zu stehen [Sou94].

[3] Wie weit das System tatsächlich gegenüber einem OODBMS eingeschränkt ist, hängt von der Investition in die oberste Schicht ab. Wird ein passendes OODBMS mit einer Integrationsmöglichkeit für Altdaten gefunden, ist die Leistung natürlich fast so gut, wie die des objektorientierten Datenbanksystems. Einschränkungen gibt es u.U. nur bei der Performance. Muß man ein Lösung selbst bauen, wird man nur die für Anwendungen unverzichtbaren Leistungen implementieren.

[4] Aufwendig zu implementieren sind z.B. geschachtelte Transaktionen und eine eingebettete Objekt-Query-Sprache (OQL [ODMG93])

```
void beispiel ( String kundenName ) {

try                          // Datenbankfehler werden über den C++-Exception-
{                            // Mechanismus bearbeitet

       Transaction trans;                   // Anlegen und starten
       trans.begin()                        // einer Transaktion

              // Deklaration der Variablen
              Ref<Kunde> alterKunde;       // Ref ist ein persistenter Pointer
              Set<Ref<Vertrag>> vertraege;  // Set<Ref<Vertrag>> repräsentiert
                                            // eine Menge persistenter Objekte

              // Holen des Kunden
              alterKunde = Kunde::getByName(kundenName);  // Hier wird nur eine Objekt-
                                            // Id geladen; Dies ist eine Methode
                                            // der 'Datenbank'

              // Dereferenzieren der Verträge
              vertraege = alterKunde->abgeschlVertraege;  // Durch das erste Verfolgen
                                                   // des Pointers wird das eigentliche
                                                   // Objekt dereferenziert

              // Anlegen eines neuen Kunden
              Ref<Kunde> neuerKunde = new(Kunde);

              neuerKunde->Name = ...               // das Objekt wird durch belegen der
                                                   // Attribute verändert;

              // Übernahme der Verträge
              neuerKunde->abgeschlVertraege = alterKunde->abgeschlVertraege;
              neuerKunde->markModified();          // Objekt kennzeichnen, damit es
                                                   // beim commit zurueckgeschrieben
                                                   // wird. (Methode der 'Datenbank')

              // Löschen des alten Kunden
              alterKunde->requestDelete();         // Methode der 'Datenbank'

              // Beenden der Transaktion
       trans.commit()
}

catch ... // Fehlerbehandlung wird hier nicht dargestellt
};
```

Abbildung 1: Beispiel für die Programmierschnittstelle des Integrationsmodells

Das Beispiel in *Abbildung 1* zeigt die folgenden Aktionen:

- Holen eines *Kunden* anhand seines Namens aus der Datenbank.
- Dereferenzieren der *Verträge*, die dieser *Kunde* abgeschlossen hat.
- Anlegen einer neuen Instanz der Klasse *Kunde*.
- Übernahme der *Verträge* durch den neuen *Kunden*.
- Löschen des alten *Kunden*.

Der Programmierer auf der Ebene der Anwendungskern-Objekte (Business Objects) muß hier mit den persistenten, in der Datenbank abgelegten Objekten nicht wesentlich anders umgehen, als mit flüchtigen (also nicht persistenten) Objekten. Ein Unterschied ist trotzdem zu bemerken. Es fällt auf, daß die Klassen *Kunde* und *Vertrag* nicht nur fachliche Methoden und Attribute besitzen, sondern auch

Methoden der 'Datenbank', wie z.B. `getByName`, `markModified` oder `requestDelete`. Diese Methoden werden durch Vererbung von einer abstrakten Basisklasse und Reimplementierung erworben. Eine solche Lösung ist noch nicht ideal. Dann wäre überhaupt kein Unterschied mehr zwischen persistenten und flüchtigen Objekten sichtbar. Sie ist aber pragmatisch vertretbar. Für eine weitergehende Diskussion sei auf [ODMG93] verwiesen.

2.2 Schichtenmodell

Um die in Abschnitt 2.1 beschriebene Außenschnittstelle zu implementieren, kann man mit Abstrichen die Leistungen einer ODMG-konformen, objektorientierten Datenbank implementieren. Diese Schnittstelle wird im folgenden als *Objekt-Schnittstelle* bezeichnet.

