Vorteile von Frameworks zur Entwicklung objektorientierter, datenbankbasierter Softwarelösungen mit Oracle Business Components for Java

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Informatiker

an der

Fachhochschule Köln
Abteilung Gummersbach
Fachbereich Informatik
Studiengang Allgemeine Informatik

1. Prüfer: Prof. Dr. rer. nat. Heide Faeskorn-Woyke
2. Prüfer: Dipl.-Math. Stefan Scheidt

Erstellt von:

Markus Lünsmann
Vosselstr. 14
51645 Gummersbach
Matrikel-Nr.: 1101755617

Gummersbach, 26. August 2001

Inhaltsverzeichnis

Seite

Anhangverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Aufl.	-	Auflage
bzw.	-	beziehungsweise
engl.	-	englisch
et al.	-	et alii (dt. und andere)
evtl.	-	eventuell
API	-	Application Programming Interface
d.h.	-	das heißt
ggf.	-	gegebenenfalls
GUI	-	Graphical User Interface
Hrsg.	-	Herausgeber
IDL	-	Interface Definition Language
o.ä.	-	oder ähnliche
OMT	-	Object Modeling Technique
OOA	-	objektorientierte Analyse
OOD	-	objektorientiertes Design
S.	-	Seite
UML	-	Unified Modeling Language
Vgl.	-	Vergleiche
z.B.	-	zum Beispiel

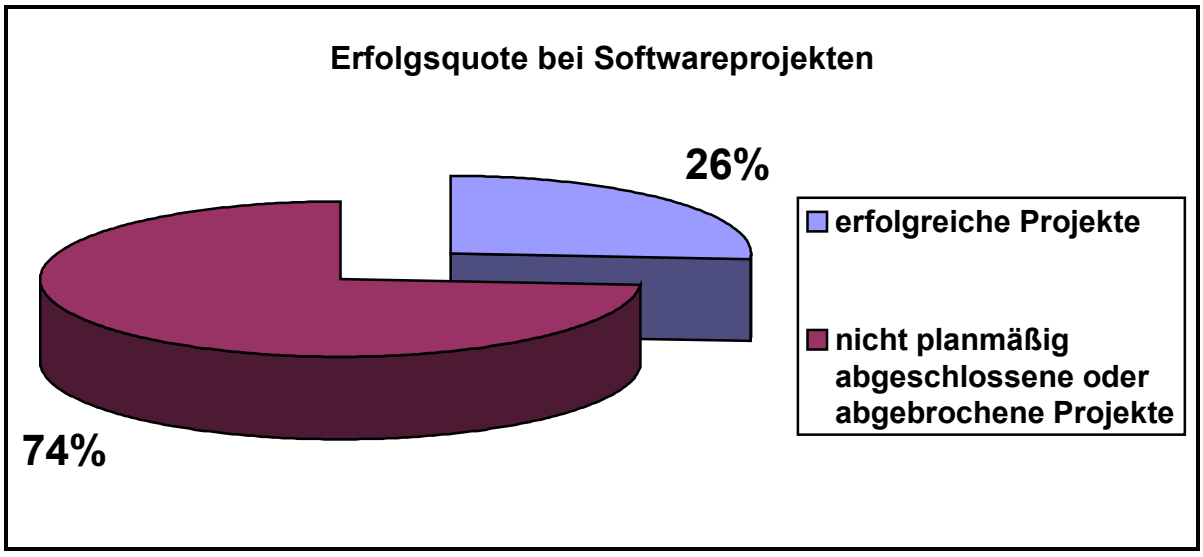
1 Einleitung / Problemstellung und Vorgehensweise

Softwaresysteme unterstützen uns heute in weiten Bereichen der Wissenschaft und Wirtschaft. Diese Systeme werden dabei zunehmend komplexer und unübersichtlicher. Durch die stark wachsenden Ansprüche an Software und die rapide Entwicklung in heutigen Softwaresystemen werden die Rufe nach Methoden laut, welche eine einfache, unkomplizierte und schnelle Softwareentwicklung möglich machen.

Die Vorgehensmodelle der konventionellen Softwarearchitektur werden diesen Ansprüchen nicht mehr gerecht und fordern vom Softwaredesigner bzw. Softwarearchitekten neue und effektivere Lösungswege.

Hierbei wurde und wird teilweise noch immer die objektorientierte Softwareentwicklung als Allheilmittel gesehen. Leider haben die Erfahrungen gezeigt, daß sie, die in sie gesetzten Erwartungen nicht erfüllen konnte. Viele dieser angeblich so fortschrittlichen Projekte scheitern. Eine Studie der amerikanischen Standish Group ergab, daß 1998 nur 26% der Softwareprojekte erfolgreich abgeschlossen wurden. Erfolgreich meint hierbei die Einhaltung des gesetzten zeitlichen und finanziellen Rahmens. Ob diese Tatsache nun auf technische oder andere Probleme zurückzuführen ist, sei erst einmal dahingestellt.

Abbildung 1.1: Erfolgsquote bei Softwareprojekten



Quelle: The Standish Group, *Chaos Report*, 1998

Bedeutet dies nun, daß das Schicksal objektorientierter Technologien besiegelt ist? Müssen Unternehmen nun auch auf die durchaus positiven Effekte der Objektorientierung verzichten und neue Entwicklungen abwarten? Oder gibt es Mittel und Wege die technischen Probleme geschickt zu lösen? Diese Fragen sollen im Laufe dieser Diplomarbeit vom technischen Standpunkt aus geklärt werden.

Es werden verschiedene Ansätze der objektorientierten Softwareentwicklung, wie z.B. Komponenten oder Muster diskutiert und auf Vor- und Nachteile untersucht. In diesem Zusammenhang sollen der Begriff des Frameworks und die Einsatzmöglichkeiten von Frameworks geklärt werden.

Abschließend werden Teilkomponenten eines Frameworks zur Entwicklung objektorientierter Individualsoftware entwickelt. Mit Unterstützung dieser Teilkomponenten und ihrer Komposition mit ergänzenden Komponenten kann ein Framework entwickelt werden, welches die Anforderungen an eine moderne Softwareentwicklung erfüllen kann. Das Framework nutzt hierbei Techniken der Oracle Business Components for Java, um datenbankbasierte Enterprise Applikationen entwickeln zu können.

2 Grundlagen der objektorientierten Softwareentwicklung

In diesem Kapitel werden die Grundlagen der objektorientierten Softwareentwicklung behandelt. Hierbei wird lediglich auf die technischen Aspekte der Softwaremodellierung eingegangen. Die der eigentlichen Entwicklung vorausgehenden Schritte, wie die *Requirements*- bzw. *Problemanalyse* sowie die *Use Case-Modellierung* und andere planerische Aspekte sollen hier als bekannt vorausgesetzt werden. Eine Vorgehensweise mit sehr praktischem Ansatz und vielen Erfahrungswerten hierzu findet sich in [OOTC97].

Objektorientierte Softwareentwicklung basiert im wesentlichen auf der Abstraktion der realen Welt. Der Begriff der Entwicklung bezieht sich in diesem Kontext auf das Durchlaufen der Analyse-, Entwurfs- und Implementierungsphase. Der wichtigste Faktor in diesem Fluß ist Identifikation und die Organisation der Konzepte und Grundlagen einer Anwendungsdomäne, die es abzubilden gilt. Die Implementierung spielt somit zwar eine unentbehrliche, jedoch nicht so wichtige Rolle wie eine korrekte Analyse und ein korrekter Entwurf. Das erfolgreiche Durchlaufen dieser beiden Phasen erleichtert eine Implementation deutlich und fördert weitere Punkte der Entwicklung, wie z.B. Wartung, Weiterentwicklung, Integration und Wiederverwendbarkeit. Die Konzepte in einer deutlichen und einheitlichen Notation darzustellen, deckt außerdem schon einen großen Teil der Projektdokumentation ab.

Neben diesen Strukturierungs- und Konstruktionsansätzen existieren auch andere Möglichkeiten, die Entwicklung von komplexen Softwarelandschaften zu vereinfachen. Hier wäre beispielsweise ein veränderter Entwicklungsprozeß zu nennen, bei dem Softwareentwicklung nicht als Produktionsprozeß unter Verwendung eines Phasenmodells, sondern als Lern- und Kommunikationsprozeß aller beteiligten Gruppen unter Einsatz einer evolutionären Vorgehensweise gesehen wird (vgl. [Floyd87]). Genauso können sich Veränderungen an der Organisationsstruktur des Unternehmens, das die Anwendungssysteme entwickelt, positiv auswirken (vgl. [Birrer95] und [Kilberth94]). Beide Faktoren werden allerdings im weiteren nicht untersucht. Die Diplomarbeit konzentriert sich ausschließlich auf den softwaretechnischen Bereich.

Die Methoden der objektorientierten Softwareentwicklung sind fast ebenso mannigfaltig wie die Menge der darüber verfassten Literatur. Allerdings haben sich einige populäre Ansätze, wie der der *Object Modeling Technique* (OMT) [Rumbaugh91] und die darauf aufbauenden Techniken der Unified Modelling Language (UML) [Oestereich98] und des Rational Unified Process (http://www.rational.com) herauskristallisiert. Die OMT beschreibt eine Methodologie für die objektorientierte Entwicklung und eine grafische Notation für die Repräsentation objektorientierter Konzepte. Es wird zunächst ein Modell einer Anwendungsdomäne erstellt, welchem dann in der Entwurfsphase Implemtierungsdetails hinzugefügt werden. Diese beiden Schritte finden sich in den meisten Methodologien zur objektorientierten Entwicklung und werden als *objektorientierte Analyse* (OOA) und das sich daran anschließende *objektorientierte Design* (OOD) bezeichnet.

Diese strukturierte Vorgehensweise zielt nicht zuletzt auch darauf hin, robusten, möglichst fehlerfreien Sourcecode zu schreiben, welcher auch in späteren Projekten Verwendung finden kann. Die Wiederverwendbarkeit von Software kann somit zu einem sehr entscheidenden Faktor im Zeitrahmen eines Softwareprojektes werden. Leider sind immer noch viele Firmen weit davon entfernt ihre Software konsequent wiederzuverwenden und erfinden Lösungen immer wieder aufs neue.

Eine strukturierte Modellierung ermöglicht es außerdem, einen komplexen Anwendungsbereich in kleinere, leichter zu entwickelnde Teilbereiche zu unterteilen. Anwendungssysteme können zudem einfacher in Form von Ausbaustufen, etwa durch den Einsatz von Komponententechnologien, entwickelt werden. Zusätzlich wird eine evolutionäre Vorgehensweise bei der Entwicklung integrierter Softwaresysteme für komplexe Anwendungsbereiche vereinfacht.

Die Modellbildung ist somit eine der zentralen Techniken im Softwareentwicklungsprozeß. Ein Modell dient immer einem speziellen Zweck. Es stellt entweder ein Abbild von etwas oder ein Vorbild für etwas dar. (vgl. [Lichter93])

Modelle finden sowohl in der fachlichen Analyse eines Problems, als auch in der technischen Umsetzung der Problemlösung häufige Anwendung. Wie nützlich diese Modelle dabei sein können hängt stark davon ab, wie gut diese einen Kontext abstrahieren ohne ihn zu komplex und unverständlich werden zu lassen. Eine sehr treffende Umschreibung des Modellbegriffs findet sich auch in [Reenskaug96]:

> *„We create a myriad of mental models to help us understand phenomena of interest and to master them. A model is an artifact created for a purpose; it cannot be right or wrong, only more or less useful for its purpose."*
> ([Reenskaug96], S. 33)

Unter zu Hilfenahme der Object Modeling Technique (OMT) nach [Rumbaugh91] sollen nun die Schritte zur Modellbildung in der Softwareentwicklung anhand der objektorientierten Analyse (OOA) und des objektorientierten Design (OOD) erläutert werden. Der komplexe Bereich der OOA wird hier nur der Vollständigkeit halber einbezogen und auch nur in Ansätzen beschrieben.

2.1 Objektorientierte Analyse

Die OOA wird häufig auch als das Bindeglied oder die Schnittstelle vom Entwickler zum Auftraggeber beschrieben. Sie ermöglicht es, eine einheitliche, sprachliche Grundlage zu schaffen, auf die man sich im OOD berufen kann. Somit wird zunächst eine fachliche Definition aller Prozesse und Begriffe eines Anwendungssystems festgelegt, welche sowohl für den Entwickler als auch für den Auftraggeber verständlich ist. Da das Analysemodell die Kommunikationsgrundlage zwischen Entwickler und Auftraggeber ist, muß es auch zwingend und eindeutig von den beiden Parteien verstanden werden. Missverständnissen, die teure Korrekturen der Software mit sich bringen, kann damit vorgebeugt werden. Fehler und Probleme werden frühzeitig erkannt und können relativ unproblematisch ausgeräumt werden.

Nach Abschluß der OOA sollen folgende Ergebnisse vorliegen:

- Es müssen alle relevanten Objekte bestimmt sein.
- Es müssen Begriffsdefinitionen für alle verwendeten Begriffe vorhanden sein.
- Die Anforderungen müssen durch eine Anforderungsanalyse genau spezifiziert sein.
- Prototypen der Oberfläche sollten angefertigt sein.
- Die Projektierung der Entwicklung muß definiert sein (Ausbaustufen, Arbeits-, Zeitpläne, Personaleinsatzpläne, Kostenpläne.)

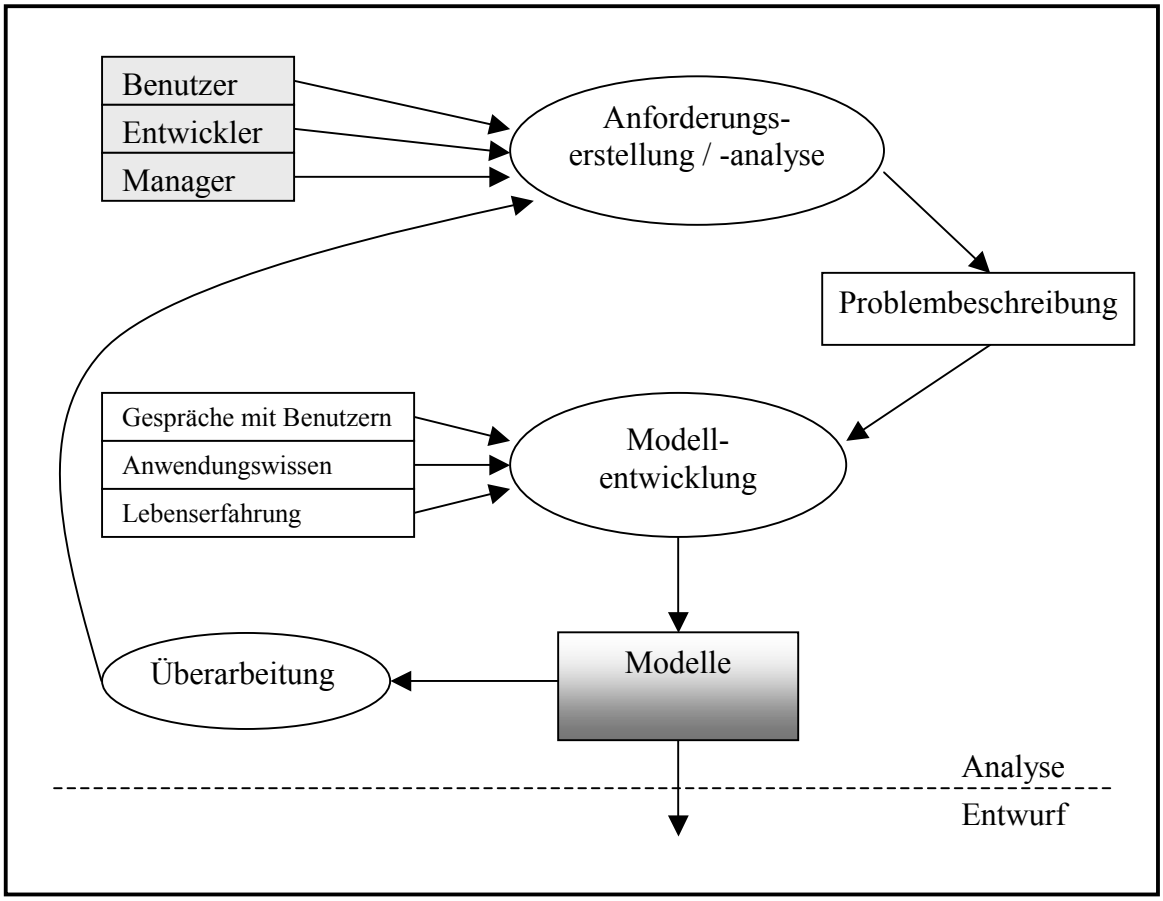
Zunächst geht es für den Systemanalytiker also darum, ein Grobmodell der Funktionen und Ereignisse der realen Welt zu bauen. Hierbei müssen deren, für das zu entwickelnde System relevanten Eigenschaften und Merkmale übernommen werden. Die Modellentwicklung erfordert eine präzise Abstraktion des Realweltszenarios. Das Modell muß sich damit beschäftigen *was* das System können muß und nicht damit *wie* es dies implementierungstechnisch realisiert. Der Systemanalytiker sollte keine Gedanken, die eine spätere Implementierung berücksichtigen in die Systemmodellierung einfließen lassen.

Abbildung 2.1 verdeutlicht die einzelnen Stadien während der Analysephase. Zunächst werden von Benutzer, Entwickler und/oder Manager die Anforderungen an die Software spezifiziert. Diese Spezifikation ist meistens sehr ungenau und inkonsistent, wie sich häufig in der späteren Analyse und im Modellbildungsprozeß herausstellt. Ohne einen Analyseprozeß systematisch durchlaufen zu haben ist dies jedoch nicht weiter verwunderlich, da auch die Anwendungsexperten und Auftraggeber keine absolut konkreten Vorstellungen von ihren Anforderungen haben. Diese ergeben sich in nicht unerheblichen Anteilen erst während der Analyse.

Die Ungenauigkeit der Anforderungsspezifikation führt damit zwangsläufig auch zu einer ungenauen Problembeschreibung. Diese beeinflußt in Verbindung mit diversen anderen Einflußfaktoren eine erste, vorläufige Modellentwicklung und führt schließlich zu einem ersten Modellentwurf. Aufgrund unpräziser Anforderungen ist dieser jedoch nicht perfekt. Eine iterative Vorgehensweise, welche die Erkenntnisse, die während der Analyse gemacht

wurden berücksichtigt, verfeinert nun die Anforderungsspezifikation, die Problembeschreibung und somit auch die darauf aufbauenden Modelle. Letztendlich führt dieser iterative Ansatz dazu, daß sehr präzise Modelle des Anwendungssystems erstellt werden und eine korrekte Anforderungsbeschreibung vorliegt.

Abbildung 2.1: Vorgehensweise zur Modellentwicklung



Quelle: In Anlehnung an James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy und William Lorensen: *Objektorientiertes Modellieren und Entwerfen*. Hanser, Müchen, Prentice-Hall, London, 1993, S.182

Um schon von Beginn an eine möglichst gute, systematisch entwickelte Anforderungsbeschreibung zu haben, ist es notwendig einige Grundregeln zu beachten. Die Anforderungsbeschreibung darf sich nicht an Implementierungs- oder Entwurfsaspekten orientieren, sondern muß sich strikt an die fachlichen Gegebenheiten der Anwendungsdomäne halten. Die Detailvielfalt spielt dabei keine Rolle, da auch die

Realweltszenarios unterschiedlich detailliert sein können. Nicht zuletzt deshalb sind diese Problembeschreibungen häufig nur sehr vage. Zudem werden sich Anforderungen auch als nicht tragbar herausstellen, da ihre Realisierung nur mit erheblichem Aufwand möglich ist und hohe Kosten verursacht. Im Laufe der Analyse werden auch neue Anforderungen an das System deutlich, welche vom Auftraggeber vorher nicht bedacht wurden. Der Systemanalytiker muß dem Auftraggeber Hilfestellung leisten, diese Anforderungen zu erkennen und zu beschreiben. Es sind also immer wieder Nachbearbeitungen der Anforderungsbeschreibung erforderlich, wodurch diese ein sehr dynamisches Konstrukt darstellt.

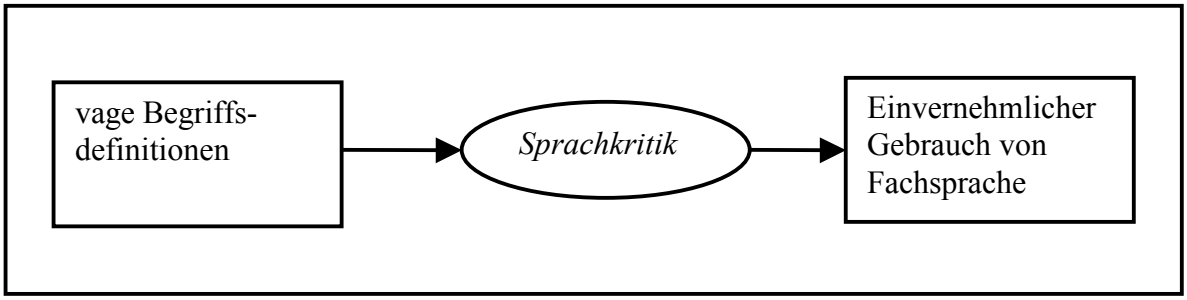
Um den komplexen Prozeß der Modellierung abzubilden, ist eine strukturierte Vorgehensweise unerläßlich. Nach [Rumbaugh91] werden während der Analysephase drei Modelle mit diesem Ansatz entwickelt:

- Objektmodell
- dynamisches Modell
- funktionales Modell

2.1.1 Das Objektmodell

Beim Entwurf des Objektmodells kommt es entscheidend darauf an, die relevanten Objekte zu identifizieren. Hierbei ist ein sprachkritischer Ansatz sehr hilfreich. Er ermöglicht es, genaue Begriffsdefinitionen zu finden, um die Probleme der Mehrdeutigkeit von Sprache zu umgehen. Die zu definierenden Objekte müssen bedeutungstragende, standardisierte Namen haben, um Mißverständnisse zu vermeiden.

Abbildung 2.2: Sprachkritik in der Objektmodellierung



Die Objektspezifikation wird am besten dem Anwendungsexperten überlassen, da er mit der realen Domäne vertraut ist. Danach werden die Merkmale und Attribute dieses Objekt festgelegt. Der Experte muß nun in der Lage sein, *Assoziationen* und *Aggregationen* zwischen den Objekten zu erkennen und diese zu nutzen, um Begriffsstrukturen herzustellen. Insbesondere bedeutet dies das Herstellen von Spezialisierungen, respektive Verfeinerungen sowie von Generalisierungen, also Abstraktionen von Objekten. Die Methoden, die ein Objektverhalten definieren, dürfen hier noch nicht Gegenstand des Modells werden, da es sonst zu unübersichtlichen und komplexen Strukturen kommt, die zu diesem Zeitpunkt noch nicht verstanden werden könnten.

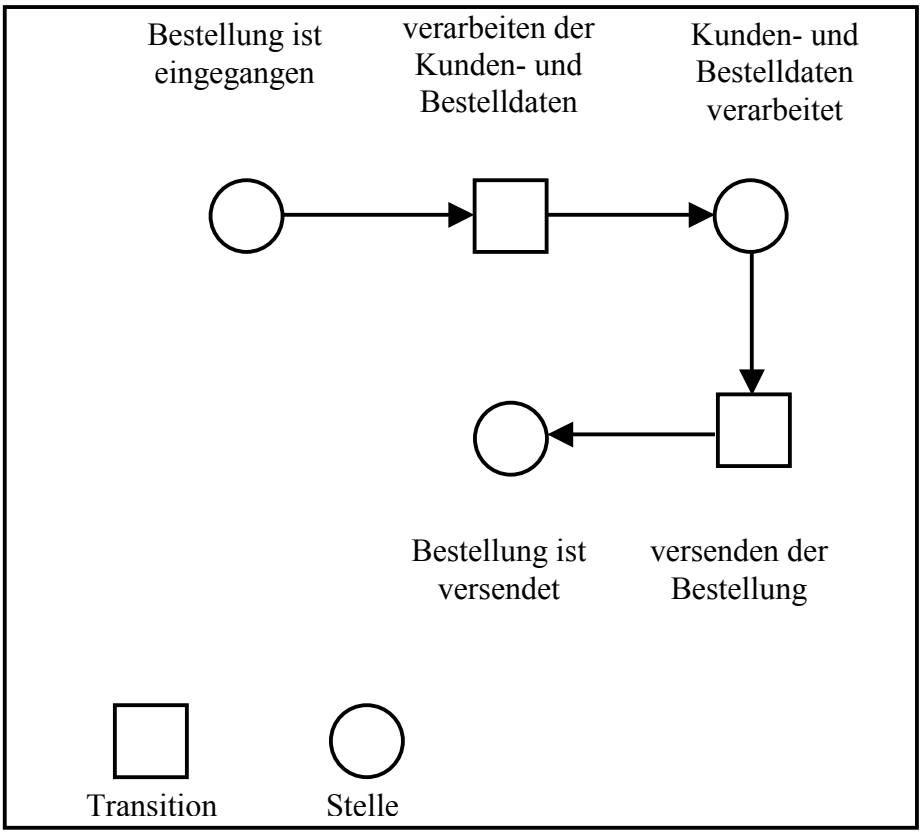
Äquivalent zur Anforderungsbeschreibung befindet sich auch das Objektmodell in einem ständigen Fluß der Rekonstruktion. Während der Analyse werden neue Objekte auftauchen und andere als überflüssig deklariert und entfernt werden.

Abschließend sollten in einem *Data-Dictionary* alle definierten Objekte mit Namen und einer ausführlichen Beschreibung festgehalten werden.

2.1.2 Das dynamische Modell

Das dynamische Modell soll alle Aspekte eines Systems berücksichtigen, die sich in Abhängigkeit von der Zeit verändern. Es spiegelt also das dynamische Verhalten der Objekte wieder. Mit Hilfe von Petri-Netzen werden die Objektinteraktionen modelliert. Zur Darstellung von Interaktions- bzw. Zustandsdiagrammen eignen sich insbesondere Stellen-Transitions-Netze, in denen die Stellen die Zustände der Objekte und die Transitionen die ausgelösten Aktionen sind.

Abbildung 2.3: Petri-Netz zur Modellierung eines Bestellvorgangs



Das Modell soll das Verhalten von Objekten bei bestimmten Ereignissen, Zuständen und Operationen abbilden. Da solche Diagramme sehr komplex und unübersichtlich werden können, gilt es einen Kompromiß zwischen Komplexität und Ausdrucksstärke zu finden.

Ein Ereignis bedeutet, daß in dem Modell eine Veränderung stattgefunden hat. Es dient somit als Reiz für andere Objekte, die diesen Reiz verarbeiten und als Reaktion darauf eine Aktion ausführen. Ein Zustand definiert den Zustand eines Objektes im Modell zwischen den Ereignissen und dient als Umgebungsinformation für das Objektverhalten bei Ereignissen. Der Zustand eines Objektes ist an die Werte seiner Attribute gebunden.

Diverse weitere Techniken zur dynamischen Objektmodellierung finden sich in [Davis88].

2.1.3 Das funktionale Modell

Im funktionalen Modell wird das Verhalten von Datenwerten im System betrachtet und abstrahiert. Es wird untersucht, welche Aktionen aus dem dynamischen Modell zu Datenveränderungen führen und wie sich dabei die Datenwerte ändern. Das funktionale Modell legt nicht die eigentliche Funktion zur Berechnung eines Datenwertes fest, sondern definiert die Algorithmen wie Datenwerte zu berechnen sind. Es spezifiziert, welche Bedeutung die Aktionen aus dem dynamischen Modell haben.

Programme mit reiner Berechnungsarbeit, also mit fast keiner Interaktion zwischen den Objekten, können *nur* im funktionalen Modell hinreichend beschrieben werden. Analog hierzu gilt für Programme mit reinem Interaktionscharakter, daß sie *nur* im dynamischen Modell treffend beschrieben werden können.

Die Berechnungen und Datenumwandlungen werden in Datenflußdiagrammen dargestellt. Ein Datenflußdiagramm enthält Prozesse zur Datentransformation, Datenflüsse, um Daten zu bewegen, Handlungsobjekte zur Datenproduktion und Datenspeicherobjekte zur Datenspeicherung. Datenflußdiagramme entstammen traditionellen Software-Entwicklungsmethodologien zur prozeduralen Softwareentwicklung. Eine weiterführende Erläuterung von Datenflußdiagrammen in der traditionellen Softwareentwicklung findet sich in [Yourdon89].

Abschließend erfordert die OOA eine genaue Dokumentation der Analyseergebnisse. Sie hilft dem Systemdesigner, bestimmte Analyseergebnisse nachzuvollziehen und erleichtert ihm somit das Verständnis der gesamten Analysemodelle.

Werden all diese Schritte gewissenhaft und detailliert durchgeführt, erleichtert dies das nachfolgend erklärte objektorientierte Design enorm. Ergebnisse aus der strukturierten Analyse können anhand einiger Regeln sehr leicht in ein Klassenmodell überführt werden.

2.2 Objektorientiertes Design

Im Anschluß an die objektorientierte Analyse (OOA) folgt in der traditionellen objektorientierten Softwareentwicklung nach [Rumbaugh91] der Systementwurf. Dieser Systementwurf soll die Gesamtarchitektur des Systems berücksichtigen und strategische Entscheidungen zur Optimierung des Systems treffen. Eine weiterführende Erläuterung dieses Entwurfsaspektes soll hier nicht gegeben werden, da er relativ unerheblich für die Entwicklung von Frameworks ist. Das objektorientierte Design (OOD) hingegen ist dafür eine unerläßliche Grundlage. Das OOD soll eine technische Umsetzung der Ergebnisse der OOA und des Systementwurfs liefern. Hierbei werden die Klassen und Klassenstrukturen im objektorientierten Sinne abgebildet und festgelegt. Das Entwurfsmodell des OOD basiert auf dem Analysemodell der OOA, enthält zudem allerdings diverse implementierungsabhängige Details.

Zur Vorgehensweise läßt sich sagen, daß man zunächst versucht synonyme Anwendungsbegriffe aus der OOA zu finden, um Objekte zu definieren. Sie repräsentieren in der technischen Umsetzung der OOA *eine* Klasse, welche sämtliche Algorithmen und Datenstrukturen dieses Objektes kapselt. Einschränkungen ergeben sich allerdings, sobald Begriffe, die ein und dasselbe Objekt bezeichnen, verschiedene Attribute oder Verhaltensweisen repräsentieren müssen. Dies ist z.B. dann der Fall, wenn sie Abhängig vom Kontext oder von der Perspektive, aus der sie betrachtet werden, unterschiedlich reagieren müssen. Eine mögliche Umsetzung für ein solches Verhalten in der OOD ist die Einfachvererbung bei gemeinsamen Eigenschaften der Objekte. Dieses Verfahren führt den Designer zu mehreren, spezialisierten Klassen eines Objekts, da er für jede Perspektive eine Klasse aus der Vaterklasse ableiten muß. Alternativ kann in diesem Fall auch eine Mehrfachvererbung stattfinden, so daß die erbende Klasse die Attribute und Verhaltensweisen von mehreren übergeordneten Klassen vereinen kann. Es ergibt sich also nur eine Klasse, die allerdings die Funktionalitäten ihrer verschiedenen Vaterklassen zur Verfügung stellt. Des weiteren werden sämtliche anderen Begriffe des Begriffssystems der OOA in Klassen des Klassensystems der OOD überführt. Attribute eines Begriffs im Begriffsmodell werden zu Attributen der Klasse. In der technischen Sicht auf das System finden sich zu den Klassen, die Begriffe der OOA widerspiegeln auch noch Objekte, die für eine Implementierung hinzugefügt werden müssen. Diese Objekte haben keinen

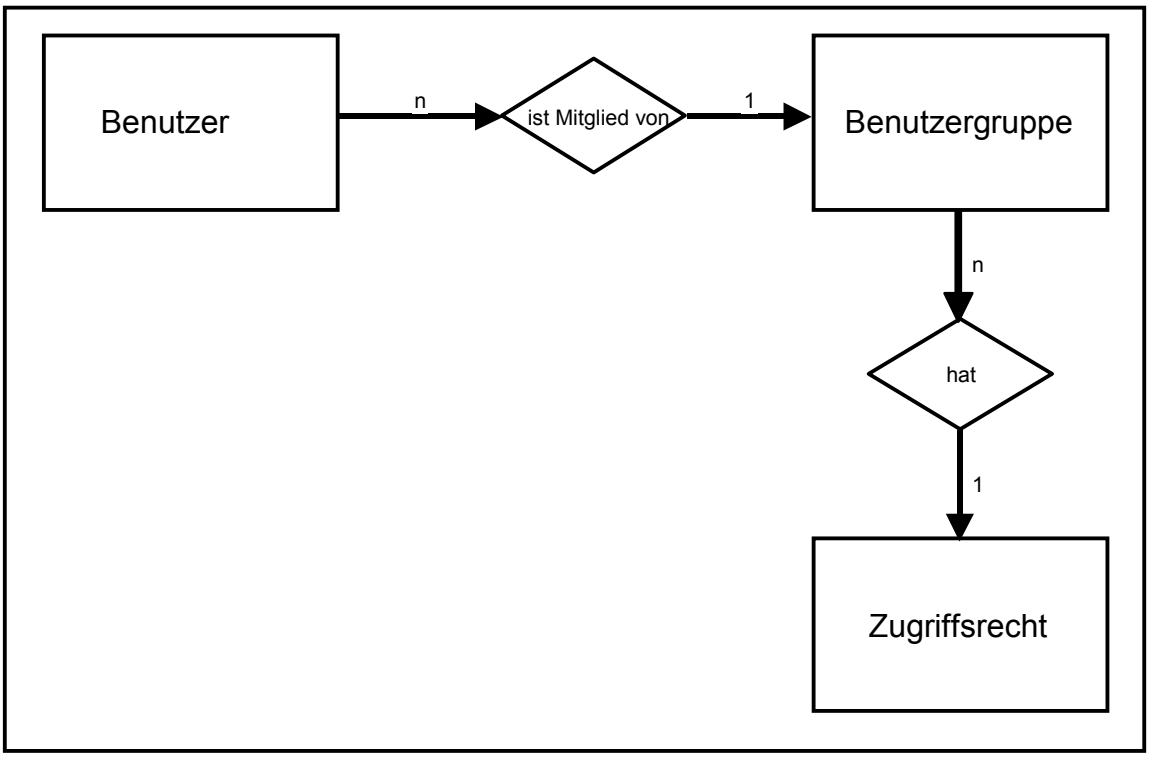
direkten Bezug zu einem Begriff der Analyse, sondern werden nur zur technischen Umsetzung in Form einer Klasse gebraucht.

Die Klassenstruktur, welche die Objektbeziehungen definiert, orientiert sich an den Begriffsstrukturen. In der Klassenstruktur finden sich Vererbungs- und Kommunikationsbeziehungen der Klassen bzw. der Begriffe der OOA. Die Vererbungsstrukturen ergeben sich aus den Begriffsbeziehungen. So werden die Spezialisierung und die Generalisierung in der Bergriffsstruktur mit den Methoden der Einfachvererbung abgebildet. Teil-Ganzes-Beziehungen erfordern eine Komponentenbildung bzw. Aufsplittung, bei der auch jeder Teil eines Oberbegriffs in eine eigene Klasse überführt wird.

Schließlich gilt es noch die Vorgänge, den Informationsfluß aus der OOA in einem technischen Modell zu erfassen. Vorgänge innerhalb des Begriffsmodells ergeben dabei die Methoden in einem Klassenmodell. Alle Klassen, die an einem Vorgang beteiligt sind, müssen untereinander kommunizieren. Ihre Methoden bilden dabei die Schnittstellen zur Kommunikation. Nur mit Hilfe dieser Methoden ist es möglich, eine Objektinteraktion und Objektkommunikation in ein Klassenmodell zu überführen.

Das Ergebnis des OOD ist zunächst eine technische Klassendefinition und eine Klassenstruktur z.B. in Form eines Entity-Relationship-Diagramms wie in Abbildung 2.4.

Abbildung 2.4: Beispiel eines Entity-Relationship-Diagramms



Darüber hinaus kann die Klassenkommunikation in einem Petri-Netz o.ä. visualisiert werden. Trotz der komplexen Möglichkeiten, einen fast perfekten Entwurf zu erstellen, ist dies nicht das Hauptanliegen des OOD. Vielmehr soll er eine einfache Wartbarkeit, Erweiterbarkeit und Implementierung ermöglichen [Rumbaugh91]. Um auch diesen Punkten gerecht zu werden, ist das Ziel die Entwicklung von sogenannten „Black Box"-Klassen. Das Prinzip, welches sich dahinter verbirgt, wird als „Information Hiding" bezeichnet. Klassen sollen soviel Eigenverantwortlichkeiten wie möglich in sich kapseln. Die Schnittstellen der Klasse bleiben nach außen sichtbar. Sämtliche internen Vorgänge, die nur die Klasse selber betreffen und zur internen Verwaltung benötigt werden, sind nach außen nicht sichtbar. Um dieses Prinzip verwirklichen zu können, ist eine saubere Schnittstellenspezifikation und -dokumentation unerläßlich.

Eine exakte Dokumentation der Entwurfsentscheidungen darf auch beim OOD nicht fehlen. Sie ermöglicht es den Implementierern bestimmte Entwurfsentscheidungen nachzuvollziehen. Es kann nicht davon ausgegangen werden, daß die Rolle von Designer

und Programmierer von einer einzigen Person repräsentiert wird. Zumindest in großen Projekten ist dies sogar unmöglich.

