

Autoreninformation:

Ein Fünf-Schichten-Modell für Business Object Frameworks

Prof. Dr. Hans Albrecht Schmid
Fachhochschule Konstanz, Fachbereich Informatik
Brauneggerstr. 55
78462 Konstanz, Germany
Tel.: +49 - 7531 - 9836-39 (bzw. -12), Fax: +49 - 7531 - 9836-13
Email: schmidha@fh-konstanz.de

Dr. Matthias Riebisch
Deutsche Post AG, AEZ Duesseldorf
Postfach 101265
40003 Duesseldorf, Germany
Tel.: +49 - 211 - 163-1298, Fax: +49 - 211 - 163-1743
Email: M.Riebisch@deutschepost.de

Dipl.-Inf. Harald Ließmann
Bayerisches Forschungszentrum für Wissensbasierte Systeme (FORWISS)
Forschungsgruppe Wirtschaftsinformatik
Am Weichselgarten 7
91058 Erlangen-Tennenlohe, Germany
Tel.: +49 - 911 - 5302-269, Fax: +49 - 911 - 536634
Email: liessmann@forwiss.de,

Dipl.-Ing. Torsten Heverhagen
Universität - Gesamthochschule Essen
Fachbereich Mathematik und Informatik
Datenverwaltungssysteme und Wissensrepräsentation
Schützenbahn 70
45117 Essen, Germany
Tel.: +49 - 201 - 183-3914, Fax: +49 - 201 - 183-2419
Email: the@informatik.uni-essen.de

Ansprechpartner:
Prof. Dr. Hans Albrecht Schmid
Adresse s.o.

Ein Fünf-Schichten-Modell für Business Object Frameworks

Hans Albrecht Schmid
Fachhochschule Konstanz
Email: schmidha@fh-konstanz.de

Matthias Riebisch
Deutsche Post AG
Email: M.Riebisch@deutschepost.de

Harald Ließmann
Bayerisches Forschungszentrum für Wissensbasierte Systeme (FORWISS)
Email: liessmann@wiso.uni-erlangen.de

Torsten Heverhagen
Universität - Gesamthochschule Essen
Email: the@informatik.uni-essen.de

Unter Mitwirkung von A. Bechtel, Haenchen&Partner, Leinfelden; W. Beneke, sd&m, Duesseldorf; A. Chugtai, Systor, Basel; R. Hefter, Kantonalbank, Zuerich; R. Knoll, RWG, Stuttgart; M. Lefering, DKV, Koeln

Inhalt

1	Einleitung	1
2	Fünf-Schichten-Modell	2
2.1	Präsentationsschicht	3
2.2	Trennung der Modellschicht in die Schichten Geschäftslogik und Geschäftsobjekte	3
2.3	Trennung der Speicherung in die Schichten Datenzugriff und Speicherung.....	4
3	Schicht 1: Präsentation	5
4	Schicht 2: Geschäftslogik, Geschäftsvorfälle (Business Logic)	7
5	Schicht 3: Geschäftsobjekte, Unternehmensmodell (Business Entities)	9
6	Schicht 4: Datenzugriff	12
7	Schicht 5: Persistenz	17
8	Zusammenfassung und Ausblick	18

Abstract

Der folgende Beitrag fordert ein Fünf-Schichten-Modell für Business Object Frameworks. Es besteht aus Präsentations-, Geschäftslogik-, Geschäftsobjekt-, Datenzugriffs- und Persistenzschicht. Die Autoren begründen, warum für Business Object Frameworks fünf statt der sonst üblichen drei Schichten nötig sind. Für jede einzelne Schicht werden Aufgaben identifiziert, Probleme beschrieben und die zu treffenden Designentscheidungen diskutiert.

We advocate a 5-layer architecture for business object frameworks with the layers: presentation, business process, business entity, data access, and data storage, instead of the more common 3-layer architecture, and give reasons why the 5 layers are required. We have collected and list, for each of the layers, the responsibilities it must fulfill and problems which may arise, together with a reasoning.

1 Einleitung

Objektorientierte Frameworks sind ein geeignetes Mittel zur Wiederverwendung von Softwarebausteinen. Soll die Anwendungsentwicklung in einem Unternehmen Framework-basiert sein, so gilt es, geeignete Frameworks und Bibliotheken zuzukaufen und eine Architektur zu definieren, wie die zugekauften Teile mit den Eigenentwicklungen zusammenspielen. Für das besondere Anwendungsgebiet betriebswirtschaftlicher Anwendungen gibt es bislang noch wenig Beispiele kommerziell erhältlicher Frameworks, daher wird gerade bezüglich der betriebswirtschaftlichen Funktionalität der Anteil an Eigenentwicklung hoch sein.

Im folgenden stellen wir ein Fünf-Schichten-Modell für Business Object Frameworks (BOF) vor. Wir begründen, warum für BOFs fünf statt der üblicherweise geforderten drei Schichten benötigt werden. Das Modell hilft u. a. bei der Auswahl geeigneter Frameworks, z. B. für die Persistenz der zu entwickelnden Geschäftsobjekte. Es gibt aber ebenso eine Orientierungshilfe zur Entwicklung einer Gesamtarchitektur unternehmensspezifischer Frameworks.

Die Fünf-Schichten-Architektur ist ein Zwischenergebnis des Arbeitskreises Frameworks der GI Fachgruppe 2.1.9. Sie vereint die von den Mitgliedern gesammelten Erfahrungen bei der Untersuchung und Entwicklung von Frameworks zur Entwicklung von kommerziellen Anwendungen. Bewußt wurde von den Details der einzelnen Lösungen abstrahiert, um allgemeine Richtlinien für eine erfolgreiche Entwicklung von BOFs zu geben.

2 Fünf-Schichten-Modell

Das bekannte 3-Tier-Client/Server-Modell (vgl. etwa Orfali et al. 1996) unterscheidet bei Geschäftsanwendungen unterschiedliche Aspekte, nämlich Präsentation, Anwendung und Speicherung. Um Anwendungen nach diesem Modell verteilen zu können, ist es notwendig, entsprechende Schichten in deren Architektur vorzusehen. Das bedeutet, die fachlichen Aspekte z. B. eines Bankkontos sollen nicht daran gekoppelt werden, wie es dargestellt oder gespeichert wird, denn ein Bankkonto kann auf verschiedene Weise verstanden werden. Auch werden bei unterschiedlichen Geschäftsvorfällen, wie etwa beim Abheben eines Betrags und bei einer Kontoänderung, unterschiedliche Teilinhalte dargestellt. Ähnliche Überlegungen gelten für die Speicherung der Kontodaten. Der Ansatz zur Trennung dieser Aspekte ist schon seit über zehn Jahren unter dem Namen MVC bekannt (Krasner et al. 1988, S. 26-49).

Die getrennten Aspekte müssen so aneinander gekoppelt werden, daß etwa beim Starten eines „Abheben“-Geschäftsvorfalles das Speicherungsobjekt die Daten aus der Datenbank in das Geschäftsobjekt einliest und das Präsentationsobjekt sie dort lesen und dem Endbenutzer sichtbar machen kann. Spätestens beim Beenden des Geschäftsvorfalles müssen die geänderten Daten vom Präsentationsobjekt durch Aufrufen fachlicher Methoden an das Geschäftsobjekt übergeben werden und vom Speicherobjekt in die Datenbank zurückgeschrieben werden.