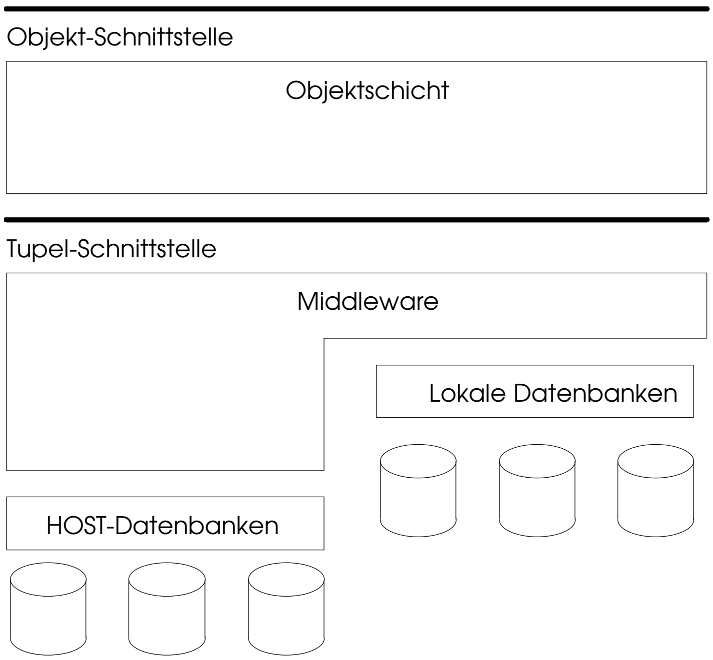

Abbildung 2: Integrationsmodell für persistente Objekte

Die Daten werden nicht oder nicht überwiegend in einem Datenhaltungssystem einer Objektdatenbank abgelegt. Vielmehr wird mehr als ein bestehendes Datenbanksystem mit relationaler oder hierarchischer Technologie verwendet. Es wird also eine Schnittstelle benötigt, die Daten aus mehr als einer Datenbank transaktionsgesichert zur Verfügung stellen kann. Diese Schnittstelle wird im folgenden als Tupel-Schnittstelle bezeichnet.

Abbildung 2 zeigt das entsprechende Schichtenmodell. Die Leistungen der Objektschicht werden in Abschnitt 2.3 genauer beschrieben. Die Anforderungen an die Tupel-Schnittstelle werden in Abschnitt 2.4 beschrieben. Das dargestellte Modell dient als Integrationsrahmen oder als Architektur für den vollständigen Eigenbau einer Lösung. Für viele der in *Abbildung 2* gezeigten Komponenten sind fertige Produkte verfügbar. Die unterschiedlichen Kategorien von Produkten werden in Abschnitt 3 vorgestellt. Sie unterscheiden sich wesentlich in dem Umfang, in dem sie die in *Abbildung 2* dargestellten Funktionen abdecken.

2.3 Objektschicht

Bei der Objektschicht kann es sich um eine käufliche Objektdatenbank oder eine selbsterstellte Lösung handeln. Ihre Aufgabe, einen möglichst großen Teil der Leistungen eines objektorientierten Datenbanksystems ähnlich zur Spezifikation [ODMG 93] zu implementieren, läßt sich in Funktionsblöcke zerlegen. *Abbildung 3* zeigt diese Funktionsblöcke im Überblick.

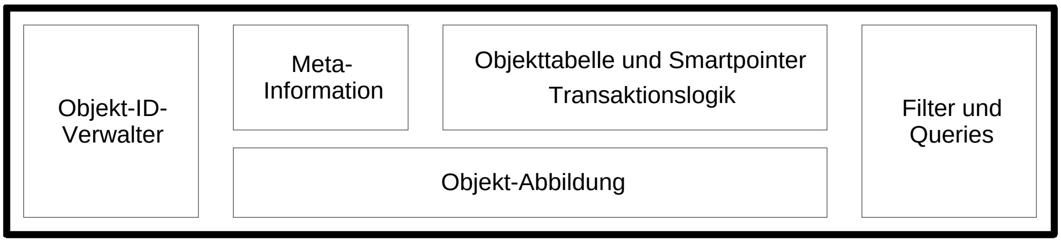

Abbildung 3: Funktionsblöcke einer OO-Zugriffsschicht

2.3.1 Funktionsblöcke

Die Objektschicht muß sich zur Ablage von Objekten der relationalen Tupel-Schnittstelle bedienen. Damit wird sie anders aufgebaut, als objektorientierte Datenbanken. Die Tupel-Schnittstelle bietet Tupel über primitiven Datentypen an. Begriffe wie komplexe Datentypen, Vererbung und Beziehungen sind auf der Ebene dieser Schnittstelle nicht bekannt.

Die Objektschicht muß also folgende Dienstleistungen erbringen, um die Instanzvariablen eines Objektes aus der Tupel-Schnittstelle zu füllen:

- Primitive Datentypen der Tupel-Schnittstelle müssen zu komplexen Datentypen der Objektinstanzdaten zusammengebaut werden. Dies geschieht als Teil der *Objekt-Abbildung*.
- Die Abbildung zwischen Attributen der Objekttypen und Feldern der Tupel-Schnittstelle muß unter Berücksichtigung von Vererbungsbeziehungen vorgenommen werden. Dies geschieht ebenfalls in der *Objekt-Abbildung* und wird durch *Meta-Information* gesteuert.

- Objektidentitäten müssen aus Schlüsseln der Tupel-Schnittstelle hergestellt werden. Dies leistet der *Objekt-ID-Verwalter*.
- Beziehungen auf Objektebene müssen aus Fremdschlüsseln und Beziehungsentitäten der Tupel-Schnittstelle hergestellt werden. Dies erledigt die *Objekt-Abbildung* in Zusammenarbeit mit dem *Objekt-ID-Verwalter*.

Die Beschreibung der internen Details der Objekt-Schicht würde den Umfang dieses Artikels sprengen. Dazu sei auf detailliertere Beschreibungen[5] verwiesen.