2.3 Wiederverwendbarkeit

Der große Vorteil des objektorientierten Ansatzes liegt in der Effizienz der Softwareentwicklung. Diese ist aber nur gegeben, wenn auch kostengünstige und robuste Software entwickelt wird. Erreicht wird diese Wirtschaftlichkeit durch den Einsatz verschiedener Wiederverwendbarkeitstechniken, für die jedoch einige Vorbedingungen erfüllt sein müssen. Hierzu zählt vor allem der strukturierte, objektorientierte Ansatz. Einer der Gründe, wenn nicht sogar der wichtigste, für einen derartigen Ansatz ist die Wiederverwendbarkeit von Softwaremodellen bzw. der Software selbst. Die Wiederverwendbarkeit bezieht sich nicht nur auf das erneute Benutzen von Software, sondern auch auf sorgfältige Modellentwürfe in der Design- und Analysephase. So kann auch die Wiederverwendung der Grundkonzepte einer entwickelten Software zu einer Produktivitätssteigerung im Entwicklungsprozeß führen.

> *„Software reuse is the use of existing software components to construct new systems. Reuse applies not only to source-code fragments, but to all the intermediate work products generated during software development, including requirements documents, system specifications, design structures, and any information the developer needs to create software."*
> (Prieto-Díaz, 1993)

Ein möglicher, wenn auch primitiver Ansatz für die Wiederverwendung von Software bildet das Übernehmen von sogenannten „code snippets“. Sie repräsentieren Code-Teile, die von bereits erstellten Softwareprojekten in neue übernommen werden. Spätere globale, d.h. projektübergreifende Änderungen, Korrekturen oder Erweiterungen dieser Codeteile sind jedoch sehr mühsam und ineffizient. Ein wesentlicher Aspekt ist die Wiederauffindbarkeit und die damit einhergehende schlechte Wartbarkeit des Codes. „Code snippets“ sind also ungeeignet für eine erfolgreiche Wiederverwendbarkeit.
Wird ein Softwareprojekt jedoch gut modelliert, bilden sich während dieses Prozesses Klassen, die bestimmte Informationen, Eigenschaften und Mechanismen kapseln. Die

nächst höhere Wiederverwendbarkeitsstufe wird als „Klassen-Retrieval“ [Jacobson97] bezeichnet und meint die Wiederverwendung von gesamten Objekten, die in vorangegangenen Projekten entworfen wurden und in neue Projekte eingebunden werden. Erforderlich dafür ist eine sorgsame Standardisierung der Klassen und ihr Einpflegen in eine Klassenbibliothek. Diese Klassen können wiederverwandt werden, sobald Ähnlichkeiten zwischen einem Anwendungsbegriff und vorhandenen Klassen in der Klassenbibliothek bestehen. Um eine möglichst hohe Trefferquote dabei zu erzielen, müssen Klassen angemessen abstrahiert und erweitert werden. Eine Klasse erfüllt dann die Erfordernisse von vielen Anwendungsbegriffen. Aus diesen Abstraktionsstufen von Klassen lassen sich Standardklassen und anschließend Klassenbibliotheken bilden.

In engem Zusammenhang mit der Methode des „Klassen-Retrieval“ steht auch die erneute Nutzung von gesamten Klassenkonstrukten. Der Vorteil daran ist, daß nicht nur Objekte und deren Eigenschaften übernommen werden, sondern auch die Vererbungshierarchien und Kommunikationsbeziehungen, die Objekte untereinander haben. Die Schnittstellen von solchen Konstrukten sollten allerdings relativ einfach gehalten werden und gut dokumentiert sein, damit eine einfache Wiederverwendbarkeit gewährleistet ist.

Die vorgestellten Mechanismen bilden jedoch nur die Grundlage für eine Produktivitätssteigerung in der Softwareentwicklung. Aufbauend auf diesen Konzepten haben sich weitere Abstraktions- und Vorgehensmodelle entwickelt, welche ein wesentlich höheres Maß an Wiederverwendbarkeit bieten. Diese Ansätze sollen in den nächsten Kapiteln näher erläutert und untersucht werden.

3 Komponentenbasierte Softwareentwicklung

3.1 Einleitung

In letzter Zeit finden sich in diversen Fachzeitschriften immer wieder Berichte über den Einsatz von sogenannten Komponententechnologien oder „Component Software“. Dies liegt im Wesentlichen daran, daß die diversen Probleme in der Softwareentwicklung, wie z.B. eine kurze Entwicklungszeit, eine gute Qualität und eine hohe Wiederverwendbarkeit, nicht oder nur unvollständig durch die objektorientierte Programmierung gelöst werden. [Yourdon96] wies damals bereits auf die starke Zunahme an komponentenbasierten Systemen hin. Heute kann ein Unternehmen es sich gar nicht mehr erlauben, Software von Grund auf neu zu entwickeln. Die Kosten dafür wären enorm. Gegenüber den traditionellen monolithischen Anwendungen soll die Komponentensoftware viele dieser Probleme besser handhaben können. Ist diese Vorgehensweise der komponentenbasierten Softwareentwicklung also das Idealbild des Entwicklungsprozesses und läßt sich der Aufwand einer Entwicklung somit minimieren? Sicherlich wird auch sie keine perfekte Lösung aller Probleme darstellen. Sollten aber nur einige dieser Probleme ansatzweise erfolgreich gelöst werden können, ist es kaum verwunderlich, daß der Ruf nach entsprechenden Komponentenmodellen immer lauter wird.

Was ist nun eigentlich komponentenbasierte Software, wozu dient sie uns und wie wird sie entwickelt? Diese Fragen sollen im Folgenden geklärt und erörtert werden.

3.2 Definition einer Komponente

Über eine genaue Definition einer Komponente sind sich die Experten noch uneinig. Es gibt jedoch diverse Aussagen, die das Verhalten und den Aufbau einer Komponente umschreiben. So beschreibt Griffel in [Griffel98]:

> *„Eine Komponente ist ein Stück Software, das klein genug ist, um es in einem Stück zu erzeugen und pflegen zu können, groß genug ist, um eine sinnvoll einsetzbare Funktionalität zu bieten und eine individuelle Unterstützung zu rechtfertigen sowie mit standardisierten Schnittstellen ausgestattet ist, um mit anderen Komponenten zusammenzuarbeiten."*
> ([Griffel98], S.31)

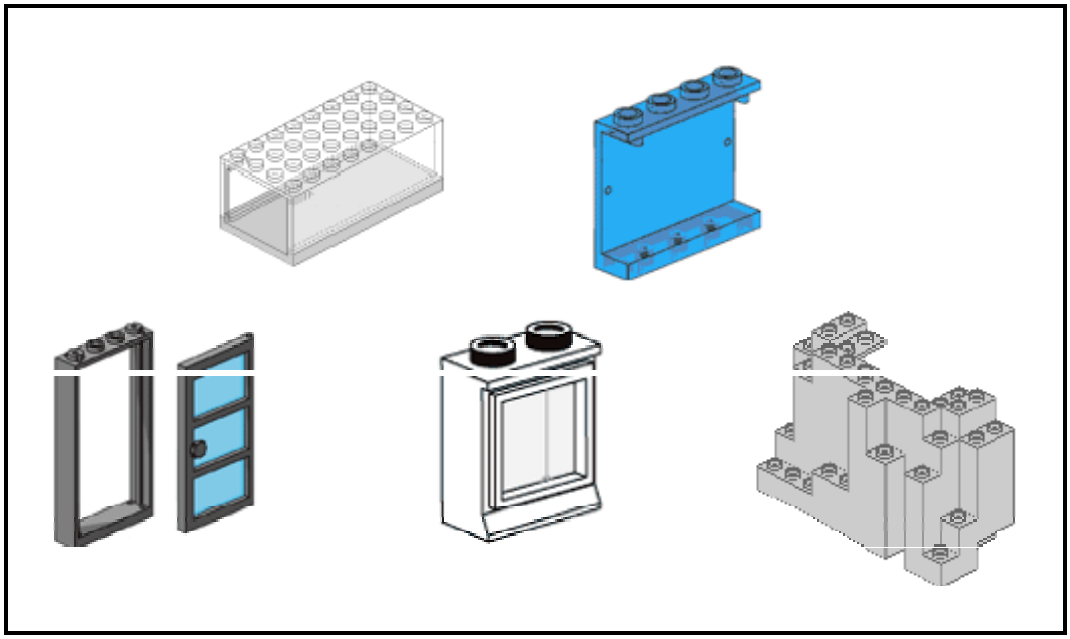
Eine andere Definition bzw. Umschreibung des Komponentenbegriffs liefert Szyperski. Er beschreibt in [Szyperski97] die Komponenten als „binary units of independent production, acquisition, and deployment that interact to form a functioning system" ([Szyperski97], S. xiii). Sein Verständnis einer Komponente fokussiert sich dabei auf ihre Unabhängigkeit. In seiner Definition soll sie sich von der Erstellung bis hin nur Nutzung ziehen, wobei die Komponente nur eingeschränkte Beziehungen mit anderen Systemen oder Komponenten haben darf. Sie soll lediglich im Rahmen einer externen Beziehung in einer Systemumgebung genutzt werden. Häufig sind jedoch auch interne Abhängigkeiten vorhanden, d.h. es werden innerhalb der Komponente andere Komponenten genutzt, um Redundanzen und damit einhergehende Fehler zu vermeiden. Schließlich soll eine Komponente aus mehreren binären Einheiten ein komplexes funktionierendes System bilden.

Eine ähnlich Anforderungsbeschreibung findet sich außerdem in [Jacobson97]. Auch hier kommt es auf die wesentlichen Charakteristika einer Komponente an. Einheitlich sind dies ihre Unabhängigkeit von anderen Komponenten, ihre Abgeschlossenheit und ihre Offenheit. Abgeschlossenheit und Offenheit scheinen sich gegenseitig auszuschließen, bedeuten jedoch unterschiedliche Eigenschaften. Mit Abgeschlossenheit meint man eine Kapselung der Funktionalitäten, also innere Konsistenz der Komponente, und mit

Offenheit, die Öffnung zur Außenwelt über wohl definierte Schnittstellen, um die Funktionalitäten nutzen zu können.

Eine allgemeingültige Definition gibt es demnach nicht. Es existieren lediglich die Bemühungen, sich auf eine gemeinsame Begrifflichkeit zu einigen, und die Ansprüche, die an eine Komponente gestellt werden, festzuschreiben. Eine weitere Möglichkeit der Komponentenbeschreibung, die vielfach in der Literatur verwendet wird, ist die „Legostein-Metapher“. Komponenten werden dabei mit Legosteinen verglichen, weil sie ähnliche Eigenschaften besitzen. Sie haben wohl definierte Schnittstellen und man muß sich nicht um die Funktionalität einer Einzelkomponente kümmern, da sie in sich konsistent ist. Überschaubare Einheiten von Komponenten werden zu großen und komplexen Systemen zusammengesetzt. Darüber hinaus müssen die Bausteine nicht-trivialen Anforderungen genügen.

Abbildung 3.1: „Legostein-Metapher“



Quelle: LEGO, Online im Internet: URL: http://shop.lego.com
(Abfrage am: 21.5.2001)

Komponenten lassen sich in zwei verschiedene Typen aufteilen. Zum einen sind dies die fachlichen Komponenten, welche auch als Geschäftskomponenten bezeichnet werden, und zum anderen die technischen Komponenten, die eine Realisierung der

Geschäftskomponenten unter Verwendung eines Komponentenmodells widerspiegeln. Die fachlichen Komponenten beziehen sich auf den Anwendungskontext und die fachlichen Anforderungen an einen Softwarebaustein. Die technische Umsetzung dieser Geschäftsobjekte und die Festlegung der Komponenteneigenschaften erfolgt schließlich in den technischen Komponenten, welche hier auch nur als Komponenten bezeichnet werden.

3.3 Einsatz von Komponenten in der Softwareentwicklung

3.3.1 Komponentenmodelle

Den ersten großen Schritt in Richtung komponentenbasierter Softwaresysteme machte Anfang der 80er Jahre die Object Management Group (OMG) mit der Entwicklung einer Spezifikation zur Verteilung und Entwicklung von Komponenten. Dieser Ansatz wird als Common Object Request Broker Architecture (CORBA) bezeichnet und ist eine der bekanntesten Spezifikationen zum Einsatz von Komponenten.

Komponentenmodelle werden heutzutage schon vielfach in der Softwareentwicklung eingesetzt. Die bekanntesten Vertreter dieser Gattung sind CORBA, die JavaBeans, die Enterprise JavaBeans (EJB), OLE, COM, DCOM, ActiveX und die Visual Basic Extensions. Die Vielzahl dieser Modelle zeigt auch hier, daß es keinen einheitlichen Standard gibt, sondern nur verschiedene Implementierungen bzw. Spezifikationen, die die Ansprüche an eine Komponente erfüllen.

Die Nutzbarkeit von komplexen Objektmodellen, wie Komponenten, wird durch diese Komponentenmodelle erheblich vereinfacht. So legt der CORBA-Standard sein Hauptaugenmerk nicht auf die Implementierung von Komponenten und deren Funktionsweise, sondern erfordert eine exakte Definition der Schnittstellen (Interfaces), über die die Komponente angesprochen werden kann. Hierfür stellt CORBA sogar eine eigene Programmiersprache, die Interface Definition Language (IDL) zur Verfügung, die nur für die saubere Definition der Interfaces verwandt wird. Dies ist einer von diversen essentiellen Aspekten des Komponentenmodells. So soll eine Komponente eine technologieneutrale, d.h. vom Gesamtsystem unabhängige Implementierung, aber ein spezifisches Interface haben. Sie muß in ihrer Implementierung also unabhängig von

Programmiersprache, Betriebssystem und Entwicklungsumgebung des Gesamtsystems sein. Des weiteren soll eine Komponente unabhängig von ihrem Speicherort angesprochen werden können. Der CORBA-Standard realisiert dies über seine Object Request Broker (ORB), welche die Verteilung von Anfragen an die Komponenten übernehmen und diese über diverse Netzwerktypen an einen beliebigen Ort weiterleiten. Die Schnittstellen der Komponenten sollen weitestgehend selbsterklärend sein und über ihre Möglichkeiten Auskunft geben, um eine einfache Wiederverwendbarkeit gewährleisten zu können. Komponenten müssen sich außerdem auch ohne großen Installationsaufwand leicht nutzen und in andere Systemumgebungen integrieren lassen.

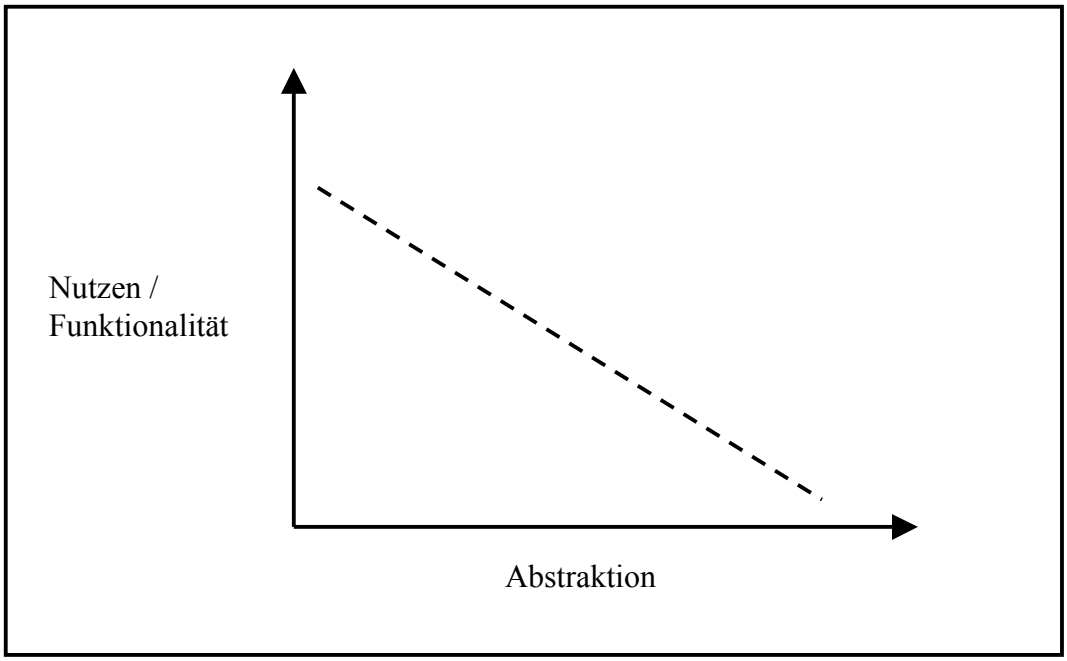
Bisher können jedoch nicht alle dieser Anforderungen von einem Komponentenmodell erfüllt werden. Es gilt hierbei eine größtmögliche Abdeckung zu finden. Die gängigsten und am wohl am häufigsten genutzten Systeme sind CORBA von der OMG und die ActiveX/DCOM-Architektur von Microsoft.

3.3.2 Komponenten im Softwareentwicklungsprozeß

Durch die Verwendung von Komponenten im Softwareentwicklungsprozeß verändert sich auch dessen Ablauf. In Anlehnung an [Heuer97] bedeutet dies, daß in der Designphase der Softwaremodellierung nicht sämtliche Objekte neu modelliert und zu einem funktionsfähigen Konstrukt zusammengesetzt werden, sondern daß man versucht Komponenten zu finden, die schon gesamte Arbeitsabläufe abbilden. Dies wird um so einfacher je abstrakter die Schnittstellen der Komponente definiert wurden ohne dabei jedoch Funktionalität einzubüßen. Hier wird auch der Anspruch an die Komponentenentwickler deutlich, in einer Komponente möglichst viel Funktionalität zu kapseln aber dennoch ein weites Feld an Einsatzmöglichkeiten abzudecken. Je höher die Abstraktion, um so besser ist auch die Wiederverwendbarkeit. Ein hohes Abstraktionsniveau schränkt allerdings auch den Nutzen einer Komponente ein. Ein Beispiel dafür wäre ein Komponente, die eine Benutzerauthentifikation auf einer grafischen Oberfläche gegen eine Datenbank durchführt. Wird diese Funktionalität abstrahiert, entsteht eine Komponente zur Benutzerauthentifikation auf grafischen Oberflächen. Es ist jedoch nicht klar, gegen welchen Benutzerdatenspeicher eine Überprüfung durchgeführt werden soll. Diese Komponente kann somit an viele

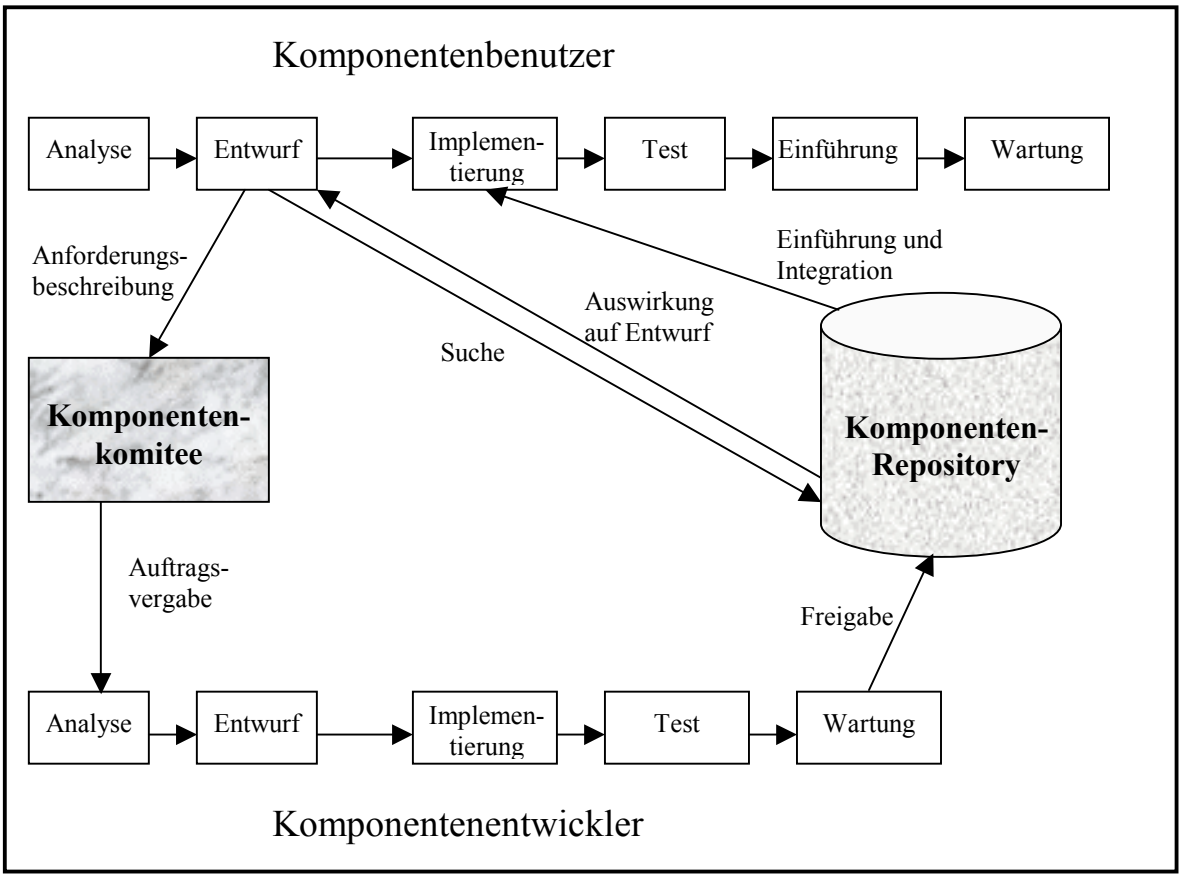
Datenspeicher, z.B. LDAP, Datenbank, Novell Directory Service (NDS) etc. angepaßt werden. Der Nutzen für eine Authentifikation gegen eine Datenbank ist indes eingeschränkt, da zunächst Aufwand entsteht, um eine entsprechende Anpassung vorzunehmen. Kapselung und Polymorphismus, als Grundsätze der objektorientierten Entwicklung spielen somit eine wichtige Rolle bei der Entwicklung und der Wiederverwendbarkeit einer Komponente.

Abbildung 3.2: Verhältnis zwischen Abstraktion und Nutzen einer Komponente



Falls keine passende Komponente in der Komponentenbibliothek (Komponenten-Repository) gefunden werden kann, wird die neue Anforderung an das Komponentenkomitee weitergeleitet. Diese Gruppe von unabhängigen Entwicklern entwickelt dann entsprechend den Anforderungen neue Komponenten mit den klassischen Modellierungstechniken. Ihr Fokus liegt aber dennoch auf der Entwicklung einer möglichst abstrakten Komponente mit entsprechender Funktion. Die fertige Komponente wird anschließend in das Komponenten-Repository eingepflegt und kann von den Designern zum Entwurf des Designmodells genutzt werden.

Abbildung 3.3: Ablauf der Komponentenbasierten Softwareentwicklung



Quelle: Ludger Bischofs: „Grundlagen der komponentenbasierten Softwareentwicklung“, Seminar Komponentenbasierte Softwareentwicklung, Universität Regensburg, 2001, S.12

3.3.3 Vorteile von Softwarekomponenten

Der wohl signifikanteste Vorteil von Komponenten ist deren Wiederverwendungspotential. Mit Hilfe komponentenbasierter Systementwicklung konnten bei Firmen wie AT&T bis zu 92% und bei Ericsson bis zu 90% Wiederverwendungsquote erzielt werden (Quelle: [Jacobson97], S.6).

Somit ergaben sich enorme Vorteile auch für das Management des Unternehmens (vgl. [Jacobson97]):

- Zeitersparnis → Reduzierung der „time to market"
- Kostenreduzierung
- Produktivitätserhöhung im Unternehmen
- Bessere Planbarkeit des Entwicklungsprozesses
- Erhöhung der Zuverlässigkeit, Betriebssicherheit
- Erhöhung der Qualität

Auch für die Komponentenentwickler ist die komponentenbasierte Softwareentwicklung äußerst wertvoll. Jede Komponente ist ein in sich abgeschlossenes Projekt und die Verteilung auf mehrere Teams fällt leichter. Die Kommunikation der Teams muß sich lediglich noch auf die Schnittstellen beziehen. Insgesamt wird die Anwendung dadurch auch weniger komplex und beherrschbarer.

Die Kapselung ganzer Prozeßabläufe führt zu einer Modellierung ganzer Anwendungsfälle (engl. use cases) innerhalb einer Komponente. Diese weitere Abstraktionsschicht macht die Anwendung wartbarer und leichter zu pflegen. Durch die Spezifikation von Schnittstellen bleibt ein System leicht erweiterbar, indem man sich nicht um Interna der Anwendung kümmern muß. Vorteile ergeben sich durch die Schnittstellen auch bei der Integration von Komponenten in Produkte von Drittanbietern oder kundenspezifische Produkte. Ein Beispiel dafür bietet der Oracle Developer, bei dem JavaBeans in die Developer-Softwareumgebung eingelagert werden können.

3.4 Zusammenfassung

Komponenten sind also eine vielversprechende Ergänzung der traditionellen objektorientierten Softwareentwicklung. Aber sinken deswegen die Anforderungen des Kunden? Werden Systeme weniger komplex? Gibt es weniger Anforderungen die Software zu ändern (engl. Change Requests)? Nein, leider nicht. Aber der Umgang mit diesen Anforderungen wird wesentlich vereinfacht und spart Zeit und Kosten. Obwohl es mittlerweile eine Vielzahl von Komponentenmodellen gibt, scheinen sich einige Standards, wie CORBA etablieren zu können. Die Verwendung dieser Modelle stellt den aktuellen

Stand der Softwaretechnik dar. Verteilte Systeme nehmen daher einen immer höheren Stellenwert im Softwaremarkt ein. Leider lassen sich die in der Theorie so einfach wiederzuverwendenden Softwarekomponenten nicht immer erfolgreich einsetzen. Zu unterschiedliche Anforderungen der Individualsoftware machen eine Wiederverwendung oft schwierig. Dennoch hat dieser produktive und effektive Ansatz zumindest aus betriebswirtschaftlicher Sicht eine große Bedeutung.

4 Muster in der Softwareentwicklung

> *„Somewhere in the deeply remote past it seriously traumatized a small random group of atoms drifting through the empty sterility of space and made them cling together in the most extraordinarily unlikely patterns. These patterns quickly learnt to copy themselves (this was part of what was so extraordinary about the patterns) and went on to cause massive trouble on every planet they drifted on to. That was how life began in the Universe."*
> (Douglas Adams: *The Hitchhiker's Guide to the Galaxy*)

Ähnlich zu den von Adams beschriebenen Mustern haben Muster auch einen massiven Einfluß auf die Softwareentwicklung. Im Wesentlichen geht es darum, atomare Teile, z.B. Klassen und Objekte so zu strukturieren, daß sie in ihrer Komposition einen bestimmten Zweck erfüllen und die Entwurfsentscheidungen so zu dokumentieren, daß sie als Entwurfsmuster für weitere Projekte dienen können.

4.1 Der Musterbegriff

Der Begriff des Musters wurde maßgeblich von Christopher Alexander geprägt. Er beschreibt in [Alexander77] zwar die Muster beim Architekturentwurf von Gebäuden und Städten, doch diese lassen sich ebenso auf den objektorientierten Entwurf übertragen. Seine Definition eines Musters ist:

> *„Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so daß Sie diese Lösung beliebig oft anwenden können, ohne sie jemals ein zweites Mal gleich auszuführen."*
> ([Alexander77], S. x)

Muster sind also sowohl in der traditionellen Architektur, als auch in der Softwarearchitektur, also dem Softwaredesign eine abstrakte Lösungsmöglichkeit für ein bestehendes Problem.

Viele Experten des Softwareentwurfs haben sich im Laufe der Zeit schon Gedanken über Lösungsmöglichkeiten bestimmter Probleme gemacht. Dadurch, daß viele dieser Probleme gleichartig sind, sind auch die Lösungen entsprechend ähnlich. Um mit einer bewährten Lösung aber alle gleich gearteten Probleme bewältigen zu können, bedarf es einer gewissen Abstraktion der Lösungen. Diese Abstraktionsebene wird als Muster bezeichnet. Aus der Vielzahl von Lösungsmöglichkeiten haben sich bewährte Prinzipien herauskristallisiert. Diese werden im Englischen auch als „best practise“ bezeichnet.

Entwurfsmuster (Design Patterns) im objektorientierten Entwurf helfen begründete Entscheidungen zwischen verschiedenen Design-Alternativen zu treffen, da die Konsequenzen ihres Einsatzes offengelegt sind. Der Entwickler braucht sich also keine Gedanken über den bestmöglichen Lösungsansatz zu machen, sondern ist lediglich verantwortlich dafür, das richtige Muster auszuwählen und zu implementieren. Kurz und prägnant wird in [Gamma96] zusammengefaßt: „Entwurfsmuster helfen Entwicklern, einen Entwurf schneller „richtig“ zu machen.“ ([Gamma96], S.2)

4.2 Aufbau eines Musters

Die „Gang of Four“ (Gamma, Helm, Johnson und Vlissides), führende Köpfe bei der Entwicklung von Entwurfsmustern, beschreibt in [Gamma96] vier grundlegende Elemente bzw. Charakteristika eines Musters:

- Musternamе
- Problemabschnitt
- Lösungsabschnitt
- Konsequenzabschnitt

Der Musternamе benennt das Muster, um es beim Softwareentwurf in das Vokabular des Entwicklers einfließen zu lassen. Verschiedene Entwickler unterhalten sich so immer über ein und dieselbe Lösungsmöglichkeit und es kommt nicht zu Mißverständnissen. Einen guten und sprechenden Namen für ein bestimmtes Muster zu finden ist eine der schwierigsten Aufgaben bei der Musterbeschreibung.

Der Problemabschnitt eines Musters legt fest, in welchem Kontext welches Problem mit diesem Muster gelöst werden kann. Hier werden die Probleme beschrieben, auf die man möglicherweise beim Softwareentwurf gestoßen ist und für die dieses Muster eine adäquate Lösung bietet. Eine Anwendung dieses Musters ist nur dann sinnvoll, wenn alle in diesem Abschnitt beschriebenen Vorbedingungen erfüllt sind. Andernfalls sollte nach einem besser passenden Muster gesucht werden.

Im Lösungsabschnitt wird der Entwurf dieses Musters beschrieben. Dazu zählen Beziehungen, Zuständigkeiten und Interaktionen der Entwurfsobjekte. Er definiert den Teil des Musters, welcher einen eigenen Entwurf überflüssig macht und an dessen Stelle verwendet werden sollte. Eine festgelegte Implementierung ist hier jedoch nicht zu finden. Es wird lediglich eine Vorlage geliefert, die auf viele verschiedene Situationen angewendet werden kann.

Der Konsequenzabschnitt beschreibt die Konsequenzen, die eine Anwendung dieses Musters mit sich bringt. Dadurch ist es einem Anwender des Musters möglich, schon vor einer konkreten Implementierung einer Lösung zu sehen, wohin diese Lösung führt und welche Ergebnisse sie liefert. Bei einem eigenen Entwurf ist dies nicht möglich. Sind die Vor- und Nachteile dieses Mustereinsatzes bekannt, kann man das Muster von anderen Entwurfsalternativen abgrenzen und überprüfen, ob es den eigenen Anforderungen gerecht wird. Die Konsequenzen einer Musteranwendung beschreiben auch den Einfluß des Musters auf Flexibilität, Erweiterbarkeit und Portabilität eines Systems.

4.3 Dokumentation eines Musters

Zusätzlich zu den in [Gamma96] erläuterten Punkten über Aufbau eines Musters sind noch weitere Aspekte zu berücksichteigen, um eine umfassende Dokumentation eines Musters zu erstellen. Im Gegensatz zur herkömmlichen grafischen Notation eines objektorientierten Entwurfs ist zur Dokumentation eines Musters die schriftliche Form geeigneter. Um auch dabei einen Standard festzulegen, muß ein einheitliches Format dieser Beschreibung festgelegt werden. In der Literatur finden sich diverse Beschreibungsmöglichkeiten für ein Muster. Häufig erweitern diese jedoch nur die von der „Gang of Four“ vorgeschlagene Dokumentationsmethode.

Tabelle 4.1: Dokumentationsformen von Mustern

Mustername und Klassifizierung	Der Mustername vermittelt knapp und präzise den wesentlichen Gehalt des Musters. Die Klassifizierung ordnet das Muster einer bestimmten Oberklasse zu.
Zweck	Der Zweckabschnitt ist eine kurze Darstellung, der folgende Fragen beantworten soll: Was macht das Muster? Was sind sein Grundprinzip und Zweck? Welche spezifischen Fragestellungen oder Probleme im Entwurf behandelt es?
Auch bekannt als	Welche anderen Namen für das Muster gibt es?
Motivation	Der Motivationsabschnitt schildert ein Entwurfsproblem und wie die Klassen- und Objektstrukturen des Musters das Problem lösen.
Anwendbarkeit	Beschreibt in welchen Situationen das Muster angewendet werden kann. Es benennt Problemsituationen, in denen das Muster helfen kann und wie man diese Situation erkennt.
Struktur	Das Strukturdiagramm besteht aus einer grafischen Repräsentation der Klassen im Muster, dargestellt in einem objektorientierten Klassenmodell (oft in UML-Notation).
Teilnehmer	Dieser Abschnitt beschreibt die am Muster beteiligten Klassen und Objekte sowie ihre Zuständigkeiten.
Interaktionen	Beschreibt, wie die Komponenten zur Erfüllung der gemeinsamen Aufgabe zusammenarbeiten.
Konsequenzen	Dieser Abschnitt diskutiert, wie das Muster seine Ziele zu erreichen sucht, welche Vor- und Nachteile sich durch die Anwendung des Musters ergeben, welche Ergebnisse zu erwarten sind.
Implementierung	Dieser Abschnitt beschreibt die Tips und Techniken zur Implementierung sowie die Fallen, die bei ihr auftreten können.
Beispielcode	Diskutiert Codefragmente, wie das Muster z.B. in C++ oder Smalltalk implementiert werden kann.
Bekannte Verwendungen	Dieser Abschnitt führt Beispiele für das Muster auf, die in echten Systemen zu finden sind.
Verwandte Muster	Dieser Abschnitt setzt das Muster in Bezug zu anderen Mustern, diskutiert die relevanten Unterschiede und erläutert, mit welchen Mustern das Muster zusammen verwendet werden kann.

Quelle: Vgl. [Gamma96], S.7-9

Es hat sich also eine Art Standard gebildet. Die einheitliche Grundlage für die Dokumentation eines Musters ist unumgänglich, da es dem Softwaredesigner möglich sein muß, aus einem vorhandenen Katalog (engl. Repository) von Mustern wählen zu können. Dabei muß er die Muster leicht miteinander vergleichen und gegeneinander abgrenzen können.

4.4 Klassifikation von Mustern

Bei näherer Betrachtungsweise der Muster verschiedener Kataloge fällt auf, daß sich die Muster erheblich im Umfang und in ihrer Abstraktionsebene unterscheiden. Es gibt Muster die bei der Strukturierung von Softwaresystemen unterstützen. Andere wiederum helfen ein Kommunikationsproblem von Subsystemen oder Komponenten beim Softwareentwurf zu lösen. Darüber hinaus existieren auch Muster, die ein kontextsensitives Problem programmiersprachenabhängig lösen oder zumindest Vorschläge für dessen Lösung liefern. Muster sind auch nicht immer unabhängig von ihrer Anwendungsdomäne. Einige beschreiben allgemeingültige Sachverhalte, wohingegen andere speziell auf eine Domäne zugeschnitten sind (Vgl. [Buschmann98], S.11f). So gibt es z.B. Muster, die bestimmte Transaktionen bei einem Finanzdienstleister beschreiben. Eine Übertragung auf eine andere Anwendungsdomäne ist nicht ratsam.

Nun stellt sich die Frage, wie man Muster in einer passenden Struktur klassifiziert und strukturiert. Zunächst lassen sich laut [Buschmann98] drei grobe Kategorien von Mustern erkennen:

- *Architekturmuster*
- *Entwurfsmuster*
- *Idiome*

Eine Kategorie umfaßt Muster, die Softwaresysteme desselben Umfangs betreffen oder auf derselben Abstraktionsebene liegen. Diese Unterteilung läßt sich auf wichtige Phasen im Softwareentwicklungsprozeß abbilden. Architekturmuster werden zu Beginn der Entwurfsphase benötigt, um grobe Grundstrukturen festzulegen. Entwurfsmuster helfen während der gesamten Entwurfsphase wichtige Entwurfsentscheidungen zu treffen. Idiome beziehen sich auf das Ende der Entwurfsphase und werden häufig bei der Implementierung

verwendet. Im Hinblick auf das weite Einsatzgebiet von Entwurfsmustern werden diese im Folgenden am intensivsten behandelt.

4.4.1 Architekturmuster

Wie der Name schon vermuten läßt, beschäftigen sich Architekturmuster mit der Architektur eines Softwaresystems. Dieser Typ von Mustern beschreibt den Aufbau, also die Struktur, von Softwaresystemen. Sie bilden Vorlagen für die konkrete Anwendungsarchitektur, unterteilen das System in Subsysteme und spezifizieren deren Zuständigkeitsbereich. Architekturmuster befinden sich auf der höchsten Abstraktionsebene eines Systems. Sie stellen somit ein Grundgerüst der Anwendung dar und sind Ausgangspunkt für die gesamte Systemarchitektur.

Das Model-View-Controller-Muster ist das wohl bekannteste Beispiel für ein Architekturmuster. Es spezifiziert zunächst lediglich die Struktur eines Softwaresystems. Die Beziehungen zwischen den in diesem Muster festgelegten Subsystemen beschreiben hingegen Entwurfsmuster.

4.4.2 Entwurfsmuster

Die Entwurfsmuster sind Vorlagen zur Verfeinerung der Zuständigkeiten und der Kommunikation der Subsysteme. Sie modellieren Kommunikationsmodelle der Komponenten und legen fest nach welchen Regeln diese Kommunikation stattfindet. Es werden also Beziehungen der Subsysteme zueinander beschrieben.

Die Abstraktionsebene der Entwurfsmuster bildet die mittlere Schicht der drei Abstraktionsebenen. Sie ist spezieller als Architekturmuster, aber dennoch programmiersprachenunabhängig und somit allgemeiner als Idiome.