Bei dem vorgestellten Fünf-Schichten-Modell werden sowohl die fachliche wie auch die Speicherungsschicht weiter unterteilt. Die fachliche Schicht enthält geschäftliche Anwendungsabläufe (wie z. B. das Abheben von einem Bankkonto) und die eigentlichen Geschäftsobjekte (wie etwa das Bankkonto). Die Eigenschaften der Anwendungsabläufe unterscheiden sich so stark von denen der Geschäftsobjekte, daß eine Trennung dieser Aspekte erforderlich und zweckmäßig ist.

Bei kommerziellen Anwendungen werden die Daten typischerweise in unterschiedlichen Datenbank- oder auch Dateisystemen gespeichert; eine persistente Speicherung in objektorientierten Datenbanksystemen wird bisher selten durchgeführt. Daher muß zunächst eine Abbildung der Geschäftsobjekte auf eine - zumeist relationale - Datenbankschnittstelle vorgenommen werden, bevor man mit Datenbankkommandos die Geschäftsobjekte aus der Datenbank lesen oder in sie schreiben kann. Die beiden Aspekte, Abbildung der Geschäftsobjekte auf Datenbankschnittstelle und Zugriff auf die Datenbank, sind voneinander unabhängig, so daß sie i.d.R. in verschiedenen Schichten getrennt bearbeitet werden.

Somit sehen wir für BOF die Schichten Präsentation, Geschäftslogik, Geschäftsobjekte, Datenzugriff und Persistenz als geeignet an (siehe Abbildung 1).

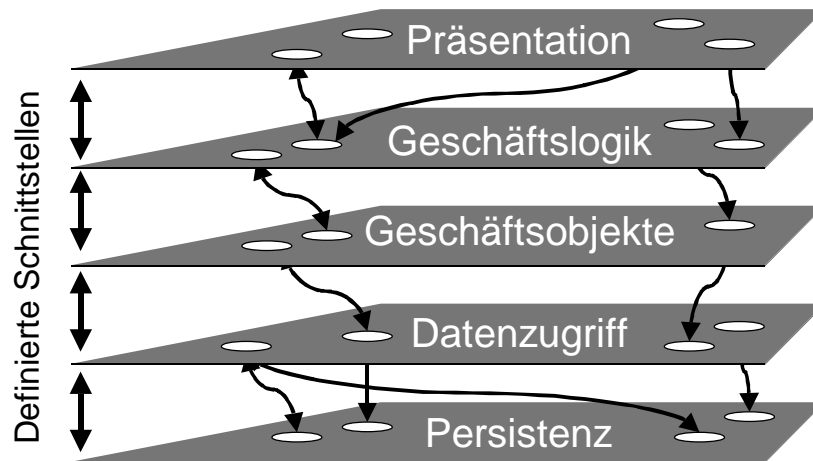


Abbildung 1: Schichtenmodell

Die Schichten sollten jedoch nicht mit den Ebenen („Tiers“) einer Client/Server-Architektur verwechselt werden. Betrachtet man die vorgestellten fünf Schichten unter dem Aspekt einer 3-Tier-Architektur, so könnte z. B. die Geschäftslogik- und die Geschäftsobjektschicht auf den Anwendungs-Tier verlegt und die Datenzugriffs- sowie die Persistenzschicht in der Speicherung zusammengefaßt werden. Der Client, dem nur noch die Präsentation obliegt, wäre somit als Thin Client zu verstehen. Soll der Client mehr Funktionalität (Fat Client) erhalten, kann in ihm beispielsweise zusätzlich die Applikationslogik implementiert werden.

2.1 Präsentationsschicht

Zur Erstellung der Präsentation kann eine auf der gegebenen Plattform vorhandene GUI-Klassenbibliothek verwendet werden, wie etwa AWT oder Swing für Java-Plattformen. Es ist in der Regel nicht zweckmäßig und zu aufwendig, eine GUI-Schicht selbst zu erstellen. Üblicherweise versucht man die Darstellung von anwendungsspezifischen Objekten mit Hilfe von GUI-Widgets der GUI-Klassenbibliothek zu realisieren. Da es sich bei kommerziellen Anwendungen hauptsächlich um die Ein- und Ausgabe von textuellen Daten handelt, verwendet man dazu hauptsächlich vorhandene Textfelder zur Darstellung und Eingabe von Zeichenfolgen.

2.2 Trennung der Modellschicht in die Schichten Geschäftslogik und Geschäftsobjekte

Mit der Trennung von Geschäftslogik und Geschäftsobjekten tragen wir der Tatsache Rechnung, daß reale Geschäftsobjekte wie ein Bankkonto in verschiedenen Geschäftsvorfällen verwendet werden. Ein Geschäftsobjekt sollte genau die fachlichen Regeln

und Prozeduren enthalten, die unabhängig von einem konkreten Geschäftsvorfall zu sehen sind, d. h. in vielen Geschäftsvorfällen vorkommen.

Das Vorgehen ist bei der Modellierung von Unternehmen üblich. In eher funktional geprägten Methodiken (vgl. z. B. ARIS (Scheer 1992)) sind Geschäftsprozesse als Abfolge von Funktionen, die auf bestimmte Daten zugreifen, definiert. Auch objektorientierte Methoden, wie beispielsweise Use Cases (Jacobson et al. 1994) trennen zwischen dem in einem Use Case definierten Geschäftsprozeß und den daran beteiligten Objekten. Diese Differenzierung geht in Architekturen, die nur eine Anwendungsschicht fordern verloren.

Mit der Trennung der Schichten Geschäftslogik und Geschäftsobjekte erreichen wir, daß letztere unabhängig von der Logik, in der sie zum Einsatz kommen, realisiert werden können. Ein Geschäftsvorfall muß die daran beteiligten Geschäftsobjekte kennen; er legt sie an und nutzt üblicherweise die von den Geschäftsobjekten bereitgestellten Dienste. Geschäftsobjekte müssen somit nicht auf elementarer technischer Ebene entwickelt werden, sondern enthalten überwiegend fachliches Wissen.

2.3 Trennung der Speicherung in die Schichten Datenzugriff und Speicherung

Die objektorientierte Repräsentation der Geschäftsobjekte wird in der Zugriffsschicht oft auf eine relationale Datenbankschnittstelle abgebildet. Es kann aber genauso der Fall sein, daß gar nicht direkt auf ein Datenbanksystem zugegriffen wird, sondern daß über die - z. B. auf einem Host bereits vorhandenen - Transaktionen eines Transaktionsmonitors wie CICS nur indirekt auf die Daten zugegriffen werden kann. Innerhalb der Zugriffsschicht ist bekannt, welche Felder eines Geschäftsobjekts welchen Spalten von welchen Relationen in einer relationalen Datenbank, bzw. welche Ergebniswerte welchen Transaktionen für den Zugriff auf einen Host entsprechen. Darüber hinaus ist definiert, welche Felder zur Identifikation eines Geschäftsobjekts dienen und als Schlüssel bei der Speicherung verwendet werden. Im einfachsten Fall entsprechen die Felder eines Geschäftsobjekts einer Zeile, einer Relation oder den Daten einer Transaktion. In komplexen Fällen können die Felder eines Geschäftsobjekts verschiedenen Zeilen verschiedener Relationen entsprechen.

Die Speicherungsschicht stellt eine Schnittstelle zur Datenspeicherung bereit und implementiert diese auf einem oder mehreren verschiedenen Datenbanksystemen. Sie sollte, wenn möglich, eine von einem bestimmten Datenbanksystem unabhängige Schnittstelle bereitstellen. Darüber hinaus sollte eine standardisierte, objektorientierte Schnittstelle wie z. B. JDBC und/oder ODBC verwendet werden, da dann die Zugriffsschicht einfacher zu realisieren und die Speicherungsschicht in Form von unterschiedlichen Treibern meist schon vorhanden ist.