Über die oben beschriebenen Dienstleistungen hinaus sind noch Leistungen zu erbringen, die nicht zur Abbildung von Objekten auf eine Tupel-Schnittstelle gehören:

- Es müssen Objekte zur *Transaktionsverwaltung* zur Verfügung stehen.
- Die Eindeutigkeit von Objekten im flüchtigen Speicher muß durch eine *Objekttabelle* [Hah+95] gesichert werden.
- Über einen *Smart-Pointer*-Mechanismus [Str91] muß vermieden werden, daß mit dem Laden eines Objektes alle Objekte aus der Datenbank mitgeladen werden, die mit dem ersten berührten Objekt in irgendeiner transitiven Beziehung stehen.
- Dialoge zur Auswahl von Objekten aus einer potentiell sehr großen Menge müssen effizient unterstützt werden. Dazu ist es bei geringer Bandbreite der vorhandenen Leitungen nicht sinnvoll, schon komplette Objekte zu laden. Wenn man in einem Auswahldialog nur wenige Attribute sehen möchte, sollte selbst von solchen Objektausschnitten immer nur der Teil beschafft werden, der wirklich in einem Auswahldialog zu sehen ist. Blättern muß ein geblocktes Nachlesen auslösen. Es werden also *Filter* benötigt, um Queries zu formulieren, mit denen man gezielt nach Objekten suchen kann und die lediglich einen geringen Anteil der Objekte aus der Datenbank beschaffen. Mit Hilfe von *Iteratoren*, die auch Cache-Funktionalitäten bereitstellen, kann dann in beiden Richtungen auf der Datenmenge navigiert werden.

2.3.2 Diskussion

Solange in rein objektorientierten Programmierumgebungen ohne den Zugriff auf relationale und hierarchische Altdaten gearbeitet werden soll, ist es ausreichend, ein objektorientiertes Datenbankprodukt einzusetzen. Ein solches Produkt ist zum

5 Die einzelnen Funktionen sind in diversen Papers genauer beschrieben. Siehe dazu zum Beispiel [Col+95, Hah+95, Kel95a, Kel95b, Kel+95, Lip95].

Beispiel anhand des Standards [ODMG93] evaluierbar. Die internen Details müssen nicht bekannt sein.

Wenn Daten aus hierarchischen und relationalen Datenhaltungssystemen verwendet werden müssen, die zugleich auch von konventionell entwickelten Anwendungen parallel genutzt werden sollen, finden sich selten passende Produkte. Es wird dann Detailwissen über die Schnittstelle der Objektschicht zu ihren Speichermanagern benötigt. Eine reine Blackbox-Sicht reicht nicht mehr aus. Die Schnittstelle zu den Speichermedien kann quer durch Produkte laufen oder sie muß selbst auf der Basis von Produkten implementiert werden Die Schnittstelle kann also recht unterschiedlich aufgebaut sein. Weitergehendes Wissen über die Funktionen objektorientierter Datenhaltungssysteme und über die Abbildung auf relationale Konzepte ist erforderlich, wenn man selbst Teile implementieren oder integrieren muß. Produktkategorien und die Kriterien für ihren Einsatz werden in Abschnitt 3 diskutiert.

2.4 Tupel-Schnittstelle

Die oben beschrieben Leistungen einer Objektschicht können basierend auf einer relationalen *Tupel-Schnittstelle* erbracht werden [Col+95]. Die *Tupel-Schnittstelle* integriert relationale und hierarchische Datenbanken zu einer einheitlichen relationalen Sicht auf die Unternehmensdaten. Dies ist eine Teilaufgabe bei der Föderierung von Datenbanksystemen [Kim95]. Dabei erfüllt die *Tupel-Schnittstelle* die folgenden Anforderungen:

- Die *Tupel-Schnittstelle* bietet je Tupel einer Datensicht mindestens die Operationen read, insert, update, delete und read multiple für verschiedene Schlüssel. Diese Operationen sind identisch mit dem, was einfache Zugriffsschichten auf relationalen Datenbanken bieten.
- Die *Tupel-Schnittstelle* abstrahiert von der Herkunft der Daten aus mehreren Datenbanksystemen. Sie bietet ihrem Benutzer das Bild einer einzigen, lokalen Datenbank.
- Die *Tupel-Schnittstelle* bildet hierarchische Datenbestände auf relationale Tupel ab.
- Dem Benutzer der *Tupel-Schnittstelle* stehen Benutzertransaktionen über mehrere Operationen zur Verfügung. Dies ist i.a. nicht trivial zu garantieren, wenn zum Beispiel Datenbestände aus einem Host mit Transaktionssystem (IMS oder CICS) verwendet werden sollen.

Für die Implementierung einer *Tupel-Schnittstelle* gibt es zahlreiche Optionen, die von föderierten Datenbanksystemen über DRDA-Produkte [Orf+94] bis zu remote-SQL-Schnittstellen [Sal93] reichen. Diese Produkte sind in *Abbildung 2* unter dem Begriff Middleware als ein Block zusammengefaßt.