Einen Schritt weiter in der Unterteilung von Mustern geht [Gamma96]. Die Klasse der Entwurfsmuster wird dort noch weiter aufgegliedert. Ein Entwurfsmuster kann demnach verschiedene Ausprägungen annehmen. Es kann ein *Erzeugungsmuster* (engl. creational pattern), ein *Strukturmuster* (engl. structural pattern), aber auch ein *Verhaltensmuster* (engl. behavioral pattern) sein.

Ein Kriterium bei der Aufteilung in diese Kategorien ist die *Aufgabe*, welche das Muster zu erfüllen hat. Es kann die Aufgabe haben, Objekte zu erzeugen, deren Struktur zu verwalten oder deren Verhalten zu steuern. Ein Erzeugungsmuster legt z.B. neue Objekte an. Diese Objekte und Klassen können von einem Strukturmuster zusammengesetzt werden. Das Verhaltensmuster beschreibt dabei die Interaktion und Kommunikation der Objekte, also die Beziehungen untereinander.

Ein weiteres Kriterium ist der *Gültigkeitsbereich* von Mustern. Muster können sich sowohl auf Klassen, als auch auf Objekte beziehen. Klassenbasierte Muster beziehen sich auf Klassen und ihre Subklassen, wie sie zur Entwurfszeit festgelegt werden. Diese Klassenbeziehungen werden durch Vererbungshierarchien beschrieben. Anders werden objektbasierte Muster eingesetzt. Sie definieren die Objektbeziehungen und ihre Interaktionen zur Laufzeit des Systems. Einen Katalog von den wichtigsten und bekanntesten Mustern findet sich in [Gamma96].

Tabelle 4.2: Dimensionen der Ausprägung von Entwurfsmustern

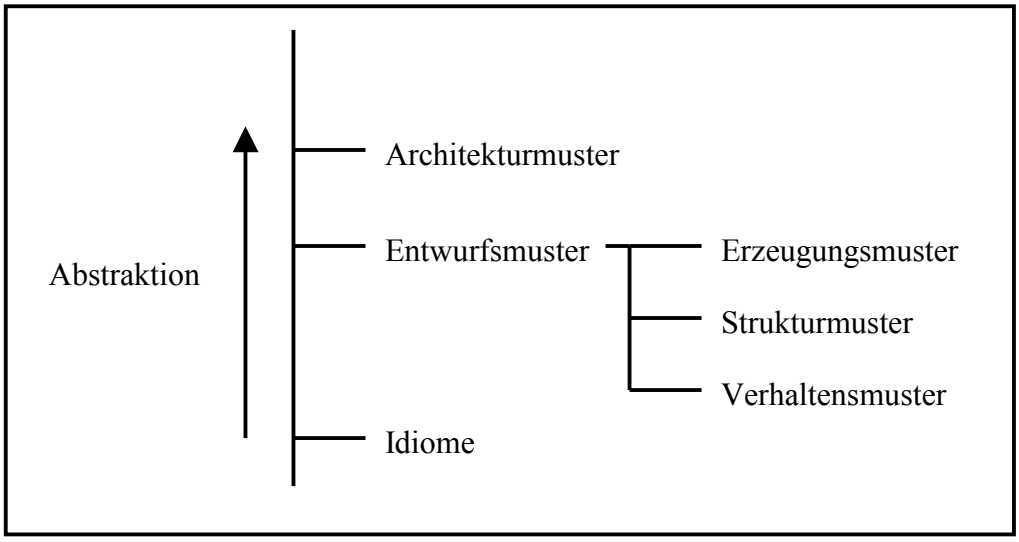
		Aufgabe		
		Erzeugungsmuster	**Strukturmuster**	**Verhaltensmuster**
Gültigkeits-bereich	**klassen-basiert**	Fabrikmethode	Adapter (klassenbasiert)	Interpreter Schablonenmethode
	objekt-basiert	Abstrakte Fabrik Erbauer Prototyp Singleton	Adapter (objektbasiert) Brücke Dekorierer Fassade Fliegengewicht Kompositum Proxy	Befehl Beobachter Besucher Iterator Memento Strategie Vermittler Zustand Zuständigkeitskette

Quelle: In Anlehnung an Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl., Addison-Wesley, Bonn, 1996, S.12

4.4.3 Idiome

Leider lassen sich nicht alle Probleme des Softwareentwurfs auf hohen Abstraktionsebenen lösen. Softwareentwurf bedeutet auch eine Schnittstelle zwischen abstraktem Entwurf und Implementierung zu schaffen. Die gleiche Aufgabe wird den Idiomen zu teil. Sie berücksichtigen in ihrer Beschreibung auch wichtige Implementierungsdetails. Der Nachteil solcher Idiome ist, daß ihre Einsatzmöglichkeiten und Leistungen stark eingeschränkt sind. Sie können nicht mehr programmiersprachenübergreifende Probleme lösen. Allerdings sind auch solche Muster notwendig, wenn es darum geht, stark optimierte Entwürfe zu erstellen. Idiome sind somit sehr stark an das zu implementierende Softwaresystem angepaßt und nehmen dem Designer mehr Arbeit ab, als das von abstrakteren Mustern getan werden kann.

Abbildung 4.1: Klassenstruktur von Mustern



4.5 Zusammenfassung

Zusammenfassend läßt sich über die Verwendung von Mustern in der Softwareentwicklung sagen, daß sie einen großen Teil zur vereinfachten Handhabung von Entwurfsprozessen beitragen. Sie unterstützen die Wiederverwendung von Lösungen, dokumentieren existierende und erprobte Entwurfserfahrungen und benennen und erklären wichtige Entwürfe. Der Designer wird von ihnen maßgeblich bei der Auswahl von

verschiedenen Entwurfsalternativen unterstützt. Werden die Muster sorgfältig ausgewählt, helfen sie nicht nur *eine*, sondern auch die *richtige* Entscheidung zu treffen.
Dennoch liegt noch ein Großteil der Verantwortung beim Designer. Die richtige Auswahl eines Musters zu treffen ist nicht immer wesentlich einfacher als einen eigenen Entwurf zu erstellen.

5 Frameworks

5.1 Was ist ein Framework?

Einleitend wurden nun schon verschiedene Wiederverwendungstechniken vorgestellt. Letztendlich ist ein Framework auch nur eine weitere Art der Softwarewiederverwendung. In der Softwareentwicklung gehen die Fähigkeiten dabei jedoch über die bisher vorgestellten Techniken hinaus und erweitern diese, um eine möglichst *effiziente* Wiederverwendung von Software zu ermöglichen. Ein Framework im allgemeinen Sinne ist ein wiederverwendbares System, welches in fertige und halbfertige Subsysteme untergliedert ist. Es legt dabei die Struktur dieser Systeme und Subsysteme fest. Die Wiederverwendbarkeit wird dabei durch einen abstrakten aber festgelegten Rahmen und die halbfertigen Subsysteme gewährleistet. Der abstrakte Rahmen legt somit ein Muster für eine breite Vielfalt von Anwendungsfällen fest. Die Abstraktion innerhalb des Frameworks darf gerade soweit gehen, daß es zwar einfach zu verwenden, aber dennoch für eine große Familie von Anwendungsfällen passend ist. Es gibt dem Benutzer die Möglichkeit, das Design und auch Teile der Implementierung eines Systems wiederzuverwenden. Die Subsysteme im Framework bilden den Ansatzpunkt für eine anwendungsspezifische Erweiterung oder Implementierung des Frameworks. Das Herleiten einer spezifischen Anwendung heißt *Instanziierung* des Frameworks.

Der Begriff des Frameworks bezieht sich in dieser Definition nicht allein auf ein Framework im softwaretechnischen Sinne. Seine Eigenschaften lassen sich auch auf andere Anwendungsdomänen übertragen. Bei der Konfiguration eines Automobils z.B. wird der Rahmen, also das Zusammenspiel der Einzelkomponenten, schon vorgegeben. Das Auto hat fest definierte Komponenten, wie die Karosserie, deren Funktionsweise und Schnittstellen, z.B. zum Motor und den Rädern, vorbestimmt sind. Welche Farbe und Form (Fließ- oder Stufenheck) diese schließlich aber annimmt bleibt den Wünschen des Kunden überlassen. Sie ist somit ein halbfertiges Subsystem des Autos, dessen Ausprägung nicht vordefiniert ist.

Im objektorientierten Sinne bedeutet das, daß ein fertiges Subsystem durch konkrete Klassen bzw. Klassenstrukturen repräsentiert wird. Halbfertige Subsysteme definieren

lediglich die Schnittstellen und minimale Funktionalität und werden über abstrakte Klassen abgebildet. Der Entwickler muß, um das Framework zu instanziieren, diese konkreten Klassen ggf. kombinieren und die abstrakten Klassen ableiten.

Das Framework bestimmt die Architektur einer Anwendung und definiert die grobe Struktur, seine Unterteilung in Klassen und Objekte, die jeweiligen zentralen Zuständigkeiten, die Zusammenarbeit der Objekte sowie den Kontrollfluß. Diese Entwurfsparameter werden vom Framework festgelegt und erlauben es dem Benutzer, sich auf die spezifischen Details der Anwendung zu konzentrieren. Von der Anwendung unabhängige Entwurfsentscheidungen braucht der Designer, der das Framework benutzt, also nicht mehr zu treffen (vgl. [Gamma96], S.31).

Häufig wird der Frameworkbegriff auch synonym für die Begriffe Referenzmodell, Architektur und Programmvorlage (engl. template) verwendet. Dies ist allerdings nicht ganz korrekt, da jeder dieser Begriffe für sich nur einen Teilbereich oder Funktionalität eines echten Frameworks abdeckt.

Ein Framework ist also eine wiederverwendbare, halbfertige Anwendung, die durch Spezialisierung und Implementierung dazu genutzt werden kann, angepaßte Anwendungen für bestimmte Anwendungsdomänen zu entwickeln [Fayad99a].

5.2 Abgrenzung von Frameworks zu anderen Wiederverwendbarkeitstechniken

Die Anwendungsprogrammierung im objektorientierten Umfeld kann durch den Einsatz verschiedener Wiederverwendungstechniken enorm beschleunigt und verbessert werden. Zu diesen Wiederverwendungstechniken zählt der Einsatz von Klassenbibliotheken, von komponentenbasierten Systemen und von Entwurfsmustern (engl. Design Patterns). Jede dieser Techniken zielt auf einen bestimmten Teil des Softwareentwicklungsprozesses, der wiederverwandt werden kann. D.h. Muster dienen dazu, eine bestimmte Entwurfsentscheidung nicht mehrmals treffen zu müssen. Man kann sich also auf erprobte Muster verlassen und diese erneut einsetzen. Softwarekomponenten haben als Kontext weniger den Entwurf, als eine konkrete Implementierung einer Funktionalität. Sie stellen in der Regel einen Dienst zur Verfügung, welcher später gekapselt genutzt und wiederverwendet werden kann. Auch Klassenbibliotheken beziehen sich auf die Wiederverwendbarkeit von Code. Sie enthalten jedoch sehr kleine Einheiten von Funktionalität. Eine Kapselung als eigener Dienst ist bei korrektem Entwurf des Systems nicht möglich. Zudem fehlt in der Klassenbibliothek eine Spezifikation der Kommunikationsprozesse.

Jede Technik für sich bietet also ganz bestimmte Vor- und Nachteile bei der Anwendungsentwicklung. Leider stellen sie keine befriedigende Lösung dar, um den Anforderungen an aktuelle Entwicklungsprojekte gerecht zu werden. Warum sollte man die Vorteile der konventionellen Technologien nicht verbinden und gemeinsam nutzen?

Frameworks integrieren Komponenten, um bestimmte statische Dienste zur Verfügung zu stellen. Die Komponenten und ihre Schnittstellen bilden den Modularen Aufbau des Frameworks. Sie ermöglichen eine Untergliederung des Frameworks in Systeme und Subsysteme. Der modularisierte Aufbau verleiht dem Framework die Fähigkeit einfach neue Komponenten einzubauen oder aus dem Rahmen herauszuziehen. Frameworks erben somit die Erweiterbarkeit eines komponentenbasierten Softwaresystems.

Zusätzlich zu den Komponenten geben Frameworks allerdings auch noch die Struktur dieser Komponenten vor. Das Framework spezifiziert auch schon große Teile der Komponenteninteraktion. Diese Beziehungen stellen sicher, daß die Funktionalitäten des

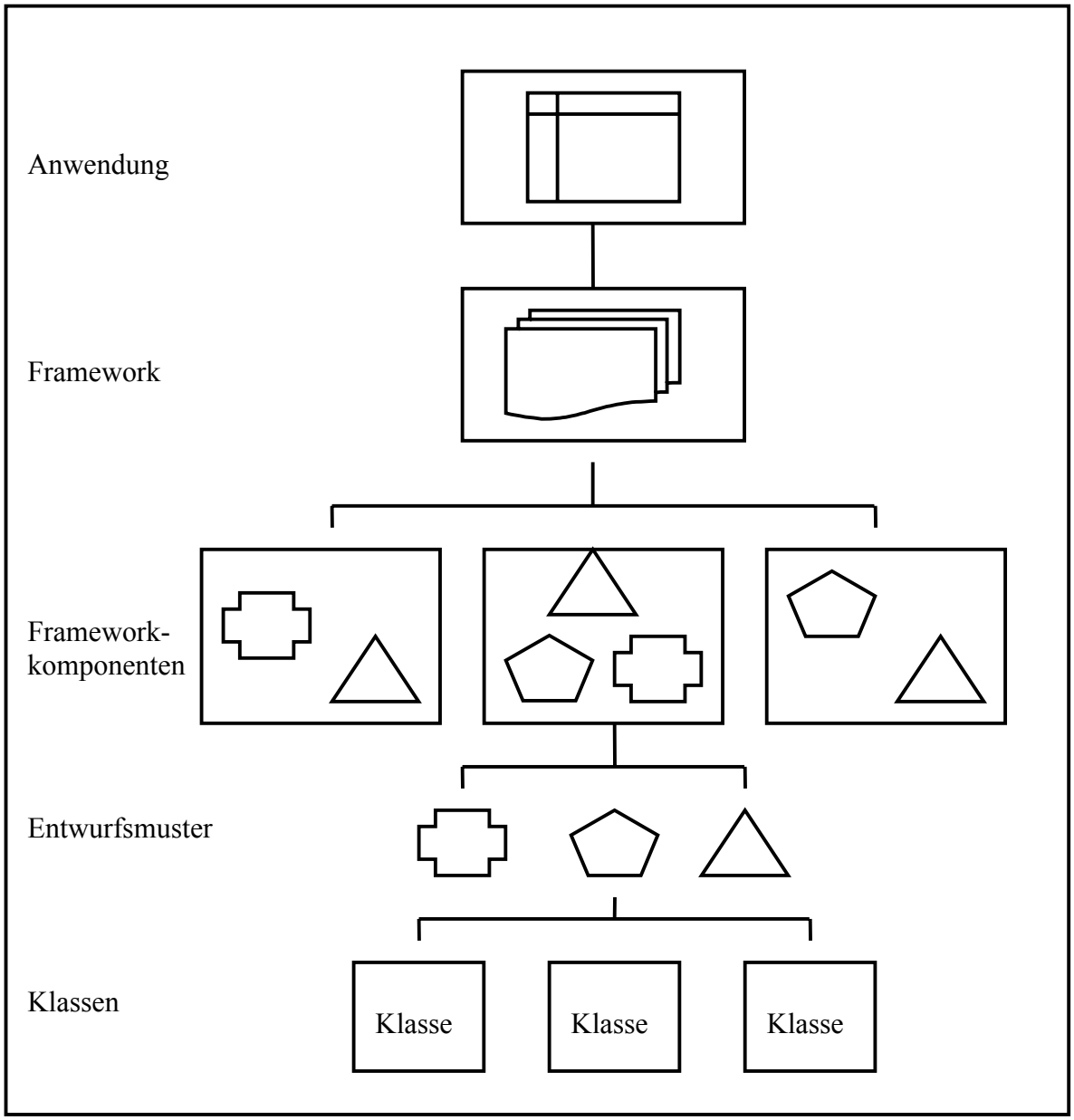
Frameworks durch eine festgelegte Komponentekommunikation gegeben sind. Die Punkte an denen das Framework instanziiert, also wirklich auf eine Anwendung angepaßt werden kann, werden bei der Frameworkerstellung entworfen. Das Framework enthält dadurch auch schon wesentliche Teile des Softwareentwurfs in seinen halbfertigen Subsystemen. Dem Benutzer wird zwar die Freiheit eingeräumt, das Framework für seine Anwendungsentwicklung zu nutzen, aber er muß sich dabei an die Entwurfsentscheidungen halten, die beim Frameworkdesign getroffen wurden. Diese Entscheidungen sind dementsprechend robust und vermeiden adäquat zu den Entwurfmustern eine Fehlentscheidung des Entwicklers beim Entwurf der endgültigen Anwendung.

Zusammenfassend bedeutet dies, daß Frameworks sowohl Eigenschaften eines komponentenbasierten Systems haben, als daß sie auch die Entwurfsvorlagen in Form von Mustern beinhalten. Ralph E. Johnson beschreibt diese Enthaltenseinsbeziehung folgendermaßen:

> *„Frameworks = (Components + Pattern)*" ([Johnson97], S.39)

Eine klare Abgrenzung findet demnach also nicht statt. Die Struktur gestaltet sich eher in Form einer Beziehung zwischen der Anwendung als spezifischen Instanziierung des Frameworks, dem Framework selber, den Entwurfsmustern, Komponenten und Klassen. Klassen arbeiten anhand von Entwurfsmustern zusammen. Sie vereinigen sich dabei zu Komponenten, die bestimmte Funktionalitäten zur Verfügung stellen. Auch die Komponenten selber werden entsprechend von bewährten Architekturmustern konfektioniert. Zusammen bilden alle Komponenten das eigentliche Framework.

Abbildung 5.1: Struktureller Aufbau eines Frameworks



Quelle: In Anlehnung an IBM: *Building Object-Oriented Frameworks*, S. 10

5.3 Vor- und Nachteile von Frameworks in der Softwareentwicklung

> *„Object-oriented application frameworks are a promising technology for reifying proven software designs and implementations in order to reduce the cost and improve the quality of software. A framework is a reusable, semi-complete application that can be specialized to produce custom applications.“*
> ([Fayad99a], S.4)

Wie schon in [Fayad99a] erwähnt, haben Frameworks den großen Vorteil, daß sie bereits fertig entworfene und implementierte Softwarestrukturen nutzen können. Dadurch wird vermieden, falsche Entwurfsentscheidungen zu treffen oder Code zu entwickeln, welcher sich als nicht geeignet herausstellt. Durch einen genau durchdachten Entwurf des Frameworks und die ständige Überarbeitung während des Instanziierungsprozesses entwickelt sich mit der Anzahl der Anwendungen ein sehr robustes und qualitativ hochwertiges Framework. Zudem ist der Entwickler nicht gezwungen Grundfunktionalitäten, die in jeder Anwendung vorkommen, jedesmal neu zu implementieren.

Da beim ersten Entwurf einer Anwendung, welche heutzutage zumeist hoch komplex ist, kein perfekter Entwurf möglich ist, und im nachhinein viele Entwurfsentscheidungen revidiert werden müssen, ist der Einsatz eines getesteten und stetig weiterentwickelten Frameworks unerläßlich. Leider rechnet sich der enorm hohe Einsatz zur Entwicklung eines guten Frameworks erst nach der Entwicklung entsprechend vieler Anwendungen. Um einen entsprechend hohen Qualitätsstandard zu erreichen, ist es für das Framework notwendig, mit ihm mehrere Anwendungen zu entwickeln, die Fehler bei der Frameworkentwicklung aufdecken. Diese müssen demnach nur einmal im Framework und nicht bei jeder Anwendung ausgebessert werden.

Ein weiterer Aspekt eines Frameworks ist dessen Komposition. Demzufolge soll das System aus mehreren autarken Komponenten bestehen. Sie sollen das System in kleinere, frei kombinierbare Einheiten unterteilen. Zusätzlich müssen Mittel gegeben sein, die eine Verbindung zwischen diesen Elementen herstellen können. Das Gesamtsystem muß sich also aus Komponenten und Verbindungsstücken zusammensetzen lassen.

Doch welchen Vorteil bietet die Komposition von Elementen? Von Vorteil ist dementsprechend, daß sich ein Framework in eine Menge von autarken Elementen, wie Komponenten und Klassenbibliotheken, aufspalten läßt. Es ist somit sehr viel übersichtlicher als eine in sich geschlossene, komplexe Anwendung. Einzelne Elemente des Frameworks können flexibel genutzt, angepaßt oder verworfen werden.

Die Wiederverwendung von Design und Implementierung vieler Anwendungskomponenten findet auf einer sehr hohen Ebene statt. Demnach können die auf einem Framework basierenden Anwendungen Design und Code wesentlich effektiver wiederverwenden, als dies mit konventionellen Technologien möglich war. Ein Beispiel dafür ist der heute schon häufige Einsatz von wiederverwendbaren Bibliothekskomponenten zur Entwicklung graphischer Benutzeroberflächen. Frameworks bieten infolgedessen den derzeit höchsten Grad an Wiederverwendbarkeit.

Die Abstraktionsstufen innerhalb eines Framework geben ihm zusätzlich ein gewisses Maß an Unabhängigkeit. Dies kann z.B. die Unabhängigkeit von gewissen Anwendungsplattformen oder einer grafischen Anwendungsoberfläche sein, indem oberflächenunabhängige Schnittstellen zum Aufruf der Funktionalität geschaffen werden.

Erfreulicherweise sind Frameworks auch in ihrer Anwendungsvielfalt nicht zwangsläufig auf bestimmte Anwendungsdomänen festgelegt. Sie können z.B. sowohl als Entwicklungsgrundlage für eine grafische Benutzungsoberfläche in der Finanzbranche dienen, als auch für ein Prozeßsteuerungssystem in der Automobilindustrie.

Einerseits vereinfachen Frameworks also die Anwendungsentwicklung durch die bereitgestellten Funktionalitäten. Andererseits erhöhen sie aber auch die Abstraktionsebene der Anwendungsentwicklung, infolgedessen die programmierten Anwendungen eine bessere Übersichtlichkeit besitzen. Folglich werden die Anwendungen wesentlich wartbarer [Bensberg01]. Eine einheitliche Grundlage aller mit dem Framework entwickelten Anwendungen ermöglicht auch anderen Entwicklern ein leichtes Verständnis der Gesamtanwendung. Die nötige Wartung bezieht sich nicht mehr auf alle Komponenten, sondern nur noch auf die flexiblen, erweiterbaren Einheiten des Frameworks.

Die durch den Benutzer anpaßbaren Komponenten des Frameworks sind durch Schnittstellen festgelegt. Die Festlegung von Punkten an denen eingegriffen werden kann, macht es möglich, Frameworks in komfortable Werkzeuge zur Anwendungsentwicklung zu integrieren. In ihnen müssen diese Schnittstellen nur noch implementiert und erweitert werden. Ein Beispiel dafür bieten viele, auf die Entwicklung von grafischen Oberflächen abgestimmten Werkzeuge. Sie können z.B. mit Hilfe der JavaBeans Spezifikation Frameworks instanziieren und einfache Anwendungen generieren.

Die Erweiterbarkeit ist ein weiterer Vorteil von Frameworks. Sie bieten dem Entwickler feste Punkte an denen er eingreifen und das Framework konfigurieren und erweitern kann. Pree beschreibt dies in [Pree97] mit seinem Hot-Spot-orientierten Entwurfsansatz, auf den später noch genauer eingegangen wird. Ebenso liefert [Fayad99a] einen entsprechenden Ansatz, bei dem sogenannte *Hooks* entworfen werden. Sie dokumentieren einen Teil einer Frameworkschnittstelle in Form eines bestimmten Schemas und beschreiben die genaue Vorgehensweise für einen Entwickler, der das Framework an dieser Stelle implementieren will. Es bleibt für den Entwickler, der das Framework erweitert, also immer sichergestellt, daß das Framework sich immer in einem konsistenten Zustand befindet, solange er sich an die definierten Schnittstellen (engl. Interfaces) hält.

Diese Technik führt bei ausgereiften Frameworks laut [Weinand89] zu einer Reduktion des zu schreibenden Quellcodes von bis zu 90% gegenüber Software, die mit Hilfe einer konventionellen Funktionsbibliothek geschrieben wurde [Pree97].

Die Programmierung von Anwendungen kann durch den Einsatz von Frameworks signifikant erleichtert und verbessert werden. Insbesondere objektorientierte Programmierung mit der Möglichkeit, Objekte leicht ändern und erweitern zu können, vermag dem Einsatz von Frameworks einen deutlichen Vorteil gegenüber herkömmlichen Wiederverwendungstechniken zu geben.

5.4 Klassifizierung von Frameworks

5.4.1 Typisierung von Frameworks anhand von Erweiterungstechniken

Im folgenden sollen lediglich die objektorientierten Anwendungsframeworks (engl. Application Frameworks) untersucht und eingeordnet werden. Sie lassen sich auf unterschiedliche Arten unterteilen. Zunächst ist es möglich, eine Einordnung anhand der zu benutzenden Erweiterungstechniken vorzunehmen. Es werden dabei folgende Typen von Frameworks unterschieden (vgl. [Johnson88]):

- *Black-Box*
- *White-Box*
- *Gray-Box*

5.4.1.1 Black-Box Frameworks

Black-Box Frameworks zeichnen sich dadurch aus, daß sie Schnittstellen definieren, an denen der spätere Anwendungsentwickler mit Hilfe der Objektkomposition schnittstellenkonforme Komponenten anbringen kann. Über diese Schnittstellen wird das Gesamtsystem somit konfektionier- und erweiterbar. Funktionalität wird wiederverwendet, indem zunächst Komponenten, die dem speziellen Interface entsprechen, erzeugt und anschließend z.B. über Strategie-Muster [Gamma96] eingebunden werden [Fayad99a].

Konfigurationen und Anpassungen an die spezifischen Bedürfnisse des Anwenders werden in Black-Box Frameworks häufig durch sogenannte Konfigurationsobjekte erzielt. Sie werden auch vom Framework bereitgestellt und erlauben dem Anwender nur bedingte und vorher festgelegte Eingriffsmöglichkeiten.

Der Black-Box Ansatz ist damit ein Idealfall eines Frameworks. Der Anwender muß sich nur noch um die Konfiguration des Systems kümmern und das Framework stellt eine robuste Anwendung zur Verfügung. Dieser etwas blauäugige Ansatz ist in der Realität leider fast nie zu erreichen. Oft ist ein solches Framework durch einen zu starren Entwurf nicht mehr nützlich für die Entwicklung einer breiten Anwendungspalette. Es muß also schon bei der Entwicklung des Frameworks auf sehr viele verschiedene Anwendungsfälle Rücksicht genommen werden. Folglich ist diese Art von Frameworks zwar einfach zu benutzen aber ihre Entwicklung ist extrem schwierig und aufwendig.

5.4.1.2 White-Box Frameworks

Der White-Box Ansatz bei der Frameworkentwicklung basiert sehr stark auf den Techniken der objektorientierten Softwareentwicklung. Insbesondere sind das die Vererbung und das dynamische Binden. Mit diesen Hilfsmitteln wird eine einfache und flexible Art der Erweiterung möglich gemacht. An Stellen, die zur Erweiterung des Frameworks vorgesehen sind, werden dabei Basisklassen, die das Framework zur Verfügung stellt, abgeleitet und Methoden mit Hilfe von Mustern, wie dem Template-Muster [Gamma96] überschrieben. Diese Methoden werden z.B. in Form von Hooks [Fayad99a] dokumentiert, wobei ihre Auswirkungen auf das Gesamtsystem beschrieben werden. Eine genaue Dokumentation bzw. Kenntnis des Frameworks ist für den Entwickler von essentieller Bedeutung, da er sonst nicht abschätzen kann, wie sich ein Eingriff in den Frameworkablauf auswirkt.

Zusammengefaßt läßt sich also sagen, daß White-Box Frameworks einen sehr flexiblen aber evtl. hoch komplexen Rahmen zur Erweiterung bzw. Instanziierung eines Frameworks liefern.

5.4.1.3 Gray-Box Frameworks

Die Klasse der Gray-Box Frameworks versucht Vorteile der beiden anderen Ansätze zu nutzen um deren Nachteile zu eliminieren. Das bedeutet für ein gutes Gray-Box Framework, daß es genug Flexibilität und Erweiterbarkeit mitbringt, aber auch die Fähigkeit besitzt, überflüssige Informationen und Arbeitsabläufe vor dem Benutzer zu verbergen [Fayad99b]. Johnson und Foote beschreiben das Verhältnis zwischen Black- und White-Box Frameworks folgendermaßen:

> *„In fact, as the design of a system becomes better understood, black-box relationships should replace white-box ones. Black-Box relationships are an ideal towards which a system should evolve.“*
> ([Johnson88], S. 26)

Alle Frameworks befinden sich in einem ständigen Entwicklungsprozeß. Damit geht meistens auch eine, der Aussage von Johnson und Foote entsprechende Entwicklung

einher. Teile des Frameworks werden zunächst als offene Komponenten gemäß des White-Box Ansatzes entwickelt. Je mehr Applikationen mit Hilfe des Frameworks entwickelt werden, desto mehr fertig implementierte Komponenten entstehen für diese Schnittstellen. Ist der Bedarf an neuen Komponenten gedeckt, kann man die bisher entwickelten nach entsprechender Bearbeitung in das Framework integrieren. Man bewegt sich also häufig auf einem Mittelweg zwischen White- und Black-Box Frameworks.

5.4.2 Typisierung von Frameworks anhand von Anwendungsbereichen

Neben der technischen Unterteilung von Frameworks in Black- und White-Box-Frameworks muß auch eine Unterteilung durchgeführt werden, die den Anwendungskontext berücksichtigt. Hierbei kommt es darauf an welche Aufgaben das Framework übernehmen soll und wobei es den Entwickler unterstützen muß. Fayad, Schmidt und Johnson unterteilen Frameworks in drei Bereiche [Fayad99a]:

- System Infrastructure Frameworks
- Middleware Integration Frameworks
- Enterprise Application Frameworks

Im folgenden beziehe ich mich auf die System Infrastructure und Middleware Integration Frameworks als technische Frameworks und auf die Enterprise Application Frameworks als fachliche Frameworks.

5.4.2.1 Technische Frameworks

Technische Frameworks haben die Aufgabe, die Entwicklung der technischen Systeminfrastruktur einfach, portabel und effizient zu gestalten. Ein Beispiel für System Infrastructure Frameworks sind Betriebssysteme [Campbell93] oder Systeme zur Kommunikation auf technischer Ebene [Schmidt97]. Des weiteren stellen technische Frameworks in Form von Middleware Integration Frameworks eine Möglichkeit zur Verteilung von Anwendungen und deren Komponenten auf mehrschichtige Architekturen dar. Diese Frameworks helfen Entwicklern dabei, ihre Applikationen in Komponenten zu unterteilen, wiederzuverwerten und zu erweitern [Fayad99a].

Mehrschichtige Architekturen sind heutzutage der Inbegriff von modernen Unternehmensanwendungen. Gerade im Bereich des Internets greifen viele Firmen auf die Vorteile verteilter Systeme zurück. Die Verteilung einer Applikation z.B. auf Client, Server und Datenbank erfordert dabei immer wiederkehrende Mechanismen, um zwischen diesen Schichten zu kommunizieren. Eine große Hilfe und der einzige Garant für echte Wiederverwendbarkeit sind dabei Frameworks, die z.B. die CORBA Technologie nutzen um entfernte Komponenten anzusprechen. Die Ausprägungen, bzw. die Funktionen die diese Frameworks bieten sind extrem unterschiedlich. Allerdings beruhen sie im wesentlichen auf einem einheitlichen Standard.

Verteilte Komponenten sind nicht die einzige Anwendungsmöglichkeit solcher Frameworks. Es werden auch Frameworks angeboten, die die Kommunikationsprozesse zwischen Middleware und Datenbank steuern und ein objektrelationales Abbilden von relationalen Datenstrukturen ermöglichen. Ein Beispiel für ein solches Persistenzframework sind die *Business Components for Java* von ORACLE, die in Kapitel 5.5 näher beschrieben werden.

5.4.2.2 Fachliche Frameworks

Die Klasse der fachlichen oder Enterprise Application Frameworks [Fayad99a] wird durch ihre fachlichen Komponenten charakterisiert. Fachliche Komponenten beziehen sich auf bestimmte Anwendungsdomänen. Dies muß nicht notwendigerweise genau eine Domäne pro Framework sein. Häufig lassen sich auch domänenübergreifende, immer wiederkehrende Komponenten identifizieren. In der Softwareentwicklung könnten dies z.B. eine Benutzerverwaltung, Mehrsprachenfähigkeit oder einfach eine grafische Oberfläche sein. Diese allgemeinen fachlichen Frameworks werden durch spezialisiertere, von einer Anwendungsdomäne abhängige Frameworks ergänzt. In einem solchen Fall kapselt das Framework Funktionalitäten, die genau für einen Anwendungsbereich passend sind. In der Finanzwirtschaft gibt es z.B. Frameworks die Konten verwalten und Zinsberechnungen durchführen. Ein weiteres Beispiel wäre ein Framework zur Planung von Telekommunikationsnetzwerken für große Telefongesellschaften.

Die charakteristischste Eigenschaft von fachlichen Frameworks ist ihre Fähigkeit, Probleme fachlicher Natur möglichst gut zu lösen. Sie konzentrieren sich dabei nicht auf die technische Umsetzung ihrer Lösungen, sondern verwenden dafür bestenfalls technische Frameworks.

Einen bedeutenden Ansatz zur Entwicklung von fachlichen Unternehmenslösungen wurde in diesem Umfeld 1994 von IBM bzw. diversen Softwareherstellern gemacht. Sie entwickelten zunächst auf Basis von C++ und der CORBA Interface Definition Language (IDL) ein Business Framework zur Erstellung von Geschäftsanwendungen im AS/400 Umfeld. 1996 wurde das sogenannte „San-Francisco-Projekt“ komplett auf Java umgestellt. Die Ziele des Projekts sind die Reduzierung der Komplexität, die Entlastung von technischer Basisarbeit sowie die Definition solider inhaltlicher Objektgrundlagen.

Mittlerweile hat IBM das Projekt treffenderweise in die „San Francisco“ Geschäftsprozeß-Komponenten (engl. Business Objects) umbenannt. Diese Komponenten liefern diverse Algorithmen für immer wiederkehrende Probleme des Unternehmensalltags. Sie bilden die Grundlage für Anwendungen in der Buchhaltung, dem Bestellwesen, dem Vertrieb oder der Warenwirtschaft. Das Framework erlaubt dem Benutzer bzw. dem Anwendungsentwickler auf einfache Art und Weise bestehende Komponenten zu nutzen. Sind die benötigten Business Objects noch nicht vorhanden, können sie über bestimmte Entwurfsmuster problemlos in das Framework integriert werden.

5.5 Business Components for Java – ein technisches Persistenzframework

Die *Business Componentes for Java* (BC4J) der Firma ORACLE sind ein technisches Persistenzframework, daß die Entwicklung von datenbankgetriebenen Java Applikationen auf Basis von Komponenten deutlich vereinfacht. Es beruht auf den Standardtechnologien Java und XML und unterstützt eine flexible Anwendungsverteilung auf mehrschichtige Architekturen. Durch die Oberflächen- und Clientunabhängigkeit können Java Swing-, Servlet- und JavaServer Page Clients leicht angebunden werden. Persistenzframeworks haben die Eigenschaft datenbasierte Modelle in funktionale Modelle zu transferieren. Besonders in der objektorientierten Entwicklung ist es wichtig, ein relationales Datenbankmodell in ein Objektmodell zu überführen, mit dem innerhalb eine Anwendung

gearbeitet werden kann. Der Anwendungsentwickler möchte sich nicht darum kümmern müssen wie Daten zwischen der Anwendung und der Datenbank ausgetauscht werden. Er möchte eine leicht zu benutzende Schnittstelle mit der ein schneller Datenzugriff gegeben ist.

Das Framework bringt bereits eine Vielzahl von immer wieder benötigten Grundfunktionalitäten mit. Datenbanktypische Mechanismen einer Applikation, wie z.B. Connection Pooling, Caching, Locking, Master-Detail-Synchronisation werden dem Anwender abgenommen und laufen unsichtbar vom Anwendungsentwickler ab. Er ist lediglich dafür verantwortlich, gewisse Rahmenbedingungen zu schaffen und die Bausteine zu konfigurieren. Die Komponenten werden in Form einer Java Klassenbibliothek mitgeliefert. Ein großer Vorteil eines technischen Frameworks ist eine entsprechende Toolunterstützung. Sie ist bei BC4J durch eine Integration in den ORACLE JDeveloper gegeben. Zusätzlich wird BC4J auch noch von dem ORACLE 9i Application Server unterstützt.

Eine interessante Gemeinsamkeit findet sich zwischen BC4J und Jacobsons Einteilung von Objekten bei der Analyse eines Anwendungsbereichs. Er gebraucht in [Jacobson92] eine Kategorisierung der Gegenstände in „entity objects“, „interface objects“ und „control objects“. Die „entity objects“ modellieren bei ihm diejenigen Informationen eines Systems deren Lebensdauer hoch ist, die „interface objects“ das Verhalten und die dargestellten Informationen der Benutzungsschnittstelle und die „control objects“ die Funktionalität des Systems, die weder an „entity objects“ noch an „interface objects“ hängt.

BC4J nimmt eine ganz ähnliche Unterteilung vor. Es teilt seine Komponenten auf in:

- *Entity Objects*
- *View Objects*
- *Application Modules*

Der Softwareentwickler kann sich dadurch sowohl eine bildhafte Vorstellung des Softwaresystems machen, als auch das System besser konstruieren und warten. Entity Objects in BC4J entsprechen dem gleichnamigen Konstrukt von Jacobson. Sie sind für die

langlebige Datenhaltung verantwortlich. View Objects stellen die eigentliche Benutzungsschnittstelle des Anwendungsentwicklers dar und sind mit den „interface objects“ zu vergleichen. Das Application Module kapselt, ähnlich den „control objects“ sämtliche Funktionalität, die weder in View noch in Entity Objects vorhanden ist. Diese drei Komponenten sollen nun näher erläutert werden.