3 Schicht 1: Präsentation

Die Präsentationsschicht stellt die Schnittstelle zum Benutzer dar. Sie dient dazu, den Benutzer durch die Anwendung zu führen, ihm die von der Anwendung zur Verfügung gestellte Funktionen und Daten anzuzeigen und seine Eingabeaktionen entgegenzunehmen. Ihre Aufgabe ist die Unterstützung der Geschäftsvorfälle, so daß die Elemente dieser Schicht sich an den in Schicht 2 definierten Prozessen orientieren.

Innerhalb eines BOF ist eine der Hauptaufgaben dieser Schicht die Darstellung von Geschäftsobjekten. In den meisten Fällen wird dies mit Hilfe von Masken der Benutzungsoberfläche realisiert. Klassische Funktionen, die von solchen Masken bereitgestellt werden, sind

- Eingabe eines neuen Objektes,
- Anzeigen eines vorhandenen Objektes,
- Bearbeiten eines vorhandenen Objektes,
- Löschen eines vorhandenen Objektes und
- Suchen nach/Selektieren von vorhandenen Objekten.

Diese „klassischen“ Funktionen sollen in kommerziellen Anwendungen aber nur Bausteine für die eigentlichen Geschäftsaktivitäten darstellen. Weiterhin sollte man unter „Bearbeiten eines vorhandenen Objektes“ nicht einfach das Ausfüllen von Attributen verstehen. Vielmehr handelt es sich dabei um die Ausführung fachlicher Dienste eines Geschäftsobjektes. Das, was man „bearbeitet“ (also in den Textfeldern einer Maske editiert), sind Parameter, die ein fachlicher Dienst zur Ausführung benötigt.

Meist wird in einer Maske nur eine Aktivität oder ein Bezug zu einem einzigen Geschäftsobjekt dargestellt. Gegenstand eines Geschäftsvorfalles sind aber manchmal nicht nur die Attribute eines Geschäftsobjekts, sondern auch weitere Objekte, die für die Ausführung des Dienstes notwendig sind und beispielsweise über Referenzen angesprochen werden. Die Präsentationsschicht des BOF muß es dem Entwickler daher ermöglichen, auf mehrere Geschäftsobjekte zugreifen zu können.

Geschäftsobjekte werden zum einen oft aus weiteren Objekten zusammengesetzt (aggregiert), zum anderen besitzen sie Beziehungen zu anderen Geschäftsobjekten. Aggregation tritt zum Beispiel in den Stücklisten von Produkten auf, die beschreiben, aus welchen Bestandteilen sich ein Produkt zusammensetzt, oder auch bei der Verknüpfung zwischen einem Geschäftsobjekt Vertrag und den entsprechenden Kunden. Die Unterscheidung zwischen Aggregation und Beziehung sollte auch in der Darstellung der Objekte Ausdruck finden.

Häufig wird angestrebt, daß die aggregierten Bestandteile eines Objektes in einer Maske vollständig angezeigt und bearbeitet werden können. Referenzierte Objekte

werden dagegen nur teilweise dargestellt und können bei Bedarf in weiteren Masken vollständig eingesehen und/oder bearbeitet werden. Ein BOF sollte deshalb dem Anwendungsentwickler die verschiedensten Möglichkeiten für das Mapping von Geschäftsobjekten auf Masken oder Fenster zur Verfügung stellen, ohne ihn dabei einzuschränken. Automatismen für dieses sogenannte GUI-Mapping wurden beispielsweise von Balzert eingeführt und untersucht, haben sich bisher aber nicht in der Praxis bewährt (Balzert 1994, S. 22-35).

Die Verwendung von Maskengeneratoren für die Anwendungsentwicklung hat den Nachteil, daß man auf softwareergonomische Aspekte wenig Einfluß hat. Die Nachbearbeitung der Masken ist meist so aufwendig, daß auf den Einsatz von Maskengeneratoren verzichtet wird. Ein BOF sollte aber Werkzeuge bereitstellen, die die Verwaltung von GUI-Elementen und deren Zuordnung zu Geschäftsobjekten und -aktivitäten unterstützt.

Geschäftsobjekte werden häufig in verschiedenen Sichten dargestellt. Diese beeinflussen die Darstellung und Erreichbarkeit sowohl von Geschäftsvorfällen und fachlichen Diensten als auch von Attributen. Jede Sicht wird von Kontextbedingungen und vom Zustand des Geschäftsvorfalles beeinflusst. Zum Kontext gehören z. B. Berechtigungen der Benutzergruppen für Bearbeitung und Zugriff auf die Geschäftsobjekte. Entsprechend der Berechtigung eines Benutzers sollte sich auch die Darstellung der Geschäftsobjekte anpassen. Zum Kontext gehören außerdem der aktuelle Geschäftsvorfall sowie der Mandantenbezug. Auch der Zustand von Fachobjekten kann zum Kontext gehören, wie z. B. der Zustand eines Kontos „im Haben“, „im Soll“, „im Dispolimit“ oder „im Soll überzogen“. Der Kontext der Darstellung eines Objektes ändert sich während der Bearbeitung einer Geschäftsaktivität nicht.

Masken einer kommerziellen Anwendung sollen aber nicht nur den fachlichen Zustand eines Objektes anzeigen (z. B. Auftragszustand), sondern auch den Zustand der gerade laufenden Aktivität oder Unteraktivität (z. B. ungespeicherte Änderungen). Angezeigt werden solche Zustände durch das Deaktivieren von Funktionen der Benutzungsoberfläche, was als grauer Button oder inaktiver Menüeintrag sichtbar wird. Gibt es z. B. keine ungespeicherten Änderungen, dann ist auch die Funktion „Speichern“ inaktiv.

Werden Attributwerte von Objekten eingegeben oder verändert, so ist es notwendig, diese Werte auf Konsistenz mit anderen eingegebenen oder schon vorhandenen Werten zu überprüfen. In der Präsentationsschicht werden daher Konsistenzprüfungen auf der Ebene einzelner Eingabefelder durchgeführt. Diese beziehen sich z. B. auf den Typ einzelner Attribute und ihre Wertebereiche. Weniger häufig werden auch Zusammenhänge zwischen verschiedenen Attributen in der Präsentationsschicht überprüft, wie z. B. der Bezug zwischen Eingangsdatum und Ausgangsdatum. Konsistenzprüfungen für Beziehungen werden auf jeden Fall in den Schichten

Geschäftslogik und Geschäftsobjekte durchgeführt, um die Speicherung inkonsistenter Zustände zu verhindern. Der Nachteil von Konsistenzprüfungen nur in tieferen Schichten liegt darin, daß sie erst beim Speichern eines Objektes ausgeführt werden. Zu diesem Zeitpunkt hat der Benutzer vielleicht schon eine umfangreiche Maske ausgefüllt, in der er dann nach Inkonsistenzen und deren Folgefehlern suchen muß.

4 Schicht 2: Geschäftslogik, Geschäftsvorfälle (Business Logic)

Zur Entwicklung betriebswirtschaftlicher Anwendungen ist es wichtig, daß Geschäftsabläufe möglichst einfach und in direkter Entsprechung zu ihrer Spezifikation realisiert werden können. Die Unterstützung muß hier weiter gehen, als lediglich Callbacks für Buttons bereitzustellen. Unsere Forderung ist, daß ein BOF Mechanismen zur Realisierung von Geschäftsvorfällen bereitstellt, wie etwa zum Abheben eines Betrags, einer Kontoänderung oder der Kündigung einer Krankenversicherung. Geschäftsvorfälle sind die größten unterstützten Geschäftslogikeinheiten. Ein Ablauf von Geschäftsvorfällen bildet einen Geschäftsprozeß. Die explizite Modellierung einzelner Geschäftsprozesse durch ein BOF ist nicht erforderlich. Die in kommerziellen Anwendungen benötigte Flexibilität zur Änderung von Workflows wird schon dann unterstützt, wenn ein BOF nur die grundlegenden Elemente bereitstellt, deren Abfolge leicht zu ändern ist.