Je nach der Mächtigkeit der eingesetzten Produkte kann es für die Systemintegration nötig sein, sich näher mit diesem „Middleware-Block" auseinanderzusetzen. Abschnitt 3 geht näher auf die verwendbaren Produktkategorien ein.

2.5 Nutzungseinschränkungen des Integrationsmodells

Es ist nicht zweckmäßig, mit der hier dargestellten Architektur Massenupdates[6] durchzuführen, wenn kein dazu geeignetes Produkt für die Objektschicht (z.B. OODBMS mit optimierendem Speichermanager) vorhanden ist. Solche Operationen werden besser unter Umgehung der integrierten Zugriffsschicht in einzelnen Methoden von Objekten codiert, weil sie direkt durch eine Datenbank effizienter ausgeführt werden können. Integrierte Zugriffsschicht-Produkte sind typisch für Dialogsysteme geeignet. Dort wird meist nur auf *ein* Objekt und seine Umgebung zugegriffen. Batchprozesse sollen möglichst nahe an den verwendeten Daten ausgeführt werden und auf keinen Fall auf einem entfernten Client. Für Batch und Dialogverarbeitung kann zum Beispiel eine Instanz der kompletten Zugriffsschicht auf den Clients für die Dialogtransaktionen und eine codeidentische für die Batchverarbeitung auf einem Host installiert werden [Col+95].

2.6 Weiterführende Literatur

Es verwundert nicht, daß es zum Thema dieses Artikels noch keine integrierte Gesamtdarstellung, z.B. in Buchform, gibt. Teilaspekte werden durch Artikel abgedeckt. Einige Autoren befassen sich mit der Frage, wann besser relationale und wann besser objektorientierte Datenbanken zu verwenden sind, z.B. [Bur94, Kim95, Sto94]. Innerhalb objektorientierter Datenbankansätze können noch Klassifizierungen nach der Speicherungsart und Verwendung der Objekte vorgenommen werden (siehe [Kim95, Sou94]). Das Problem der Integration relationaler und objektorientierter Datenbanktechnologie wird von vielen Autoren andiskutiert, u.a. [Bur94, Kim95, Gra95]. Mit der Abbildung von objektorientierten auf relationale Konstrukte befassen sich u.a. [Hah+95, Kel94b, Pre+94]. Architekturen für Zugriffsschichten findet man u.a. bei [Hah+95, Kel+95, Lip95, NeXT].

[6] Unter Massenupdates versteht man z.B. SQL-Statements der Art update .... where, die mit einem Statement eine große Menge von Datenbanksätzen manipulieren.

3 Produktkategorien

Auf dem Markt finden sich zahlreiche Produkte, mit denen Teile des oben dargestellten Integrationsrahmens realisierbar sind. Dieser Abschnitt dient als kurzer Überblick für die Suche nach geeigneten Produkten. Teilweise sind solche Produkte auch schon für andere Aufgaben in den Unternehmen vorhanden und können in eine Lösung für persistente Objekte integriert werden. Wir können weder eine vollständige noch eine aktuelle Produktliste angeben. Dazu ist der Markt zu stark in Bewegung. Wir beschränken uns daher auf Klassen von Produkten und geben jeweils Beispiele an, ohne den Anspruch zu haben, alle Produkte zu nennen.

3.1 RDA-Produkte

RDA (Remote Data Access)- Produkte [Orf+94] stellen einen Zugang zu einer entfernten relationalen Datenbank her. Sie erlauben es, SQL-Befehle entfernt auf einem oder mehreren Datenbank-Servern ausführen zu lassen. Sie sind dafür geeignet, Teile der Tupel-Schnittstelle zu realisieren. RDA-Produkte bieten nicht immer auch eine Lösung für den simultanen Zugang zu mehreren Datenbanken, für verteilte Transaktionen oder gar für den Zugang zu hierarchischen Datenbeständen.

3.2 Objektifizierte relationale Datenbanken

Objektifizierte relationale Datenbanken stellen die Meta-Begriffe[7] der Schnittstelle relationaler Datenbanken als Klassenbibliothek zur Verfügung. Sie benötigen meist ein RDA-Produkt mit dynamischem SQL, um diese Dienstleistung zu bieten. Sie sind ebenfalls geeignet, um Teile der Tupel-Schnittstelle zu realisieren.

Typische Produkte aus dieser Kategorie sind zum Beispiel DBtools.h++ von Rogue Wave [www.roguewave.com] oder ein entsprechendes Framework von Taligent [Cot+95].

3.3 Föderierte Datenbanken

Föderierte Datenbanken bilden eine Schicht über mehrere, physisch getrennte Datenbanksysteme [Kim95]. Sie geben ihrem Anwender die Fiktion, eine einzige Datenbank zu benutzen. Die DRDA-Architektur [Orf+94] von IBM implementiert Aspekte eines föderierten Datenbanksystems. Föderierte Datenbanken werden auf

[7] Zum Beispiel Table, Row, Query, usw.

der Basis verschiedener SQL-Varianten angeboten und integrieren teilweise auch hierarchische Datenbanken (DRDA). Sie sind geeignet, um Teile der Tupel-schnittstelle zu realisieren. Beispiele für Produkte sind der DataJoiner von IBM [www.software.ibm.com/data/dbtools/datajoin.html], UniSQL/M [www.unisql.com] und die DRDA-Architektur [Orf+94].