5.5.1 Entity Objects

Entity Objects sind in BC4J die Schnittstelle zur Datenbank. Sie entsprechen exakt den Tabellen in der Datenbank. Ein Entity Object steht mit genau einer Tabelle in Beziehung und umgekehrt. Die Spalten der Tabelle werden bei ihrer Erzeugung zu den Attributen der Entity Objects. Das Framework realisiert über die Entity Objects verschiedene datenbanknahe Mechanismen, wie z.B. Persistenz und Caching. Der Anwendungsentwickler kann in Entity Objects auch eigene Anwendungs-, bzw. Geschäftslogik zentral ablegen. Sie wird unabhängig von jeglicher Präsentationsform der Daten durchlaufen. Praktisch besteht das Entity Object aus einer XML-Datei, die den Aufbau, also seine Struktur beschreibt und ggf. einer Java Datei, in der die Implementierung von Geschäftslogik vollzogen werden kann. Der Entwickler hat in seiner Anwendung keinerlei direkte Zugriffsmöglichkeiten auf Entity Objects. Sie sind für ihn nicht transparent und erfüllen nur die ihnen zur Designzeit zugewiesenen Aufgaben. Sie kapseln die Informationen eines Datensatzes. Eine Instanz eines Entity Objects entspricht einem Datensatz aus der Datenbank. Die Spalten füllen dabei die Attribute des Entity Objects.

Entity Objects werden untereinander durch sogenannte Assoziationen (engl. Associations) verknüpft. Sie entsprechen den Fremdschlüsselbeziehungen der in der Datenbank abgelegten Tabellen (engl. Foreign-Key). Somit ist eine Datenmodellierung auf Anwendungsebene mit 1:1, 1:n, n:1 und n:m Verknüpfungen möglich.

Bei der Erzeugung von Entity Objects und Associations gibt es zwei alternative Vorgehensweisen. Im einen Fall legt der Anwendungsentwickler wie gewohnt sein Datenmodell an und erzeugt per SQL die Tabellen und Schlüsselbeziehungen. Anschließend generiert er sich daraus die Entity Objects und Associations. Im anderen Fall

modelliert der Entwickler die Daten auf BC4J-Ebene. Er legt werkzeugunterstützt nach einem bestimmten Vorgehensmodell Entity Objects und Associations an und generiert aus diesen Informationen ein zugehöriges Datenbankschema. In beiden Fällen steht nach Abschluß dieser Arbeit ein Datenmodell im Anwendungskontext zur Verfügung.

5.5.2 View Objects

Nun geht es noch darum anwendungsabhängige Zugriffsschnittstellen, die View Objects, anzulegen. View Objects sind in etwa vergleichbar mit Views innerhalb einer Datenbank. Sie repräsentieren lediglich ein beim Entwurf festgelegtes SQL-Statement, mit dem vordefinierte Abfragen an die darunterliegende Entity Schicht abgesetzt werden können. Für den Benutzer hat es den Anschein, er arbeite direkt auf der Datenbank. In der Realität delegieren View Objects jedoch nur an die Entity Schicht, welche schließlich die Kommunikation mit der Datenbank übernimmt. Um eine hohe Performance des Systems erreichen zu können, werden alle durch eine View Object Anfrage erstellten Entity Objects in einem Cache-Speicher auf der logischen Mittelschicht gehalten. Ein erneuter Abruf der gleichen Informationen kann somit wesentlich schneller erfolgen.

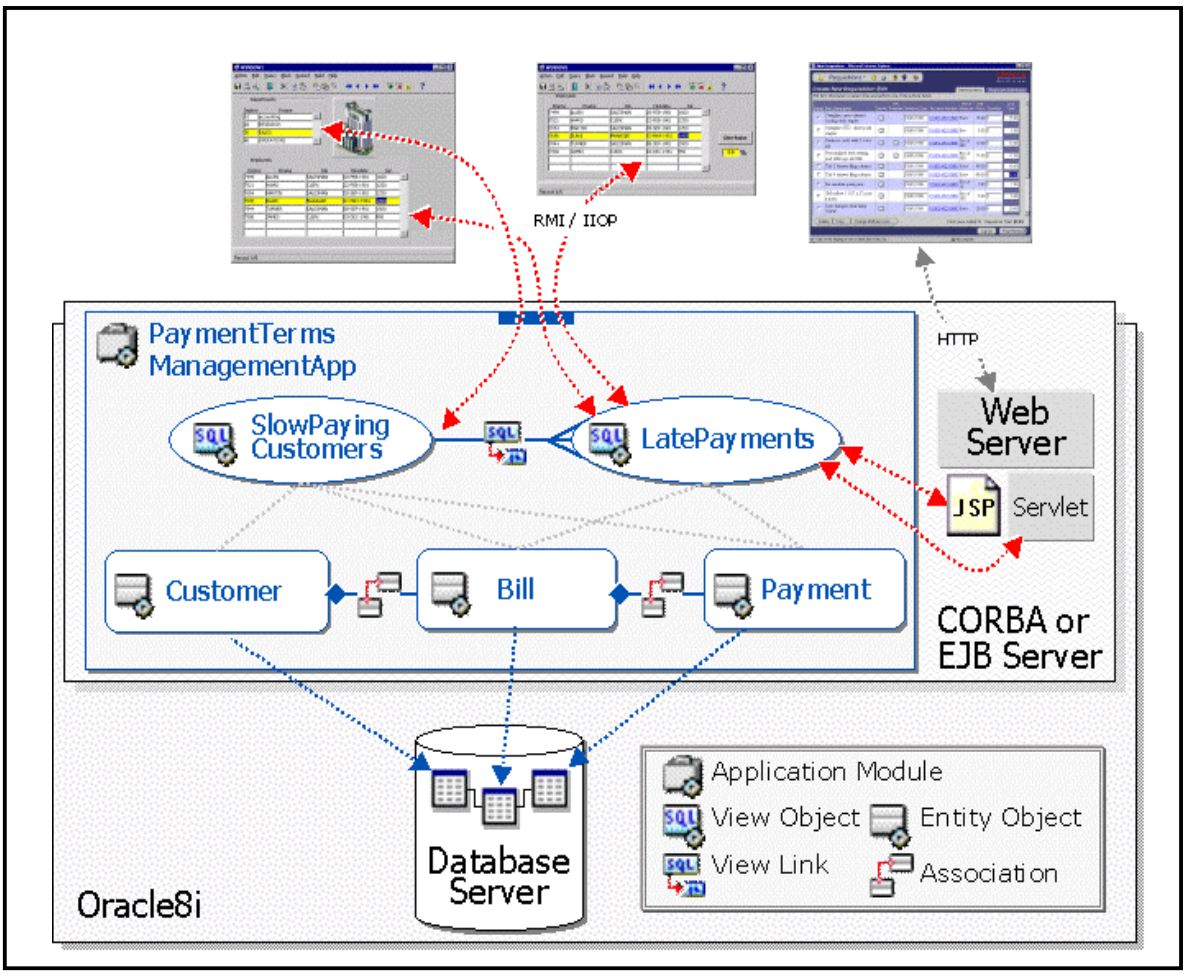
View Objects bieten dem Anwendungsentwickler weitreichende Zugriffsmöglichkeiten auf die Entity Daten an. Datensätze können gelesen, manipuliert und wieder zurück in die Datenbank geschrieben werden. Sie können auch zusätzliche Geschäftslogik aufnehmen, die nur ausgeführt wird, wenn der Datenzugriff über dieses eine View Object erfolgt. View Objects basieren nicht zwangsläufig auf nur einem Entity Object. Es ist auch eine Zuweisung von mehreren oder gar keinem Entity Object möglich. Dem Entwickler sind beim Entwurf viele Erweiterungs- und Anpassungsmöglichkeiten gegeben. View Objects werden untereinander ähnlich den Associations über die View Links verknküpft.

View Links müssen nicht unbedingt den Associations entsprechen. Sie sind frei definierbar und der Entwickler kann mit ihnen logische Verknüpfungen modellieren. Dazu zählen z.B. Master-Detail-Beziehungen zwischen den Datensätzen. Der Benutzer muß sich in der Applikation dann nicht mehr damit beschäftigen zu einem Master passende Details zu selektieren. Wählt der Benutzer auf der Oberfläche einen Masterdatensatz aus, paßt das Detail View Object die Menge der selektierbaren Datensätze entsprechend an.

5.5.3 Application Modules

Analog zu Jacobsons „control objects" gibt es bei BC4J ein Konstrukt, welches eine zentrale Zugriffsschnittstelle auf alle Komponenten des BC4J Systems anbietet. Das sogenannte Application Module kann neben der Verwaltung der View Objects auch sehr viel weitere Funktionalität implementieren. Sämtliche Funktionen, die in der Mittelschicht ausgeführt werden sollen, kann das Application Module kapseln. Der Entwickler kann beim Entwurf eines Application Modules angeben, welche View Objects für eine Applikation bereitgestellt werden sollen. Nur die in einem Application Module enthaltenen View Objects sind von der Applikation nutzbar. Es ist also eine Art Container für die Mittelschichtkomponenten. Application Modules sind außerdem die einzigen Objekte die bei einem verteilten Softwaresystem als entfernte Objekte, z.B. über CORBA angesprochen werden können. Sie sind hierarchisch strukturierbar. Sogenannte „Nested" Application Modules sind Kinder eines „Root" Application Modules. Die Schachtelung kann beliebig tief fortgeführt werden. An dem, in der Hierarchie an oberster Stelle stehenden „Root" Application Module, hängt die eigentliche Datenbanktransaktion in Form eines Transaktionsobjekts, welches Aktionen auf der physikalischen Datenbankverbindung durchführen kann. Dazu zählen unter anderem auch die „Commit" und „Rollback" Aktionen.

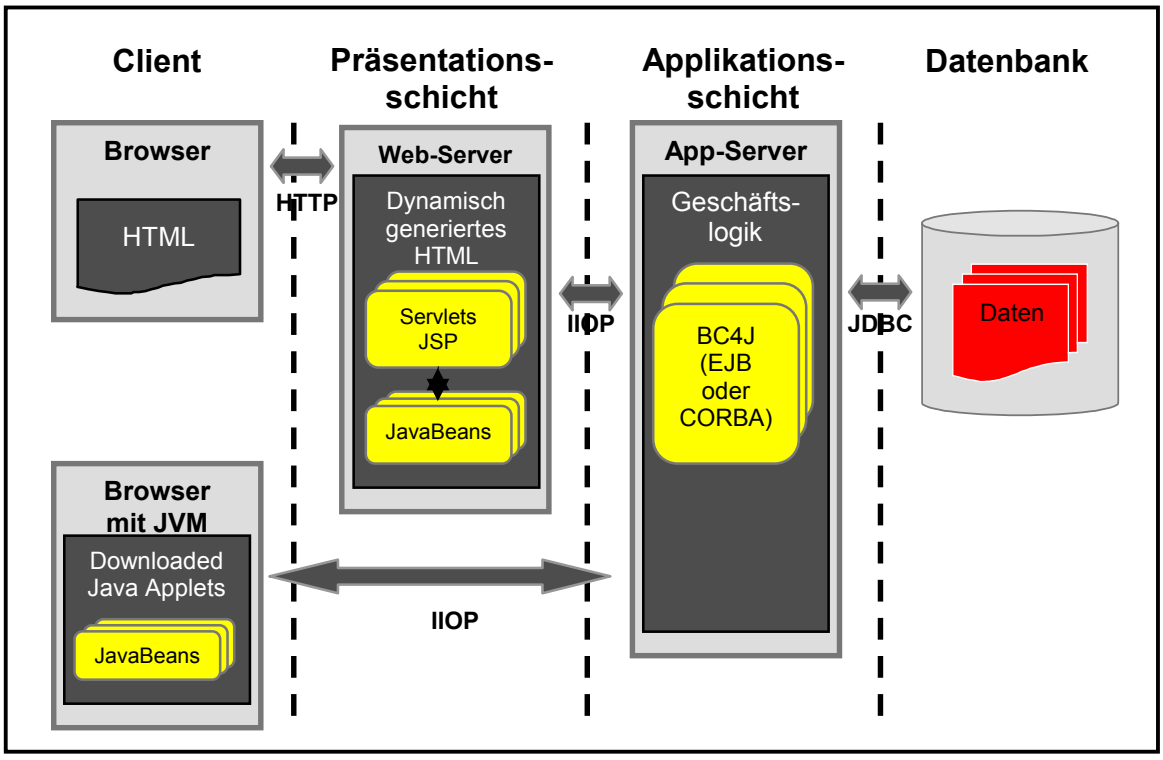
Abbildung 5.2: Zusammenspiel von BC4J Komponenten innerhalb einer Anwendung



Quelle: Powerpointpräsentation Oracle Business Components for Java, Opitz Consulting, 2000

Aus Entwicklersicht stellen die Business Components for Java eine enorme Arbeitserleichterung dar. Er kann sich auf die Implementierung von Logik und Präsentation der Daten konzentrieren. Der aufwendige Datenzugriffsmechanismus wird mit seinen komplexen Funktionalitäten komplett von BC4J übernommen. Des weiteren bringt BC4J durch seinen Cachingmechanismus Performancevorteile bei der intensiven Nutzung von Daten aus einer Datenbank. BC4J ist im Hinblick auf verteilte System sehr flexibel, da es auf Standards, wie den Enterprise JavaBeans oder CORBA, zur logischen und physikalischen Schichtentrennung beruht.

Abbildung 5.3: Beispiele von Architekturentwürfen mit BC4J



Quelle: Powerpointpräsentation Oracle Business Components for Java, Opitz Consulting, 2000

5.6 Vorgehensmodell bei der Anwendung von Frameworks

Wie bei vielen anderen Sachverhalten im Frameworkumfeld gibt es auch bei der Entwicklung von Applikationen mit Frameworks keine wohldefinierten Methoden oder Vorgehensmodelle. Erschwert wird ein solcher methodischer Ansatz auch noch durch eine Vielzahl von individuellen Frameworks, die alle ein unterschiedliches Vorgehen erfordern. Taligent beschreibt in [Taligent95] drei unterschiedliche Arten ein Framework zu benutzen:

- Es werden lediglich Funktionalitäten genutzt, die das Framework von sich aus zur Verfügung stellt. Im allgemeinen kommen hierfür nur Black-Box Frameworks in Frage, bei denen Komponenten mittels Kompositionsmechanismen zu einer Applikation zusammengefügt werden.

- Dem Framework werden individuelle Funktionalitäten hinzugefügt, indem flexible Elemente des Rahmens den Bedürfnissen angepaßt werden. Diese Art der Benutzung bezieht sich auf die White-Box Frameworks, welche dem Benutzer viele Freiheiten bei der Anpassung lassen. Die Ergänzung des Frameworks wird notwendig, sobald die vorhandenen Komponenten nicht mehr sämtliche Funktionalitäten abdecken.
- Das Framework wird den eigenen Bedürfnissen angepaßt und mit individuellem Quellcode modifiziert. Ein solch tiefer Eingriff in das Framework selber erfordert intimste Kenntnisse über die Funktionalität und den Kontrollfluß innerhalb der Komponenten.

Meistens wird es wahrscheinlich dazu kommen, das Framework mit Hilfe der dynamischen Komponenten des White-Box Ansatzes zu erweitern und an individuelle Bedürfnisse anzupassen. Die reine Black-Box Benutzung hingegen ist oft nicht ausreichend für Anforderungen komplexer Applikationen, in denen viele verschiedene Komponenten gebraucht werden. Auch der Eingriff in das Framework selber wird außer bei unternehmensintern entwickelten Frameworks eher selten vorkommen. Ein solch detailliertes Wissen über das Framework ist bei umfangreichen Frameworks fast unmöglich zu erlangen.

5.6.1 Analyse

Wie auch in der traditionellen Softwareentwicklung wird zunächst eine Analysephase durchlaufen. Hauptsächlich wird dabei eine Anforderungsanalyse durchgeführt, welche die Ansprüche und Bedürfnisse der Anwendung sammelt und analysiert.

Die Ergebnisse der Analyse werden verwandt, um ein Konzept der Anwendungsarchitektur zu entwerfen. Die Funktionalitäten der Anwendung werden somit bestimmten Komponenten zugewiesen. Es erfolgt also eine Klärung der Verantwortlichkeiten und der Beziehung der Komponenten untereinander. Sobald diese Phase abgeschlossen ist, muß geprüft werden, welche Frameworks für diese Anwendung am geeignetsten sind.

5.6.2 Evaluierungsprozeß

Ein besonders wichtiger Prozeß bei der frameworkbasierten Entwicklung ist die Auswahl eines passenden Frameworks. Zunächst sollte sich ein Unternehmen fragen, ob es überhaupt eine Auswahl treffen muß. Tatsache ist, daß es zur Zeit nicht für alle Domänen oder Verwendungszwecke abgestimmte Frameworks gibt. Hinzu kommen auch die Anforderungen an die zu entwickelnde Applikation. Wie stark schränken diese die Auswahl ein? Eine Applikation in einer bestimmten Anwendungsdomäne, die zwingend auf C++ und CORBA auf verteilten Plattformen zum Einsatz kommen soll, kann die Auswahl an passenden Frameworks schon auf ein Minimum zusammenschrumpfen lassen.

Welche Kriterien sind nun aber für die richtige Frameworkwahl entscheidend? Fayad nennt einige einfach zu evaluierende Eigenschaften die den Auswahlprozeß beeinflussen können:

- Auf welchen Plattformen kann das Framework eingesetzt werden?
- Mit welchen Programmiersprachen kann die Anwendung entwickelt werden?
- Welche sonstigen Standards müssen erfüllt werden?
- Gibt es eine ausreichende Werkzeugunterstützung bei der Anwendungsentwicklung?
- Ist die Dokumentation ausreichend und qualitativ hochwertig?
- Gibt es eine „Community“, in der sich Entwickler zum Erfahrungsaustausch zusammengeschlossen haben?

Diese meist technischen Fragen sind relativ leicht in Erfahrung zu bringen. Schwieriger wird es allerdings herauszufinden, ob das Framework problemadäquat ist. Es soll einfach zu benutzen, aber dennoch leistungsfähig genug sein, um die Anwendungsprobleme zu lösen. Dazu sollte zunächst eine Liste mit funktionalen Anforderungen der Applikation erstellt werden. Es ist nicht notwendig, daß das Framework diese alle erfüllt. Die Hauptsache ist, es ist erweiterbar genug, um entsprechende Funktionalitäten einzubauen.

> „*No framework is good for everything, and it can be hard to tell whether a particular framework is well suited for a particular problem.*“
> ([Fayad99a], S.20)

Leider ist die Erweiterbarkeit eines Frameworks schwer zu evaluieren. Deshalb bietet es sich an eine Anwendungsentwicklung testweise durchzuführen, bei der spezielle Funktionalitäten erweitert werden müssen. Durch die dabei entstehenden Kosten ist dies jedoch nur sinnvoll, wenn das Framework auch entsprechend häufig eingesetzt werden soll. Generell läßt sich sagen, daß eine häufige Anwendung des Frameworks viel Flexibilität und Leistungsfähigkeit voraussetzt. Bei einmaligem oder seltenem Einsatz muß das Framework hingegen einfach zu erlernen und anzuwenden sein. Wichtigstes Entscheidungskriterium bleibt letztendlich, ob sowohl der Entwicklungsprozeß als auch die Anwendung signifikant verbessert werden.

5.6.3 Erlernen von Frameworks

Der Lernprozeß bei der Benutzung eines Frameworks ist wesentlich aufwendiger als bei herkömmlichen Wiederverwendbarkeitsansätzen. Im Gegensatz zu Klassenbibliotheken oder Einzelkomponenten, bei denen nur die Funktionalität einzelner Klassen oder Komponenten erlernt werden muß, kommt es bei Frameworks auf die Zusammenhänge an. Es müssen also gesamte Klassenkonstrukte einer näheren Untersuchung unterzogen werden. Anders als bei konkreten Klassenbibliotheken enthalten Frameworks zusätzlich eine Vielzahl abstrakter Klassen und Interfaces. Die Mischung aus konkreten, statischen und dynamischen, flexiblen Teilen des Frameworks erschweren deutlich ein Erlernen der Frameworkstruktur [Fayad99a]. Der Anwender braucht also ein „big picture“, den Überblick über das Gesamtsystem, der Funktion, Interaktion und den Verantwortlichkeiten der Elemente.

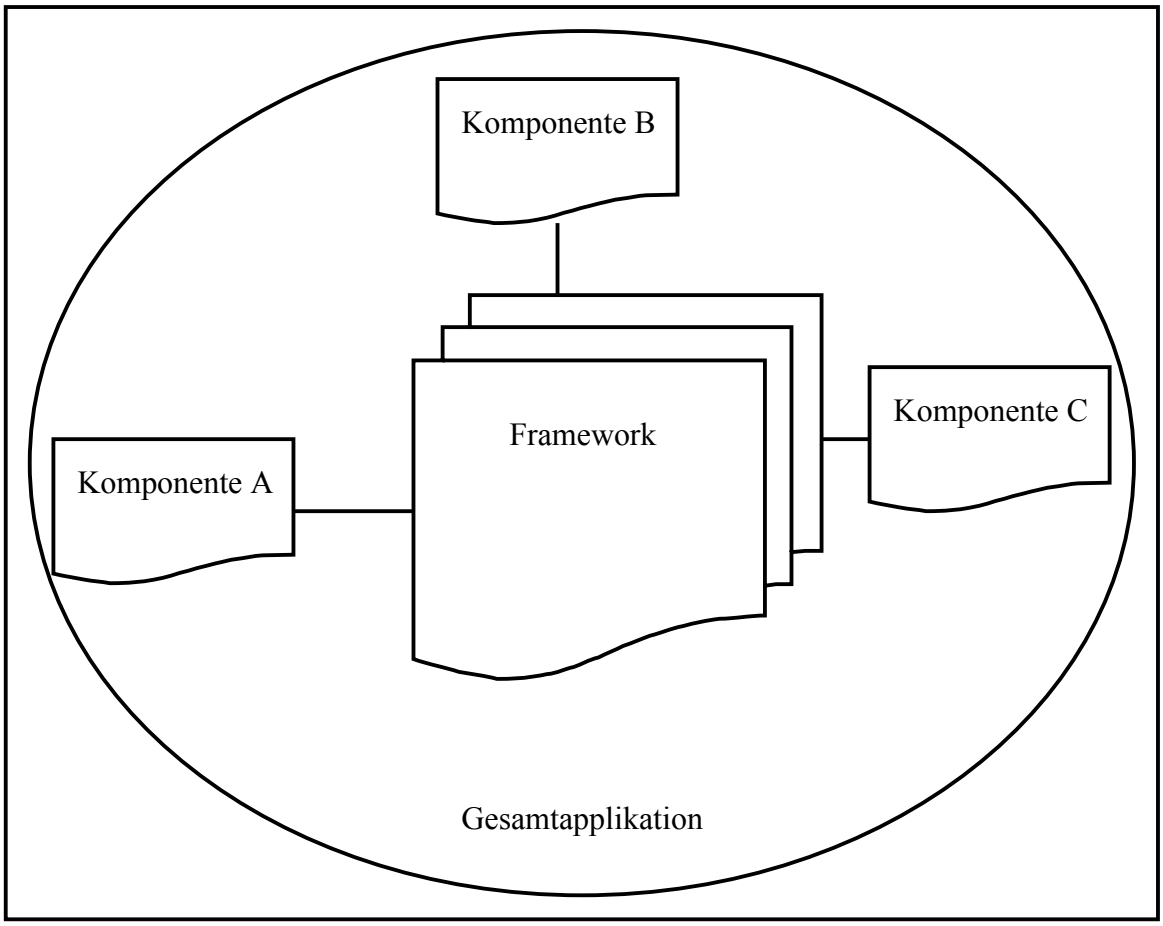
Eine gute und ausgefeilte Dokumentation ist also unabdingbar. Sie sollte sowohl den Zweck, also das Anwendungsgebiet des Frameworks, als auch dessen Anwendung und Arbeitsweise dokumentieren. Eine wesentliche Erleichterung stellt der Einsatz von Beispielen dar. Das „learn by example“ kann viele Zusammenhänge besser verdeutlichen als das Lernen anhand einer Dokumentation. Ohne Beispiele und Fallstudien wären sehr komplexe Frameworks unbrauchbar. Zudem machen sie die Einsatzgebiete des Frameworks deutlich und zeigen Grenzen auf, an denen das Framework für eine Applikation nicht mehr geeignet ist. Sie geben häufig erst die entscheidenden Einblicke in den Kontrollfluß innerhalb des Frameworks. Es kommt dabei besonders auf das Objektverhalten und die Objektinteraktion an [Fayad99a].

Teilweise bringen Frameworks auch sogenannte „Kochbücher“ mit, in denen die Anwendung von Frameworks Schritt für Schritt erklärt wird. Fayad rät dazu, zunächst eine kleine auf das Framework zugeschnittene Anwendung beispielhaft zu implementieren und diese dann mit Hilfe eines Kochbuchs an speziellere Bedürfnisse anzupassen. Diese erste Benutzung eines Frameworks ist der schwierigste Teil im Lernprozeß. Er kann aber idealerweise durch einen Mentor in Form eines Frameworkexperten unterstützt werden. Es ist nicht zwingend erforderlich, wenn nicht sogar unmöglich, alle Details des Frameworks zu verstehen. Gute Frameworks können viele Funktionalitäten in Komponenten kapseln, ohne daß ein Anwender intime Kenntnisse über deren Funktionsweise erlangen muß.

5.6.4 Design und Implementierung

Anhand des Designs der Applikation läßt sich nun sagen, an welchen Stellen Komponenten des Frameworks nicht genug Funktionalität bieten und erweitert werden müssen. Ansatzpunkte könnten z.B. der Zukauf oder die Entwicklung entsprechender Komponenten sein, um den Funktionsumfang des Framework zu erweitern.

Abbildung 5.4: Komponentenkomposition in der Anwendungsentwicklung



Weitere Anpassungen sind möglich, sobald das Framework dynamische Elemente enthält, mit deren Hilfe Erweiterungen seiner Komponenten durchgeführt werden können. Fayad beschreibt diese Erweiterungen als *application-specific increments* (ASIs). ASIs durchlaufen die aus der konventionellen Softwareentwicklung bekannten Strukturen, Design und Implementierung. Während dieser Prozesse bleibt der enge Kontakt zur Frameworkdokumentation und ihren Entwurfsregeln unerläßlich [Fayad99a].

An dieser Stelle des Entwicklungsprozesses wird auch der enorme Vorteil der frameworkbasierten Entwicklung deutlich. Der Anwendungsentwickler hat nur noch wenig Quellcode zu implementieren, da große Teile schon durch Funktionen des Frameworks abgedeckt sind. Auch die Entwurfsentscheidungen bleiben dem Entwickler größtenteils erspart, da sie bereits bei der Frameworkentwicklung getroffen wurden.

5.6.5 Qualitätssicherung und Test

Das Testverfahren ist auch bei frameworkbasierter Software ein wichtiger Bestandteil der Anwendungsentwicklung. Leider sind diese Tests nicht gerade einfacher im Gegensatz zu herkömmlicher Software. So stammen Fehler nicht zwangsläufig aus den eigenen Quellcodeteilen, sondern können auch innerhalb des Frameworks auftreten, weil durchgeführte Modifikationen sich nicht an dokumentierte Regeln zur Frameworkerweiterung gehalten haben. Eine Entkopplung von Eigenentwicklung und Framework kann bei der Fehlersuche sehr nützlich sein. Fehler können beim *Debugging* und der Fehleranalyse dann eindeutig bestimmten Bereichen zugeordnet werden.

Die Fehlersuche im eigentlichen Framework gestaltet sich dabei oft am schwierigsten. Wie auch bei Klassenbibliotheken ist ein Debugging des Frameworks mit sehr viel Kenntnissen und Aufwand verbunden. Ein großer Vorteil ist es, wenn das Framework schon passende Testverfahren liefert. Tritt ein Fehler innerhalb des Frameworks auf, hat der Benutzer drei Möglichkeiten diesen zu behandeln: er behebt ihn, wenn der Quellcode verfügbar ist, er umgeht in, indem er einen *Workaround* entwickelt, oder er verwirft das Framework und wählt ein anderes.

Schließlich werden dann alle Elemente der Anwendung zusammengefügt und durchlaufen die Testphase als Ganzes.

5.7 Der Frameworkentwicklungsprozeß

In Kapitel 5.6 wurde die Anwendungsentwicklung mit Hilfe von Frameworks untersucht. Dabei ergaben sich bzgl. der Effizienz der Anwendungsentwicklung viele Vorteile. Doch sind diese Vorteile noch so markant, wenn man die Frameworkentwicklung selber mit in Betracht ziehen muß? Leider sind nicht für alle Anwendungsfälle und –domänen passende

Frameworks zu kaufen. Dem Entwickler bleiben schließlich, wenn überhaupt, nur noch zwei Wahlmöglichkeiten. Zum einen kann er die Applikation ohne jegliche Hilfsmittel der Wiederverwendung entwickeln oder er begibt sich daran, ein eigenes, sehr gut auf die Anwendung abgestimmtes Framework, zu entwickeln. Der größte Vorteil bei einer Eigenentwicklung eines Frameworks ist die Tatsache, daß das Framework nahezu 100% passend für die zu entwickelnde Applikation ist. Der Frameworkentwickler kann dieses genau auf seine Anforderungen abstellen und kennt sich zudem extrem gut mit Aufbau und Kontrollfluß des Frameworks aus. Ein späterer Eingriff und die Frameworkmodifikation fallen somit deutlich leichter. Außerdem kann die Kostenintensive Lern-, Evaluierungs- und Einarbeitungsphase entfallen. Leider hat eine Eigenentwicklung nicht nur Vorteile, sie birgt auch extreme Nachteile. Dazu zählen größtenteils die lange Entwicklungsdauer und die aufwendige Konzeptionsphase.

Um eine Entscheidung zu fällen, kommt es für den Entwickler maßgeblich darauf an, ob er das zu entwickelnde Framework wiederverwenden kann oder nicht. Er benötigt dabei eine rein wirtschaftliche Betrachtungsweise, bei der der Aufwand und die anfallenden Kosten dem Nutzen bzw. Gewinn gegenübergestellt werden müssen. Eine aufwendige Eigenentwicklung ist also nur ratsam, wenn entsprechend viele Applikationen mit ihr entwickelt werden können.

Hat der Entwickler sich für die Neuentwicklung eines Frameworks entschieden, stellt sich die Frage wie diese anzugehen ist. Im folgenden wird dieser Frameworkentwicklungsprozeß beschrieben und Unterschiede zur herkömmlichen objektorientierten Entwicklung herausgearbeitet.

5.7.1 Abstraktion anwendungsspezifischer Modelle

Der wahrscheinlich wichtigste Grundgedanke bei der Frameworkentwicklung ist die Abstraktion. Da Frameworks zu Beginn ihrer Entwicklung nur sehr unspezifizierte Anforderungen an die spätere Funktionalität bieten, ist der einzige Weg zu einem wiederverwendbaren Framework die Abstraktion von exemplarischen Anwendungsfunktionalitäten. Anhand von Beispielen müssen Anforderungen an das Framework dokumentiert werden. Die Anzahl der untersuchten Beispielanwendungen

sollte möglichst hoch sein, um eine breite Vielfalt von Anwendungsfällen abdecken zu können.

Allerdings darf nicht aus den Augen verloren werden, auf welche Art von Anwendungen sich das Framework spezialisieren soll. Das ist notwendig, damit das Framework immer noch seinem Zweck, der Wiederverwendbarkeit dient, aber eine Benutzung nicht zu komplex und aufwendig wird. Frameworkdesign ist also immer ein Balanceakt zwischen Wiederverwendbarkeit und Nützlichkeit.

Die Anwendungsanforderungen an das Framework bilden die Grundlage für den Abstraktionsprozeß. Es muß versucht werden, Gemeinsamkeiten der Beispielanwendungen zu finden oder, falls diese nicht vorhanden sind, soweit zu abstrahieren, bis entsprechende Gemeinsamkeiten gefunden werden.

Über Abstraktionen lassen sich auch schon Schlüsse über einen möglichen Entwurf des Frameworks ziehen. Abstraktionen die starke Ähnlichkeiten, also ein geringes Abstraktionsniveau innerhalb ihrer Anwendungsdomäne aufweisen, sind Kandidaten für einen Black-Box Entwurf eines Frameworks. Sie bilden darin fertige Komponenten, die nur geringfügig verändert werden können, aber dennoch ausreichende Funktionen anbieten. Müssen allerdings stärkere Abstraktionen vollzogen werden, weil Prozesse verschiedener Anwendungen zwar den gleichen Zweck erfüllen, dies jedoch auf unterschiedliche Art und Weise implementieren, ist das ein Kennzeichen für einen White-Box Entwurf. Es müssen also Schnittstellen für die unterschiedlichen Implementierungen geschaffen werden.

Ein Beispiel für eine solche Abstraktion wären zwei Anwendungen, die beide einen Mechanismus zum Anmelden von Benutzern an das System benötigen. Bei beiden Applikationen ist eine Anmeldung über einen Benutzernamen und ein Paßwort notwendig. Leider besitzen beide Anwendungen unterschiedliche Anforderungen an den eigentlichen Anmeldevorgang. In einem Fall soll eine Authentifikation des Benutzers an einem Applikationsserver zentral erfolgen und im anderen Fall muß der Anwender seine Identität gegenüber einer Datenbank kenntlich machen. Der Frameworkentwickler hat nun die

Aufgabe diesen Prozeß auf einen gemeinsamen Nenner zu bringen, indem er die Einzelanforderungen so verallgemeinert, daß sie als gemeinsame Anforderung an ein Framework definiert werden können.

Mit Hilfe der Abstraktion lassen sich also Anforderungen an ein Framework definieren. Die Abstraktionsprozesse finden in der Regel während der Analysephase der Frameworkentwicklung statt. Umgesetzt werden sie im anschließenden Design und der Implementierung.

5.7.2 Analyse der Anwendungsdomäne

Zunächst läßt sich sagen, daß die Frameworkentwicklung sich nicht grundsätzlich von traditionellen Entwicklungstechniken unterscheidet. Am einfachsten läßt sie sich mit der Entwicklung von herkömmlicher wiederverwendbarer Software vergleichen. Es dürfen also keine anwendungsspezifischen Details in das Framework einfließen. Vielmehr geht es darum, die Grundlage für den späteren Anwendungsentwickler zu schaffen, solche Details möglichst einfach implementieren zu können. Dabei darf das Framework jedoch nicht zu sehr in Richtung eines generell gültigen Softwarekomplexes gestaltet werden. Die Übersichtlichkeit würde dabei verloren gehen und die Anwendungsentwicklung unnötig erschwert.

Da das Framework später Bestandteil einer Applikation sein wird, orientiert es sich in seiner Funktionsweise sehr stark an den Anwendungen, die mit seiner Hilfe entwickelt werden sollen. Dies ist aber nur sehr begrenzt möglich, weil viele dieser Applikationen noch gar nicht definiert sind. Es muß also analysiert werden, welche Probleme in einem Anwendungsumfeld auftreten könnten und typisch für dieses sind. Für diese Probleme könnte es also Anforderungen an eine softwaretechnische Lösung geben. Das Framework muß also auch eine Grundlage für diese Lösung liefern können. Folglich wird durch die Analyse des Anwendungsbereichs eine einigermaßen exakte Beschreibung der Anforderungen an das Framework stark vereinfacht. Immer wiederkehrende Probleme von vielen Anwendungen sind z.B. eine Benutzeroberfläche, eine Benutzerverwaltung, deren Pflege und ein Mechanismus der den Benutzer bei der Behandlung unterschiedlichster Aufgaben unterstützt.

Am leichtesten ist die Analyse zu vollziehen, wenn schon diverse Anwendungen als Beispiele für domänentypische Applikationen vorliegen. Deren charakteristische Gemeinsamkeiten können dann leicht als Anforderungen für das Framework definiert werden.

Schwierigkeiten bereiten hingegen Anwendungsumfelder, von denen der Frameworkentwickler wenig Informationen besitzt. Er muß zuerst die Probleme erkennen und anschließend potentielle Lösungskonzepte durchdenken, bevor er eine Anforderung spezifizieren kann.

Glücklicherweise sind heutzutage viele der ganz allgemeinen Anforderungen an eine Software bekannt und auch schon als Lösungen in unterschiedlichster Weise implementiert. Deshalb wird die Analyse auch als *Solution Domain Analysis* bezeichnet. Sie bereitet somit, wie auch in der konventionellen, objektorientierten Softwareentwicklung, die Entwurfsphase vor, indem sie Anforderungen an die Software, also das Framework, festlegt.

5.7.3 Entwurfsansätze und deren Implementierung

Die Aufgabe des Frameworkentwicklers ist es, das Framework so zu entwerfen, daß es den Anforderungen des Anwendungsentwicklers gerecht wird. Dieser möchte das Framework leicht benutzen können und es muß ihm soviel Implementierungs- und Entwurfsaufwand wie möglich abnehmen. Eine weitere Anforderung ist die einfache und doch genügend umfangreiche Erweiterungsmöglichkeit des Frameworks. Es müssen also flexible Schnittstellen geschaffen werden, um den Funktionsumfang des Frameworks ausbauen zu können.

Die komplexen Anforderungen machen es trotz dieser aufwendigen Vorüberlegungen unmöglich ein Framework beim ersten Design perfekt zu entwerfen und zu implementieren. Eine iterative Vorgehensweise ist also unumgänglich [Johnson88]. Folgende Gründe für eine iterative Entwicklung führt Fayad in [Fayad99a] an:

- Analyse der Anwendungsdomäne (engl. *Domain Analysis*)
- Entwurf der flexiblen Komponenten
- Abstraktion

Da die Domain Analysis keinen kompletten Überblick über die gesamte Anwendungsdomäne schaffen, kommt es während und auch nach Entwurf des Frameworks zu neuen Anforderungen des Anwendungskontextes. Um möglichst viele dieser Anforderungen schon im Framework und nicht erst in der Anwendung berücksichtigen zu können ist eine Überarbeitung des Entwurfs unabdingbar.