Ein Geschäftsvorfall läßt sich in den (bedingten) Ablauf von einzelnen (atomaren) Geschäftsaktivitäten unterteilen. Zum Beispiel teilt sich der Geschäftsvorfall „Kündigung einer Krankenversicherung“ in die Geschäftsaktivitäten „Gültigkeit Kündigung prüfen“, „wenn gültig, Risiko prüfen“, „wenn Risiko gut, Versuch Kündigung rückgängig zu machen“, „sonst Kündigung akzeptieren“ auf. Ein Geschäftsvorfall ist zwischen der Ausführung von Geschäftsaktivitäten unterbrechbar und wiederaufnehmbar. Somit kann der Geschäftsvorfall „Kündigung einer Krankenversicherung“ aus verschiedenen Gründen nach der Gültigkeitsprüfung der Kündigung unterbrochen und an einem anderen Tag wieder aufgenommen werden.

Die Festlegung von Transaktionsgrenzen ist eine der wichtigen Aufgaben dieser Schicht. Eine Geschäftsaktivität soll in Bezug auf ihre Auswirkungen atomar oder, anders ausgedrückt, eine „unit of work“ sein. Sie stellt damit eine fachliche Transaktion dar, die entweder in ihrer Gesamtheit durchgeführt wird und einen konsistenten Zustand als Ergebnis liefert oder keine Auswirkungen hat. Sie wird technisch durch eine Datenbank-Transaktion realisiert. Eine Geschäftsaktivität ist nicht abrechbar und wiederaufnehmbar; sie soll während ihrer Durchführung durch ein „cancel“-Kommando zurückgesetzt werden können. Es kann jedoch der Fall auftreten, daß ein Sachbearbeiter bei der Durchführung einer Geschäftsaktivität unterbrochen wird und sie am Bildschirm in einem Zwischenzustand stehen bleibt, bevor er sie nach längerer Zeit weiterführt und beendet (langdauernde fachliche Transaktion).

In einer Geschäftsaktivität kann mehr als ein Geschäftsobjekt bearbeitet werden, wie etwa eine Abteilung mit ihren Mitarbeitern oder ein Kunde mit seinen Versicherungsverträgen.

Eine Geschäftsaktivität läßt sich in eine Folge, oder möglicherweise auch Menge von Unteraktivitäten aufteilen, die keine „unit of work“ darstellen. Die Unteraktivitäten werden von einem Sachbearbeiter nacheinander ausgeführt. So besteht der Geschäftsvorfall „Betrag von Konto abheben“ aus einer einzigen Geschäftsaktivität, die in die Folge der Unteraktivitäten „Identität des Abhebenden prüfen“, „Berechtigung des Abhebenden prüfen“ und „Betrag vom Konto entnehmen“ gesplittet werden kann. Bei der letzten Unteraktivität wird der oben angesprochene Dienst des Geschäftsobjekts aufgerufen. Wird eine Unteraktivität erfolgreich abgeschlossen, geht es jeweils mit der nächsten Unteraktivität weiter. Optional kann vorgesehen werden, daß eine Unteraktivität nur bedingt, d. h. in Abhängigkeit vom Resultat einer vorhergehenden Unteraktivität, ausführbar und damit anwählbar ist, oder daß, falls sinnvoll, mehrere Unteraktivitäten anwählbar sind und somit der Sachbearbeiter größtmögliche Freiheit bei der Ausführungsreihenfolge hat. Unteraktivitäten rufen fachliche Dienste von Geschäftsobjekten auf (siehe Schicht 3).

Ein BOF sollte häufig vorkommende Aktivitäten als Unteraktivitäten oder als fachliche Dienste von Geschäftsobjekten anbieten. Beispiele dafür sind das Anlegen eines neuen Geschäftsobjekts mit Eingabe der entsprechenden Daten, die Suche eines Geschäftsobjektes in der Datenbank und die Auswahl eines Geschäftsobjektes aus einer Auswahlliste. Ebenso sollte es optional vordefinierte parametrisierbare Standard-Aktivitätsabläufe geben, wie etwa die Suche eines Geschäftsobjektes in der Datenbank und, falls mehr als eines den Suchkriterien entsprechen, die anschließende Auswahl aus einer Auswahlliste. Sie bestehen aus einer Abfolge von Unteraktivitäten, die in der gleichen Weise in verschiedenen Geschäftsaktivitäten vorkommt und sich somit wiederverwenden läßt. Die Parametrisierbarkeit (z. B. bezüglich der Klasse des Geschäftsobjekts und der Suchkriterien) ist zur Anpassung an eine konkrete Geschäftsaktivität erforderlich.

Sinnvollerweise wird die Präsentation mit der vorgestellten Menge von Geschäftsvorfällen, Geschäftsaktivitäten und Unteraktivitäten verbunden. Einem Geschäftsvorfall ist ein Fenster oder Unterfenster mit Anzeige und Auswahlmöglichkeit der einzelnen Geschäftsaktivitäten zugeordnet, welche im (Unter-) Fenster des zugehörigen Geschäftsvorfalles gezeigt wird. Dieses stellt, wenn vorhanden, zugehörige Geschäftsobjekte dar, oder die einer Unteraktivität zugeordnete Maske zeigt das entsprechende Geschäftsobjekte an.

Ein BOF stellt kein Workflowsystem dar; die mit ihm realisierten Geschäftsvorfälle sollten jedoch von einem Workflow-Ablauf-Steuerungssystem aufgerufen werden können. Es erscheint sinnvoll, die Steuerung eines Geschäftsprozesses mit einer separaten Menge von Objekten oder einem Workflow-Management-System zu

realisieren und damit Geschäftsvorfälle in der richtigen Abfolge aufzurufen. Im Gegensatz dazu erscheint es jedoch kaum sinnvoll, mit einer Workflow-Ablauf-Steuerung einen einzelnen Geschäftsvorfall zu realisieren und über diese feingranulare Geschäftsaktivitäten aufzurufen.

5 Schicht 3: Geschäftsobjekte, Unternehmensmodell (Business Entities)

In dieser Schicht werden fachliche Objekte mit ihrem Verhalten, ihren Attributen und Zuständen modelliert. Daten und Zustände werden gekapselt und sind über fachliche Dienste zu beeinflussen. Das Verhalten des Geschäftsobjekts wird durch fachliche Dienste festgelegt, da diese die Zustandsübergänge steuern. Beispiele für diese sind „Kontostand erhöhen“, „Fahrtroute errechnen“ oder „Prognose erstellen“. Durch Aufruf solcher Dienste müssen sich die Geschäftsvorfälle vollständig abdecken lassen. Nach deren Durchführung muß die Konsistenz der Geschäftsobjekte gewährleistet sein. Diese Schritte eines fachlichen Dienstes sind durch Transaktionen abgedeckt, da sie Bestandteil der Geschäftsaktivitäten sind.