3.4 Objektorientierte Datenbanken mit relationalem Gateway

Einige objektorientierte Datenbanken bieten die Möglichkeit, ausgewählte Objekte über einen speziellen Speichermanager in einer relationalen Datenbank ablegen zu lassen. Als Projektlösungen werden speziell maßgeschneiderte Speichermanager angeboten. Einen solchen Speichermanager kann man auch für eine Tupel-Schnittstelle maßschneidern lassen. OODBMS mit relationalem Gateway sind geeignet, um die Objektschicht zu realisieren. Zu beachtende Problempunkte sind die Integration hierarchischer DB-Systeme, Reengineering, das Problem der Objektidentität [Kel94a] und die Anbindung an Host-Transaktionssysteme. Auch werden „klassische" Host-Umgebungen nur selten unterstützt.

Produkte aus dieser Kategorie sind z.B. das OODBMS ONTOS [www.ontos.com] oder der Odapter (Objekt Adapter) zur OpenDB Architektur von HP [www.hp.com].

3.5 Objektorientierte Zugriffsschicht-Produkte

Auf dem Markt finden sich auch Produkte, die dem Aufrufer die Schnittstelle eines OODBMS oder ähnliche Mechanismen bieten und zur Speicherung der Objekte ausschließlich fremde Hintergrundspeicher benutzen und über kein eigenes Speichermanagement verfügen. Problempunkte sind die Unterstützung von Host-Datenbanken (z.B. DB2) und die Frage der Anbindung an Transaktionssysteme wie IMS-TM oder CICS.

Typische Produkte hier sind Persistence [www.persistence.com] oder das NeXT Enterprise Objects Framework [www.next.com].

3.6 Objekt/relationale Datenbanken

Neben objektorientierten Datenbanken gibt es auch sog. Objekt/Relationale Datenbanken [Kim95], die relationale Datenbanken um benutzerdefinierte Datentypen, Vererbung, Stored Procedures und weitere Konstrukte anreichern. Kim [Kim95] diskutiert ausführlich die Unterschiede zu objektorientierten Datenbanken. Objekt/Relationale Datenbanken orientieren sich an dem noch nicht verabschiedeten SQL3-Standard. Solche Produkte bieten (zusammen mit einer Zugriffsschicht) die Dienste der Objektschicht an. Auf der Ebene der Tupelschnittstelle wird teilweise auch eine Datenbankföderierung angeboten (zum Beispiel von UniSQL [www.unisql.com]). Ein weiteres Produkt aus dieser Kategorie ist Illustra [www.illustra.com].

4 Fallstudie: Persistente Objekte bei einer Großbank

In diesem Abschnitt wird die in einem Projekt bei einer Großbank implementierte Lösung vorgestellt und diskutiert. Zunächst werden die relevanten Anforderungen und die Ausgangssituation geschildert. Dann werden Faktoren genannt, die den Einsatz fertiger Produkte nicht ratsam gemacht haben. Die letztlich gewählte Lösung wird am Ende des Abschnitts vorgestellt.

4.1 Ausgangssituation, Anforderungen und Systemumgebung

Das Projekt ist Teilprojekt eines Vorhabens zur Einführung objektorientierter Softwareentwicklungs-Methoden bei einer Großbank. Es ist dabei eine der Projektaufgaben, C++-Objekte persistent zu machen.

Es sollen nicht nur neue Datenbestände entstehen. Es soll zugleich die Möglichkeit bestehen, weiter auf bestehende Datenbestände in relationalen und hierarchischen Datenbanken zuzugreifen. Außerdem muß gewährleistet werden, daß auch traditionelle, mit 3GL-Sprachen neu entwickelte Anwendungen die neuen Datenbestände aus den objektorientierten Anwendungen verwenden können. Der Zugriff auf bestehende Datenbestände muß so erfolgen, daß bestehende Anwendungen, die diese Daten ebenfalls nutzen, auf keinen Fall angepaßt oder auch nur recompiliert werden müssen.

Die Systemumgebung ist typisch für einen Großanwender. Es ist ein Client/Server Konzept mit einem MVS-Host als Zentralrechner vorhanden. Programmierumgebung auf dem Host ist IMS-TM, PL/I, DB2 und IMS-DB. Es sind Clients unter OS/2 vorhanden, die bisher mit C und neuerdings mit C++ programmiert werden. Diese sind in LANs organisiert. Es gibt LAN-Server, die ebenfalls Datenbanken (DB2/2) beherbergen. Die Clients sind über APPC mit dem Host verbunden. Den Clients muß ein integriertes Bild der Unternehmensdaten geboten werden.

Es ist ein mehrschichtiges Client/Server-Modell mit Clients, Abteilungsrechnern und HOST vorhanden. Die Leitungen zwischen HOST und Abteilungsrechnern zwingen durch ihre geringe Bandbreite (9600 Baud) zu rationellem Umgang mit Datentransfers (geblockte Transfers, Verkehrsminimierung). Die angestrebte Lösung für persistente Objekte soll nach einer Erprobungsphase in mehreren

Pilotprojekten zum Unternehmensstandard avancieren. Es existierten umfangreiche Erfahrungen mit Client/Server-Anwendungen unter C mit IMS-Transaktionen.