Es liegt in der Natur des Frameworks, sich mit den ständig wechselnden Komponenten der verschiedene Anwendungen zu befassen. Um eine Integration der sich wahrscheinlich ändernden Komponenten zuzulassen, müssen flexible Schnittstellen geschaffen werden. Gerade deren Entwicklung ist aber der aufwendige und unübersichtliche Teil des Frameworks. Eine Anpassung der Interfaces ist deswegen auch häufig im nachhinein noch notwendig.

Der letzte Punkt, dem Fayad Bedeutung zumißt, ist der Abstraktionscharakter des Frameworks. Der Frameworkentwickler versucht anhand von Beispielen der Analyse eine abstrakte Implementierung des Frameworks zu erreichen. Diese Abstraktion geht oft aber nicht weit genug. Werden neue Beispiele bzw. Anwendungsfälle in Betracht gezogen, müssen zur Anpassung höhere Abstraktionsebenen entworfen werden. Sie machen das Framework genereller und wiederverwendbarer [Fayad99a].

Frameworks befinden sich also in einem Fluß ständiger Iteration und Anpassung. Sie werden für den Anwendungsentwickler immer wertvoller, weil immer mehr komplexe Funktionalitäten durch das Framework abgedeckt und einfach wiederverwendet werden können.

All diese Überlegungen haben dazu geführt, daß sich unterschiedliche Entwurfsansätze entwickelt haben. Wie schon bei der Beschreibung von White- und Black-Box Ansätzen erläutert, beginnen die meisten Frameworkentwürfe mit einem White-Box Entwurf und versuchen während des Iterationsprozesses immer mehr fertige Komponenten zu erstellen, welche dann nur noch zusammengefügt werden müssen. Der White-Box Ansatz erfordert eine intensive Auseinandersetzung mit den Schnittstellen, an denen das Framework erweitert und angepaßt werden kann. Zumeist geschieht das durch die Erweiterung abstrakter Basisklassen, welche eine Grundfunktionalität abstrahieren, aber nicht implementieren.

Da der Entwurf passender Schnittstellen schwierigste Entwurfsentscheidungen des Entwicklers fordert, beschäftigen sich viele Entwürfe mit Methoden zur Schnittstellenerstellung und ihrer Beschreibung.

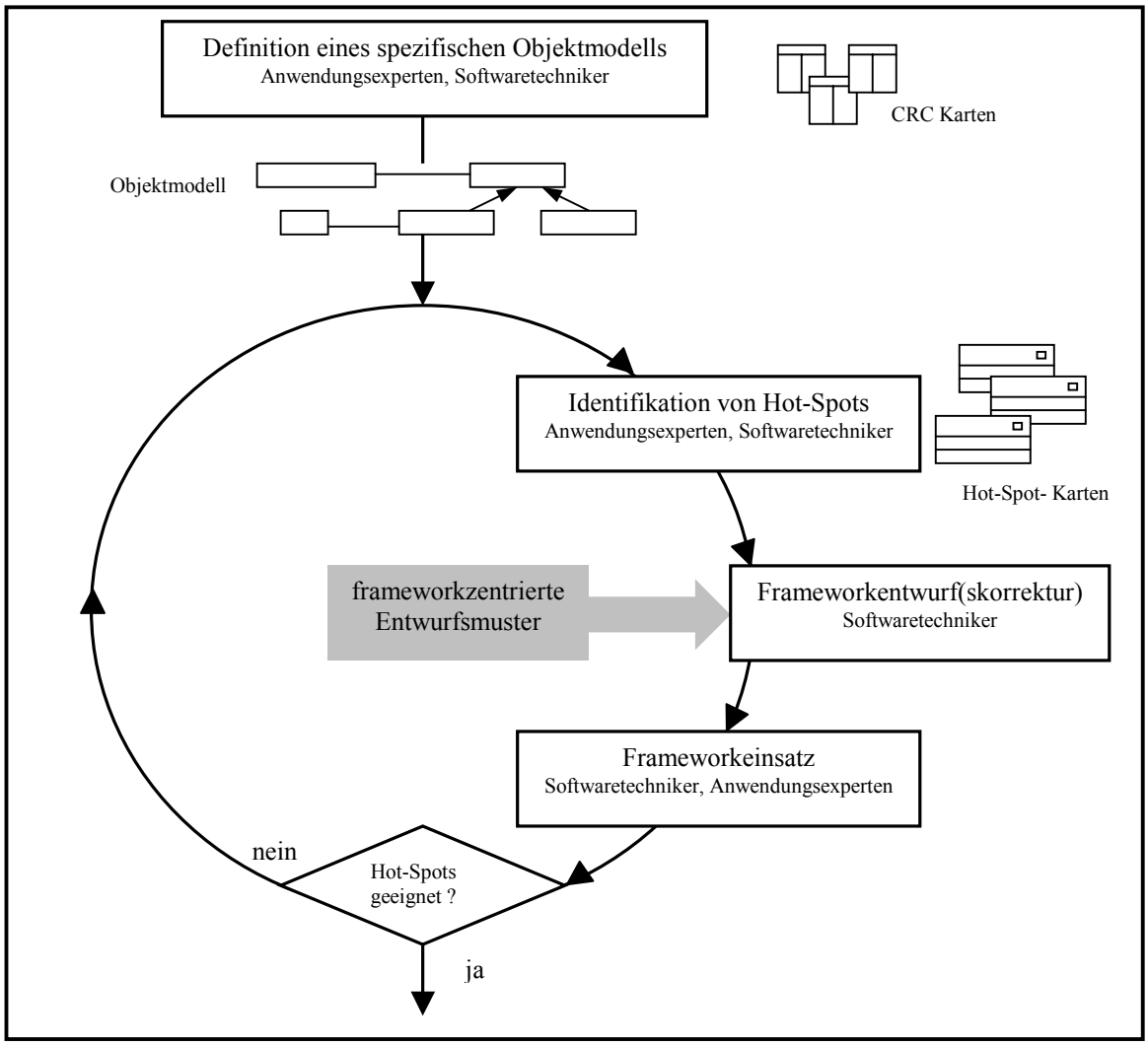
Pree entwickelt in [Pree97] einen Ansatz, der durch die Verwendung von sogenannten Hot-Spots eine Beschreibung von Einschubmethoden vereinfacht und für den Anwendungsentwickler transparenter macht.

Pree prägt in [Pree97] den Begriff der *Hot-Spots*, um die flexiblen erweiterbaren Stellen des Frameworks zu beschreiben. Er verfolgt mit ihnen einen Ansatz, um die Verwendung von sogenannten *Einschubmethoden* zu vereinfachen und für den Anwendungsentwickler transparenter zu machen. Eine Art und Weise diese Hot-Spots zu beschreiben und zu dokumentieren wird in Kapitel 5.7.5 erläutert.

Bevor die Hot-Spots selber entworfen werden können, müssen sie zunächst innerhalb des Frameworks identifiziert werden. Unter Zuhilfenahme von exemplarischen Anwendungsfällen sind die Stellen des Frameworks zu finden, an denen eine flexible Handhabung der Funktionalitäten nötig ist. Die unüberschaubare Anzahl von potentiellen Anwendungsfällen führt zu einer iterativen Entwicklung der Hot-Spots. Teilweise werden sie beim ersten Entwurf als solche erkannt und können implementiert werden. Im Zuge der Anwendungsentwicklung tauchen immer wieder neue Hot-Spots auf oder bestehende

müssen überarbeitet werden. Abbildung 5.5 verdeutlicht diese iterative Vorgehensweise im Frameworkentwicklungsprozeß.

Abbildung 5.5: Entwicklungsprozeß mit Hot-Spots

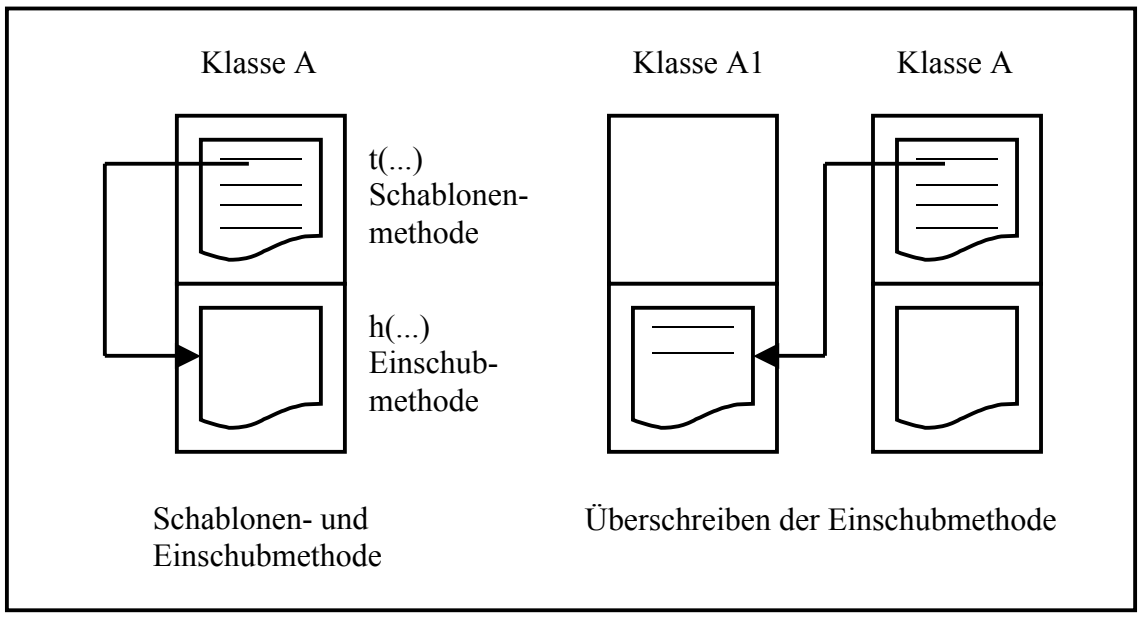


Quelle: In Anlehnung an [Pree97], S.66

Hot-Spots sind durch den Gebrauch von Einschubmethoden gekennzeichnet. Einschubmethoden bilden genau die Stellen ab, an denen eine Spezialisierung stattfinden soll und kann. Pree benutzt in diesem Zusammenhang auch den Begriff der *Schablonenmethoden*. Sie zeichnen sich dadurch aus, daß sie als zum Framework gehörende Methoden den Aufruf von Einschubmethoden einleiten. Im Umfeld einer objektorientierten Entwicklungsumgebung findet eine Realisierung, also Implementierung,

der Einschubmethoden als abstrakte Methoden statt. Eine abstrakte Methode ist eine Methode, welche lediglich den Methodenrumpf darstellt, diesen aber nicht implementiert. Sie ist vergleichbar mit einem Platzhalter für eine konkrete, anwendungsspezifische Funktionalität. Ihre Implementierung bleibt dem Frameworkbenutzer vorbehalten. Sie erfolgt durch ein bestimmtes Entwurfsprinzip, welches auch in [Weinand89] als *Prinzip der Vererbung enger Schnittstellen* (engl. principle of narrow inheritance interface) beschreiben wird. Dieser Ansatz fordert vom Frameworkentwickler den Entwurf von Interfaces, welche durch eine bestimmte Vorlage oder Schablone (engl. *template method*) implementiert werden.

Abbildung 5.6: Zusammenspiel von Schablonen- und Einschubmethoden



Quelle: In Anlehnung an [Fayad99a], S.382

Die Implementierung der Schnittstellen bedarf dabei einer Mischung von Methoden statischer und flexibler Funktionalität. Statische Methoden (Abbildung 5.6, t(...)) sind vollständig implementiert und delegieren Teile ihrer Aufgaben an abstrakte Methoden (Abbildung 5.6, h(...)), durch die der Anwendungsentwickler die Möglichkeit besitzt, das Framework anzupassen. Sie werden im objektorientierten Umfeld mittels Vererbung der abstrakten Klasse (Abbildung 5.6, Klasse A) in einer konkreten Klasse (Abbildung 5.6, Klasse A1) implementiert. Je weniger dieser Einschubmethoden eine Framework also

besitzt, um so einfacher ist es zu benutzen. Leider leidet auch die Flexibilität eines Frameworks, wenn nicht genügend Einschubmethoden zur Verfügung stehen. In einem solchen Fall wäre eine Erweiterung, um komplexe, zusätzliche Funktionalitäten nur noch erreichbar, wenn auch die Schablonenmethoden überschrieben werden würden. Die Kosten für eine Anpassung und Wiederverwendung würden dadurch enorm ansteigen.

Eine weitere Herausforderung bei der Entwicklung von Frameworks ist es, eine passende Komposition von Schablonen- und Einschubmethoden zu finden. Wie müssen die Methoden idealerweise miteinander interagieren? An diesem Punkt in der Entwurfsphase eines Frameworks kommt der Einsatz von Entwurfsmustern zum tragen. Wie schon in Kapitel 4 vorgestellt, sind Muster ein Mittel bewährte Entwurfsentscheidungen treffen zu können, um eine Revidierung des Entwurfs überflüssig zu machen. Eine bedeutende Rolle bei der Auswahl eines adäquaten Entwurfsmusters spielt dabei die Kenntnis des Entwurfsmusterkatalogs der „Gang of Four“, wie er in [Gamma96] veröffentlicht wurde. Eine Zusammenstellung der gängigen Entwurfsmuster findet sich im Anhang dieser Arbeit.

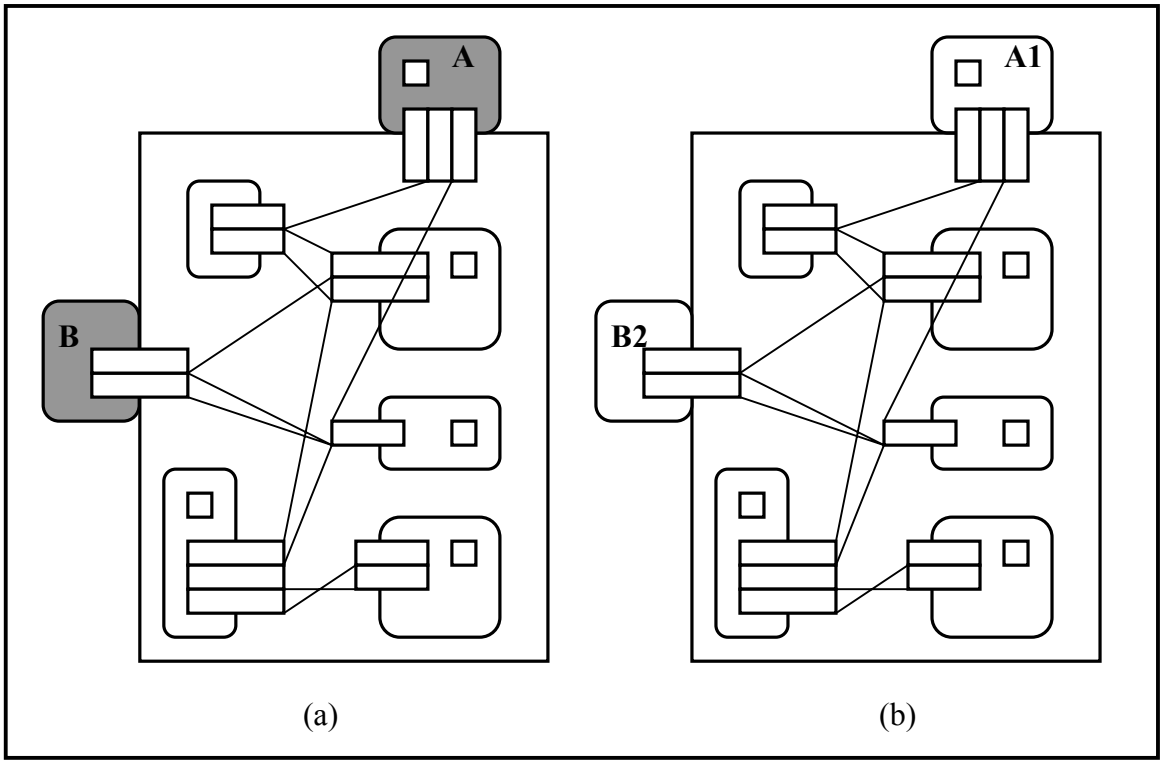
Beim Frameworkentwurf sind die Muster *Abstract Factory*, *Builder*, *Observer*, *State* und *Strategy* von besonderer Bedeutung. Sie alle bieten dem Entwickler Methoden zur Verknüpfung statischer und dynamischer Teilbereiche einer Softwareumgebung. So ist das *Factory Pattern* z.B. dafür verantwortlich, Instanzen von Objekten zur Laufzeit zu erzeugen, welche aber zur Designzeit noch nicht festgelegt werden müssen. In Java wird dieser Mechanismus in Zusammenhang mit der *Reflection API* verwandt, um zur Laufzeit Klassen in das Framework zu integrieren.

Ein weiteres oft benutztes Muster ist das *Observer Pattern* (Beobachter). Es sorgt dafür, daß ein Objekt überwacht wird und bestimmte Änderungen innerhalb dieses Objekts an den oder die Beobachter weitergegeben werden. Somit können Abhängigkeiten automatisch aktualisiert werden und es wird ein einheitlicher aktueller Stand des Systems gewährleistet.

Musterkenntnisse sind also Kernkompetenzen eines guten Frameworkentwicklers. Sie reichen jedoch nicht aus, um ein komplettes Framework zu entwerfen. Zu einem Framework gehören nicht nur die Stellen der dynamischen Erweiterbarkeit, sondern auch starre Komponenten, die einen Großteil zum Wiederverwendungscharakter beitragen. Komponenten sind dabei elementare Teile des Frameworks, welche eine Vielzahl von Standardfunktionalitäten anbieten. In Black-Box Frameworks sind sie sogar der einzige Ansatzpunkt, um ein Framework den Anforderungen des Anwendungsentwicklers anzupassen. Aus vielen Implementierungen der Einschubmethoden entwickeln sich im Laufe der Zeit, fertige wiederverwendbare Komponenten, die nur noch in der richtigen Struktur zusammengefügt werden müssen. Je mehr Anwendungen mit dem Framework entwickelt wurden, um so mehr läßt sich der Komponentencharakter immer wieder benötigter Implementierungen erkennen. Die Schablonenmethoden stellen dann nur noch Schnittstellen für die Komposition von Komponenten zur Verfügung.

Wie bereits in Kapitel 3 erwähnt, haben komponentenbasierte Systeme einen hohen Wiederverwendungswert. Anforderungen der Anwendungen bei denen schon während des Entwurfs eines Frameworks absehbar ist, daß sie für sämtliche Applikationen identisch sind, können als Komponenten in eine Framework integriert werden. Sie können später in einer speziellen Anwendung aktiviert oder deaktiviert werden um deren Funktionsumfang festzulegen.

Abbildung 5.7: Frameworks vor und nach der Spezialisierung



Quelle: In Anlehnung an [Fayad99a], S.381

Abbildung 5.7 verdeutlicht die Interaktion zwischen Schnittstellen und Komponenten innerhalb eines Frameworks. (a) zeigt dabei ein unspezialisiertes Framework, wie es meist in uninstanziierter Form vorliegt. (b) hingegen stellt das instanziierte und an spezielle Anforderungen angepaßte Framework dar, wie es innerhalb eines Applikationssystems zu finden ist. Das Framework kapselt diverse Komponenten mit statischer Funktionalität (*frozen spots* [Pree97]) und bietet Schnittstellen zur Spezialisierung an seinen Hot-Spots. Die Klassen A1 und B2 bezeichnen eine anwendungsspezifische Modifikation von Methoden durch Vererbung.

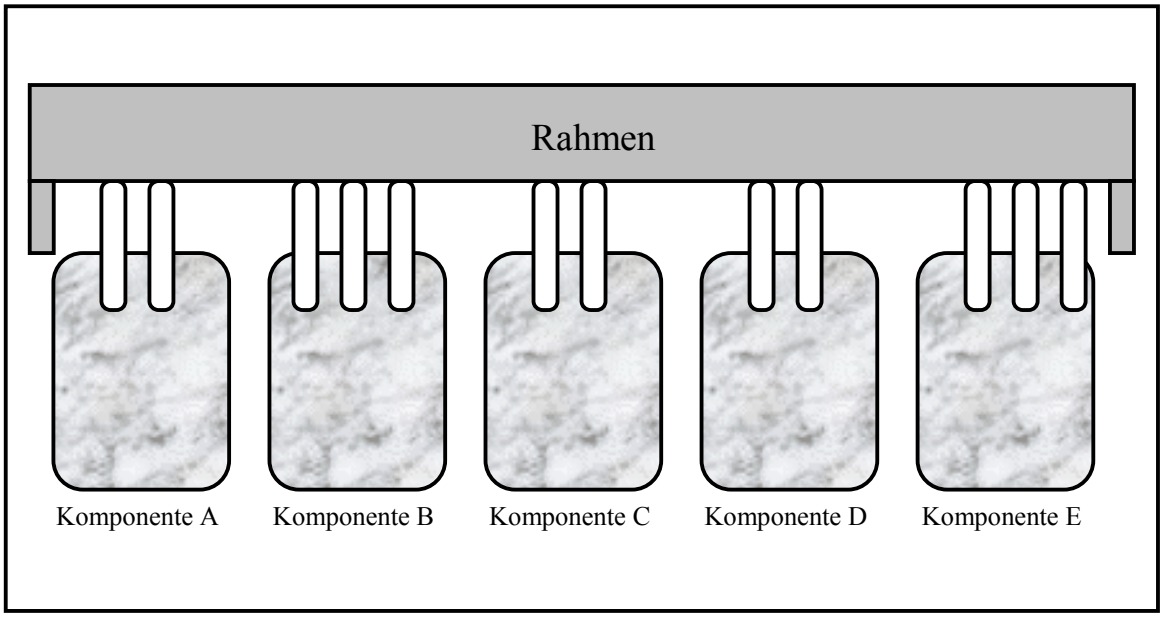
Einen weitestgehend identischen Ansatz bei dem Entwurf von Schnittstellen beschreibt Fayad in [Fayad99a]. Er gebraucht dabei den Begriff der „Hooks“ zur Beschreibung von Einschubmethoden. Seine Abhandlung fokussiert den Dokumentationsaspekt allerdings stärker und beschäftigt sich weniger mit dem eigentlichen Entwurf.

5.7.4 Frameworkarchitektur – Ein Zusammenspiel der Komponenten

Nachdem nun auf den komponentenbasierten Ansatz von Frameworks eingegangen wurde, gilt es noch zu klären, wie die Komponenten untereinander interagieren und kommunizieren.

Die bloße Komponentenkomposition innerhalb eines Frameworks macht es noch nicht zu dem was es eigentlich sein sollte, ein System zum Zusammenspiel der Komponenten. Frameworks müssen auch dafür Sorge tragen, daß die enthaltenen Komponenten sich sinnvoll in eine Umgebung integrieren. Aufgaben bzw. Funktionen lassen sich nicht immer vollkommen in einzelnen Komponenten zusammenfassen. Es müssen also Wege geschaffen werden, die einen eindeutigen Informationsfluß zwischen den Komponenten gewährleisten und ihre Einzelfunktionen zu einem komplexen System mit vielen Funktionalitäten erweitern. Über Schnittstellen alleine ist dies nicht möglich. Folglich sind nicht nur adäquate Komponenten zusammenzusetzen um ein Framework zu schaffen, sondern es muß ein Applikationsrahmen entwickelt werden, der die Dienste der Komponenten nutzbar macht.

Abbildung 5.8: Interne Struktur eines Frameworks



Der Rahmen koordiniert die Abläufe innerhalb des Softwaresystems, indem er gezielt die Schnittstellen der Komponenten anspricht und gewonnene Informationen weiterverteilt. Er ist also eine Art Verbindungsstück zwischen den Komponenten des Frameworks. Seine

Entwicklung ist die eigentliche Herausforderung des Frameworkentwicklers und fordert von ihm eine exakte Modellierung von Prozeßabläufen.

Eine Anpassung des Rahmens innerhalb eines Frameworks ist später nur mit großem Aufwand zu vollziehen. Deshalb ist ein guter Entwurf mit entsprechenden Freiräumen zur späteren Anpassung (Hot-Spots) unabdingbar.

5.7.5 Dokumentation

Während der systematischen Frameworkentwicklung entstehen schon diverse Dokumente, die im Anschluß an die Implementierung in eine Dokumentation umgesetzt werden können. Der Frameworkentwickler muß sich, um sein Ziel eines einfach zu verwendenden Frameworks verwirklichen zu können, während der ganzen Entwicklungsphase in die Rolle des Frameworkbenutzers versetzen. Im Gegensatz zur herkömmlichen, traditionellen Softwareentwicklung ist es bei der Frameworkentwicklung nicht immer möglich, mit einem Kunden zu kommunizieren, da dieser potentielle Kunde noch gar nicht existiert. Frameworkentwickler müssen sich also selbst ihre Anforderungen definieren.

Bei der Benutzung des Frameworks muß sichergestellt werden, daß der Anwender das Framework genauso benutzt wie es sich der Entwickler gedacht hat. Dazu muß er im Laufe der Entwicklung Verwendungshinweise spezifizieren. Diese verkürzen den Lernprozeß erheblich und machen das Framework damit wiederverwendbarer. Die Zeit ein Framework zu verstehen, darf die Zeit, die benötigt wird um eigenständige Applikation zu entwickeln, nicht übersteigen. Die strukturierte Erarbeitung der Verwendungshinweise ist auch für den Entwickler hilfreich. Sie zwingt ihn dazu, sich genaue Gedanken über die Verwendung des Frameworks zu machen.

Eine gute Dokumentation der flexiblen Frameworkteile macht es dem Anwender wesentlich leichter, passende Einsprungpunkte in das Framework zu lokalisieren. Der Entwickler muß bei der Entwicklung definieren, für welche Anforderungen Hooks [Fayad99a] und Hot-Spots [Pree97] nützlich sind, für welchen Anwendungsbereich sie gedacht sind und welche anderen Komponenten am Prozeß beteiligt sind. Der Anwender kann die Auswirkungen seines Eingriffs in das System somit abschätzen.

Die Benutzung eines Hooks scheint zunächst nicht schwer zu sein. Allerdings muß sie nicht immer in Form einer einfachen Klassenvererbung stattfinden. Es kann durchaus vorkommen, daß der Anwender komplexe Klassenstrukturen mit vielen beteiligten Komponenten anpassen muß. Ohne eine exakte Schnittstellendokumentation wäre das unmöglich. Die Dokumentation eines Hooks bietet dem Entwickler die Chance, sein Wissen an den Anwender weiterzugeben. Sie kapselt das Wissen über die Verwendung des Hooks zur Anwendungsentwicklung.

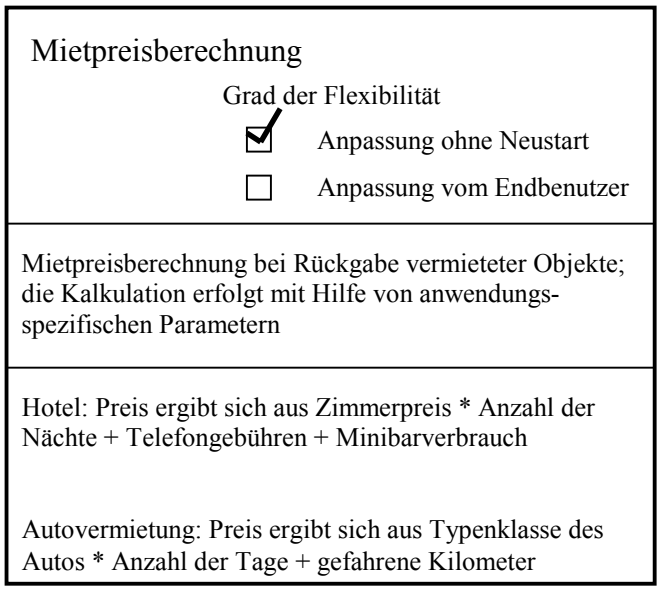
Wie sieht die Dokumentation eines Hooks nun aus? Die Beschreibung eines Hooks unterteilt sich in mehrere Punkte. Fayad legt in [Fayad99a] ein einheitliches Format zu ihrer Definition fest.

Tabelle 5.1: Beschreibung von Hooks

Informationstyp	**Beschreibung**
Name	ein eindeutiger Name, der den Hook identifiziert
Anforderungen	Anforderungen, für die der Hook zuständig ist und welche Probleme er lösen kann
Typ	Typisierung anhand der Eingriffsmöglichkeiten in das System, z.B. Funktion hinzufügen, Funktion ersetzten, Funktion ein-/ausschalten und Funktion erweitern. Des weiteren die Art der Benutzungsunterstützung, wie z.B. Auswahl aus bestimmten Komponenten, Musterunterstützter oder freier Programmiereingriff
Bereich	Teile des Frameworks die beeinflußt werden
Gebrauch	Komponenten, die zusätzlich benötigt werden um diesen Hook zu gebrauchen
Teilnehmer	Komponenten die von diesem Hook benutzt werden
Änderungen	Hauptbereich. Beschreibung des Kontrollflußes innerhalb aller teilnehmenden Komponenten
Einschränkungen	Einschränkungen die der Hook selber oder seine Benutzung hat
Kommentare	beliebige zusätzliche Kommentare

Ein ähnliches Ziel verfolgt Pree mit seinen *Hot-Spot-Karten*. Sie sollen eine gemeinsame sprachliche Basis zwischen Anwendungsexperten und Softwaretechnikern bilden [Pree97]. Sie gliedern sich in Datenkarten, auf die hier mangels Einsatzmöglichkeit nicht näher eingegangen wird, und Funktionskarten, welche eine flexible Funktionalität des Frameworks dokumentieren. Auf ihnen wird zunächst der Funktionsname und der gewünschte Grad an Flexibilität festgelegt. Flexibilität bedeutet in diesem Zusammenhang zum einen, daß der Hot-Spot zur Lauf- oder Designzeit verändert werden kann, und zum anderen, ob der Endbenutzer die Möglichkeit hat den Hot-Spot anzupassen. Der wichtigste Punkt auf einer Hot-Spot-Karte faßt seine Funktionalität zusammen. Sie sollte abstrakt beschrieben werden. Schließlich werden noch mindestens zwei verschiedene Anwendungssituationen exemplarisch angeführt.

Abbildung 5.9: Beispiel einer Funktionskarte für Hot-Spots



Quelle: In Anlehnung an [Pree97], S.69

Zusammenfassend läßt sich sagen, daß eine strukturierte Vorgehensweise bei der Dokumentation des Frameworks sowohl für den Anwender, als auch für den Frameworkentwickler eine große Arbeitserleichterung ist. Beiden ist es somit möglich, den Überblick über ein solch komplexes System zu behalten.

5.8 Zusammenfassung

Frameworks bieten Lösungen für eine Menge verwandter Probleme. Sie bestehen aus einer Anordnung von Klassen die den Kontroll- und Informationsfluß und die Interaktion zwischen Komponenten zur Laufzeit steuern. Die Lösung anwendungsspezifischer Probleme geschieht mit Hilfe von flexiblen Frameworkkomponenten (Hot-Spots/Hooks), welche ohne Eingriff in das Framework durch den Anwender angepaßt werden können. Im Gegensatz zu Entwurfsmustern sind Frameworks nicht nur theoretische Konstrukte, sondern fertige Implementierungen der Muster für eine Menge von Applikationen. Frameworks sind idealerweise hochgradig wiederverwendbar, aber dennoch leicht zu benutzen. Es werden neben der Implementierung auch Analyse, Entwurfsentscheidungen und Architektur wiederverwandt.

Frameworks unterteilen sich in unterschiedliche Kategorien. Aus Sicht der zu verwendenden Erweiterungstechniken gibt es Black- und White-Box Frameworks, sowie deren Abstufungen. Eine weitere Klassifizierung besteht bei der Aufteilung in verschiedene Anwendungsbereiche. Technische Frameworks unterstützen bei der technischen Realisierung einer Applikation oder eines weiteren Frameworks. Fachliche Frameworks leisten Hilfestellungen in bestimmte Anwendungsdomänen. Sie lösen fachliche Probleme.

Bevor Frameworks angewandt werden können, müssen sie aufgrund ihres Komplexitätsgrades erlernt werden. Der Anwender muß wissen, an welchen Stellen und auf welche Art und Weise er das Framework seinen speziellen Anforderungen anpassen kann. Ein wichtiges Hilfsmittel für ihn ist dabei eine gute Dokumentation der anpaßbaren Punkte.

In der Analysephase des Entwicklungsprozesses werden nicht nur Anforderungen an eine Applikation analysiert, sondern es wird ein kompletter Anwendungsbereich, eine Anwendungsdomäne, untersucht. Sie soll Aufschluß über mögliche Anforderungen an Applikationen geben. Auch die Entwurfsphase weicht von der herkömmlichen Vorgehensweise ab. Die Entwicklung von Frameworks erfordert wesentlich umfangreichere Überlegungen während des Entwurfs, als dies bei traditionellen

Applikationen der Fall ist. Demzufolge ist die Frameworkentwicklung mit einem hohen Zeit- und Kostenaufwand verbunden und rentiert sich nur bei entsprechend häufiger Nutzung.

Frameworks erfreuen sich heutzutage in der schnellebigen Softwarebranche einer großen Beliebtheit und sind mittlerweile eine häufig genutzte Methode um Softwaresysteme wiederzuverwenden.

6 Beschreibung der entwickelten Komponenten

Hintergrund für die Entscheidung ein eigenes Framework zu entwickeln war die Tatsache, daß die Opitz Consulting GmbH seit mehreren Jahren erfolgreiche Projekte im Oracle Forms Developer Umfeld entwickelt. Hierzu wurde im Laufe der Jahre eine sogenannte Softwareproduktionsumgebung (SPU) geschaffen. Die SPU definiert neben einem Vorgehensmodell zur Erstellung datengetriebener Applikationen mit Forms auch diverse Mechanismen zur Implementierung von Programmteilen, welche sich in diversen Projekten wiederfinden. Ein Beispiel hierfür ist die Suchmaske, die eine leicht anpaßbare Suche auf den Feldern einer Datenbank ermöglicht. In nahezu jedem Projekt ist eine solche Suchfunktionalität gefordert. Damit diese in einem Projekt abgebildet werden kann, muß die in der SPU beschriebene Komponente lediglich noch durch einige notwendige Parameter modifizieren werden, um die Suchmaske den spezifischen Anforderungen anzupassen. Die SPU spezifiziert unter anderem also eine Art Komponentenmodell für die Formsentwicklung.

Neben den programmtechnischen Funktionalitäten sind in einer solchen Umgebung auch die dafür benötigten Datenmodelle enthalten. Ein zusätzlicher Entwicklungsaufwand wird dadurch vermieden. Der erneute Projekteinsatz der SPU führt folglich auch zu einer enormen Qualitätsverbesserung, weil bereits gewonnene Erfahrungen gesammelt und strukturiert abgelegt werden und neue Erkenntnisse einer Weiterentwicklung der SPU zu Gute kommen.

All diese Vorteile die das bisherige Projektgeschäft effizient unterstützt haben sind allerdings auf die Anwendungen in Zusammenhang mit dem Oracle Forms Developer abgestimmt. Der heutige Stand der Applikationsentwicklung entspricht aber nicht mehr dem zur Zeit der SPU Entwicklung. Zwar werden noch immer viele Anwendungen mit den Oracle spezifischen Werkzeugen entwickelt, doch der Trend geht immer mehr zu offenen und weniger proprietären Anwendungen.

In diesem Zusammenhang nimmt Java mittlerweile eine solide Stellung im Softwaremarkt ein. Dies ist vor allem daran zu erkennen, daß sich auch Unternehmen wie Oracle bemühen eine entsprechende Einbindung in ihre Produktpalette zu schaffen. Das Thema Java

Anwendungsentwicklung nimmt auch bei der Firma Opitz Consulting GmbH einen immer bedeutenderen Stellenwert ein. Leider können solche Anwendungen nicht im Entferntesten mit dem Aufwand einer Formsapplikation entwickelt werden. Das führt dazu, daß ein entsprechendes Projekt sowohl für den Kunden als auch für das entwickelnde Unternehmen wesentlich kostenintensiver wird als bisher. Ein Schritt zum Einsatz solch offener Technologien wird somit erheblich erschwert. Um diesem Nachteil vorzubeugen hat die Firma Opitz Consulting GmbH sich dazu entschlossen ein neue Produktionsumgebung für das Java Applikationsumfeld zu entwickeln. Dazu sollten die Erfahrungen aus der bereits bestehenden SPU als auch die Erkenntnisse der Softwarebranche bei der Frameworkentwicklung genutzt werden. Frameworks erscheinen hier als *das* geeignete Mittel um eine der SPU entsprechende Wiederverwendbarkeit zu schaffen.

Die Entwicklung eines Java Frameworks gestaltet sich dabei allerdings um ein vielfaches komplexer als bei der SPU. Mit der Zeit sind zu den bei der SPU Erstellung vorhandenen Funktionalitäten von Softwaresystemen viele neue Möglichkeiten der Softwarenutzung entstanden. Gerade Java hat in diesem Zeitraum gravierende technologische Fortschritte gemacht. Zu nennen seien hier nur einmal Möglichkeiten der Softwareverteilung über mehrschichtige Architekturen. Diese Anforderungen an eine Software müssen natürlich auch in dem zugehörigen Framework abgedeckt werden. Da hierfür kein geeignetes Frameworkprodukt existierte wurde der Beschluß gefaßt eine angemessene Eigenentwicklung durchzuführen.

Im folgenden werden Komponenten des bei der Firma Opitz Consulting GmbH entwickelten Frameworks mit Namen „Apollo“ vorgestellt. Es soll die Grundlage zur Erstellung objektorientierter, datenbankbasierter Enterprise Applikationen in Java bilden. Die grafische Oberfläche wird dabei mit Java Swing Komponenten realisiert. Durch den Einsatz der Oracle Business Components for Java kommt in dem „Apollo“-Framework eine echte JDBC-Persistenzschicht zum Einsatz, die es möglich macht, die entwickelte Applikation gemäß dem Java 2 Enterprise Edition Blue Print zu verteilen..