Die von einem Geschäftsobjekt angebotenen fachlichen Dienste sind so zu gestalten, daß sie von vielen Geschäftsvorfällen (wieder-)verwendet werden können. Eine stilistische Einheitlichkeit unterstützt ihre Wiederverwendung. Auch hier gilt der Grundsatz: Dienste sollen so einfach wie möglich und so komplex wie nötig gestaltet werden. Komplexere Dienste, die nur von einem oder wenigen Geschäftsvorfällen benutzt werden, sollen durch den Geschäftsvorfall selbst oder durch eine Geschäftsaktivität gebildet werden. So sollte ein Geschäftsobjekt „Bankkonto“ keinen Dienst „Betrag abheben“ anbieten, da dies ein spezifischer Geschäftsvorfall zwischen zwei Konten ist. Es gibt ähnliche Geschäftsvorfälle wie „Scheck einreichen“ oder „Überweisung bearbeiten“, bei denen auch Geld vom Konto entnommen wird. Ein Dienst, der allen Vorfällen gemeinsam ist und daher vom Geschäftsobjekt zur Verfügung gestellt werden sollte, ist das Entnehmen eines Betrags vom Konto. Dieser überprüft, ob nach den mit dem Konto verbundenen Geschäftsregeln und dem Kontostand das Entnehmen des gewünschten Betrags überhaupt zulässig ist. Damit muß diese Überprüfung nicht mehr in jedem Geschäftsvorfall vorgenommen werden. Dies hat die Vorteile, daß die Überprüfung wiederverwendet wird, stets einheitlich ist und Abänderungen nur an einer Stelle durchgeführt werden müssen.

Die Geschäftsobjekte prüfen die Konsistenz ihrer Attribute (einfache und zusammengesetzte Attribute, Zustände, Beziehungen zwischen Attributen, Beziehungen zu anderen Geschäftsobjekten) bei der Durchführung von fachlichen Diensten. Diese Prüfung wird selbst dann wiederholt, wenn sie in anderen Schichten wie der Präsentationsschicht bereits ausgeführt wurde. Die Ausnahmebehandlung erfolgt idealerweise von beiden Schichten aus in den anfordernden Geschäftsvorfällen. Die Realisierung ist von den Möglichkeiten der Programmiersprache abhängig, in Java

ist dies mit dem Erzeugen von Exceptions innerhalb von Methoden relativ elegant möglich.

Außerdem können spezielle Dienste für Geschäftsobjekt-übergreifende Konsistenzprüfungen erforderlich sein, die von Geschäftsvorfällen aus angefordert werden. So sind z. B. häufig Prüfungen eines Adreßbestands auf Dubletten erforderlich.

Fachliche Dienste sollen zwar einfach sein, sie müssen jedoch immer einen Umfang haben, der die Erhaltung der Konsistenz erreichen läßt. Set-Methoden als einfachste Zugriffsmethoden stellen dies nicht sicher. Werden bei der Veränderung einer Adresse Straße und Stadt nacheinander und unabhängig verändert, ist der Bezug zwischen Straße und Stadt zwischenzeitlich nicht konsistent.

Bei der Darstellung von komplexen Geschäftsobjekten werden in manchen Fällen auch einfache Zugriffsmethoden auf Attribute und Zustände der Geschäftsobjekte benötigt. Dies tritt immer dann auf, wenn Redundanzen in der Schicht enthalten sind. Andere Geschäftsobjekte müssen diese Methoden während der Ausführung ihrer fachlichen Dienste benutzen, um Redundanzen herzustellen. Die Zugriffsmethoden können also nicht „private“ bezogen auf das Objekt sein, im Sinne der Programmiersprache C++. Solche Dienste sind von anderen Schichten nicht zugreifbar und damit private Dienste der Schicht 3. Bei der Benutzung der Methoden muß das anfordernde Geschäftsobjekt die Konsistenz sicherstellen.

Als Beispiel soll hier die Redundanz durch doppelte Verkettung zwischen den Geschäftsobjekten Arzt a und Patient p angeführt werden. Der Arzt hat $0..n$ Patienten, der Patient hat 0 oder 1 Arzt. Der fachliche Dienst des Patienten p „löse Verbindung zum Arzt“ löscht das Attribut $p.artzt$ und muß den Verweis auf sich aus der Kollektion $a.patienten$ entfernen. Dieses Entfernen erfordert eine direkte Zugriffsmethode wie $removePatient(p)$. Erst damit ist die Konsistenz hergestellt und die Transaktion vollständig. Ein Aufruf des fachlichen Dienstes des Arztes „löse Verbindung zu Patient p “ würde wiederum den Eintrag $p.artzt$ löschen müssen.

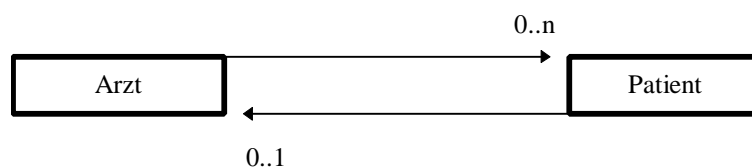


Abbildung 2: Beziehung zwischen den Geschäftsobjekten Arzt und Patient

Komplexe Geschäftsobjekte werden dadurch gebildet, daß einfachere Geschäftsobjekte mit jeweils eigenem Verhalten durch Aggregation verbunden werden. Ein Beispiel ist hier die Darstellung einer Maschine als Aggregation mehrerer Teile.

Schwächere Beziehungen zwischen Geschäftsobjekten werden durch Assoziationen wie Referenzen und Benutzung gebildet. Mit diesen Beziehungen werden alle

relevanten Beziehungen der abzubildenden fachlichen Objekte modelliert. Beispiele sind hier die Beziehungen zwischen Kunde und Bank und zwischen Rechnungsposition und Artikel.

Das Zugriffsverhalten wird vom BOF so umgesetzt, daß es von der Art der Beziehung abhängt, z. B. beim Löschen oder beim Kopieren von komplexen Geschäftsobjekten. Schwächere Beziehungen führen nicht zum Löschen verbundener Objekte, bei Aggregationen ist meist das Löschen aller enthaltenen Objekte erforderlich. Die Art der Beziehung wirkt sich damit implizit auch auf die Grenzen von Transaktionen aus. In den Programmiersprachen wird die Unterscheidung von Beziehungen unterschiedlich gut unterstützt. In C++ würde dies beispielsweise wie folgt implementiert werden:

```
class Patient {
    Adresse privatAdresse; //Aggregation
    Arzt *hausarzt; //Referenz
}
```

Die in Geschäftsobjekten gekapselten Daten und Beziehungen müssen häufig versioniert werden. Ein Kunde kann z. B. seine Hauptwohnung an mehreren Adressen nacheinander haben. Die Historie von Attributen kann durch eine Kollektion von Datum-Wert-Paaren implementiert werden. In vielen Fällen muß die Gültigkeit von Geschäftsobjekten zeitlich begrenzt sein. Hier kann eine Umsetzung über Kollektionen von Geschäftsobjekten oder über Beziehungen erfolgen.

Geschäftsobjekte werden von verschiedenen Geschäftsvorfällen benutzt. Dazu sind verschiedene Sichten auf die Geschäftsobjekte notwendig, wobei jeweils unterschiedliche Attribute und Beziehungen dargestellt werden. Die Sicht auf ein Geschäftsobjekt hängt von der Rolle ab, die es im aktuellen Zustand des Geschäftsvorfalles einnimmt. Die Sicht wird auch von den Beziehungen zu anderen Geschäftsobjekten und ggf. den Attributen und Zuständen beeinflusst. Zustände führen häufig dazu, daß sich die Rolle eines Geschäftsobjekts verändert. Beispiele sind hier der Kunde in der Rolle als Sicherheitsgeber und in der als Wertpapierverkäufer, ein Arzt in den Rollen angestellter und selbständiger Arzt sowie ein Konto im Guthaben oder im Soll.