Aufgabe war es, im Rahmen der neu zu erstellenden Softwareproduktionsumgebung, eine Programmierschnittstelle für persistente Objekte zu schaffen, die für den Programmierer möglichst nahe an der ODMG-Spezifikation [ODMG93] liegt. Dabei soll der Programmierer auf der Ebene fachlicher Objekte möglichst wenig davon wahrnehmen, daß eine Datenbank vorhanden ist. Die Orientierung an [ODMG93] soll den Weg zu Produkten offen halten.

4.2 Woran Produkte heute noch scheitern

Das Team hatte stets das Ziel, den „Eigenbau“ einer Lösung zu vermeiden, da es nicht zu den Kerngeschäften einer Bank gehört, komplexe, systemnahe Software zu entwickeln oder sie auch nur speziell entwickeln zu lassen. Deshalb wurden im großen Umfang Produkte gesichtet. In der Evaluierungsphase, die parallel zur Spezifikation einer Eigenlösung durchgeführt wurde, konnten keine passenden Produkte gefunden werden. Die Gründe dafür sollen hier genannt werden, da sie auch als Checkliste für eine Produktauswahl in ähnlichen Situationen verwendet werden können:

- Die meisten Lösungen unterstützen DB2 [Orf+94] nicht. Selbst wenn zum Beispiel die OS/2 Variante DB2/2 unterstützt wird, wird meist auf dynamic-SQL [Sal93] aufgesetzt. In vielen Unternehmen ist die Benutzung von dynamic-SQL für DB2-Produkte jedoch durch Programmierkonventionen verboten. Dies wird zum Beispiel durch interne Verfahren für Autorisierung und Transaktionskontrolle begründet, die static-SQL voraussetzen.
- Keine Produktlösung auf der Ebene der Objektschicht konnte IMS schreibend und lesend auf der passenden Client-Plattform (OS/2) unterstützen.
- Die meisten Produkte für die Objektschicht, die sich auf relationale Datenbanken stützen, sind nicht in der Lage, mit beliebigen relationalen Tabellenschemata umzugehen. Viele Produkte sind für den *Forward-Engineering-Fall* gebaut, also für den Neubau von Systemen auf der „grünen Wiese“. Sie sind nicht speziell für den Aufsatz auf bestehende, beliebig schlecht normalisierte Datenbestände vorgesehen.
- In objektorientierten Datenbanken spielt der Begriff der Objektidentität [Cat94] eine unverzichtbare Rolle. Wenn Produkte voraussetzen, daß in alle zu verwendenden Datenbanktabellen nachträglich ein Feld für eine Objektidentität eingefügt werden muß, können sie nicht verwendet werden. Bestehende Anwendungen können ein solches Feld nicht kennen. Sie müssen es aber fort-

schreiben. Lösungen, bei denen deshalb tausende bestehender Programme angepaßt werden müßten, sind nicht praktikabel.

- Wir haben kein Produkt gefunden, das in der Lage war, mit einem klassischen Transaktionssystem (IMS oder CICS) als Datenserver umzugehen. Das Besondere an diesen Transaktionssystemen ist, daß der Abschluß jedes Aufrufs (jeder Transaktion) an den Server automatisch auch alle zugehörigen offenen Datenbanktransaktionen abschließt. Es ist also nicht ohne weiteren Aufwand möglich, einzelne Datenbanktransaktionen in einer logischen Transaktion über mehrere Aufrufe des Transaktionsservers zu bündeln.
- Keines der gesichteten Produkte für die Objektschicht war in der Lage, mit einer vielschichtigen Client/Server Architektur umzugehen, also gleichzeitig auf lokale Datenbestände, Abteilungsserver und den MVS-Enterprise-Server zuzugreifen. Diese Fähigkeit ist aber für eine Bank, die über eine solche Architektur verfügt, eine wesentliche Anforderung.

Aus den genannten Gründen mußte der Weg einer Eigenentwicklung beschritten werden.

4.3 Gewählte Architektur

Nachdem Produkte als Lösung ausgefallen waren, wurde eine eigene Lösung (siehe *Abbildung 4*) implementiert. Diese ist, wie in Abschnitt 2 beschrieben, aufgebaut. Hier sollen deshalb nur noch die Aspekte diskutiert werden, die speziell gestaltet werden mußten. Dies geschieht im Schichtenmodell von unten nach oben.

Basis für die Zugriffe sind konventionelle Datenzugriffsmodule (ZS-Module) auf dem MVS-Host. Diese sind in PL/I realisiert, werden aus Beschreibungen automatisch generiert und bieten die üblichen Funktionen einer relationalen Zugriffsschicht (read, insert, update, delete, read-multiple). Die Module sind so geschrieben, daß sie auch von PL/I-Host-Anwendungen aus genutzt werden können.