Ziel dieser Entwicklung ist es, ein leistungsfähiges Framework zu erstellen, welches die softwaretechnische Umsetzung diverser Anwendungsfälle extrem erleichtert. So sollen die Entwicklung von typischen Benutzungsszenarien, der Datenanbindung und die Erzeugung der Oberfläche (GUI-Komponenten), die fast bei jeder Anwendung in ähnlichen Art und Weise vorkommen, wesentlich vereinfacht werden. Standardfunktionalitäten, wie die Pflege von Metadaten, z.B. Benutzern und deren Berechtigungen, sind ebenfalls gängige Anforderungen an Enterprise Applikationen und werden im Rahmen des Frameworks abgebildet.

Darüber hinaus ist eine strikte Trennung des Datenbereichs (application component) und des Darstellungsbereichs (presentation component) der Anwendung Ziele bei der Entwicklung des „Apollo“-Frameworks. Die Entwicklung einer Schnittstelle zwischen diesen Komponenten ist Teil der im folgenden beschriebenen Frameworkkomponenten und stellt einen wiederverwendbaren Entwurf dar. Durch eine Vielzahl abstrakter und konkreter Klassen, die vom Framework bereit gestellt werden, können diese Klassen einfach abgeleitet und an die jeweilige Anwendungsschicht angepaßt werden.

Dieses Kapitel enthält eine Einführung in wesentliche Teilkomponenten des „Apollo“-Frameworks. Hierzu werden die Anforderungen an die Komponenten, deren Funktionsweise, die Anwendungsarchitektur und das zugrundeliegende Datenmodell beschrieben, sowie eine kurze Benutzeranleitung gegeben.

6.1 Authentifikation – Die Benutzerverwaltung

6.1.1 Anforderungen an die Benutzerverwaltung

Das Framework soll eine grundlegende Benutzerauthentifikation gegen die Datenbank zur Verfügung stellen, da diese in fast sämtlichen datenbankbasierten Systemen benötigt wird. Bevor ein Anwender mit der Anwendung, die mit dem Framework entwickelte wurde, arbeiten kann und Zugriff auf eine Datenbasis bekommt, soll er sich gegenüber der Anwendung anmelden und im System bekannt machen. Diese Anmeldung bildet gleichzeitig die Grundlage für die in Kapitel 6.2 eingeführte Berechtigungsverwaltung.

Um eine eindeutige Identifikation des Systembenutzers vornehmen zu können, muß er dem System seine Identität beweisen. Er soll dies über einen öffentlichen Benutzernamen und ein geheimes, nur ihm bekanntes Paßwort durchführen. Das Systems sollte weitestgehend vorhandene Funktionalität nutzen, so daß auf einfache Art und Weise Sicherheits- und Wartungsmechanismen von bestehenden Systemen genutzt werden können. Im Falle des Frameworks sind dies die Mechanismen der Oracle Datenbank.

Ebenso wie die Datenbank soll auch das „Apollo“-Framework eine Gruppierung von Benutzern in der Benutzerverwaltung umsetzen können. Dazu müssen sowohl datenbankseitig die nötigen Datenstrukturen definiert sein, als auch programmseitig Funktionen implementiert werden, welche auf diesen Strukturen arbeiten. Diese Funktionalitäten kommen allerdings hauptsächlich bei der in Kapitel 6.2 beschriebenen Berechtigungsverwaltung zum Tragen.

Um eine Verwaltung der Rechte auf die Datenbasis des Systems weitestgehend unkompliziert zu gestalten, gilt es zunächst eine Anmeldung des Benutzers gegenüber der Datenbank als Benutzerdatenbasis möglich zu machen. Demzufolge muß der Anwender dem System sowohl den Benutzernamen seines Datenbankschemas als auch dessen Paßwort bekannt machen.

6.1.2 Funktionsweise der Benutzerverwaltung

Gleich welches Authentifizierungsverfahren der Systementwickler für seine Anwendung wählt, muß der Benutzer immer ein einheitliches Interface an der Oberfläche angezeigt bekommen. Hierzu wird direkt nach dem Starten der Anwendung in der Klasse `ApplicationManager` die Methode `doLogin()` der Klasse `SessionManager` aufgerufen, welche für die Anzeige des in Abbildung 6.1 gezeigten Anmeldedialogs verantwortlich ist. Folgende Abbildung zeigt den Anmeldedialog, über den sich ein Benutzer mit seinem Benutzernamen und seinem Passwort anmelden kann.

Abbildung 6.1: Anmeldedialog des „Apollo“-Frameworks

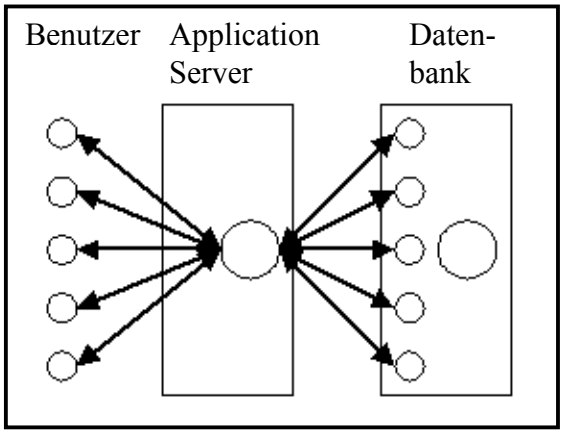
Quelle: Screenshot des „Apollo”-Frameworks

Die `SessionManager` Klasse ist die zentrale Benutzer- und Sitzungsverwaltungsinstanz der Applikation. Sie speichert den Namen des angemeldeten Benutzers und sein Paßwort. Sie kann somit genutzt werden, um eine Art Single-Sign-On Funktionalität bei der Anbindung an andere, externe Systeme zu implementieren. Benutzername und Paßwort könnten so z.B. für eine Anbindung an Oracle Workflow genutzt werden, ohne daß der Benutzer sich an zwei verschieden Systemen anmelden muß.

Nachdem der Benutzer seinen Schemanamen und das dazu gehörige Paßwort eingegeben hat, werden diese Informationen in der `connect()` Methode der `SessionManager` Klasse dazu genutzt, eine JDBC Verbindung zur Datenbank aufzubauen. Bevor dies jedoch geschieht, stellt die `connect()` Methode eine Verbindung zur logischen BC4J Mittelschicht her, um sich eine Instanz des BC4J Application Modules zu besorgen, dessen Transaktion schließlich mit der Datenbank verbunden wird. Aufgrund der optionalen Mehrschichtarchitektur kann dies bedeuten, eine Verbindung unter einem einheitlichen

Benutzernamen zu einem Application Server, einem CORBA ORB o.ä. herstellen zu müssen.

Abbildung 6.2: Authentifikationsarchitektur des „Apollo“-Frameworks

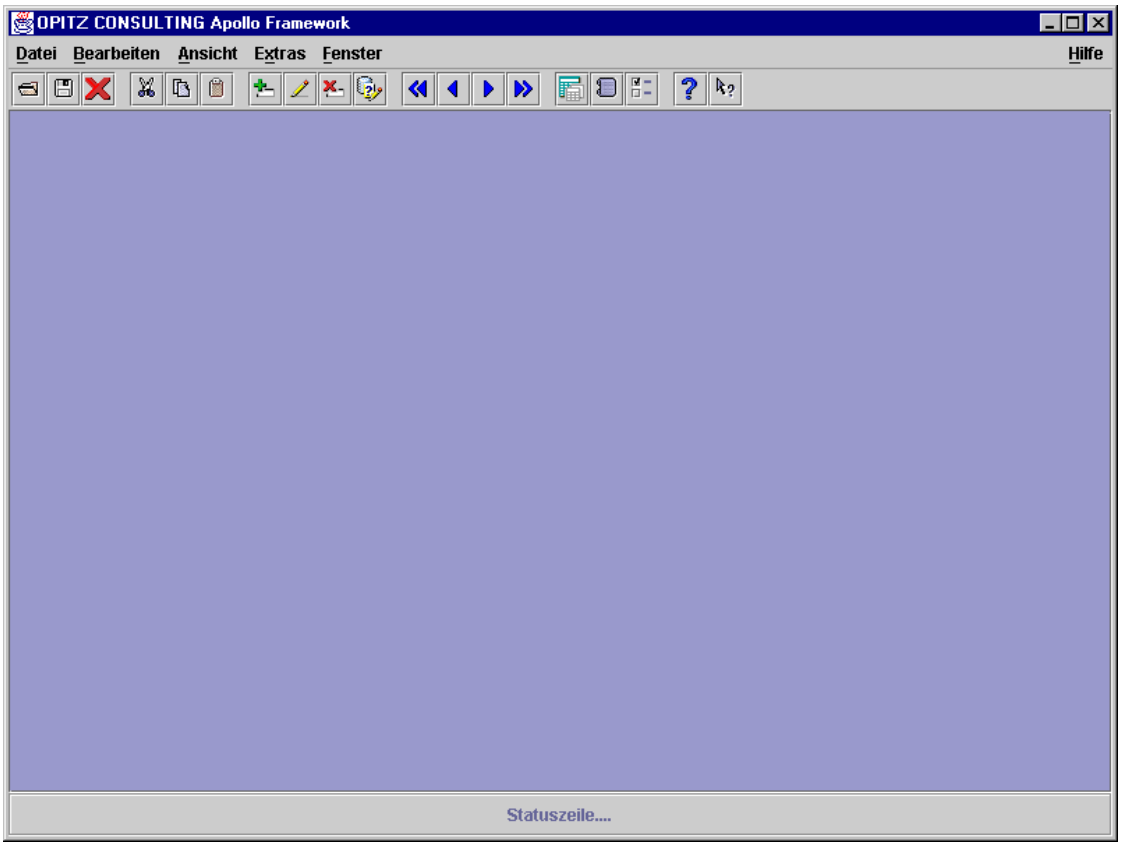


Quelle: Dokumentation des „Apollo”-Frameworks

Kann die anschließende Datenbankverbindung hergestellt werden, ist der Benutzer unter seinem Namen bei der Datenbank, aber auch bei der Applikation, angemeldet. Wenn andernfalls die Verbindung fehlschlägt, also von der Datenbank abgelehnt wird, ist diese Kombination von Benutzername und Paßwort falsch. Es wird ein Fehler angezeigt und der Login Dialog erneut gestartet. Der Anwender hat jetzt wieder die Chance seine korrekten Informationen einzutragen.

Eine erfolgreiche Anmeldung am System führt den Benutzer anschließend in die eigentliche Oberfläche des „Apollo“-Frameworks.

Abbildung 6.3: Applikationsrahmen des „Apollo“-Frameworks

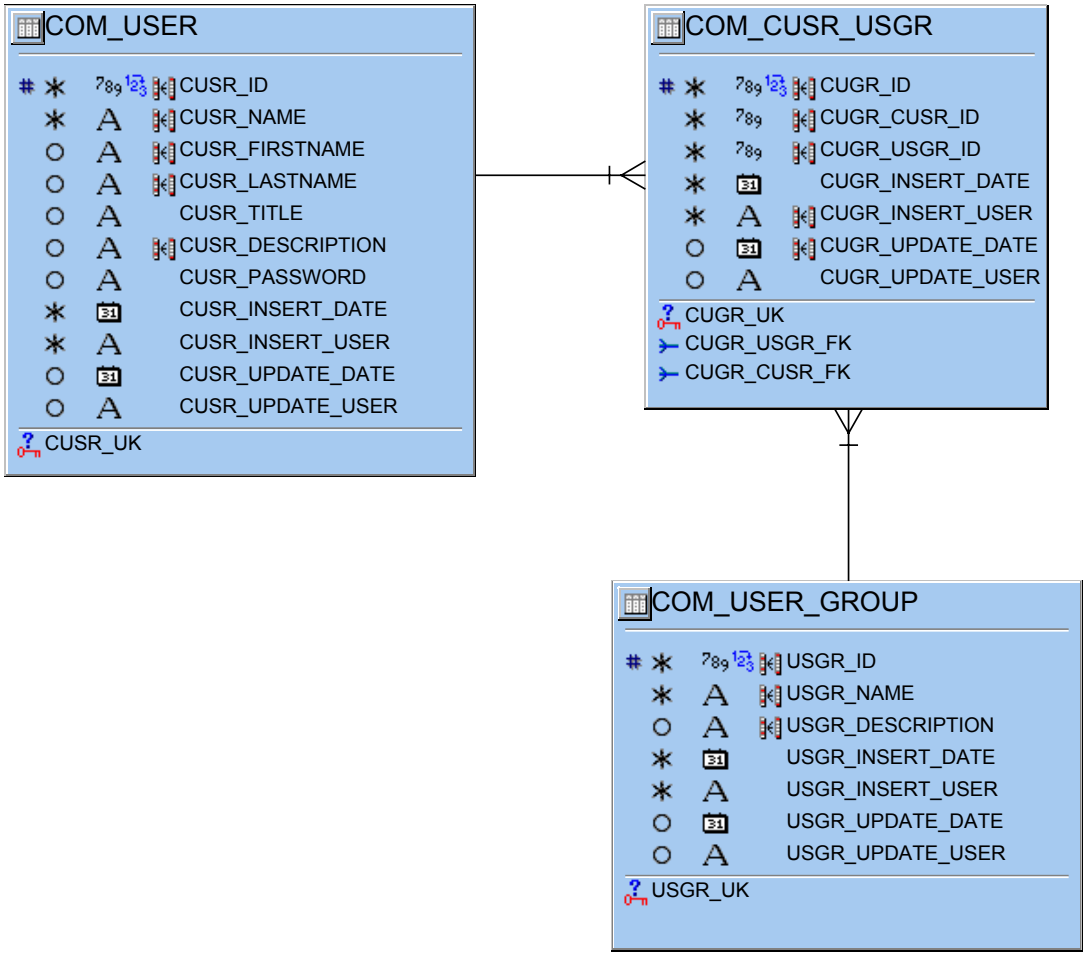


Quelle: Screenshot des „Apollo”-Frameworks

6.1.3 Datenmodell

Die Abbildung 6.4 zeigt das Datenmodell der „Apollo“ Benutzerverwaltung. In der Tabelle COM_USER werden die Benutzer des System mit ihren Informationen hinterlegt. Die COM_USER_GROUP Tabelle enthält die Benutzergruppen, für die später auch Berechtigungen vergeben werden können. COM_CUSR_USGR ist die Zuordungstabelle, welche die n:m Beziehung abbildet.

Abbildung 6.4: Datenmodell der Benutzerverwaltung



6.2 Autorisation – Die Berechtigungsverwaltung

6.2.1 Anforderungen an die Berechtigungsverwaltung

Neben der zuvor beschriebenen Benutzerverwaltung soll das „Apollo“-Framework auch eine grundlegende Berechtigungsverwaltung ermöglichen. Sie soll weitestgehend flexibel und einfach anzupassen sein. Zunächst sollen Berechtigungen auf beliebige Objekte zu vergeben sein.

Anlehnend an die Berechtigungsverwaltung der Datenbank erfordert auch eine im wesentlichen datengetriebene Applikation einen entsprechenden Autorisationsmechanismus. Neben den üblichen Berechtigungen selektieren, einfügen, ändern und löschen ist noch eine weitere Art, die Ausführberechtigung nötig. Sie bietet dem Anwendungsentwickler die Möglichkeit Rechte auf ausführbare Objekte zu vergeben. Wichtig ist dies z.B. bei der Rechtevergabe auf Maskenebene, wie sie aus dem Umfeld von Internetanwendungen bekannt ist. Dabei wird einem Benutzer durch eine Zugriffsbeschränkung auf bestimmte Masken erlaubt oder verweigert mit den Funktionalitäten auf der entsprechenden Maske, Daten zu manipulieren.

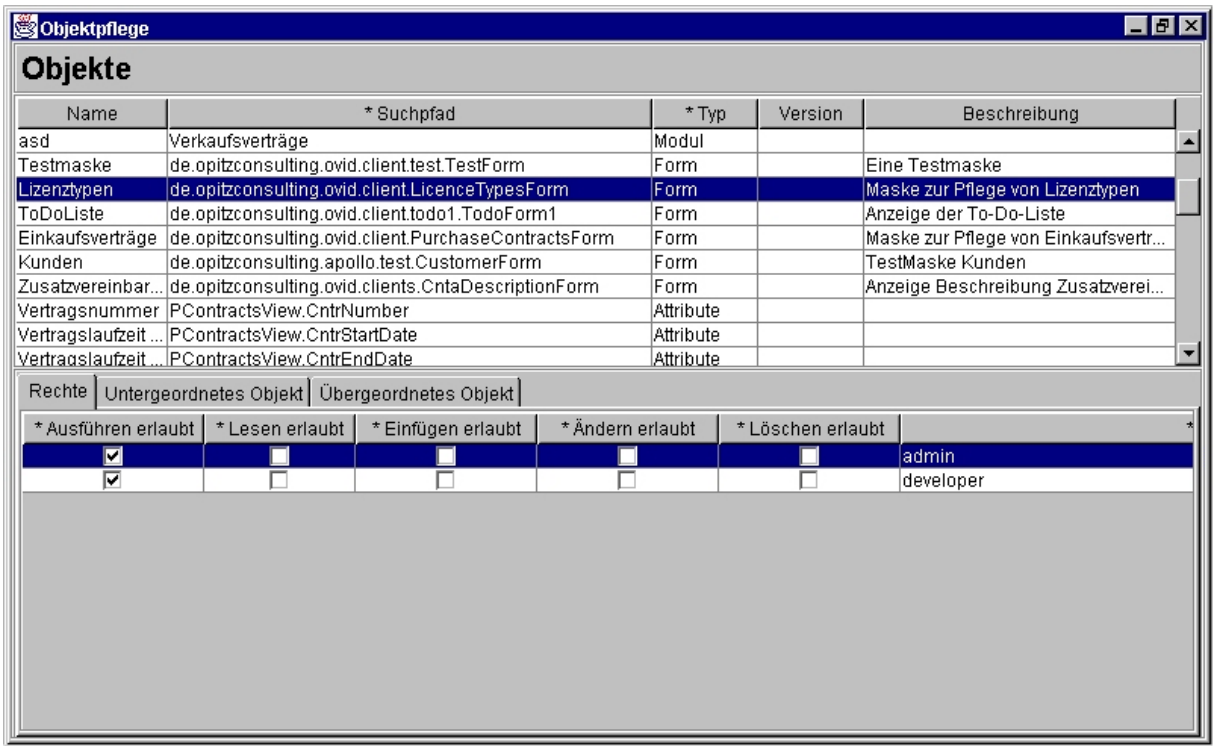
Die Berechtigungsverwaltung gliedert sich also in zwei Bereiche. Zum einen muß sie eine flexible Datenstruktur liefern, mit der man deklarativ Berechtigungen auf Objekte hinterlegen kann, und zum anderen sind programmtechnische Funktionen zum Überprüfen dieser Rechte notwendig.

Grundsätzlich sollen im Framework keine Rechte auf Benutzerebene vergeben werden. Der Fall, daß genau ein Benutzer des Systems eine fachliche Rolle im Anwendungssystem abdeckt, kommt in der Realität eines Anwendungssystems eher selten vor. Deshalb werden die Benutzer in verschiedene Benutzergruppen unterteilt. Die Objektberechtigungen sollen dann auf Ebene der Benutzergruppen vergeben werden. Dadurch wird ebenfalls eine Pflege des Systems übersichtlicher und einfacher handhabbar. Die benutzerspezifische Rechtevergabe kann somit immer noch abgedeckt werden, indem pro Systembenutzer genau eine Rolle bzw. Benutzergruppe gepflegt wird.

6.2.2 Funktionsweise der Berechtigungsverwaltung

Bevor die Berechtigungsverwaltung programmseitig genutzt werden kann muß die Datenstruktur des Autorisationsmechanismus gefüllt werden. Im „Apollo“-Framework werden dazu entsprechende Maske zur Systemverwaltung bereitgestellt. Hier können Systembenutzer angelegt, geändert, gelöscht und Benutzergruppen zugeordnet werden. Auf diese Benutzergruppen lassen sich über die Systempflegemasken Berechtigungen vergeben.

Abbildung 6.5: Berechtigungspflegemaske des „Apollo“-Frameworks



Quelle: Screenshot des „Apollo”-Frameworks

Um ein Recht auf ein Objekt vergeben zu können, muß dieses Objekt und der entsprechende Objekttyp in der Datenbank existieren. Es wird also zunächst ein Typ eines Objektes in der Datenbank angelegt. Bei dem Objekt Stammdatenverwaltungsmaske könnte dies z.B. der Typ Maske sein. Ebenso kann es sinnvoll sein Typen für bestimmte BC4J-Komponenten anzulegen, um Rechte auf ein Application Module oder ein View Object zu vergeben. Die Objekttypen dienen hauptsächlich dazu zu definieren mit welchen Rechtearten ein Objekt ausgestattet werden kann. So macht es zwar beispielsweise zwar Sinn ein BC4J View Object mit einem Recht zur Datenselektion zu versehen, jedoch

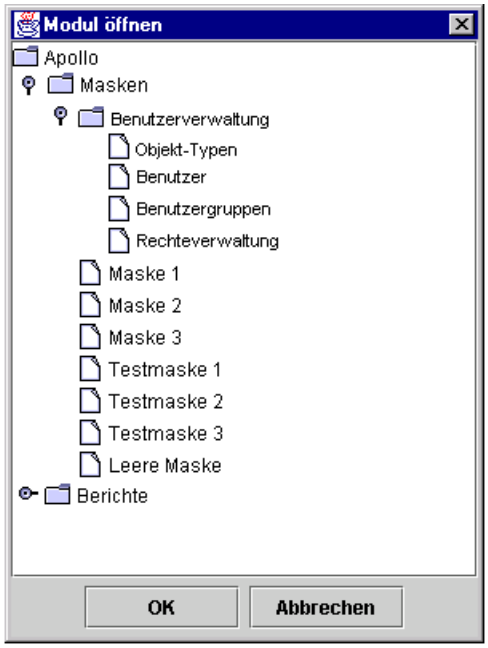
erscheint es wenig plausibel ebenfalls eine Selektionsrecht auch eine Bildschirmmaske zu vergeben. Für sie wäre ein Recht zum Ausführen der Maske zweckmäßiger.

Wenn ein adäquater Objekttyp in der Datenbank angelegt ist, kann ein Objekt diesen Typs in die Objekttabelle aufgenommen werden. Im Anschluß daran kann einer Benutzergruppe ein Recht auf dieses Objekt eingeräumt werden. Des weiteren können Objekte hierarchisch strukturiert werden, indem sogenannte Objektcontainer angelegt werden.

Schließlich gilt es dem Anwendungsentwickler eine programmtechnische Schnittstelle an die Hand zu geben. Im eigentlichen Anwendungscode muß er einfach auf die komplexe Struktur der Berechtigungen zugreifen können. Das „Apollo“-Framework bietet hierfür eine Schnittstelle in der Form, daß der Benutzer den `AuthorizationService` nutzt, um bestimmte Berechtigungen zu einem Objekt zu erfragen. Der `AuthorizationService` stellt Methoden in der Form von `canExecute()`, `canSelect()`, `canInsert()`, `canUpdate()`, `canDelete()` und `getUserGroups()` zur Verfügung. Über diese Methoden kann der Entwickler Informationen über Berechtigungen eines Benutzers erfragen. Allen `can...()`-Methoden wird lediglich der zu überprüfende Objektname sowie der Benutzername übergeben. Das Ergebnis des Methodenaufrufs liefert dem Entwickler eine einfache Aussage, ob der angegebene Benutzer ein übereinstimmendes Recht besitzt. In Abhängigkeit von der Antwort kann er dann programmseitige Entscheidungen treffen, um z.B. bestimmte Objekte auf der Oberfläche zur Anzeige zu bringen oder nicht.

Das „Apollo“-Framework nutzt diesen einfachen Mechanismus, um ein Startmenü aufzubauen. Dieses Menü bedient sich der `canExecute()`-Methode indem es einen Menüpunkt mit einem Verweis auf eine Maske nur dann anzeigt, wenn der Benutzer auch ein Recht zum Ausführen der Maske hat. Dies implementiert auf simple Art und Weise eine Berechtigungsverwaltung auf Maskenebene.

Abbildung 6.6: Menü „Maske öffnen“ des „Apollo“-Frameworks

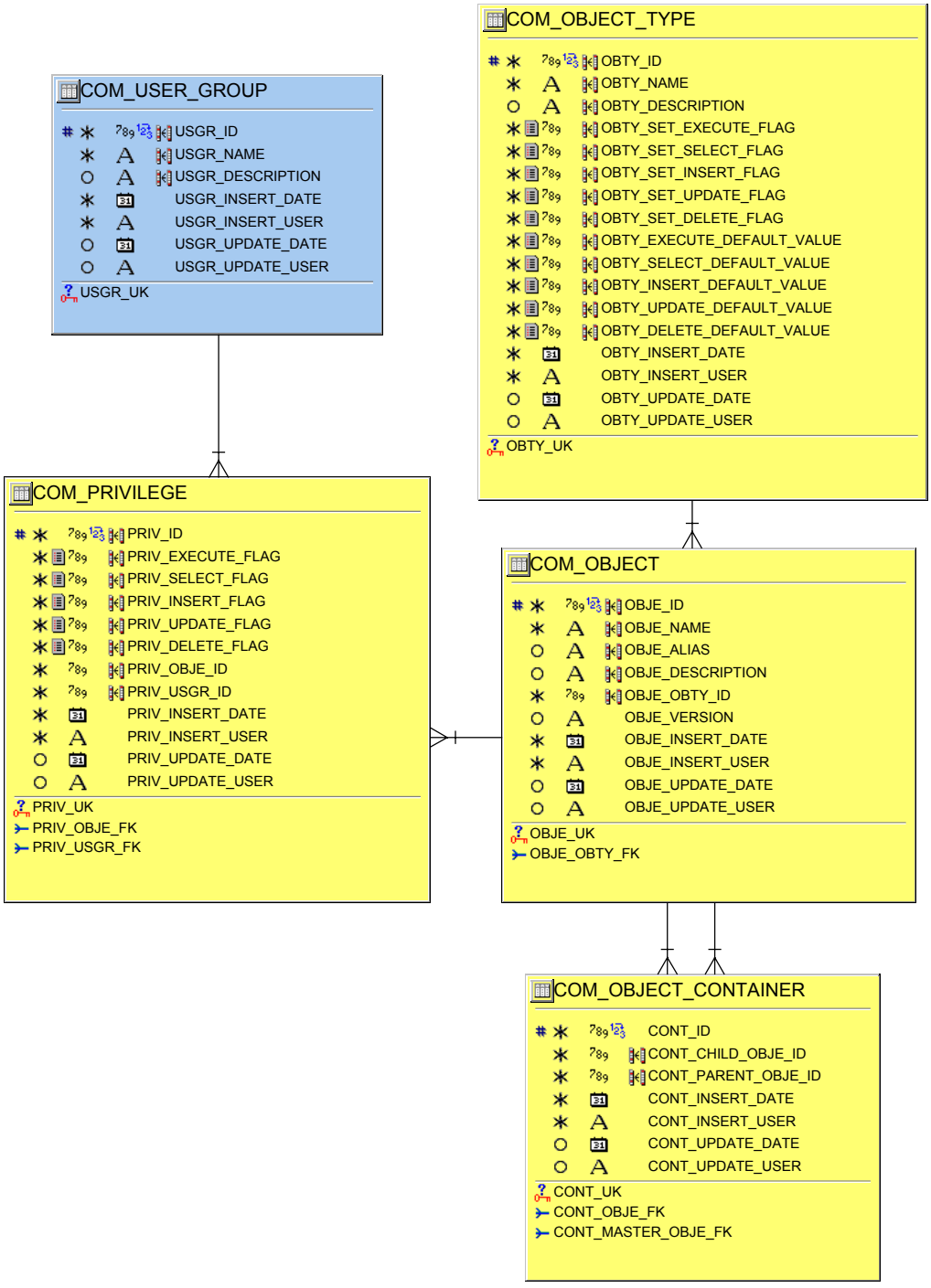


Quelle: Screenshot des „Apollo”-Frameworks

6.2.3 Datenmodell

Abbildung 6.7 stellt das Datenmodell der „Apollo“ Berechtigungsverwaltung dar. Sie umfaßt dabei auch eine die Tabelle COM_USER_GROUP der Benutzerverwaltung, da die Berechtigungen für diese Benutzergruppen vergeben werden. Sowohl das hier abgebildete Datenmodell, als auch das aus Abbildung 6.4 der Benutzerverwaltung sind Teil des Datenmodells zur Systempflege des „Apollo“-Frameworks, welches im Anhang dieses Dokuments enthalten ist. Die COM_PRIVILEGE Tabelle enthält die Berechtigungen eines Benutzers auf eine bestimmtes Objekt. Diese Objekte sind in der COM_OBJECT Tabelle abgelegt und werden über die COM_OBJECT_CONTAINER Tabelle hierarchisch Strukturiert, indem hier eine n:m Selbstreferenz abgebildet wird. In COM_OBJECT_TYPE sind die verschiedenen Objekttypen, die eine Applikation besitzen kann mit ihren Standardberechtigungen hinterlegt.

Abbildung 6.7: Datenmodell der Berechtigungsverwaltung



6.3 Applikationsweites Sessionmanagement

6.3.1 Anforderung an ein Sessionmanagementsystem

Eine essentielle Anforderung an das „Apollo“-Framework ist eine Verwaltung sowohl von Datenbankverbindungen (engl. Transactions), als auch von Verbindungen vom Client zur Mittelschichtarchitektur. Ein Benutzer des Systems soll zwingend seine einmal aufgebauten Verbindungen während seiner ganzen Anwendungssitzung (engl. Session) beibehalten. Somit ist im Falle einer Datenbankverbindung gewährleistet, daß eine Verbindung von Mittelschicht zu Datenbank nur einmal pro Session aufgebaut werden muß. Besonders wichtig ist dies für die Performanz des Gesamtsystems. Je nach Netzwerkumgebung kann eine erneute Anmeldung an die Datenbank mehrere Sekunden dauern.

Die Aufrechterhaltung der Datenbankverbindung ist ebenso für die gesamten Lockingmechanismen auf Datenbankebene von besonderer Bedeutung. Wird in der Applikation ein Datensatz modifiziert wird dieser auf der Datenbank gelockt. Diese Reservierung des Datensatzes für einen Benutzer wird erst dann wieder aufgehoben, wenn ein Commit- oder Rollbackbefehl an die Datenbank gesendet wird. Um einen solchen Lock eines Datensatzes in der gesamten Applikation bis zu einem Commit oder Rollback aufrecht zu erhalten, muß auch die Datenbankverbindung gehalten werden.

Solange mit der Applikation gearbeitet wird soll also immer wieder auf derselben Datenbankverbindung gearbeitet werden können. Erst wenn die Anwendung beendet wird kann auch die Datenbankverbindung getrennt und zuvor ggf. ein Commit ausgeführt werden.

Neben den Vorteilen der Aufrechterhaltung von Datenbanktransaktionen ist auch eine Verbindungsverwaltung für die Verbindung zur Mittelschicht enorm wichtig. Die Mittelschicht wird in dem „Apollo“-Framework durch die Business Components for Java repräsentiert. Die Nutzung dieser Mittelschichtarchitektur setzt voraus, daß während der kompletten Session ein und dieselbe Instanz eines Application Modules genutzt wird. Sollte diese Instanz im Laufe der Sitzung dereferenziert und somit zerstört werden, ist eine Zugriff auf die Daten der logischen Mittelschicht nicht mehr möglich. Die einzige

Ausnahme bildet dabei der Fall, daß die in der Anwendung bearbeiteten Daten regelmäßig in die Datenbank geschrieben, also persistent gemacht werden.

Neben den Verbindungsdaten müssen auch weitere Informationen Systemweit einfach zugreifbar sein. Es müssen z.B. häufig Informationen zu dem gerade angemeldeten Benutzer abgefragt werden.

Eine applikationsweite Sitzungsverwaltung ist also unabdingbar für eine performante und flexible Datenbankapplikation.

6.3.2 Funktionsweise des Sessionmanagementsystems

Das Sessionmanagement innerhalb des „Apollo"-Frameworks findet zentral in einer Klasse statt. Dies ist die Klasse `SessionManager`. Sie nutzt ein wiederverwendbares Designmuster der objektorientierten Softwareentwicklung, wie in Kapitel 4 beschrieben. Sie ist als Singletonpattern implementiert. Konkret bedeutet dies, daß im Verlauf der Applikation niemals mit einer zweiten Instanz dieser Klasse gearbeitet werden kann. Der Konstruktor dieser Klasse kann durch seine eingeschränkte Sichtbarkeit nicht von anderen Klassen oder von `SessionManager` abgeleiteten Klassen aufgerufen werden. Er ist als `private` deklariert.

Da kein Aufruf an den Konstruktor durchgeführt werden kann, wird dem Applikationssystem eine andere Schnittstelle angeboten. Die `SessionManager`-Klasse implementiert hierzu die öffentliche (engl. `public`) Methode `getInstance()`, um an eine Instanz dieses Objektes zu gelangen. Es wird dabei klassenintern sichergestellt, daß beim ersten Aufruf dieser Methode eine neue Instanz erzeugt und intern gespeichert wird. Jeder weitere Aufruf von `getInstance()` führt daraufhin lediglich zu einer Rückgabe der bereits erzeugten Klasseninstanz.

Der zuvor beschrieben Mechanismus liefert somit die Grundlage für eine applikationsweit vorhandene Instanz der `SessionManager`-Klasse. Nun gilt es noch weitere Objekte in dieser Instanz zu referenzieren, damit auch diese während des gesamten Programmablaufs verfügbar sind. Wichtigstes Objekt innerhalb des „Apollo"-Frameworks ist dabei eine

Instanz eines `ApplicationModule`. Sämtliche datenbasierten Transaktionen auf Mittelschicht- und Datenbankseite werden über dieses Modul abgewickelt. Das Framework beruht darauf, ein einziges Business Components `ApplicationModule` zu nutzen, um sich mit einer Datenquelle zu verbinden. Weitere benötigte Module werden diesem Modul als sogenanntes „nested“ `ApplicationModule` untergeordnet. Die zentrale Methode des `SessionManager` ist `getApplicationModule()`, welche eine Referenz auf das beim Starten des Systems erzeugten `ApplicationModule` zurückliefert.

Ein Business Components `ApplicationModule` verwaltet intern eine Datenbanktransaktion. Im Framework wird diese Transaktion beim Systemstart initialisiert und mit der Datenbank verbunden. Konkret bedeutet dies für alle datengetriebenen Applikationskomponenten, daß sie die Methode `getApplicationModule()` aufrufen müssen, wenn sie ebenfalls die applikationsweite Transaktionsverwaltung nutzen möchten. Zusätzlich stellt der `SessionManager` auch weitere Funktionen bereit. Einige beziehen sich dabei auf die eigentliche Transaktionsverwaltung und führen `commit()`, `rollback()`, `disconnect()`, `postChanges()`, `isTransactionDirty()` und `validate()` aus. Der Applikationsentwickler kann damit über eine einfache Schnittstelle auf diese Komponenten zugreifen und z.B. eine Datenbanktransaktion bei Anwendungsabbruch schließen oder einen Autocommitmechanismus implementieren.

Neben der Transaktionsverwaltung kann auch auf Funktionen der datengetriebenen Mittelschicht zugegriffen werden. Die Methoden lock() und isLocked() implementieren eine Lockingfunktionalität au der BC4J Mittelschicht. Eine Abbildung auf die Datenbank erfolgt damit automatisch durch BC4J.

Da der `SessionManager` auch für den gesamten Loginmechnismus verantwortlich ist, enthält er auch die systemweit benötigten Funktionen, um den aktuellen Benutzer und sein Paßwort zu erfragen. Dies wäre zum Beispiel bei der Anbindung an externe Systeme notwendig, wenn diese eine zweite Benutzerauthentifikation erfordern.

Zusammenfassend läßt sich sagen, daß der `SessionManager` eine globale Ressourcen- und Sessionverwaltung für die Applikation durchführt und dem Entwickler einfache Zugriffsschnittstellen auf anwendungsweit benötigte Funktionen bietet.

6.4 Connectoren – Datengetriebene Oberflächenelemente

6.4.1 Anforderungen an die grafische Oberfläche

Es liegt in der Natur datengetriebener Applikationen, daß diese Informationen aus einer Datenbank an die Präsentationsoberfläche des Anwendungsbenutzers bringen müssen. Im Falle des mehrschichtig konzipierten „Apollo"-Frameworks findet ein Unterteilung in drei logische Schichten statt. Im Hintergrund arbeitet die Oracle Datenbank und verwaltet dort sämtliche Datenstrukturen des Systems. Die logische Mittelschicht wird mittels des Business Components Persistenzframeworks realisiert. Der Datenaustausch zwischen diesen beiden Schichten ist somit schon durch BC4J abgedeckt. Die dritte Schicht des Systems ist ein Java Client. Er übernimmt die Darstellung der Oberfläche, welche Informationen in Java Swing und somit in grafischer Form auf dem Bildschirm präsentiert. Leider hat diese Oberflächenkomponente in ihrer Ursprungsform einen großen Nachteil. Sie hat von sich aus keinerlei Verbindungen zur logischen Mittelschicht.

Die essentielle Anforderung an die Oberflächenkomponenten, wie z.B. Textfelder, ist also, daß sie eine Schnittstelle zur BC4J Schicht beinhalten, um eventuelle Datenveränderungen auf der Mittelschicht sofort anzeigen zu können. Der Entwickler möchte sich nicht manuell um das Füllen eines Textfeldes kümmern. Das Textfeld soll hingegen selber wissen mit welcher Datenquelle es in Verbindung steht und welche Informationen hiervon dargestellt werden müssen. Dabei müssen die Oberflächenkomponenten immer die aktuellen Informationen aus der Mittelschicht anzeigen. Finden also Änderungen in der Mittelschicht statt, müssen automatisch sämtliche darauf beruhenden Swing Komponenten ihre Anzeige anpassen. Ebenso soll nach dem Eintragen von Informationen in eine Oberflächenkomponente eine Weiterleitung dieser Daten an die Mittelschicht erfolgen. Die Implementierung dieses komplexen Kommunikationsprozesses darf dabei allerdings nicht jedesmal dem Anwendungsentwickler überlassen werden. Er soll lediglich einmalig eine Konfiguration dieser Verbindung festlegen. Hierfür müssen dem Entwickler leicht zu bedienende Schnittstellen geschaffen werden.