Die Realisierung von Mehrsprachigkeit und Mehrwährungsfähigkeit fließt ebenfalls in die Implementierung von Geschäftsobjekten ein.

Die Möglichkeit, auf Mengen von Geschäftsobjekten zuzugreifen, ist in dieser Schicht zu implementieren. Eine Menge von Geschäftsobjekten enthält in der Regel nur Verweise auf Proxies (vgl. das entsprechende Design Pattern von Gamma et al. (1995)). Dadurch wird eine kostenoptimale Strategie des Zugriffs nach Bedarf möglich, die insbesondere bei großen Ergebnismengen erforderlich ist. Die Strategie kann anhand des Umfangs gesteuert werden. Im Fall der Ergebnismenge bei Suchvorgängen ist nicht unbedingt eine (aufwendige) Zählung erforderlich, häufig kann die Strategie

durch die Präzision der Suchkriterien definiert werden.

In objektorientierten Systemen ist ein eindeutiger Zugriff auf Objekte durch die Referenz gegeben. Trotzdem ist es in der Praxis vorteilhaft, einen Schlüssel in der Art einer Objekt-ID zu nutzen, um eine Verwaltung von Mengen von Objekten zu ermöglichen. Diese Objekt-ID muß systemweit für alle Geschäftsobjekte eines Typs eindeutig sein.

Die Bildung von Objekt-IDs ist auf verschiedenen Wegen möglich. Viele Geschäftsobjekte verfügen über natürliche Schlüssel, wie Rechnungsnummer, Kundennummer und Steuernummer. Für alle anderen Geschäftsobjekte ist die Bildung einer Objekt-ID als abgeleiteter Schlüssel aus einer Aggregation von Attributen möglich. Hier sind die gleichen Attribute zu nutzen, die bei relationaler Modellierung eines Geschäftsobjekts in einer Tabelle als zusammengesetzter Schlüssel verwendet würden.

Die Nutzung von logischen Schlüsseln relationaler Datenbanken als Objekt-ID ist nachteilig, wenn Änderungen z. B. im Persistenzmechanismus erfolgen sollen, wie die lokale Pufferung von Daten auf dem Client oder eine Speicherung in sog. Flat Files.

Suchobjekte stellen eine spezielle Form von Geschäftsobjekten dar. Sie enthalten die Suchkriterien, führen die Suche in der persistenten Menge von Geschäftsobjekten aus und stellen die Kollektion von gefundenen Geschäftsobjekten bereit. Sie korrespondieren mit Suchvorgängen in der Schicht Geschäftsvorfall sowie mit Filtern und Auswahllisten in der Präsentationsschicht.

Die Darstellung von Suchkriterien erfolgt beispielsweise in Form von WHERE-Klauseln in OQL-Notation (zu OQL vgl. Cattell et al. 1997). Diese Notation erlaubt komplexe Suchkriterien und ermöglicht trotzdem eine einfache Umsetzung auf relationale Strukturen in Schicht 4. Gegenüber der sich bei relationaler Speicherung anbietenden Notation in SQL hat OQL den Vorteil, daß sich Veränderungen der Persistenzschicht wie z. B. im Datenmodell oder die Realisierung der Persistenz mit anderen Technologien nicht auf die höheren Schichten auswirken. Die WHERE-Bedingungen werden aufgrund der Filtereinstellungen der Präsentationsschicht generiert. Ein Beispiel ist hier:

```
WHERE patient.krankheit = Grippe AND arzt.fachgebiet = Orthopaedie
```

Die Suche wird von den Schichten 4 und 5 durchgeführt, die eine Ergebnismenge als Kollektion zurückliefern.

6 Schicht 4: Datenzugriff

Diese Schicht bietet datenbankunabhängige Methoden zur persistenten Speicherung der

in Schicht 3 definierten Geschäftsobjekte an. Es sollte demnach keine Rolle spielen, ob die Speicherung in einem objektorientierten Datenbanksystem (ODBMS), einer relationalen Datenbank (RDBMS), in sogenannten „Flat Files“ oder durch irgendeinen anderen Mechanismus erfolgt.

Im Gegensatz zu den ODBMS, die Objekte direkt persistent abbilden, hat man im Falle von RDBMS ein vom Objektmodell differierendes Datenmodell zu erwarten. Relationale Systeme sind aber häufig bereits im Unternehmen etabliert und können bzw. sollen auch nicht durch objektorientierte ersetzt werden. Zwei Vorgehensweisen zur Integration relationaler Daten in Objektsysteme sind hier üblich (vgl. Sutherland et al. 1993, S. 326-333):

1. Direkte Verbindung zwischen den Objekten und der relationalen Datenbank, ohne deren Schema zu verändern.
2. Zwischen Objekten und RDBMS steht ein Objektmanager, der die persistenten Objekte in relationale Tabellen wandelt, so daß der Entwickler nicht selbst SQL-Kommandos an die Datenbank formulieren muß.

Schicht 4 entspricht dem in Punkt 2 genannten Vorgehen. Damit sich die Entwickler von Geschäftsobjekten möglichst wenig mit der Persistenz ihrer Objekte auseinandersetzen müssen, sollte eine Schnittstelle angeboten werden, die folgenden Forderungen genügt:

1. Erzeugen eines Geschäftsobjekts

Das Erzeugen von Geschäftsobjekte geschieht idealerweise über einen Factory-Mechanismus (vgl. das entsprechende Design Pattern von Gamma et al. (1995)). Im Falle einer relationalen Datenbank könnte für ein Geschäftsobjekt „Person“ beispielsweise in einer entsprechenden Relation „Person“ ein neuer Datensatz angelegt werden. Die Attribute der Relation bilden in der Regel eine Übermenge zu den Objektattributen des Geschäftsobjekts. Außer den direkt korrespondierenden fachlichen Attributen wird eine spezielle Identifikation gespeichert werden müssen, die in der relationalen Datenbank als Primärschlüssel gelten kann. Im Idealfall kann dies die in Schicht 3 geforderte Objekt-ID sein, u. U. kommen aber Datenbank-spezifische Kennungen hinzu. Dies kann dann sinnvoll sein, wenn z. B. aus Gründen der Zugriffsgeschwindigkeit semantisch eng verbundene Objekte (etwa durch Vererbung) in einer Relation zusammengefaßt werden. Ein Beispiel wären unterschiedliche Arten von Geschäftspartnern, z. B. Endkunden, Wiederverkäufer, Rohmaterial-Lieferanten, Zulieferer etc., die in Dokumenten wie Rechnungen oder Lieferscheinen zu finden sind. Werden häufig Listen zur Planung und Kontrolle der Geschäftsabläufe generiert, mag es sinnvoll sein, alle Objekte der aufgeführten Typen in einer einzigen Relation zu speichern.

Aufgabe dieser Schnittstellenfunktionalität ist es, das erzeugte Objekt so an Schicht

3 weiterzureichen, daß aus deren Sicht die Erzeugung vollkommen transparent bleibt.

Bestehen vielfältige Beziehungen zu weiteren Geschäftsobjekten, müssen bei der Bereitstellung einer Menge alle diese verbundenen Geschäftsobjekte ebenfalls durch die Schichten 4 geladen werden. Eine Kostenreduktion solcher Ladevorgänge kann durch Nutzung von Proxies in den verfolgten Beziehungen erfolgen. Die Tiefe der Verfolgung beim Laden ist dann steuerbar. Ein Bereitstellen erfolgt erst dann, wenn der direkte Zugriff auf Inhalte von Geschäftsobjekten erforderlich ist. Im Java-Framework San Francisco von IBM werden dazu sogenannte Stubs zur Umsetzung einer Ladestrategie genutzt, die verfolgte Tiefe der Beziehungen wird durch einen Stub Level bestimmt (IBM 1998).