Diese Module werden unter dem Transaktionsmonitor (IMS-TM) vom Client aus aufgerufen. Vom Client aus müssen meist mehrere Schreiboperationen zu einer Benutzertransaktion zusammengefaßt werden. Dazu werden sie in einer einzigen IMS-Transaktion zusammengepackt und auf dem Host ausgepackt und nacheinander ausgeführt. Die Pufferung wird von einem Kommunikations-Agenten vorgenommen. Benutzertransaktionen werden so erst möglich. Zur Kommunikation zwischen Host und PC wurde kein DRDA-Produkt, wie zum Beispiel DDCS/2 [Orf+94] verwendet, sondern APPC. Dies mag auf den ersten Blick verwundern, ist aber wirtschaftlicher, weil es nach der Rechnung *((Lizenzkosten + Installation + Hardwareaufrüstung) - Programmieraufwand - Wartungskosten)* die preiswertere Lösung war. Diese Rechnung ist sicher nicht für jeden Anwender und jede Anzahl an Lizenzen gültig. Jeder Anwender muß eine solche Kalkulation für sich selbst durchführen.

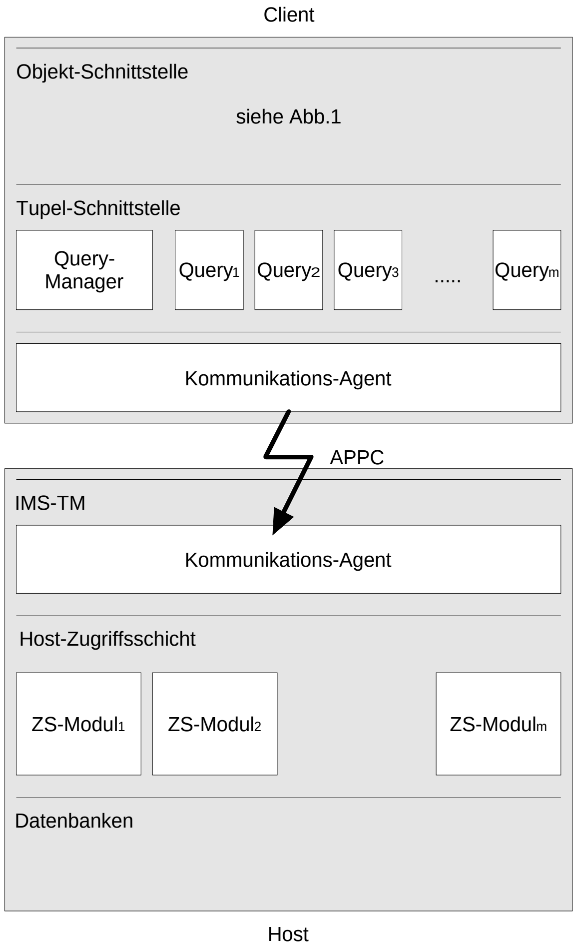

Abbildung 4: Gewählte Architektur Client/Server

Die Tupelschnittstelle auf der Seite des Clients besteht aus einer Menge von Queries, also objektifizierten Zugriffsschicht-Modulen. Um beliebige Anfragen, zum Beispiel für Query-by-Example möglich zu machen, wurde ein Query-Server eingeführt. Durch den Query-Server wird vermieden, daß eine hart codierte Beziehung zwischen höheren Schichten und Queries bestehen muß. Die Kommunikation und die Zugriffsschicht nimmt Rücksicht auf die Geschwindigkeit der Leitungen zwischen Host und PC durch geblockte Zugriffe, besonders beim Mehrsatzlesen.

Die Objektschicht ist so, wie in Abschnitt 2 beschrieben, aufgebaut. Auch hier kam jedoch kein Produkt in Frage, weil Zugriffsschicht-Produkte als Tupelschnittstelle auf dynamic-SQL aufsetzen. Durch den MVS-Host mit DB2 im

Hintergrund war dies keine Option. Eine Projektlösung eines objektorientierten Datenbankherstellers für das spezielle Backend wäre ebenfalls nicht kostengünstiger gewesen und hätte das Projekt verzögert.

Zum Schluß des Abschnitts noch ein technischer Hinweis: Ähnliche Projekte, und der Zwang, static-SQL zu verwenden, haben uns den Einsatz von Codegenerierung auch unter C++ auf dem Client nahegelegt. Generierung von kompletten Komponenten wie Queries und Zugriffsmodulen auf der Ebene von C++ hat sich jedoch aus Gründen der Wartbarkeit und Flexibilität nicht bewährt. Laufzeit-Repositories und Templates sind einer Generierung vorzuziehen. Dies wird auch durch den Aufbau anderer Architekturen [Kel+95, Lip95, Wal+95, NeXT] bestätigt.

5 Erfahrungen und Zusammenfassung

Der gewählte und im Projekt implementierte Ansatz hat sich im ersten Pilotprojekt bewährt. Daß keine, für eine Bank voll passenden Produkte verfügbar sind, ist eigentlich schwer verständlich. Für Integrationstechnologien sollte ein erheblicher Markt bei Großanwendern vorhanden sein. Man sollte gerade angesichts der kurzen Innovationszyklen von etwa 6 Monaten auf dem Markt für objektorientierte Datenbanktechnologien bei eigenen Projekten besonders auf die Modularisierung der Software achten. Saubere, gut dokumentierte Schnittstellen ermöglichen es später, eigene Komponenten durch leistungsfähigere Produkte vom Markt auszutauschen. Einen Anhaltspunkt für sinnvolle Modularisierung bietet dieser Artikel.