Genau an diesem Punkt setzen die im „Apollo“-Framework Connectoren genannten Komponenten an. Wie der Name schon vermuten läßt, schlagen sie die Brücke zwischen den Java Swing Komponenten und der BC4J Mittelschicht. Sie sind einfach wiederverwendbare Softwarekomponenten und lassen sich leicht den Wünschen des Entwicklers anpassen bzw. sind entsprechend erweiterbar.

Eine Beschreibung der für das „Apollo“-Framework denkbaren Java Swing Komponenten findet sich in der unten aufgeführten Tabelle. Ein Großteil soll bereits in der ersten Frameworkentwicklungsphase realisiert werden. Des weiteren muß eine Anbindung weiterer Elemente leicht möglich sein.

Tabelle 6.1: Beschreibung von Java Swing Komponenten im „Apollo“-Framework

Swing Komponente	Beschreibung
JTextField	Das `JTextField` ist ein Textfeld zur grafischen Anzeige einer Textzeile. Das Model der Komponente ist eine Instanz der Klasse `Document`. Es enthält den Text, verschiedene Informationen wie z.B. die Cursorposition, Länge des Textes, verarbeitet die Änderungen des Textes und erzeugt die entsprechenden `Document` Ereignisse, die vom Entwickler verarbeitet werden können. Das `Document` kann entweder im Konstruktor angegeben oder mittels `setDocument()` gesetzt werden. Der Text eines Textfeldes kann mittels `setText(String)` gesetzt und mit `getText()` gelesen werden. Mittels `setEditable(boolean)` kann dem Benutzer das Eingaben von Text erlaubt oder verboten werden. Da es von `JTextComponent` abgeleitet ist, erbt es diverse Methoden seiner Vaterklasse.
JTextArea	Die `JTextArea` ist ebenfalls von `JTextComponent` abgeleitet und hat somit weitestgehend die gleichen Methoden wie das `JTextField`. Die Anzeigekomponente erlaubt darüber hinaus die Eingabe mehrzeiliger unformatierter Texte.
JTable	Die `JTable` ist eine Tabellenkomponente, die die flexible Programmierung von Tabellen beliebiger Struktur erlaubt. Sämtliche Informationen zu den, in der Oberflächenkomponente (engl. View), angezeigten Objekten werden in diversen Models gemäß des MVC Musters verwaltet. Die View und auch das Model der Tabelle erzeugen Ereignisse, die von einem Entwickler genutzt werden können, um mit individueller Programmlogik darauf zu reagieren. Die Tabelle bietet dem Nutzer eine Vielzahl von Möglichkeiten die Tabelle seinen Bedürfnissen anzupassen. Die wichtigsten Methoden sind dabei `setValueAt(Object, int, int)`, um einen Wert in einer Tabelle an einer bestimmten Position zu setzen und `getValueAt(int,int)`, um einen Wert aus einer bestimmten Position zu lesen.

Swing Komponente	Beschreibung
JCheckBox	Die `JCheckBox` ist ein Auswahlknopf und hat zwei Zustände: ausgewählt und nicht ausgewählt. Er sieht allerdings nicht wie ein Knopf aus, sondern wie ein Kästchen, das angekreuzt werden kann, um seine Auswahl anzuzeigen. Die Methode `setSelected(boolean)` bestimmt, ob das betreffende Kästchen angekreuzt ist. Mit der Methode `setEnabled(boolean)` kann man den Auswahlknopf ein- oder ausschalten.
JComboBox	Das Auswahlfeld ist ähnlich wie ein Texteingabefeld, mit dem Unterschied, daß es zusätzlich ein Aufklappmenü enthält, aus dem verschiedene Texte ausgewählt werden können. Mit `addItem()` können neue Einträge in die Liste des Aufklappmenüs aufgenommen werden. Mit `removeItem()` werden sie wieder entfernt. Die Einträge können mit `getItemAt(int)` aufgrund ihres Index abgefragt werden. Ihre Anzahl erhält man mit `getItemCount()`. Den momentan ausgewählten Eintrag erhält man mit `getSelectedItem()`.
JTree	Die `JTree` Komponente ist grafische Repräsentation einer Baumstruktur. Die besteht aus Knotenpunkten. Zu diesen Knoten können weitere Unterknoten existieren. Die Baumstruktur kann dabei über den Konstruktor und das Model festgelegt werden.
JRadioButton	Der `JRadioButton` ist ein spezieller Auswahlknopf. Der Unterschied zu den herkömmlichen Auswahlknöpfen besteht darin, daß eine Menge von Radioknöpfen in einer Gruppe (`ButtonGroup`) zusammengefaßt wird. Es kann jeweils nur ein Radioknopf aus einer Gruppe ausgewählt werden. Die Methoden entsprechen weitestgehend denen der Klasse `JCheckBox`.

Weitere Beschreibungen der Java Swing Komponenten sowie deren Architektur und Implementierung des Model-View-Controller Entwurfsmusters würden den Rahmen dieses Dokumentes sprengen und sind in dem Java Tutorial unter http://java.sun.com zu finden.

6.4.2 Funktionsweise und Softwarearchitektur der Connectoren

Das Prinzip der Connectoren ist die Verbindung von sichtbaren Java Swing Komponenten und der datengetriebenen Mittelschicht. Umgesetzt wurde dies innerhalb des „Apollo"-Frameworks, indem eine eigene Schicht zwischen grafischer Oberfläche (GUI) und BC4J eingezogen wurde.

Mit Rücksicht auf die Anforderung eine Framework zu entwickeln war die Benutzung von bewährten, wiederverwendbaren und erweiterbaren Softwareentwicklungsmethoden unverzichtbar. Die Connectoren wurden somit unter Verwendung verschiedener Entwurfs- und Designmuster entwickelt. Besonders hervorzuheben sind hier das Model-View-Controller (MVC) Pattern und das Beobachter (engl. Observer) Pattern. Ihre Verwendung wird in der nachfolgenden Beschreibung eines Connectors deutlich. Die Muster und der komponentenartige Aufbau der Connectoren machen sie somit zu hochgradig wiederverwendbaren Teilen des Frameworks. Ihre Leistung ist allerdings nur dann vollständig gewährleistet, wenn sie auch innerhalb des „Apollo"-Frameworks genutzt werden. Sie stellen diverse öffentliche Schnittstellen zur Verfügung, um einerseits Anpassungen an den Komponenten selber durchzuführen und andererseits deren Dienste zu nutzen.

Die Connectoren teilen sich in zwei Gruppen auf. Zum einen sind dies die Connectoren, welche naturgemäß nur mit einem einzigen Datenwert verbunden werden müssen. Dazu zählt z.B. der `TextFieldConnector`, der ein Java Swing `JTextField` mit einem Datenwert einer Spalte in der BC4J Mittelschicht verbindet. Diese Art Connector wird im Framework als Attribut Connector bezeichnet. Zum anderen gibt es auch noch die Connectoren, die mehr als einen Datenwert in ihrer GUI Komponente anzeigen müssen. Populärstes Beispiel dafür ist der `TableConnector`, der eine Ansicht mehrerer Datensätze in einer Swing `JTable` verwaltet. Connectoren die auf mehr als einem Datensatz arbeiten, werden RowSet Connectoren genannt.

In Anlehnung an die in Kapitel 3 vorgestellten Methoden zur komponentenbasierten Softwareentwicklung sind auch die Connectoren als Komponenten des „Apollo"-Frameworks entwickelt worden. Sie müssen dem restlichen Framework einheitliche und

genormte Schnittstellen liefern, damit sie über diese in das Framework integriert werden können. Dieses Interface wird anschließend z.B. von den Maskenflußkomponenten von „Apollo“ benötigt, um Validierungsmechanismen zu implementieren. Eine solche Normung der Schnittstellen wurde bei der Connectoren durch das Java Interface `Connector` erreicht. Sämtliche Attribute und RowSet Connectoren implementieren diese Interfaceklasse. Eine einheitlicher Zugriff ist somit gewährleistet. Das Connector Interface deklariert die essentiellen Methoden zur Connector Initialisierung. Ein Connector erhält über die Methode `setIterator(RowSetIterator)` eine Referenz auf sein BC4J `ViewObject`, aus dem er Dateninformationen auslesen kann. Ebenso wichtig ist die Methode `setDataPanel(DataPanel)`, die dem Connector ein Konstrukt zur Verwaltung von mehreren Connectoren zuweist. Neben diesen „set...“-Methoden sind auch „get...“-Methoden vorhanden, um die zuvor gesetzten Objekte wieder beim Connector zur erfragen. Darüber hinaus besitzt ein Connector noch die Methode `applyEdits()`, welche das explizite Setzen der aktuellen Informationen der grafischen Oberflächenkomponente in die BC4J Schicht durchführt. Nach diesem Aufruf ist sichergestellt, daß keinerlei Informationen nur in der Oberflächenschicht existieren und nicht in die Mittelschicht übertragen wurden. Weitere Methoden der `Connector` Klasse sind dem in Abbildung 6.5 gezeigten UML (Unified Modelling Language) Diagramm sowie der im Anhang angefügten Java Dokumentation der Connector Klassen zu entnehmen.

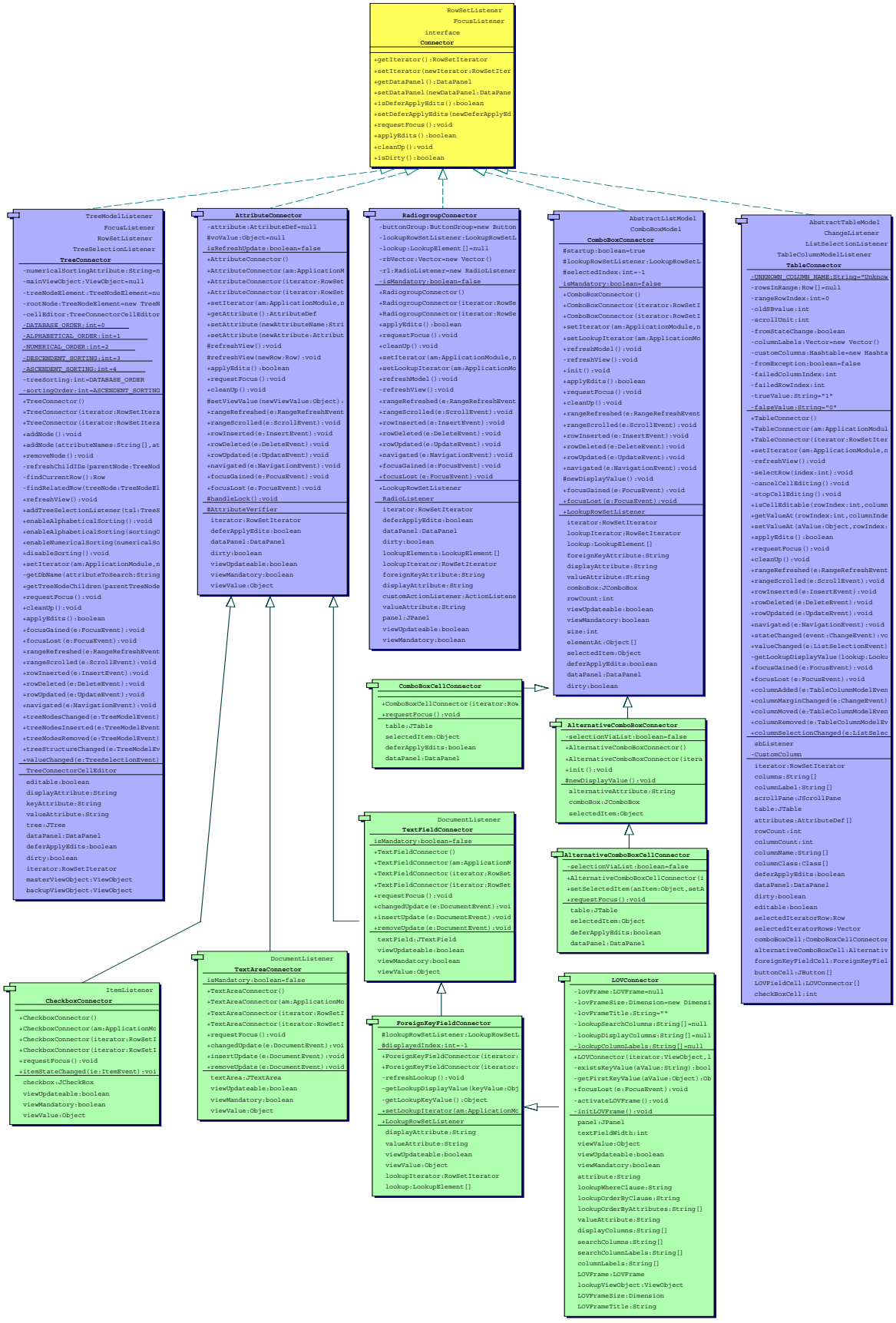
Der wichtigste Grundgedanke, um den in den Anforderungen beschriebenen Kommunikationsprozeß zwischen Oberfläche und Mittelschicht zu realisieren, ist die Verwendung des Beobachter Musters (engl. Observer Pattern) wie es in [Gamma96] beschrieben wird. Ein Connector, der das `Connector` Interface implementiert, muß auch die Methoden des `RowSetListener` und des `FocusListener` Interfaces implementieren, da das `Connector` Interface von diesen Klassen abgeleitet ist. Ein Listener ist in diesem Zusammenhang ein Objekt, das ein Ereignis (engl. Event) empfängt und daraufhin eine Aktion durchführt. Für jeden Ereignistyp gibt es einen zugehörigen Listenertyp.

Indem ein Connector sich sowohl bei seinem ViewObject als auch bei seiner Swing Komponente als Beobachter, bzw. Listener registriert, wird er von diesen über Veränderungen informiert. Wenn sich z.B. in einem `ViewObject` der aktuell betrachtete Datensatz verändert, wird der Connector davon in Kenntnis gesetzt und kann sich den aktuellen Wert aus der Mittelschicht holen und seine Swing Komponente entsprechend aktualisieren.

Neben dem Informationsfluß von Mittelschicht zu Oberfläche wird über den FocusListener auch der umgekehrte Fluß möglich. Wenn beispielsweise ein Anwender Daten in ein Textfeld der Applikation einträgt und dieses Element durch einen Mausklick in ein anderes Feld verläßt, wird der Connector darüber informiert und liest diese Daten aus, um sie in das mit ihm verbundene ViewObject einzutragen. Es findet also eine automatische Synchronisation statt, ohne daß der Entwickler diese Funktionalität programmieren muß. Das Framework nimmt ihm hier enorm viel Entwicklungsaufwand ab.

Eine detaillierte Vererbungshierarchie der Connectoren kann dem in Abbildung 6.5 dargestellten UML Klassendiagramm der Connectoren entnommen werden.

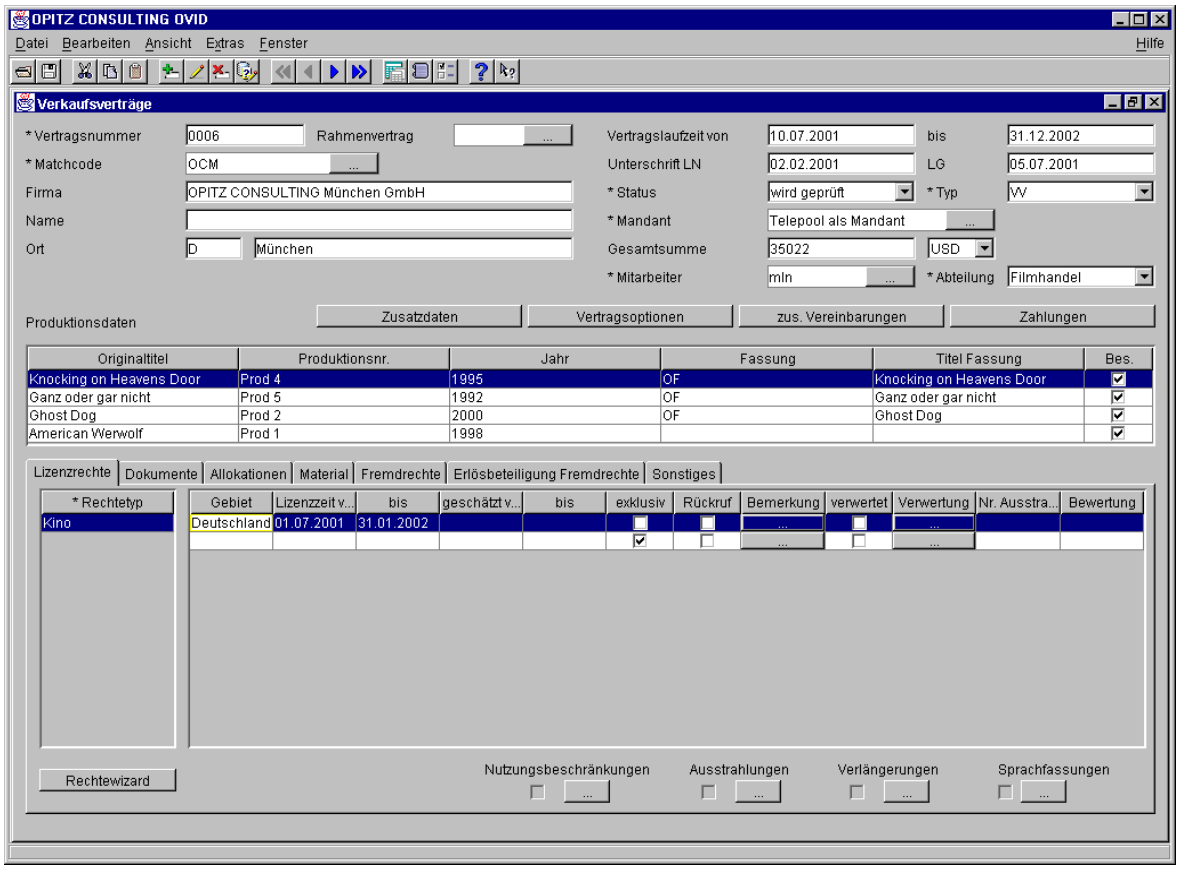
Abbildung 6.5: UML Klassendiagramm der Connectoren des „Apollo“-Frameworks



Durch den objektorientierten Entwurf der Connectoren ist eine einfache Erweiterbarkeit gegeben. Es können weitere beliebige Connectoren entwickelt und leicht in das Framework integriert werden. Für eine Integration ist lediglich die Implementierung des `Connector` Interfaces wichtig. Die Gruppe der Attribut Connectoren kann noch einfacher erweitert werden. Für sie steht bereits eine Vaterklasse `AttributeConnector` bereit. Durch eine objektorientierte Ableitung von dieser Klasse sind schon wesentliche Funktionalitäten eines Connectors realisiert. Es muß lediglich eine Anpassung an die Oberflächenkomponente stattfinden. Als Beispiel sind die `AttributeConnector` und die `TextFieldConnector` Klasse inklusive Dokumentationen im JavaDoc-Format im Anhang enthalten.

Eine Maske mit diversen im „Apollo“-Framework enthaltenen Connectoren zeigt die Abbildung 6.6.

Abbildung 6.6: Eine komplexe Maske des „Apollo“-Frameworks



Quelle: Screenshot des „Apollo”-Frameworks

Im folgenden soll als Beispiel für einen RowSet Connector die TableConnector Klasse näher untersucht werden.

6.4.2.1 Beispiel - Der TableConnector zur Anzeige einer datengetriebenen Tabelle

Der `TableConnector` ist das Verbindungsstück zwischen grafischer Oberfläche und Datenquelle. Der `TableConnector` gehört zur Klasse der sogenannten RowSet Connectoren. Er beruht nicht auf einem einzelnen Datensatz, sondern direkt auf einem ganzen Block an Datensätzen mit gleichen Attributen. Er implementiert das `Connector` Interface, um eine einheitliche Schnittstelle zum gesamten Framework zu schaffen. Somit kann er innerhalb eines DataPanels (Konstrukt zur Verwaltung mehrerer Connectoren auf einem `JPanel`) Anweisungen die aus anderen Komponenten des Frameworks stammen entgegennehmen.

Der `TableConnector` ist mit Hilfe des Model-View-Controller Entwurfsmusters implementiert. Auf Datenquellenseite arbeitet er mit dem Persistenzframework Oracle Business Components for Java (BC4J). Seine grafische Repräsentation (engl. View) wird sowohl durch eine `JTable` als auch durch eine darunterliegende `JScrollPane` realisiert. Er selber ist das Model dieser `JTable` und erweitert das `AbstractTableModel`. Er überschreibt dazu die Methoden:

- `getValueAt(int,int)`
- `setValueAt(Object,int,int)`
- `getRowCount()`
- `getColumnCount()`
- `getColumnName(int)`
- `getCoulumnClass(int)`
- `isCellEditable(int,int)`

Die Methoden `getValueAt(int,int)` und `setValueAt(Object,int,int)` sorgen dafür, daß eine Kommunikation mit der BC4J Mittelschicht stattfindet. Sie holen bzw. setzen die Werte eines BC4J ViewObjects.

Implementierung des `Connector` Interfaces:
Zu den Funktionen des `Connector` Interfaces zählt z.B. das explizite Setzen eines gerade veränderten Oberflächenwertes in der Mittelschicht durch den Aufruf der `applyEdits()` Methode beim Verlassen der Maske. Darüber hinaus müssen auch einige grundsätzliche Methoden zur Verfügung gestellt werden, damit überhaupt Daten angezeigt werden können. Die `setIterator(RowSetIterator)` Methode veranlaßt den `TableConnector` dazu, sich mit einer BC4J Datenquelle zu verbinden. Eine weitere wichtige Methode des Interfaces ist die `setDataPanel(DataPanel)` Methode. Sie speichert lediglich das `DataPanel`, in dem sich die aktuelle Instanz des Connectors befindet. Es kann mittels `getDataPanel()` anschließend wieder über das Framework ausgelesen werden. Dadurch kann das Framework einen Wechsel des aktuellen DataPanels erkennen und es dazu auffordern, die nötigen abschließenden Arbeiten vorzunehmen.

Neben dem Wechsel des gerade bearbeiteten DataPanels kann ebenso nur das aktuelle Oberflächenelement und damit der zugehörige Connector durch den Anwender geändert werden. Für diesen Fall ist die Methode `requestFocus()` notwendig. Sie veranlaßt den Connector z.B. im Falle eines Fehlers bei einem Connectorwechsel dazu, sich wieder den Fokus zu holen und zum aktuellen Connector zu machen. Wenn also ein Validierungsmechanismus z.B. auf Mittelschichtebene fehlschlägt, kann der aktuelle Connector nicht verlassen werden und die Eingabe muß korrigiert werden.

Die Methoden `isDeferApplyEdits()`, `setDeferApplyEdits(boolean)`, `cleanUp()` und `isDirty()` sind zwar implementiert, werden jedoch nicht mit Logik gefüllt, da sie für den `TableConnector` ohne Bedeutung sind. Sie liefern bestenfalls Standardwerte zurück.

Neben dem `Connector` Interface müssen auch das `RowSetListener` Interface und das `FocusListener` Interface implementiert werden. Das `RowSetListener` Interface wird implementiert, um auf Aktionen der Datenquellenseite reagieren zu können und das `FocusListener` Interface, um den Connector innerhalb des Frameworks zum *aktuellen* Connector zu machen. Der *aktuelle* Connector wird innerhalb des Frameworks

gesondert behandelt. Er ist derjenige mit dem der Anwender zu Zeit arbeitet und auf den sich die Aktionen des Anwendungsbenutzers beziehen.

Das BC4J Framework ruft bei Änderungen eines Mittelschichtobjekts die in Tabelle 6.2 aufgeführten Methoden des `RowSetListener` Interfaces auf.

Tabelle 6.2: Beschreibung der Methoden des `RowSetListener` Interfaces

Methode	**Beschreibung**
`rangeRefreshed(RangeRefreshEvent)`	Wird aufgerufen, wenn sich das Anzeigeintervall eines ViewObjects geändert hat.
`rangeScrolled(ScrollEvent)`	Wird aufgerufen, wenn das Anzeigeintervall eines ViewObjects bewegt wurde
`rowInserted(InsertEvent)`	Wird aufgerufen, wenn ein neuer Datensatz in ein ViewObject eingefügt wurde.
`rowDeleted(DeleteEvent)`	Wird aufgerufen, wenn ein Datensatz aus einem ViewObject gelöscht wurde.
`rowUpdated(UpdateEvent)`	Wird aufgerufen, wenn ein bestehender Datensatz in einem ViewObject geändert wurde.
`navigated(NavigationEvent)`	Wird aufgerufen, wenn sich der aktuelle Datensatz des ViewObjects geändert hat.

Sobald eine dieser Methoden von BC4J aufgerufen wird, bedeutet dies, daß sich in der Mittelschicht Veränderungen ergeben haben. Meistens folgt aus diesen Veränderungen, daß die Darstellung der Daten der Mittelschicht angepaßt werden muß. Es ist also eine Aktualisierung der Tabelle notwendig. Sie wird von der Methode `refreshView()` vorgenommen, indem sie die View Komponente der Tabelle dazu auffordert ihre Anzeige neu aufzubauen.

Der `TableConnector` realisiert weitere Funktionen mit dem Observer Pattern nach [Gamma96]. Er implementiert auch die Java Swing Interfaces `ChangeListener`, `ListSelectionListener` und `TableColumnModelListener` und bekommt somit die Ereignisse der `JScrollPane` und der Tabellenkomponenten `ListSelectionModel` und `TableColumnModel` mitgeteilt. So führt zum Beispiel das Verändern des angezeigten Tabellenbereichs über die Scrollleiste dazu, daß neue Datensätze aus der Mittelschicht in die Oberfläche geholt und in der Tabelle angezeigt

werden. Ebenso wird eine Auswahl einer neuen Zelle innerhalb der Tabelle von dem Connector registriert und an die BC4J Schicht weitergegeben. Dort wird ebenfalls der aktuell markierte Datensatz geändert.

Die Attribute eines Datensatzes können in einer `JTable` nicht nur in Textform angezeigt werden. Swing bietet hierzu Schnittstellen, die den Einsatz von diversen Komponenten zur Anzeige einer Tabellenspalte ermöglichen. Diese Funktionen müssen auch für die Daten einer Datenbank zur Verfügung stehen. Über die folgenden Methoden und weitere Connectoren kann auch der `TableConnector` einen solchen Mechanismus abbilden:

- `setComboBoxCell(int, ComboBoxCellConnector)`
- `setAlternativeComboBoxCell(int, AlternativeComboBoxCellConnector)`
- `setForeignKeyFieldCell(int, ForeignKeyFieldConnector)`
- `setButtonCell(int, JButton)`
- `setLOVFieldCell(int, LOVConnector)`
- `setCheckBoxCell(int)`

Er setzt entsprechend der durch den Entwickler vorgenommenen Konfiguration die passenden Anzeige- und Bearbeitungselemente der Tabelle. Somit können z.B. zweiwertige Wertelisten als Auswahlknopf in Form einer `JCheckBox` angezeigt werden.
Des weiteren müssen nicht immer sämtliche Spalten des bei der Initialisierung des TableConnetors angegebenen Spalten angezeigt werden. Eine Beschränkung der Anzeige auf bestimmte Spalten kann durch die Methode `setColumns(String[])` und die Angabe der Attributnamen erfolgen.

Alle bedeutenden Methoden wurden zuvor erklärt. Darüber hinaus enthält der `TableConnector` noch weitere Methoden zur Konfiguration sowie weitere interne Hilfsmethoden, deren Funktionsweise hier allerdings nicht näher erläutert werden soll. Sie sind im Anhang in Form des Quellcodes der `TableConnector` Klasse und deren JavaDoc Dokumentation enthalten.

Verwendung:

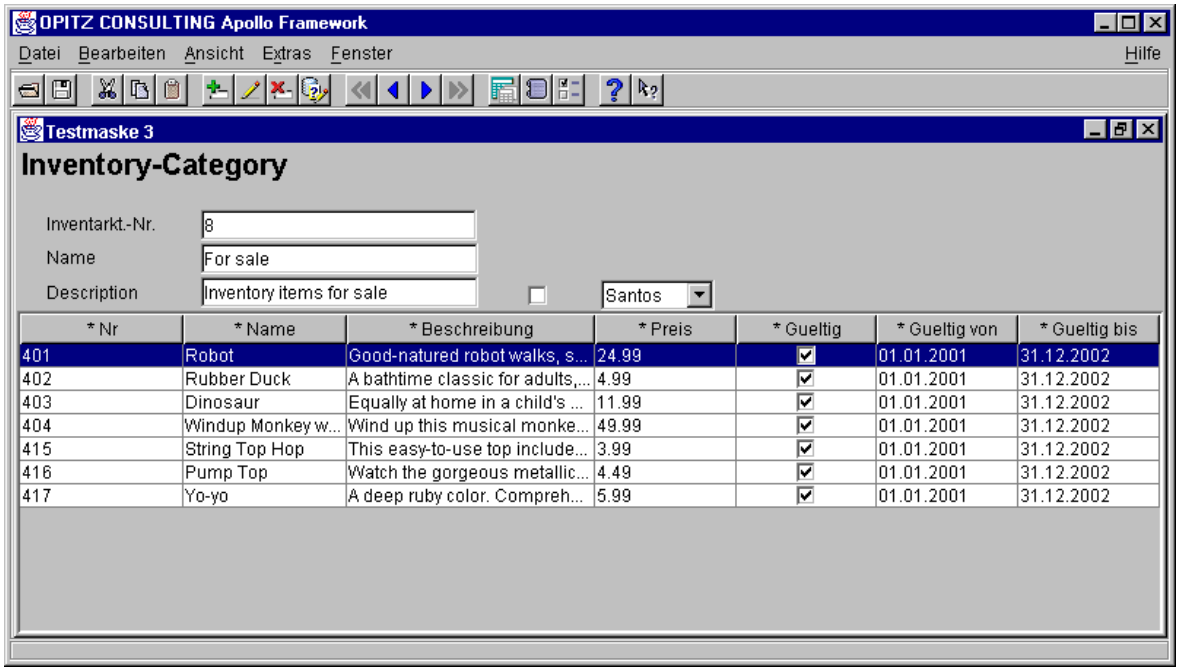
Der `TableConnector` wird zunächst über den parameterlosen Konstruktor und die entsprechenden „set...“-Methoden oder direkt über einen Konstruktor mit allen notwendigen Parametern initialisiert. Dazu zählen die `JTable`, das BC4J `ViewObject` und die `JScrollPane`. Der Java Container `JScrollPane` muß die `JTable` dabei enthalten. Im Anschluß definiert der Entwickler noch die anzuzeigenden Attribute und die Spaltenüberschriften. Ggf. können nun noch verschiedene Anzeige- und Bearbeitungsmodi für einzelne Spalten gewählt werden. Letztendlich muß der Connector nur noch dem `DataPanel` zu dem er gehören soll, hinzugefügt werden. Die `JScrollPane` kann nun wie ein normales Swing Objekt behandelt und in eine grafische Oberfläche integriert werden.

Nachfolgend wird beispielhaft die Konfiguration einer Tabellenansicht zur Anzeige von Inventargegenständen durchgeführt:

```
TableConnector detailTableConnector = new TableConnector();
JTable detailJTable = new JTable();
JScrollPane detailJScrollPane = new JScrollPane();
detailJScrollPane.getViewport().add(detailJTable, null);

detailTableConnector.setTable(detailJTable);
detailTableConnector.setScrollPane(detailJScrollPane);
detailTableConnector.setIterator(detailViewObject);

detailTableConnector.setColumns(new String[]{"InviId","InviName",
                                             "InviDescription","InviPrice",
                                             "InviOnhand","InviValidFrom",
                                             "InviValidUntil"});
detailTableConnector.setColumnLabel(0,"Nr");
detailTableConnector.setColumnLabel(1,"Name");
detailTableConnector.setColumnLabel(2,"Beschreibung");
detailTableConnector.setColumnLabel(3,"Preis");
detailTableConnector.setColumnLabel(4,"Gueltig");
detailTableConnector.setColumnLabel(5,"Gueltig von");
detailTableConnector.setColumnLabel(6,"Gueltig bis");

detailTableConnector.setCheckBoxCell(4);

addConnector(detailTableConnector);
```

Abbildung 6.8 zeigt den Screenshot einer nach dem zuvor beschriebenen Schema erstellten datengetriebenen Tabelle innerhalb des „Apollo“-Frameworks.

Abbildung 6.8: Mit dem `TableConnector` erstellte Tabelle auf einer Maske



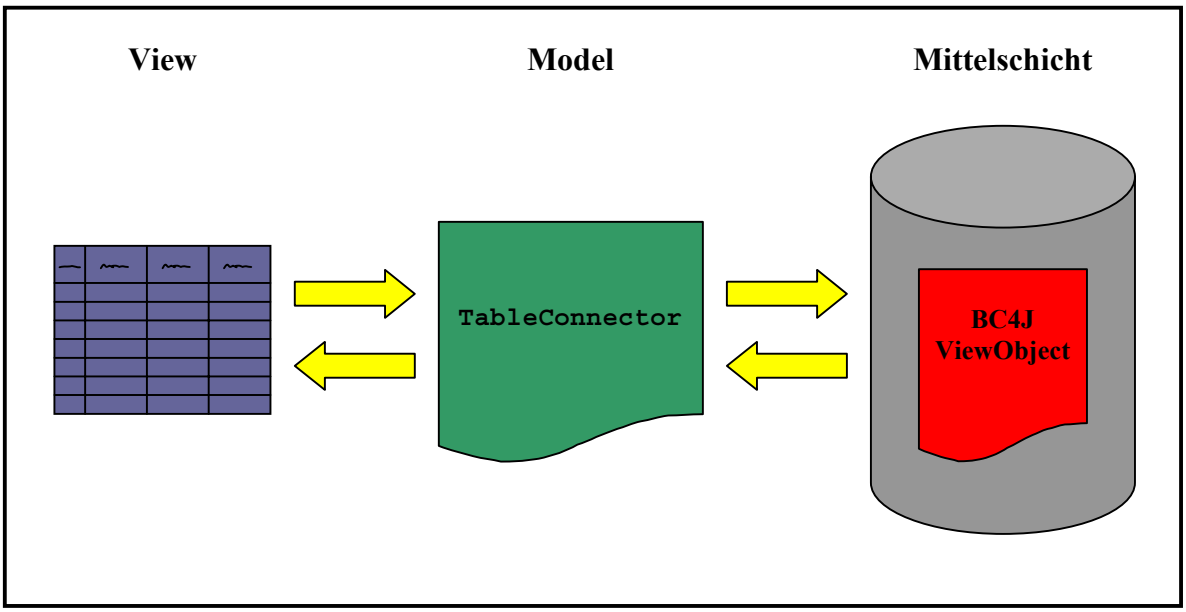
Quelle: Screenshot des „Apollo”-Frameworks

Um einen abschließenden Überblick über den Datenfluß einer Applikation zu gewinnen, wird dieser in Abbildung 6.9 dargestellt. Der Benutzer einer mit „Apollo“ erstellten Anwendung startet eine Maske, auf der eine Tabelle vorhanden ist. Diese Tabelle wird zur Anzeige gebracht. Die Tabelle oder vielmehr ihre View Komponente befragt dazu ihr Model (in diesem Fall der `TableConnector`) nach dem Inhalt der Zellen, die zur Zeit auf dem Bildschirm sichtbar sind. Der TableConnector besorgt sich nun diese Daten aus seinem `ViewObject`, speichert sie intern und liefert sie an die View zurück. Diese zeigt sie schließlich an.

Navigiert der Anwender nun auf der Tabelle oder nimmt in ihr Veränderungen vor, delegiert die View diese Aktionen wieder an ihr Model, den `TableConnector`, welcher darauf mit einer Änderung der Mittelschicht reagiert.

Ähnlich funktioniert auch der entgegengesetzte Weg. Sobald durch eine andere Komponente des Frameworks eine Veränderung an der Mittelschicht durchgeführt wurde, informiert sie den `TableConnector` darüber. Dieser reagiert darauf und fordert seine View dazu auf, die Daten neu und aktualisiert anzuzeigen.

Abbildung 6.9: Kommunikation zwischen `JTable`, `TableConnector` und BC4J



7 Schlußbetrachtung

Die Auseinandersetzung mit den Problemen im Softwareentwicklungsprozeß führte zu der Erkenntnis, daß es mittlerweile diverse Ansätze gibt, um die bekannten Schwächen der Softwareentwicklung zu beheben bzw. sie zu relativieren. Leider wurde auch klar, daß es keine Allheilmittel gegen diese Unzulänglichkeiten gibt. Erfreulicherweise finden neben den rapiden Entwicklungen der Softwaretechnologie auch immer mehr Forschungen und Verbesserungen im Bereich der Softwareentwicklung statt. Es werden immer ausgefeiltere Arten eines Vorgehensmodells entwickelt. Dabei zeichnet sich ein Trend ab, bei dem nicht nur neue, alternative Vorgehensweisen erforsch, sondern auch bereits bewährte miteinander kombiniert werden. Dies zeigt sich auch im Gesamtaufbau dieser Arbeit. Zunächst werden die grundlegenden Methoden der Objektorientierung vorgestellt, anschließend diverse Wiederverwendbarkeitstechniken diskutiert und schließlich miteinander in Kombination gebracht.

Die Softwareentwicklung unter zu Hilfenahme von Mustersystemen bildet heute eine unabdingbare Grundlage der wirtschaftlichen Softwareentwicklung. Sie zeigt sich als etabliertes Mittel, bestehende Erkenntnisse wiederzuverwenden. Besonders bei einer relativ jungen Programmiersprache, wie Java, fallen diese Muster auch in den Standardklassenbibliotheken auf. Die Java Swing Komponenten sind fast gänzlich mit Hilfe solcher Muster entwickelt worden.