Dieses Verfahren ist insbesondere dann interessant, wenn durch die in Schicht 3 beschriebenen Suchobjekte eine große Zahl von Geschäftsobjekten angefordert wird. Beispielsweise soll eine umfangreiche Artikelliste in einer Listboxen generiert werden. Würden dann bei einem Großhändler alle verfügbaren Artikel gleichzeitig aus dem persistenten Speicher geladen, könnte es zu ernsthaften Performanzproblemen kommen, die durch den Einsatz von Proxies gemildert würden.

2. Löschen eines Geschäftsobjekts

Zu unterscheiden ist zwischen dem Entfernen eines Verweises, der lokal zum Zugriff auf ein Geschäftsobjekt benutzt wird und der Zerstörung eines Geschäftsobjekts. Im ersten Fall kann das Löschen dem entsprechenden Mechanismus der Programmiersprache überlassen werden. Im zweiten Fall muß zudem der Eintrag im persistenten Speicher entfernt werden, z. B. durch ein DELETE FROM Kommando an eine SQL-kompatible Datenbank.

Problematisch beim Löschen von Geschäftsobjekten ist die Konsistenzsicherung. Abhängige Objekte müssen auf ihre weitere Daseinsberechtigung untersucht werden. Enthält beispielsweise ein Produkt eine Stückliste, so ist zu überprüfen, ob darin enthaltene Teile noch anderweitig benötigt werden oder ob sie ebenfalls gelöscht werden können.

3. Aktualisieren des Zustandes aus dem persistenten Speicher

Um ein einmal persistent erzeugtes Geschäftsobjekt zu laden, sollte eine Methode implementiert sein, die es erlaubt, den Zustand eines über den Schlüssel eindeutig zu identifizierenden Objektes zu aktualisieren bzw. wieder herzustellen. Der Schlüssel korrespondiert mit der global eindeutigen Objekt-ID.

Die Methode kann einerseits Bestandteil der Factory sein. In diesem Fall würde sie als Rückgabewert das gefundene Objekt liefern. Andererseits kann auch eine Implementierung im Geschäftsobjekt in Betracht gezogen werden, was nützlich sein

kann, wenn viele verteilte Versionen eines Geschäftsobjektes existieren, die sich über einen Observer-Mechanismus synchronisieren müssen.

4. Schreiben des aktuellen Zustandes in den persistenten Speicher

Nach einer Änderung des Zustandes eines Geschäftsobjekts, muß dieser wiederum persistent gemacht werden. Diese Funktionalität sollte Bestandteil des Objekts sein - nicht der Factory -, da sie erwartungsgemäß häufig genutzt wird, und die Factory sich sonst sehr leicht zum Flaschenhals entwickelt. Folgende Varianten sind denkbar (vgl. z. B. auch Lefevre 1998):

- Es existiert eine allgemeine Zugriffsklasse, die eine Menge von Methoden für alle denkbaren Geschäftsobjekte vorhält. Beispiele für solche Methoden sind das Erzeugen einer Verbindung zur Datenbank oder das Ausführen einer Datenbankabfrage. Wir sehen solche Methoden eher als Bestandteil einer separaten Schicht an (Schicht 5).

Ein konkretes Geschäftsobjekt würde somit ein Exemplar der für die Persistenz zuständigen Klasse enthalten und das Mapping seiner Daten mit Hilfe der verfügbaren Methoden vornehmen. Ein solches Vorgehen bringt zunächst den Vorteil, die Implementierung der für die Persistenz notwendigen Methoden nicht für jedes Geschäftsobjekt wiederholen zu müssen. Nachteilig ist jedoch, daß u. U. alle Geschäftsobjekte über das gleiche Verfahren gespeichert werden, was zu Performance-Problemen führen kann, z. B. wenn die Objekte als Binary Large Objects in eine relationale Datenbank geschrieben werden, statt die einzelnen Attribute aufeinander abzustimmen.

- Alle persistenten Geschäftsobjekte implementieren eine gemeinsame Persistenz-Schnittstelle (im Sinne eines Java Interface), die vorschreibt, welche Methoden für das Mapping der Objektdaten in die Datenbank zuständig sind und im Objekt vorhanden sein müssen. Ein solcher Ansatz, der auch sichere Objektzustände berücksichtigt, wird von Reese für die Speicherung von Java-Objekten vorgeschlagen (vgl. Reese 1997, S. 32-33).

Der Vorteil liegt darin, daß ein Mapping für jede Klasse individuell stattfindet. Möchte man mehrere Plattformen unterstützen, so ist eine abstrakte Klasse im Vergleich zu einer Schnittstelle zu bevorzugen, da so unterschiedliche Mechanismen über das in Java übliche Peer-Konzept (verwandt mit dem von Gamma et al. 1995 vorgeschlagenen Facade-Pattern) für den Benutzer transparent in die Klasse integriert werden können. Die Peers wären nach unserer Terminologie Bestandteil von Schicht 5.

5. Transaktionsverwaltung

Die von den Datenbanken bekannte Transaktionsproblematik überträgt sich nicht

nur auf objektorientierte Systeme, sondern kann durch komplexe Objektinteraktionen sogar noch verschärft werden. Es kommen insbesondere Probleme der globalen Serialisierbarkeit, der globalen Atomarität sowie der globalen Deadlock-Erkennung und –Vermeidung (vgl. Conrad 1997, S. 197-205).

Neben den üblichen flachen ACID-Transaktionen, ist es von Vorteil, auch verschachtelte oder Multi-Level-Transaktion zu unterstützen. Dabei ist für ein BOF zu entscheiden, ob diese relativ aufwendigen Mechanismen durch semantische Vereinfachung kostengünstiger implementiert werden können. In Schicht 4 sollten entsprechende Methoden zur Verfügung stehen, um zumindest die dem ACID-Prinzip entsprechenden Mechanismen „begin“, „commit“ und „rollback“ nutzen zu können, die von den meisten RDBMS angeboten werden. Sobald Transaktionsfragen semantischer Natur sind, sollten man sie allerdings in Schicht 3 oder häufiger noch in Schicht 4 klären, oft sind so auch komplexe Transaktionen zu vermeiden.

Darüber hinaus ergibt sich die Frage der Sperrgranularität, d. h. sollen für eine Transaktion einzelne Objektattribute oder ganze Objekte gesperrt werden. Auch hier ist es zweckmäßig, sich an den existierenden Möglichkeiten des Datenbanksystems zu orientieren, bevor eigene Mechanismen entwickelt werden.

Darüber hinaus können in Schicht 4 aber auch weitere (optionale) Mechanismen angeboten werden:

6. Geschäftsregeln

Geschäftsregeln, die für einen konsistenten Zustand auf u. U. unterschiedlichen Granularitätsebenen sorgen. Auf Attributebene muß z. B. bezüglich eines Datums ein definierter Wertebereich vorhanden sein, so daß etwa der 30.02.1999 als ein ungültiges Datum erkannt wird. Auf Objektebene könnte man festlegen, daß zwei Attribute bestimmten Regeln folgen müssen, z. B. daß ein Startdatum zeitlich immer vor einem Enddatum liegen muß. In einem weiteren Zusammenhang wäre zu kontrollieren, daß zwei Personen niemals die selbe Sozialversicherungsnummer haben können.