Bevor man Produkte einsetzt, sollte man sie einer sehr sorgfältigen Prüfung unterziehen. Die hier vorgestellte Architektur, die Projekterfahrungen und die Produktkategorisierung schaffen dazu eine Ausgangsbasis.

Viel Aufwand ist in der Praxis für die Bewältigung von Problemen zu erbringen, die unmittelbar mit Datenbankfunktionalität nichts zu tun haben. So ist zum Beispiel die Integration einer Zugriffsschicht in das Sicherheits- und Rechnerbetriebssystem ein nicht zu vernachlässigender Punkt. Herkömmliche Transaktionssysteme wie IMS und CICS bieten hier einfache Lösungen und gute Überwachungsmöglichkeiten, die teuer nachgebaut werden müssen, wenn man nicht auf sie verzichten möchte.

Ein wichtiger Zusatznutzen des Projektes war ein Generator für konventionelle Datenbank-Zugriffsschichten in PL/I auf dem HOST. Man kann Zugriffsmodule so konstruieren, daß sie für objektorientierte und konventionelle Anwendungen geeignet sind. Ist bei einem Anwender eine solche konventionelle Datenbank-zugriffsschicht noch nicht vorhanden, kann alleine daraus oft schon der Nutzen für das komplette Projekt gerechtfertigt werden.

6 Literatur

[Bur94] **D. K. Burleson**: *Practical Application of Object-Oriented Techniques to Relational Databases*; John Wiley & Sons, 1994.

[Cat94] **Rick G. G. Cattell**: *Object Data Management*; Addison-Wesley 1994.

[Col+95] **Jens Coldewey, Wolfgang Keller:** *Objektorientierte Datenzugriffe auf dem VAA Datenmanager*, GDV Bonn, 1995 - auch über Compuserve, GO CASEFORUM, File oozs.doc.

[Cot+95] **Cotter, Pottel**: *Inside Taligent Technology*, Addison Wesley 1995.

[Gra95] **Ian Graham:** *Migrating to Object Technology*, Addsion-Wesley 1995.

[Hah+95] **Wolfgang Hahn, Fridtjof Toennissen, Andreas Wittkowski**: *Eine objektorientierte Zugriffsschicht zu relationalen Datenbanken*, Informatik Spektrum 18(Heft 3/1995); Seite 143-151, Springer Verlag 1995

[ODMG93] **Rick G. G. Cattell (Ed.) et. al.**: *Object Database Standard (ODMG 93);* Morgan Kaufmann Publishers, 1993.

[Kel95a] **Wolfgang Keller:** *Problems Reengineering RDBMS to Object Oriented Databases,* Compuserve , GO CASEFORUM, File oidpr.doc.

[Kel95b] **Wolfgang Keller:** *Associations in Object Oriented Access Layers* Compuserve, GO CASEFORUM, File relpro.exe.

[Kel+95] **Michael Keller, Thomas Stalzer:** *Ein Erfahrungsbericht über den Einsatz von VisualAge und Vaser*, OBJEKTspektrum, Vol 2(4), Juli/August 1995.

[Kim95] **Won Kim (Editor)**: *Modern Database Systems,* ACM Press 1995.

[Lip95] **Peter Lipps:** *Enterprise Objects Framework - Fachspezifische Objekte in Open Step*, OBJEKTspektrum, Vol 2(5), September/Oktober 1995.

[NeXT] **NeXT Inc.:** Diverse Unterlagen zum NeXT Enterprise Objects Framework findet man im Internet (World Wide Web) unter www.next.com.

[Orf+94] **Robert Orfali, Dan Harkey**: *Client/Server Survival Guide;* Van Nostrand Reinhold 1994.

[Pre+94] **William Premerlani, Michal R. Blaha**: *An approach for reverse engineering of relational databases;* Communications of the ACM, May 1994, p42(9).

[Sal93] **Joe Salemi**, *PC Magazine's Guide to Client/Server Databases*; Ziff-Davis Publishing, 1993.

[Sou94] **Jiri Soukup**, *Taming C++ - Pattern Classes and Persistence for large Projects*; Addison-Wesley, 1994

[Sto94] **Michael Stonebreaker**, *Object-Relational Database Systems*, Proceedings ObjectWorld 94, London 1994.

[Str91] **Bjarne Stroustroup**: *The C++ Programming Language (2nd. Ed.)*, Addison Wesley 1991.

[Wal+95] **Kim Walden, Jean-Marc Nerson**: *Seamless Object Oriented Software Architecture*, Prentice Hall, 1995.

[www.xxx.com] Hierbei handelt es sich um Verweise auf die World Wide Web-Seiten verschiedener Produkthersteller. Dort findet man meist ausführliche White-Papers, die technische Aspekte der genannten Produkte erläutern.