In den letzten Jahren haben auch viele der beschriebenen Komponentensysteme den Markt erobert. Auch sie sind also nichts neues mehr. Häufig erfüllen sie jedoch ihren eigentlichen Zweck der hochgradigen Wiederverwendbarkeit nicht. Die Praxis hat gezeigt, daß viele Softwaresysteme nur sehr rudimentären Gebrauch von Komponenten machen. Zu diesen Themen erscheinen zwar ununterbrochen neue Artikel in diversen Entwicklerzeitschriften und in der Fachliteratur, deren Umsetzung in die unternehmerische Praxis ist jedoch eher selten. Vielmehr scheinen sie einen Blick in die Zukunft der Softwareentwicklung aufzuzeigen. Der Einsatz neuer komponentenbasierter Systeme wie z.B. CORBA ruft häufig noch eine Art Mißtrauen hervor. Dieses ist angesichts der erschreckend hohen Anzahl gescheiterter Softwareprojekte aber auch kein Wunder, wenn die enormen Entwicklungskosten solcher Anwendungen betrachtet werden.

Vielversprechend sind allerdings die Ansätze bei der Entwicklung von Frameworks. Mittlerweile entstehen Frameworks auch in der kommerziellen Softwareentwicklung und nicht ausschließlich in den Forschungslaboren von Universitäten und Großunternehmen. Sie haben dennoch einen signifikanten Nachteil. Ihre Entwicklung ist extrem aufwendig und teuer. Sie lohnt sich also erst, wenn der mehrfache Einsatz dieses Frameworks vorhersehbar ist.

Betriebswirtschaftlich betrachtet kann der Einsatz neuer weitestgehend unbekannter Technologien leicht zu einem Desaster führen. Ist dieses Risiko im Hinblick auf die Chancen einer solchen Anwendung dann überhaupt tragbar? Generell läßt sich hierfür wohl keine Antwort finden, aber unter Betracht der hier diskutierten und analysierten Aspekte von Softwarewiederverwendbarkeit besteht die berechtigte Hoffnung, mit diesen Technologien enorme Kosteneinsparungen in der teuren Individualsoftwareentwicklung zu erzielen.

Weiterhin hat die Erfahrung im Umgang mit Softwareprojekten gezeigt, daß häufig eine chaotische Vorgehensweise in der Entwicklung von Software mit objektorientierten Technologien vorherrscht. Das Potential dieser Systeme wird verkannt und somit ist ebenfalls der Nutzeneffekt dieser Technologien nicht mehr vorhanden. Eine traditionelle Vorgehensweise würde mit weniger Risiko und Aufwand zu dem gleichen Ergebnis führen.

Wichtig erscheint also die *konsequente* Nutzung von Wiederverwendbarkeitstechnologien im objektorientierten Umfeld. Erst dann schöpfen frameworkbasierte Systeme ihre vollen Möglichkeiten aus.

Die heutzutage als Produkte vertriebenen Frameworks sind den traditionell entwickelten Softwaresystemen bei weitem überlegen. Die Entwicklung zu sogenannten „Plug-and-Play“ Systemen, wie sie in den letzten Jahren in der Hardwarebranche stattgefunden hat, setzt sich nun augenscheinlich auch im Softwarebereich fort. So können viele der aktuellen Frameworks schon durch einfachen Austausch von Komponenten angepaßt werden.

Zu bemerken ist allerdings auch, daß der Markt zwar enorm viele technische Frameworks von diversen Herstellern bietet, aber kaum fachliche oder domänenspezifische Frameworks zu finden sind. Eine deutliche Erweiterung dieses Angebotes wäre sehr wünschenswert. Es hat den Anschein, als würden zwar viele dieser Systeme firmenintern zur Anwendungsentwicklung genutzt, aber keines erreicht einen verkaufsfertigen Produktstatus. Einen ersten Ansatz und die Vorreiterrolle auf diesem Gebiet hat hier das „San Francisco“ Framework von IBM. Ansonsten bleibt es den Firmen selbst überlassen, ein geeignetes Framework zu entwickeln. Da die Unternehmen diese Frameworks nicht extern vermarkten, leidet somit oft die Qualität eines solchen Frameworks.

Ähnliche Erfahrungen wurden auch bei der Entwicklung des „Apollo“-Frameworks gemacht. Oft waren auch hier die Überlegungen aus Kosten- und Zeitgründen sehr stark einem konkreten Anwendungszweck untergeordnet. Dennoch wurden große Teile des Frameworks sehr flexibel und erweiterbar entworfen und implementiert. Es dient somit als Grundlage für die Anwendungsentwicklung von Individualsoftwaresystemen im Client-Server Umfeld. Es unterstützt dabei auch die Verwendung von mehrschichtigen Architekturen mit Komponentensystemen wie CORBA und Enterprise Java Beans (EJB).

Am deutlichsten hat die Entwicklung des „Apollo“-Frameworks jedoch gezeigt, daß das in diesem Dokument vorgestellte Vorgehensmodell bei der Frameworkentwicklung ein Idealbild darstellt, welches sich in der Praxis nur sehr schwer umsetzen läßt. Nur durch eine iterative Vorgehensweise und die Nachkonzeptionierung während der Realisierungsphase war eine Implementierung im Sinne des Frameworks möglich.

Zur Zukunft von Frameworks in der objektorientierten Softwareentwicklung läßt sich sagen, daß sie einen immer größeren Stellenwert bei der Entwicklung komplexer Anwendungssystem einnehmen werden. Mit ihrer Hilfe lassen sich viele Aufwände auf eine einmalige, wenn auch aufwendigere Durchführung beschränken. Unternehmen können somit einen enormen Marktvorteil erreichen, weil sie Applikationen durch kürzere Entwicklungszeiten kostengünstiger anbieten können.

Es soll an dieser Stelle auch herausgestellt werden, daß Frameworks keine Wunder vollbringen können und auch kein Allheilmittel gegen sämtliche Unannehmlichkeiten der Softwareentwicklung bieten. Sie werden in der Zukunft jedoch eine signifikante Position im Entwicklungsprozeß einnehmen und die Verwendung konventioneller Systeme einschränken.

Anhang

Literaturverzeichnis

[Alexander77] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel: *A Pattern Language.* Oxford University Press, New York, 1977.

[Bensberg01] Frank Bensberger, Lofi Dewanto: „Mille Plateaux". *JAVA Magazin Nr.4*, 2001, S. 74-80.

[Buschmann98] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: *Pattern-orientierte Software-Architektur: Ein Pattern-System.* Addison-Wesley-Longman, Bonn (et al.), 1998.

[Birrer95] A. Birrer, W. R. Bischofberger, T. Eggenschwiler: „Wiederverwendung durch Framework-Technik - vom Mythos zur Realität". *OBJEKTSpektrum Nr.5*, 1995, S.18 - 26.

[Campbell93] Roy H. Campbell, Nayeem Islam: „A technique for documenting the framework of an object-oriented system", *Computing Systems*, Herbst 1993.

[Davis88] Alan M. Davis: „A comparison of techniques for the specification of external systems behavior". *Comunications of the ACM 31*, September 1988, S.1098-1115.

[Fayad99a] Mohamed E. Fayad, Douglas C. Schmidt, Ralph E. Johnson: *Building Application Frameworks*. 1. Aufl., John Wiley & Sons, Inc., New York , 1999.

[Fayad99b] Mohamed E. Fayad, Ralph E. Johnson: *Domain-Specific Application Frameworks*. 1. Aufl., John Wiley & Sons, Inc., New York , 1999.

[Floyd87] C. Floyd: „Outline of a Paradigm Change in Software Engineering". In: *Computer and Democracy - A Scandinavian Challenge*. Hrsg.: G. Bjerknes, P. Ehn und M. Kyng. Gower Publishing Company Limited, Avebury, 1987, S.191 - 210.

[Gamma96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl., Addison-Wesley, Bonn, 1996.

[Griffel98] Frank Griffel: *Componentware - Konzepte und Techniken eines Softwareparadigmas*. 1. Aufl., dpunkt.verlag, Heidelberg, 1998.

[Heuer97] H. Heuer-Hasenpatt, B. Hollunder, H.-B. Kittlaus, N. Schumacher: „Bausteinorientierte Anwendungsentwicklung", *OBJEKTSpektrum Nr.3*, 1997.

[Jacobson92] Ivar Jacobson M. Christerson, Patrik Jonsson, G. Övergaard: *Object-oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley Publishing Company, Reading, MA., 1992.

[Jacobson97] Ivar Jacobson, Martin Griss, Patrik Jonsson: *Software Reuse – Achitecture, Process and Organization for Business Success*. ACM Press, New York, 1997.

[Johnson88] Ralph E. Johnson, B. Foote: „Designing Reusable Classes", *The Journal of Object-Oriented Programming*, Juni/Juli 1988, S.22 - 35.

[Johnson97] Ralph E. Johnson: „Frameworks = (Components + Patterns)". *Comunications of the ACM 40*, Oktober 1997, S.39.

[Kilberth94] K. Kilberth, G. Gryczan, H. Züllighoven, D. Bäumer, R. Budde, K. Hasbron-Blume, K.-H. Sylla, V. Weimer: *Objektorientierte Anwendungsentwicklung*. Vieweg, 1994.

[Lichter93] H. Lichter: *Entwicklung und Umsetzung von Architekturprototypen für Anwendungssoftware*. Verlag der Fachvereine Zürich, 1993.

[Oestereich98] Bern Oestereich: *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified modeling language*. 4. Aufl., R. Oldenbourg Verlag, München, 1998.

[OOTC97] IBM Object-Oriented Technology Center: *Developing object-oriented software: an experience-based approach*. Prentice Hall, Englewood Cliffs, 1997.

[Pree97] Wolfgang Pree: *Komponenetenbasierte Softwareentwicklung mit Frameworks*. 1. Aufl., dpunkt.verlag, Heidelberg, 1997.

[Reenskaug96] T. Reenskaug, Per Wold, Odd Arild Lehne: *Working with Objects*. Manning, Greenwich, 1996.

[Rumbaugh91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen: *Object-Oriented Modelling and Design*. Prentice-Hall, Englewood Cliffs, 1991.

[Schmidt97] Douglas C. Schmidt: „Applying Design Patterns and Frameworks to Develop Object-Oriented Communication Software“, In: *Handbook of Programming Languages*. Hrsg.: Peter Salus. Mcmillan Computer Publishing, New York, 1997.

[Szyperski97] C. Szyperski: *Component Software Beyond Object-Oriented Programming*. 1. Aufl., Addison-Wesley Publishing Company, Reading, MA., 1997.

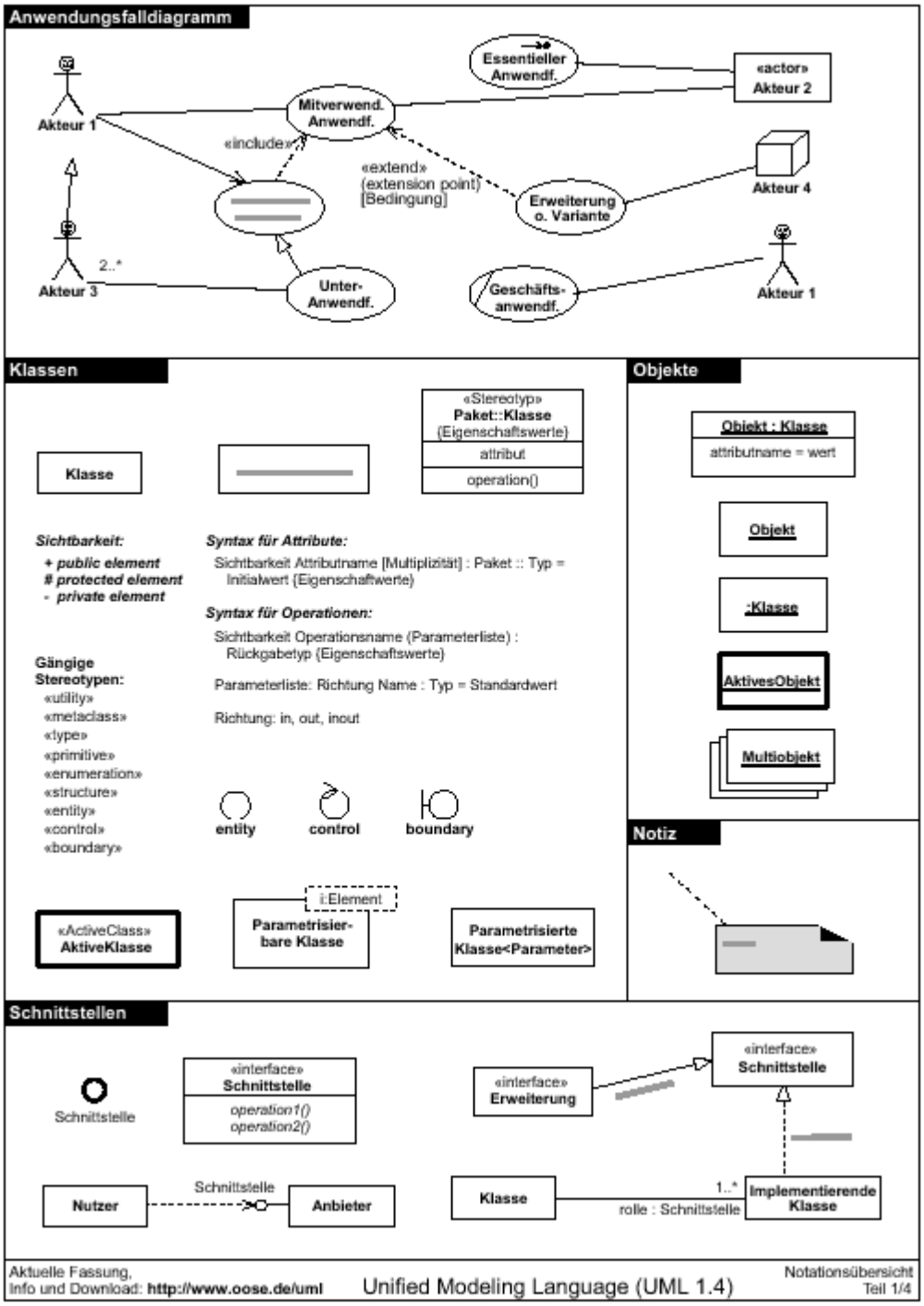
[Taligent95] Taligent: *The Power of Frameworks*. 1. Aufl., Addison-Wesley Publishing Company, Reading, MA., 1995.

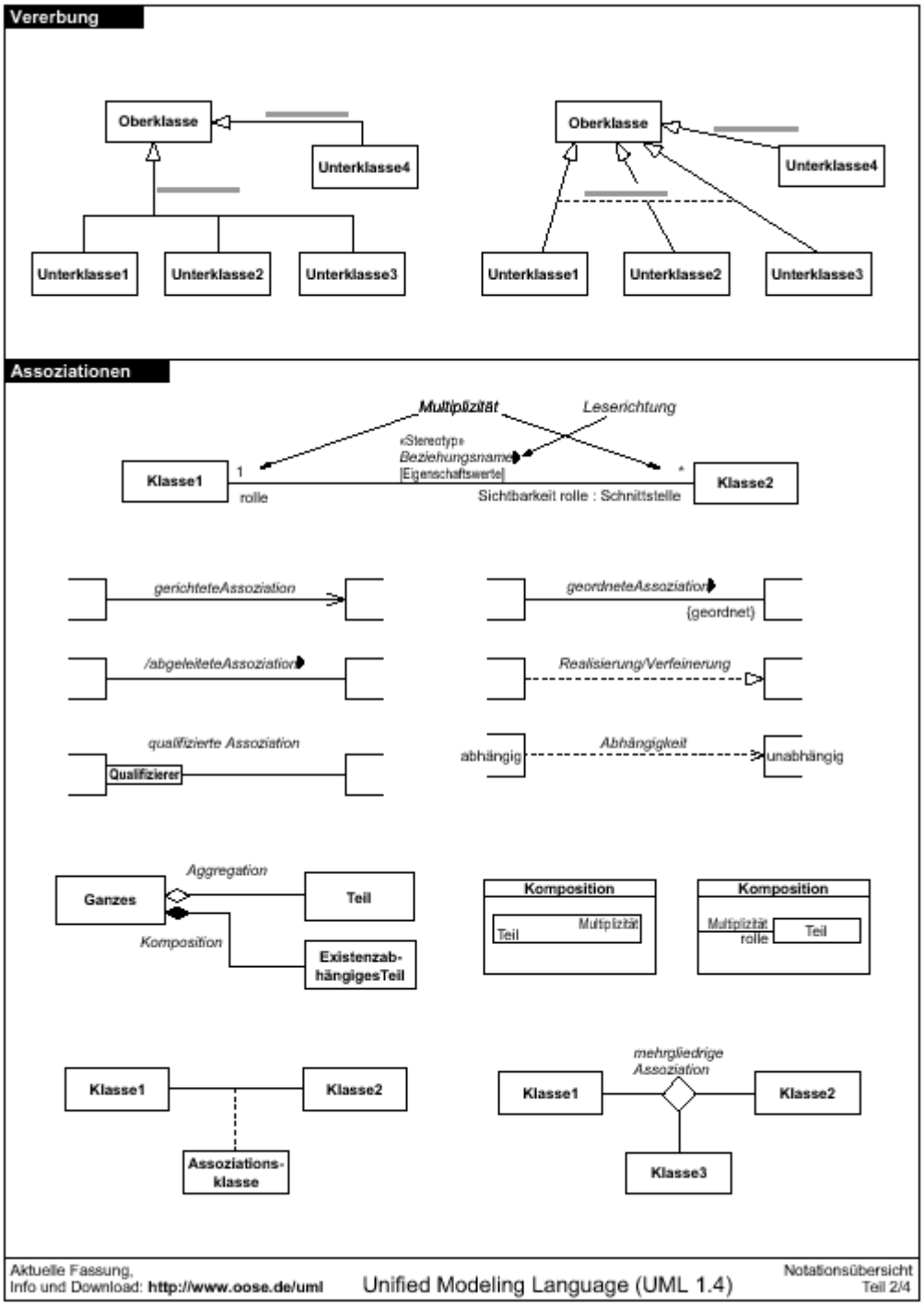
[Weinand89] A. Weinand, E. Gamma, R. Marty: „Design and Implementation of ET++, a Seemless Object-Oriented Application Framework“, *Structured Programming*, Springer Verlag, 1989.

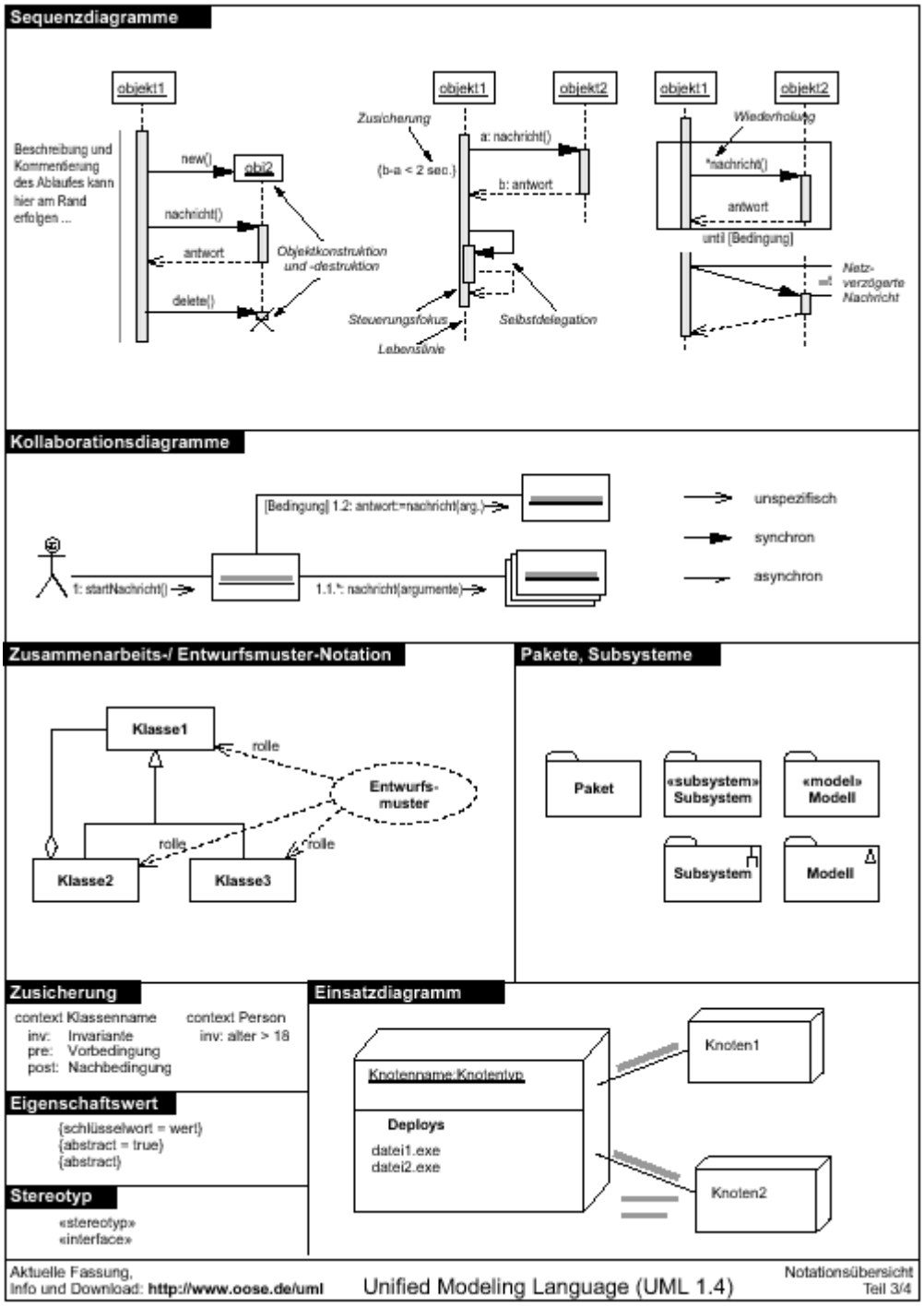
[Yourdon89] Edward Yourdon: *Modern Structured Analysis*. Yourdon Press, Englewood Cliffs, 1989.

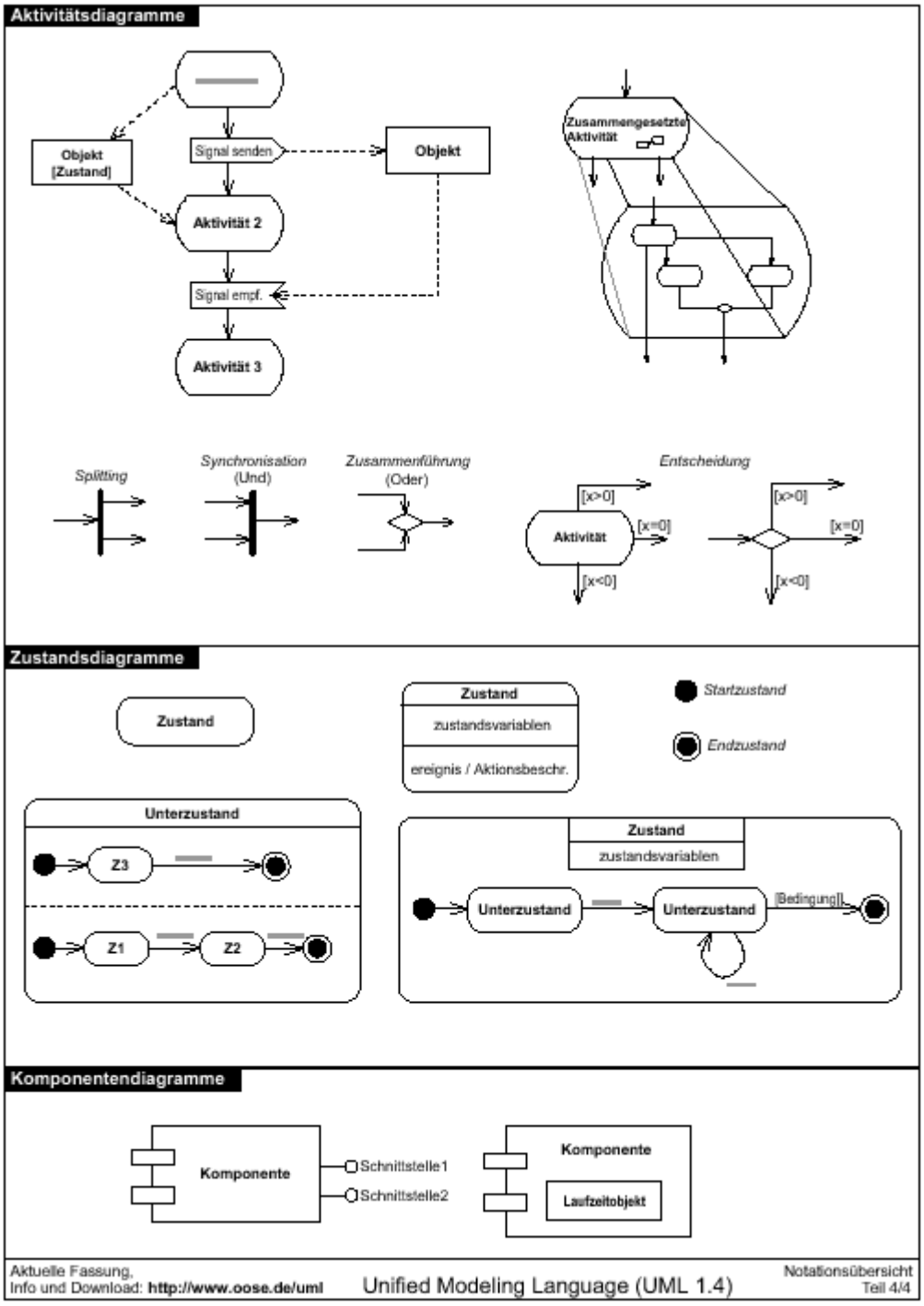
[Yourdon96] Edward Yourdon: *Rise and Resurrection of the American Programmer*. Yourdon Press, Prentice Hall, Englewood Cliffs, 1996.

Die UML-Notation




Vererbung
Oberklasse
Unterklasse4
Unterklasse1
Unterklasse2
Unterklasse3
Oberklasse
Unterklasse4
Unterklasse1
Unterklasse2
Unterklasse3
Assoziationen
Multiplizität
Leserichtung
«Stereotyp»
Beziehungsname
[Eigenschaftswerte]
Klasse1
1
rolle
*
Klasse2
Sichtbarkeit rolle : Schnittstelle
gerichteteAssoziation
geordneteAssoziation
{geordnet}
/abgeleiteteAssoziation
Realisierung/Verfeinerung
qualifizierte Assoziation
Qualifizierer
abhängig
Abhängigkeit
unabhängig
Aggregation
Ganzes
Teil
Komposition
Existenzab-
hängigesTeil
Komposition
Multiplizität
Teil
Komposition
Multiplizität
rolle
Teil
Klasse1
Klasse2
Assoziations-
klasse
mehrgliedrige
Assoziation
Klasse1
Klasse2
Klasse3
Aktuelle Fassung,
Info und Download: http://www.oose.de/uml
Unified Modeling Language (UML 1.4)
Notationsübersicht
Teil 2/4


Sequenzdiagramme
objekt1
Beschreibung und Kommentierung des Ablaufes kann hier am Rand erfolgen ...
new()
obj2
nachricht()
antwort
delete()
Objektkonstruktion und -destruktion
objekt1
objekt2
Zusicherung
(b-a < 2 sec.)
a: nachricht()
b: antwort
Steuerungsfokus
Selbstdelegation
Lebenslinie
objekt1
objekt2
Wiederholung
*nachricht()
antwort
until [Bedingung]
Netz-verzögerte Nachricht
Kollaborationsdiagramme
[Bedingung] 1.2: antwort:=nachricht(arg,)
1: startNachricht()
1.1.*: nachricht(argumente)
unspezifisch
synchron
asynchron
Zusammenarbeits-/ Entwurfsmuster-Notation
Klasse1
rolle
Entwurfs-muster
rolle
rolle
Klasse2
Klasse3
Pakete, Subsysteme
Paket
«subsystem» Subsystem
«model» Modell
Subsystem
Modell
Zusicherung
context Klassenname
inv: Invariante
pre: Vorbedingung
post: Nachbedingung
context Person
inv: alter > 18
Eigenschaftswert
{schlüsselwort = wert}
{abstract = true}
{abstract}
Stereotyp
«stereotyp»
«interface»
Einsatzdiagramm
Knotenname:Knotentyp
Deploys
datei1.exe
datei2.exe
Knoten1
Knoten2
Aktuelle Fassung,
Info und Download: http://www.oose.de/uml
Unified Modeling Language (UML 1.4)
Notationsübersicht
Teil 3/4



Quelle: Bernd Oestereich, UML-Notationsübersicht,
Online im Internet: URL: http://www.oose.de/uml/ (Abfrage am: 26.8.2001)

Der Musterkatalog nach Gamma

In der folgenden Tabelle soll wird kein ausführlicher Musterkatalog dargestellt, wie in Gamma beschreibt. Es soll vielmehr eine Übersicht über die 23 von Gamma in [Gamma96] beschriebenen Muster gegeben werden.

Tabelle A.1: Beschreibung der von Gamma erwähnten Muster

Musterbezeichnung	**Musterzweck**
Abstract Factory	Fasse Familien zusammengehöriger Klassen, möglicherweise aus verschiedenen Klassenhierarchien, zusammen und schreibe eine Klasse, deren Methoden Exemplare dieser Klassen erzeugen.
Adapter	Falls Klassen nützlich sein könnten, aber ihre Schnittstelle ungeeignet ist, schreibe eine Klasse mit guter Schnittstelle, deren Methoden die schlechte Schnittstelle benutzen.
Bridge	Bilde unabhängige Hierarchien für Schnittstellen und Implementierungen, die an der Wurzel zusammenhängen.
Builder	Trenne den Konstruktionsprozeß eines komplexen Objekts von dessen Repräsentation, so daß man die Repräsentation frei wählen kann.
Chain of Responsibility	Lasse eine Aufgabe nicht direkt an einen Bearbeiter übergeben, sondern lasse sie durch eine Hierarchie von Bearbeitern wandern, bis sich einer zuständig fühlt.
Command	Kapsle einen Befehl als ein Objekt. Dies ermöglicht es, Klienten mit verschiedenen Anfragen zu parametrisieren, Operationen in eine Queue zu stellen, ein Logbuch zu führen und Operationen rückgängig zu machen.
Composite	Zerlege Baumstrukturen in Teilbäume und behandle diese wie einzelne Blätter. Füge Objekte zu Baumstrukturen zusammen, um Teil-Ganzes-Hierarchien zu repräsentieren. Das Kompositionsmuster ermöglicht es Klienten, sowohl einzelne Objekte als auch Kompositionen von Objekten einheitlich zu behandeln.
Decorator	Baukastenprinzip: Ein Objekt wird in weitere Objekte, die Funktionalität hinzufügen, eingebunden, anstatt Unterklassen für die erweiterte Funktionalität zu schreiben.
Facade	Schreibe eine Klasse, die zur Bedienung eines komplexen Systems dient. Sie greift aus einer Ansammlung von Objekten die wichtigen Methoden heraus und macht sie nach außen verfügbar. Dadurch wird von der Implementierung (Strukturierung des Problems in Klassen) abstrahiert.

Factory Method	Definiere eine Klassenschnittstelle mit Operationen zum Erzeugen eines Objekts, aber lasse Unterklassen entscheiden, von welcher Klasse das zu erzeugende Objekt ist. Fabrikmethoden ermöglichen es einer Klasse, die Erzeugung von Objekten an Unterklassen zu delegieren.
Flyweight	Wenn es zu aufwändig ist, so viele Objekte zu erzeugen, wie man gern möchte, lasse ein Objekt viele Bestandteile verwalten.
Interpreter	Mache Kommandosprachen flexibel, indem die Grammatik der Sprache eingegeben und interpretiert wird.
Iterator	Ermögliche den sequentiellen Zugriff auf die Elemente eines zusammengesetzten Objekts, ohne seine zugrundeliegende Repräsentation offenzulegen.
Mediator	Ein Objekt vermittelt zwischen mehreren Objekten, damit nicht jeder jeden kennen muß.
Memento	Ein Objekt benutzt ein anderes, um seinen aktuellen Zustand zu speichern. Es kann damit später diesen Zustand wieder herstellen.
Observer	Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so daß die Änderung des Zustands eines Objekts dazu führt, daß alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.
Prototype	Lasse eine allgemeine Klasse Objekte erzeugen, indem sie vorgegebene Prototypen kopiert.
Proxy	Kontrolliere den Zugriff auf ein Objekt mit Hilfe eines vorgelagerten Stellvertreterobjekts.
Singleton	Von manchen Klassen darf es nur ein Exemplar geben. Lasse diesen Mechanismus von der Klasse selbst verwalten.
State	Lasse das Verhalten eines Objekts sich mit dem Zustand ändern. Führe eine abstrakte Klasse mit mehreren konkreten Unterklassen ein. Operationen ersetzen ein Exemplar der einen Unterklasse durch ein Exemplar einer anderen Unterklasse.
Strategy	Packe verschiedene Algorithmen für das gleiche Problem in verschiedene Klassen mit gemeinsamer Schnittstelle. Ein Benutzer kann nach Wunsch die Klassen austauschen.
Template Method	Beschreibe einen allgemeinen Algorithmus, lasse aber konkrete Schritte offen. Die Schritte sind abstrakte Methoden, die in Unterklassen überschrieben werden.
Visitor	Lasse die Operationen zu den Elementen kommen statt umgekehrt: Schreibe eine abstrakte Klasse für Operationen mit konkreten Unterklassen.

Quelle: In Anlehnung an Klaus G. Barthelmann, Online im Internet: URL: http://www.informatik.uni-mainz.de/~barthel/Java/Lektionen_/12/Lektion12.html (Abfrage am: 26.8.2001)

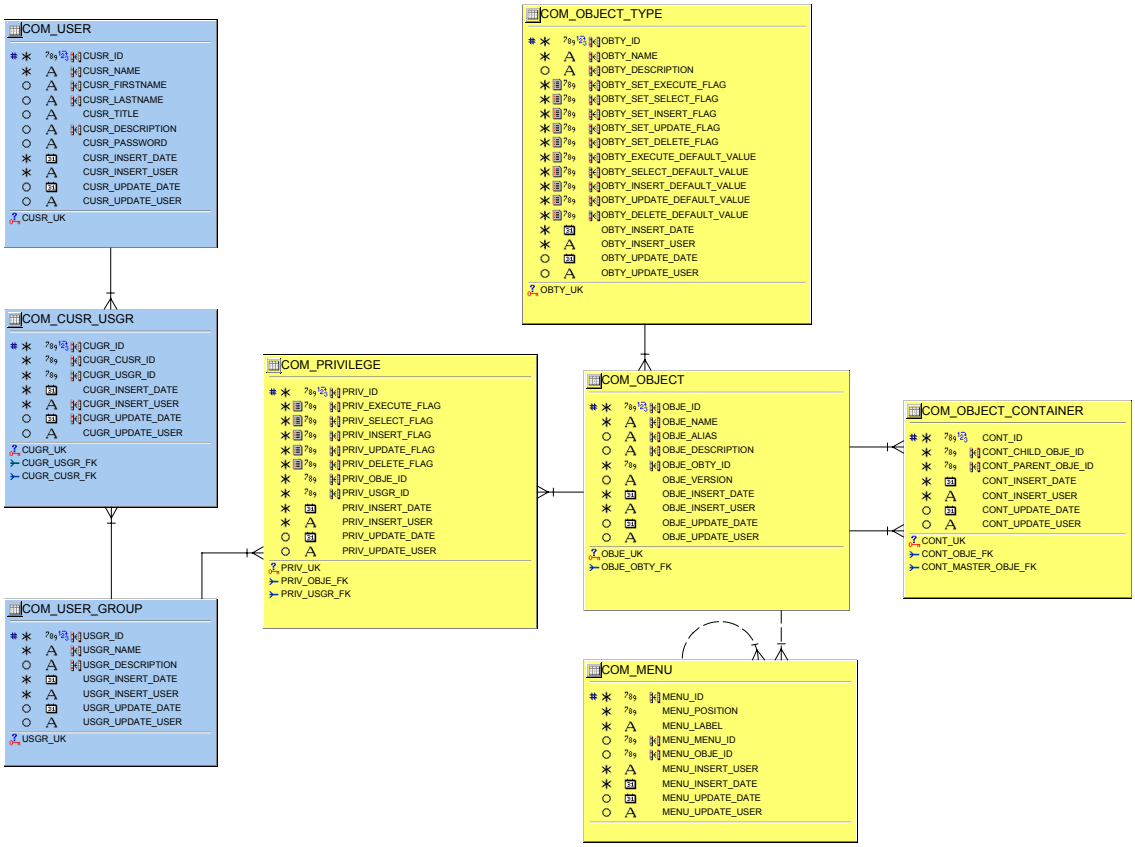
Java Quellcodedokumentation der Connectoren

Die Dokumentation findet sich auf der CD-ROM im Anhang dieses Dokumentes.

Quellcode der Klasse `AuthorizationService`

Der Quellcode findet sich auf der CD-ROM im Anhang dieses Dokumentes.

OCF (Opitz Consulting Framework) Datenmodell



Quellcode der Klasse `SessionManager`

Der Quellcode findet sich auf der CD-ROM im Anhang dieses Dokumentes.

Quellcode der Klasse `Connector`

Der Quellcode findet sich auf der CD-ROM im Anhang dieses Dokumentes.

Quellcode der Klasse `AttributeConnector`

Der Quellcode findet sich auf der CD-ROM im Anhang dieses Dokumentes.

Quellcode der Klasse `TextFieldConnector`

Der Quellcode findet sich auf der CD-ROM im Anhang dieses Dokumentes.

Quellcode der Klasse `TableConnector`

Der Quellcode findet sich auf der CD-ROM im Anhang dieses Dokumentes.