7. Performanzregeln

Performanzregeln, die beispielsweise festlegen, welche assoziierten Objekte sofort und welche erst bei Zugriff erzeugt werden. Oft enthalten Geschäftsobjekte Verweise auf weitere Objekte, die sich ebenfalls im persistenten Speicher befinden. Die einfachste Methode wäre, diese Beziehungen zu traversieren und alle betroffenen Objekte im flüchtigen Speicher zu erzeugen. Dies würde bei umfangreichen Kollektionen eine große Zahl an Zugriffen auf den persistenten Speicher bedeuten, was die Performanz eines Systems in der Regel stark belastet. Ein intelligenterer, aber aufwendigerer Mechanismus ist die oben genannte Technik

der Stub Levels.

Ein besonderes Augenmerk verdient das Umsetzen der Objektdaten in den persistenten Speicher, was allgemein als Mapping oder Schema Mapping bezeichnet wird. Als Beispiel seien die IBM San Francisco Frameworks genannt, die in ihrem Foundation Layer einen transparenten Mechanismus zur Persistenz der San Francisco Business Objects bieten (vgl. IBM 1998). Ein ähnliches Vorgehen findet man auch bei Smalltalk bzw. Java RMI. Kern dieses Vorgehens in den San Francisco Frameworks sind die beiden Methoden

```
externalizeToStream(BaseStream) und  
internalizeFromStream(BaseStream) .
```

Demnach wird der Inhalt eines Geschäftsobjekts serialisiert, so daß sich unterschiedliche persistente Speicher hinter San Francisco Anwendungen verbergen und diese auch ohne Änderung des Programmcodes über den Schema Mapper, ein mit dem Framework ausgeliefertes Werkzeug, gewechselt werden können.

Sollte es notwendig sein, Daten aus sogenannten Legacy-Systemen zu übernehmen oder gar via Wrapping in eine neue Anwendungslandschaft zu integrieren, so empfiehlt es sich, die entsprechenden Mechanismen in Schicht 4 zu plazieren, so daß von den darüberliegenden Schichten ein transparenter Zugriff auch auf diese Daten möglich wird.

Ein Default-Schema-Mapping kann sich als Flaschenhals erweisen, wenn Geschäftsobjekte ohne Rücksicht auf ihre Struktur, z. B. als Binary Large Objects, in die Datenbank geschrieben werden. Besser ist es, wenn sich die Objektattribute in den Datenbankattributen zu widerspiegeln. Der Schema-Mapper, der als grundlegender Baustein der Schicht 4 betrachtet werden kann, legt demnach nicht unwesentlich das Laufzeitverhalten des Frameworks fest. Da sich der Anwendungsentwickler in den oberen Schichten nicht mehr mit den Persistenzmechanismen auseinandersetzen möchte und sollte, muß ein Default-Mapping sorgfältig gewählt werden.

Generell ist zu überlegen, ob für BOFs Basismechanismen, wie sie in Schicht 4 angeboten werden, selbst entwickelt werden müssen. Es existiert bereits eine Reihe von kommerziellen Lösungen für diese Aufgaben, so daß vor einer Eigenentwicklung in jedem Fall eine Evaluierung der verfügbaren Produkte erfolgen sollte.

7 Schicht 5: Persistenz

Die Aufgabe der untersten Schicht besteht in der physikalischen Abbildung der Geschäftsobjekte in den persistenten Speicher. Im Fall der vom Betriebssystem unterstützen Flat Files bestünde die Aufgabe vielleicht nur darin, einen entsprechenden Stream zur Speicherung bereitzustellen. In der Regel wird es sich bei komplexeren

Applikationen aber um RDBMS oder ODBMS handeln, die als persistenter Speicher dienen. Als Exoten mögen z. B. Betriebssystemmechanismen vorhanden sein, die ein direktes Abbilden des flüchtigen in den persistenten Speicher möglich machen. Die AS/400 bietet mit dem Single-level Store Persistence Model einen solchen Mechanismus an (vgl. IBM 1998, S. 26).

Generell wird die Persistenzschicht zumindest zwei Klassen anbieten, die den Zugriff auf die Datenbank regeln:

1. Datenbank-Manager

Diese Klasse verwaltet alle Verbindungen zu einem Datenbanksystem. Ihr obliegt die Auswahl der entsprechenden Treiber, um auf unterschiedliche Datenbanksysteme zugreifen zu können. Über den Treiber ist es möglich, eine Verbindung zum Datenbank aufzubauen. Typische Methoden dieser Klasse sind:

```
OpenDatabase(url, uid, password)
CloseDatabase()
CreateDBConnection()
DeleteDBConnection()
```

2. Anfragen-Manager

Diese Klasse verwaltet eine Verbindung zur Datenbank. Während dessen werden eine oder mehrere Anfragen (auch Query oder Statement) an die Datenbank gerichtet. Eine Verbindung kann auch als eine Transaktion interpretiert werden, auf der Rollback- und Commit-Mechanismen möglich sind. Typische Methoden dieser Klasse sind:

```
CreateQuery(AbfrageString)
ExecuteQuery()
CloseQuery()
DeleteQuery()
```

Schicht 5 sollte in der Regel für ein BOF nicht neu entwickelt werden, da Basismechanismen dieser Art z. B. mit JDBC oder ODBC bereits zur Verfügung stehen. Für ODBMS existieren die Standards Object Definition Language (ODL) und Object Query Language (OQL) von der Object Database Management Group (ODMG), die bereits in einigen ODBMS-Produkten Verwendung finden.

8 Zusammenfassung und Ausblick

Die Fünf-Schichten-Architektur stellt ein erstes Ergebnis des Arbeitskreises

Frameworks dar. Sie gibt Richtlinien, wie eigene BOF zu entwickeln bzw. wie kommerziell verfügbare BOF auf ihre Verwendung für eigene Zwecke zu überprüfen sind. Ziel des Arbeitskreises ist es, weiteres Wissen zur Entwicklung und Evaluierung zu Sammeln.

Literaturverzeichnis

Balzert, H. (1994): Das JANUS-System: Automatisierte, wissensbasierte Generierung von Mensch-Computer-Schnittstellen. Informatik - Forschung und Entwicklung 17(1994)9, S. 22-35.

Cattell, R.G.G./Barry, D.K./ Bartels, D./Berler, M./Eastman, J./Gamerman, S./Jordan, D./Springer, A./Strickland, H./Wade, D. (Hrsg.) (1997): The Object Database Standard: ODMG 2.0. San Mateo 1997.

Conrad, S. (1997): Föderierte Datenbanksysteme. Konzepte der Integration. Berlin u.a. 1997.

Gamma, E./Helm, R./Johnson, R./Vlissides, J. (1995): Design Patterns. Reading 1995.

IBM (Hrsg.) (1998): San Francisco Concepts & Facilities. Rochester 1998.

Jacobson, I./Ericsson, M./Jacobson, A. (1994): The Object Advantage – Business Process Reengineering with Object Technologie. Menlo Park 1994.

Krasner, G.E./Pope, S.T. (1988): A Cookbook for Using the Model-view-controller User Interface Paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(1988)3, S. 26-49.

Lefevre, M. (1998): Designing a Persistence Object Model from Patterns. Journal of Object-Oriented Programming 10(1998)2, S. 17-28 u. 80.

Orfali, R./Harkey, D./Edwards, J. (1996): The Essential Client/Server Survival Guide. New York u.a. 1996.

Reese, G. (1997): Database Programming with JDBC and Java. Cambridge u.a. 1997.

Scheer, A.-W. (1992): Architektur integrierter Informationssysteme – Grundlagen der Unternehmensmodellierung. Berlin u.a. 1992.

Sutherland, J./Rugg, K./Pope, M. (1993): The Hybrid Object-Relational Architecture (HORA): An Integration of Object-Oriented and Relational Technology. Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing, S. 326-333.